Dataset Augmentation and Mixture-Of-Experts Working In Concert For Few-Shot Domain Adaptation Transfer Learning

Stanford CS224N Default Project - RobustQA track

Leif Jurvetson
Department of Computer Science
Stanford University
leijurv@stanford.edu

Rafael Esteves
Department of Computer Science
Stanford University
resteves@stanford.edu

Abstract

Despite the significant improvements in NLP in the last few years, models can still fail to work well on test sets which differ, even a small amount, from their training sets. Few shot learning is an important goal in creating generalizable neural network models. In this paper we explore ways to increase the few shot learning performance of a model by implementing a few variations meant to improve generalizability; specifically we measure the effects of data augmentation and mixture of experts on a pre-trained transformer BERT model. Mixture of experts is a technique in which separate models are trained to be responsible for different sub tasks within a problem. We find that this change is able to remove the interference between out-of-domain datasets during training and increase performance from F1 48.43 to 51.54. Data augmentation applied for NLP is a technique in which words within a piece of text are added, removed, or replaced in an effort to increase the variance in training data. This method was found to be a valuable tool in further improving expert learning, increasing the overall F1 score further to 52.07, however it did not improve the baseline model when used on its own.

1 Introduction

Recent work in Natural Language Processing has revealed that while models have made strong improvements on training results, they are not always able to generalize well to novel data points in the test set. Out-of-domain learning is a fundamental challenge in applying neural network models to real world problems which do not have large data sets easily available. Furthermore, question-answering is an essential task because it is a very good metric for measuring a models understanding of language. By applying this rigorous task to measure out-of-domain performance, we are able to create strong evaluation metrics for the generalizability of a model. Question-answering is a task in which a model receives a context and a sequence of questions about the context; a model solving this task must choose a subset of the context that correctly answers the question. Our model is built by taking a pre-trained transformer model [1], finetuning it on a large in-domain dataset, and finetuning it again on a very small out-of-domain dataset.

Our project investigated two main approaches to improve few-shot adaptation to new domains from this baseline training process. First, we implemented dataset augmentation, which replaces certain words in questions and contexts with other similar words in order to expose the model to larger and more diverse out-of-domain datasets. This experimental variation gave mixed results, and we cannot claim to have achieved repeatable statistically significant improvements on dev set metrics from dataset augmentation when implemented on its own, however we do present findings on some methods of dataset augmentation that are decidedly more or less productive than others. Second, we implemented mixture-of-experts by training several 'expert' models each on one of the out-of-domain

datasets. Using mixture-of-experts allowed the models to focus on one specific new domain at a time. We find that this change produces significant improvements over the baseline by 3 percentage points.

Finally, we were able to fit in a small amount of trials of the two approaches in conjunction. We actually find that dataset augmentation, when applied "one domain at a time", can provide meaningful and measurable improvements over the raw training set. By using mixture-of-experts to segment this out, we get a "best of both worlds" result, by only applying dataset augmentation to domains where it improves results.

2 Related Work

Mixture of experts is a procedure which was first described in Adaptive Mixtures of Local Experts [2], which was initially slow and led to poor generalization. The authors explain that by updating each expert individually, it is possible to avoid the issue of interference between tasks. More recently, the concept of mixture of experts was applied to domain adaptation in Multi-Source Domain Adaptation with Mixture of Experts [3]. The authors found that a single out-of-domain data set might be better modeled using a mixture of experts approach where each expert is trained on a individual in-domain data set. The results are particularly improved when augmented with adversarial training, specifically an adversarial loss is added to minimize the distance between the in-domain and the out-of-domain data sets. In this paper we expand on previous mixture of expert studies by using the method along with a pre-trained transformer model and data augmentation.

The BAE paper[4] describes various mutations that can be made to a sentence to adversarially perturb a model attempting to classify the sentence as real or augmented until misclassification occurs. The authors find that they are able to reduce the accuracy of transformer models such as BERT by over 80% on some data sets. Their approach consists of modifying sentences, iteratively and maliciously making the minimal number of changes in order to maximize the misclassification rate of a given classifier model, continuing until the entire sentence is mispredicted. We take inspiration from this approach, but diverge onto our own route for augmenting our data.

