

Default Final Project: RobustQA Track

Stanford CS224N {Default} Project

Rui Hu

Department of Computer Science
Stanford University
rhu2@stanford.edu

Abstract

Our goal is to build a question answering system that can adapt to unseen domains with only a few training samples from the domain.. We experimented with several approaches, including mixture of experts approach and various techniques to fine tune the pre-trained model better. Although we are able to outperform the baseline, we found that model architecture is less important when it comes to improving performance. Relevant training data is by far the most important factor. Various fine tune techniques also help to some extend.

1 Key Information to include

- Mentor: Zihan Wang
- External Collaborators (if you have any):
- Sharing project:

2 Introduction

Over the last few years, we have seen tremendous progress on fundamental natural language understanding problems. At the same time, there is increasing evidence that models learn superficial correlations that fail to generalize beyond the training distribution. On the other hand, humans can easily generalize beyond their training distribution—while not strictly from our training distribution, we can effortlessly understand novels set in fictional worlds and quickly understand the meanings of new words. How can we build NLP systems that generalize like humans? From a practical perspective, robustness to out-of-distribution data is critical for building accurate NLP systems in the real world since train and test data often come from distinct user interactions. [from official project handout]

In the project handout, we were given several research papers as suggested starting points with the warning that this is a fairly new field, so the suggested approaches may not all lead to improvements over the baseline.

We decided to try the mixture of experts approach suggested by one of the papers first because it made intuitive sense. Our original hope was that the mixture of experts will deliver solid improvements over the baseline, then we can employ fine tuning techniques suggested by a different paper to further improve performance. As we will detail later, it turned out that the mixture of expert approach does not improve the performance at all, at least in the experiments we conducted. It is far more important to group relevant training data together. Also, various fine tuning techniques also help.

3 Related Work

Section 5 of the official project handout listed all the related papers.

4 Approach

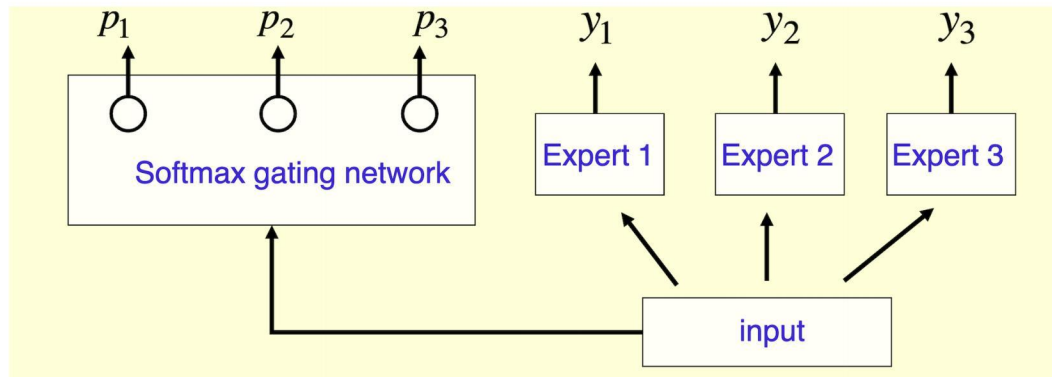
The first approach we tried was the mixture of experts architecture. The idea behind this approach is quite intuitive: we train a team of expert models, with each focusing on a specific sub-domain of knowledge, then when we encounter a case from unseen domain, it is likely that one of the experts will be able to handle the case better than a holistic model which does not have the benefit of division of labor. The architecture involves a gating network which selects appropriate expert for a new case and a team of experts with each having different focus. This approach also allows flexible way to build our solution: we can try different ways to train experts, different ways to construct the gating network and so on. Given all these reasons, we decided to go with this architecture first. For training the experts, we first tried to train them on the provided training data sets without reorganizing the data. For example, we trained an expert on data set squad, another expert on data set newsqa and so on. As we will cover in details in the experiments section, we eventually discovered this is not an optimal way to train localized experts. We then tried to reorganize the training data using sentence embedding similarities. For purpose of illustration, let's assume the training example in the training data sets has the following structure: title, context, question, answer. We go through the entire training sets, one training example at a time, and calculate a sentence embedding (we used SentenceTransformer for generating sentence embedding) on the title of this example. Once we have embeddings for all the titles in the training sets, we run a KNN classification to break them into groups. The hope was this new grouping would better reflect the similarities between training examples than the original split therefore allow us to train better experts.

We did make one more change to generate the embeddings. Instead of calculating the sentence embeddings using title, we decided to calculate embeddings using context. Context includes more information than title and will better reflect the similarities between training examples. It turned out this was the right thing to do.

For gating network, we initially used a simple multiple layer structure: two linear layers with random dropout and a final softmax layer. We trained the experts individually first, then we trained the gating network together with all the experts using all training data. The hope was the gating network would learn which expert to dispatch when given a new question. This did not work at all. As we will show in the experiments section, the gating network did not pick the best expert for a given question.

Once realizing the MLP does not work well as the gating network, we tried to act as the gating network ourselves. The basic idea is this: instead of only classifying the training data using sentence embeddings, we classify the training data together with the out of domain validation data. Say after classification, question one and two from the training data and question one from the validation data ended up in the same group, say group 1. We then put training question one and two in a group trainGroup1 and validation question one in a group valGroup1. We train an expert on trainGroup1 and apply the expert to valGroup1 to generate results. The hope was expert trained on data similar to the tests the expert will face would perform better. (note we didn't use any out of domain test data here, we want to see whether it works on validation data first)

We also tried various techniques to improve the fine tuning process: fine tune the model longer (6 epochs instead of 3), freeze or re-initialize layers of pre-trained model (random sampling from normal distribution with mean zero and standard deviation of 0.02), and different randomization (by setting different random seeds). Most of these ideas are from the fine-tuning paper listed in project handout.



5 Experiments

5.1 Data

Default final project, data provided.

5.2 Evaluation method