3 Approach

3.1 Baseline

We leverage the baseline code provided by the RobustQA default project track which uses the pre-trained DistilBERT model[5] (which is a smaller version of the original BERT model, which was a state of the art model in NLP just a few years ago[1]). From our starting point, the models are then finetuned on the in-domain data sets and subsequently on our out-of-domain data sets. The model takes in as input a question passage pair (q, p) and is expected to output probability distributions over the start and end points, p_{start} and p_{end}.

BERT has a restriction that it can only encoder context of size 512 or less. In order to allow our model to answer questions based on an arbitrarily sized input we use chunking, which splits given question passage pairs, (q, p) into chunks of size 384 with stride 128. More specifically, given $q = \{q_0, q_1, ...q_{10}\}$ and $p = \{p_0, p_1, ..., p_{500}\}$. We convert this into chunks $c1$ and $c2$ where $c1 = [CLS]q[SEP]p^1[SEP]$ with $p^1 = \{p_0, p_1, ..., p_{371}\}$ and $c2 = [CLS]q[SEP]p^2[SEP]$ with $p^2 = \{p_{128}, p_{129}, ..., p_{499}\}$. Each chunk is labelled with its start end end indices, in order to calculate its offset with the rest of the question.

Our loss function is the negative log-likelihood loss for the start and end locations. Given the optimal start and end locations i^*, j^* respectively our loss is given by equation (1)

$$\mathcal{L} = -\log p_{\text{start}}(i^*) - \log p_{\text{end}}(j^*) \quad (1)$$

Under this framework, we experimented with various tweaks to the model for the final learning on the 381 out-of-domain datapoints.

3.2 Dataset Augmentation Approach

The objective of using data augmentation was to introduce more variance into the training out-of-domain data sets in order to improve the generalization of our model. Determinng the correct

approach for dataset augmentation proved initially difficult. This was for three main reasons: a difference in goals between what we desire and what the BAE paper was trying to accomplish, difficulty finding the correct "abstraction" for augmentation as it relates to the structure of the starter code, and of the main BERT model itself. Another issue which affected this was the RobustQA track restriction that we must only use DistillBERT with no additional/external training data or pretrained data added.

Our task is different from that of the BAE paper described above because we do not use the data augmentation in an adversarial setting but a cooperative one. This means that even though the BAE paper demonstrates extremely significant improvements for "replace + insert" over just "replace", that doesn't necessarily mean we want to do that. Furthermore, with our particular setup with the starter code, inserting a token becomes quite difficult. This is specifically because of the chunking procedure described above. If we want to insert a token, we can insert it after this splitting and have a very nasty problem of shifting everything over, which entails moving one token to the next chunk, updating all start and end positions, among other things. Or, we can insert the new token to the sentence in textual form, prior to the tokenizer, and have the same complicated alignment issues (e.g. with viewing specific examples in tensorboard, and all answer pointers must be updated to match how all the text has now been shifted)

Given the lack of evidence that inserting tokens would improve performance, and the difficulty of doing so, our approach was simply to replace tokens.

To perform replacements we use a pre-trained `DistilBertForMaskedLM` and we use the same tokenizer as the baseline model. This tokenizer maps our input textual data into a sequence of token identifiers, known an `input_ids`. We apply a "sliding window" approach, in which one token at a time is replaced with `<MASK>` and the model is run on this input which then attempts to predict which token best fills the masked out token. In order to improve the performance of this, we implemented dynamic batching in which subsections of a given example are operated on. While iterating through an example and replacing tokens, we build up a Tensor of the `input_ids` and run it 32 at a time through the model. At the end of the example we are augmenting, we run potentially less than 32 at the final step. We then perform a standard softmax to calculate the specific output logits corresponding to the one index we masked out. This can be thought of as a diagonalized indexing into the output: the Xth run of the is the sentence with the Xth input token masked, therefore we want the output values at that Xth index. We sample the top 3 most likely replacements as decided by BERT for every token in the example, then remove any entries that predict the real masked token (to discard any options that change nothing from the original). We aggregate the top 100 replacements in the form of `(softmax_logit_value, index, old_token, new_token)`, sorted by the softmax logit value. Here we are picking the 100 replacements that BERT is the most confident of. For example, if there is a part of speech that truly has no valid replacement, BERT will produce a strong output for the only correct word and very low outputs for any variants. These options will naturally be sorted very low, and will be overtaken by better alternative replacements elsewhere in the example. By sorting among all possible replacements, we ensure that we aren't "forcing" the augmenter to run on tokens where it has no confident alternative token to replace it with.

We cache this information to disk at this point, using pickle. This allows us to iterate on specific dataset augmentation variants and hyperparameters without needing to recompute BERT's replacement predictions, which despite the $2\times$ time improvement from batching is still a time consuming process.

When actually performing these replacements, there are some heuristics that we use to choose which replacements we will enact. As the BAE paper describes, when a token is entirely masked out, BERT loses semantic meaning from the sentence and doesn't always reconstruct it correctly. By masking out a key but unintuitive (so to speak) word, BERT may confidently predict a filling of that mask token that has a completely opposite meaning. Concretely on our training set, we observed that naively performing replacements according just to the softmax score caused replacements such as "subsidiary" to "parent" (in the context of `minority interest ... is the portion of a subsidiary corporation's stock`), which indeed does change the meaning of the sentence. We need to preserve semantic meaning wherever possible because our task is question-answering, and we may not be able to predict which tokens are key to understanding the sentence. We then devised a superior (as we will detail in

the Results section) method for picking which token replacements should be used. The BAE paper solves this by ensuring that the overall meaning is not changed too much, by calculating a similarity metric between the original and replaced data. We wanted to do something similar, but our project requirements mandate that we not use any pre-trained model other than DistilBERT so we could not use, for example, word2vec paired with a vector similarity metric. We instead opted to unearth the parameters of our pretrained DistilBERT model itself, and extract the word embeddings from the main embedding layer. BERT uses positional embeddings as well as word embeddings, however the positional embeddings will be constant (we are just replacing in-place, not moving anything), so we opted to just look at word embeddings. By using Python reflection to extract individual layers from the module and get the embedding matrix, we were able to get a rough word2vec equivalent without violating the rules against external training. We then perform cosine similarity and re-rank our replacement options according to how similar they are to the original word. Cosine similarity is a natural choice here, and it turned out to work well with BERT's first layer embeddings (as will be discussed further in Analysis). Recall that all our replacement options at this stage are selected from the top 3 BERT predictions at each location, with the true token removed as an option; cosine similarity is only used to rank the options generated by the masked language model. We take the top K augmentations and apply them individually: for every augmentation, we copy the entire example and apply that specific replacement. In our final model, we specifically take the original dataset plus the top three distinct most cosine-similar replacements, for a total outcome of 4x dataset size. Order is then randomized and the model is trained as normal.

3.3 Mixture of Experts Approach

The approach for mixture of experts was fairly standard. We separately train three models, then create a new evaluation mode that reads from three different models on disk. These three are passed into a new `nn.Module` that we implemented that switches between the three expert models and reassembles their outputs into a `Seq2SeqQuestionAnsweringModelOutput`.

The mixture of experts task can be summarized using equation (2)

$$p(y|x) = \sum_{i=1}^{N} g_i(x) f_i(x) \quad (2)$$

where y is the question answering output, x is the question along with its context. Furthermore $g(x)$ is the output of a gating function, which produces a specific weight for each expert, and $f_i(x)$ is the i^{th} experts estimate of $p(y|x)$.

In our mixture of experts experiments we use 3 different experts, one for each out-of-domain data set. Each of these experts is fine tuned on only their respective data set using the in-domain baseline described above in section 3. The gating function was chosen to be a deterministic one hot encoding of the data set from which the question was sourced. This means that only one expert is chosen for each question. We are able to use the source of the question by labelling each data point from the set it was taken from.

4 Experiments

4.1 Data

In this project we use three in-domain reading comprehension datasets, namely Natural Questions, NewsQA, and SQuAD, and three out-of-domain datasets, specifically RelationExtraction, DuoRC, and RACE.

4.2 Evaluation method

The two evaluation metrics used in this paper are EM and F1. EM is an abbreviation for exact match which compares the model response to the labelled response and returns a boolean of whether they match or not. F1, a less strict metric, is the harmonic mean of precision and recall. As a reminder precision is a fraction of correct answers among every model answer and recall is a fraction of number of correct model answers over the number of correct answers. The F1 metric can be computed as in

Dataset	Question Source	Passage Source	Train	dev	Test
		`in-domain datasets`			
SQuAD [5]	Crowdsourced	Wikipedia	50000	10,507	-
NewsQA [7]	Crowdsourced	News articles	50000	4,212	-
Natural Questions [6]	Search logs	Wikipedia	50000	12,836	-
		`oo-domain datasets`			
DuoRC [9]	Crowdsourced	Movie reviews	127	126	1248
RACE [10]	Teachers	Examinations	127	128	419
RelationExtraction [11]	Synthetic	Wikipedia	127	128	2693

Figure 1: The train, dev, and test split of the in-domain and out-of-domain models.

equation (3).

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (3)$$

4.3 Experimental details

Learning rates used for testing ranged from 2e-7 to 3e-6, but we settled quickly on 1e-6 as a good middle ground, as that resulted in the time for the model to peak in performance to roughly match the limits of patience (approximately 45 minutes per iteration).

4.4 Results

On the test set, we achieved 59.548 F1 and 42.477 EM with our final model. As we have only submitted to `test` the once, all other numbers, and all graphs, refer to the `val` set. The final model was chosen by performance on each individual domain (as detailed in the Analysis section): the three experts were the original baseline, the baseline finetuned for `duorc`, and the baseline finetuned on our dataset augmentation of `relation_extraction`.

We report marginal improvements over baseline. The baseline provides 48.43 F1 and 33.25 EM on `val`, and our final model provides 52.07 F1 and 38.22 EM on `val`. The baseline has never been run on the `test` set, so we cannot compare there.

As judging by our position on the leaderboard as of the time of writing (Wednesday night), we are in 8th out of 57 on test for EM yet 30th for F1. We are also 8th out of 87 on val for EM, yet 14th for F1. This is difficult to investigate because we don't know what the correct examples are on the test set. We are satisfied with this EM performance, but are a little confused why our F1 performance seems to be so comparable to EM on `val` yet so much worse on `test`.

Some of our results for other versions of this model prior to mixture of experts, but including data augmentation:

Datasets	Learning Rate	Augs	Aug Mode	F1	EM
all three indomain	N/A	N/A	N/A	48.43	33.25
all three oodomain	3.00E-06	3	softmax	49.05	34.55
all three oodomain	2.00E-07	3	softmax	48.84	34.77
all three oodomain	3.00E-07	N/A	N/A	49.36	34.29
all three oodomain	3.00E-07	1	softmax	49.12	34.29
just relation	1.00E-06	N/A	N/A	71	51.56
just race	1.00E-06	N/A	N/A	40.04	28.13
just duorc	1.00E-06	N/A	N/A	43.47	33.33
all three oodomain	5.00E-07	3	softmax	48.93	33.77
all three oodomain	5.00E-07	3	cosine	49.19	34.03
all three oodomain	2.00E-06	3	cosine	49.07	34.29
all three oodomain	2.00E-06	1	cosine	49.23	34.29
all three oodomain	2.00E-06	3	cosine	49.07	34.29
all three oodomain	1.00E-06	3	cosine	49.2	34.03
just relation	1.00E-06	3	cosine	72.56	53.13

We find that looking at the final F1 and EM numbers is less elucidating, and that analyzing the graphs of F1 and EM across epochs is much more fruitful. This is because the code only saves the highest F1 score, so, for example as in Figure 6 below, we see that the cosine similarity variant scored higher than the nonaugmented variant in EM, but this was never saved, because the model that was saved was a few iterations earlier, when the F1 score peaked.

5 Analysis

It's important to remember that this is a fine tuning improvement upon a extremely high quality baseline.

As detailed earlier, our method was to iterate upon various ways to fine tune our baseline and compare the outcomes on the validation set.

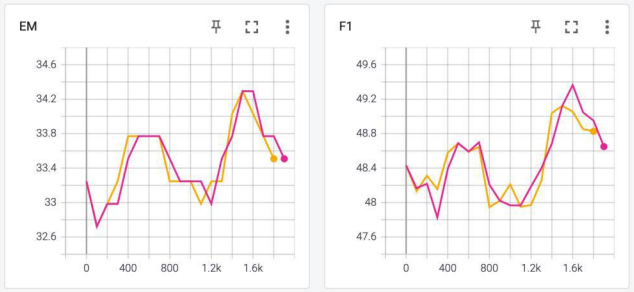

Figure 2: A sampling of two similar models trained during hyperparameter search, to demonstrate an interesting pattern in performance over time.

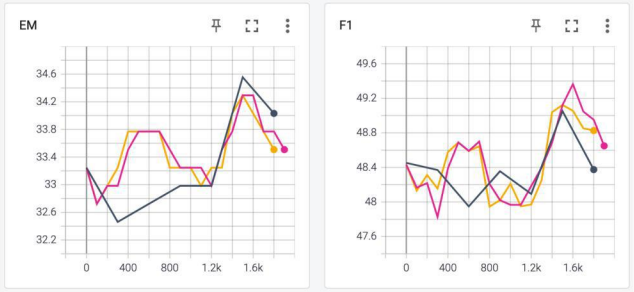

Figure 3: A third model that displayed only one of the behaviors.

As you can see in Figure 2, we were presented with a perplexing loss structure in which two humps appear. Additionally, some other approaches (see Figure 3) displayed only the first behavior and not the second.

The completion of the mixture of experts model gave us insight into why this is the case. The two humps that we see are actually two different domains, which we can separate. `relation-extraction` is responsible for the late increase and `duorc` is responsible for the first hump. We can see this by plotting the domains trained individually against this behavior, see Figure 4. Furthermore we can explain the model in Figure 3, which performs worse at the begining of training than the other two by noting that a significantly higher learning rate on `duorc`, causes the lack of the first increase in the score, both in the overall model and the expert.

This separation allows for several key benefits. First, we can choose where in the training process to select our model. The starter code picks the best F1 score to save to disk, but we can clearly see that the peak happens at a different epoch for each data set. We believe that most of our improvements from mixture of experts comes from the fact that we are able to harness the early improvement in `duorc` and the later improvement from `relation-extraction`; in other words, we are able to avoid the interference described in Adaptive Mixtures of Local Experts [2].

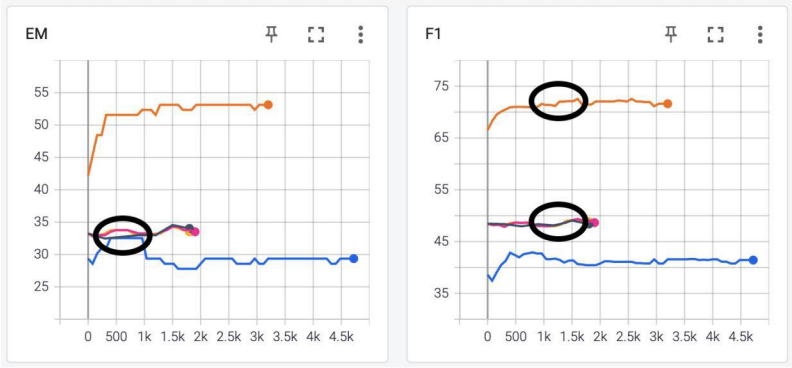

Figure 4: Same combined models in the center, blue is `duorc`, orange is `relation-extraction`. Annotations have been added to highlight the effect of the individual domains on the combined model performance: first we see `duorc`'s peak around 500 to 1k reflected in the combined model; later we see a smooth increase in `relation-extraction` also appear in the combined model.

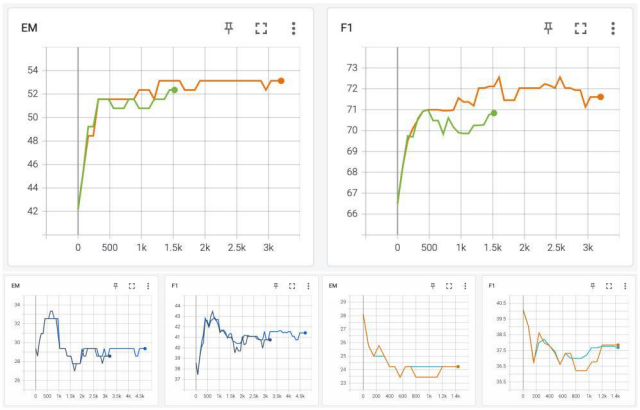

Figure 5: Training outcomes on the three oodomains: `relation-extraction`, `duorc`, and `race`, respectively. Augmented is orange/light blue/cyan, nonaugmented is green/dark blue/orange.

It appears that enabling data augmentation may actually decrease performance on the baseline model trained on the three combined domains (see Figure 6). However, when cosine similarity is used in conjunction with the masked language model for data augmentation it provides marginal improvements . In the case where data augmentation with cosine similarity was applied to the mixture of experts model, noticeable benefits were noted in the `relation-extraction` expert (see Figure 5). For `duorc`, the augmented training has a slightly lower variance in performance over time, but does not quite achieve the same scores at its best as without data augmentation. Finally, for `race`, we see a particularly difficult domain. We found that the best expert for this data set was the baseline, as none of our finetuning was able to improve `val` performance.

We also performed qualitative analysis on our embedding cosine similarity approach for dataset augmentation. Extracting an embedding layer then using its output to compare tokens is nonstandard. Embedding layers in the case of BERT are trainable parameters. One thing that we considered as a reason against this approach is that BERT may well have the primary computational power of the model in its transformer and attention layers. It's perfectly possible, theoretically, for BERT to work just as well as it does with a randomized embedding layer frozen, simply because the rest of BERT is an enormous pile of complex machine learning parameters that might learn to deal with the poor embeddings (such as in reservoir computing [6]). Word2vec is different in the sense that it is forced to produce high quality embeddings as that is its explicit target. For this reason we were careful to qualitatively look at debug output of the old and new rankings of replacements to ensure that it was a clear improvement. We specifically saw some direct token replacements (such as % becoming `percent`), replacements of parts of speech (such as `the` swapped with `a`, `that`

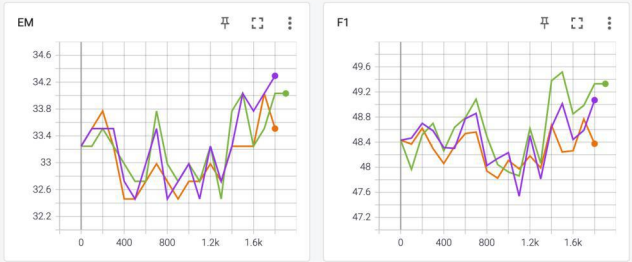

Figure 6: Purple is cosine similarity, green is no data augmentation, orange is softmax ordered augmentation. While cosine seems to be comparable to the nonaugmented model with each taking one of F1 and EM, the softmax augmented model is clearly inferior to both.

swapped with `which`), and synonyms (such as `generally` becoming `usually` or `typically`). Most importantly, the replacements with antonyms had been correctly re-sorted to near the bottom; we rarely saw them in any of the top-k samplings, and certainly never in the top three. Therefore we can conclude qualitatively that our data augmentation was improved by switching from softmax to cosine-based ranking. This conclusion is backed up by our quantitative analysis above (see Figure 6)

6 Conclusion

Our project explored ways to improve resilience of models to domain transfer and fine tuning. We report positive results from mixture of experts, and mixed results from dataset augmentation. We explored various metrics for choosing between token replacement candidates, with a clear answer in favor of cosine similarity of word embeddings.

Our final conclusion is that dataset augmentation in combination with mixture of experts can provide benefits for finetuning a model for few-shot transfer learning. We present improvements in F1 score from 48.43 to 52.07, and EM score from 33.25 to 38.22.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[3] Jiang Guo, Darsh J. Shah, and Regina Barzilay. Multi-source domain adaptation with mixture of experts. *CoRR*, abs/1809.02256, 2018.

[4] Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based adversarial examples for text classification, 2020.

[5] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[6] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

A Appendix (optional)

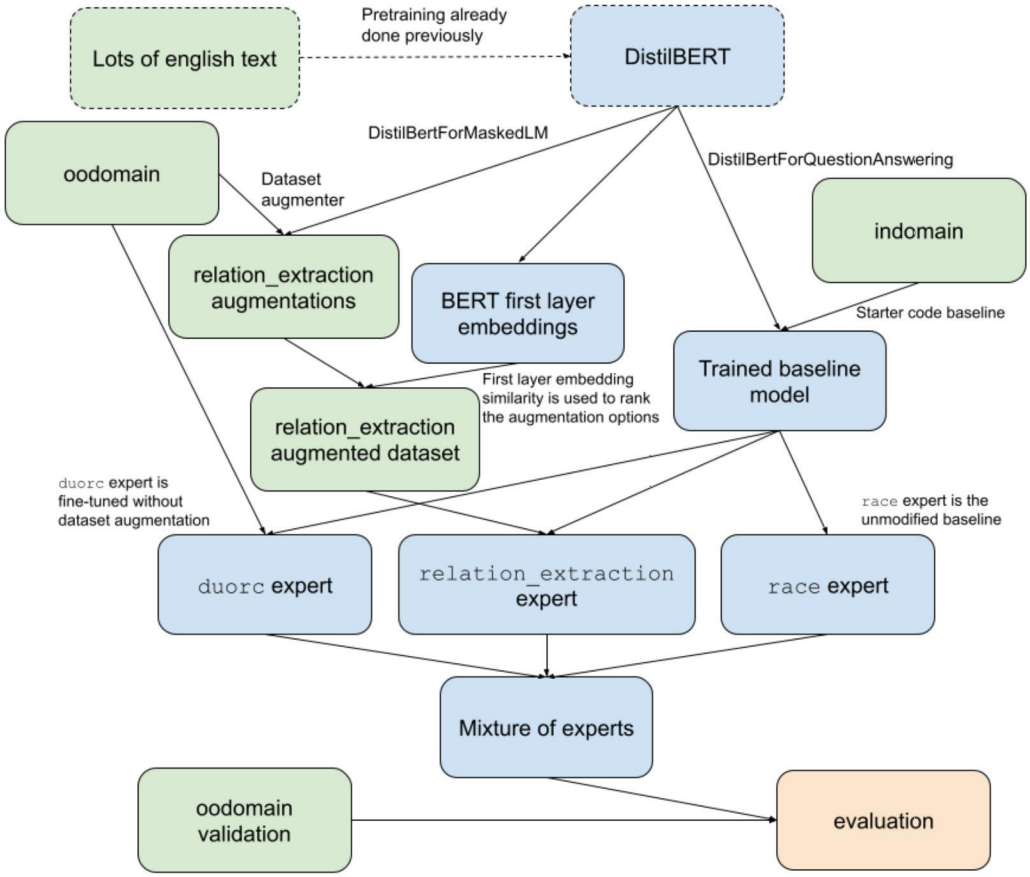

Figure 7: The data flow in our final model (as submitted to the `test` leaderboard).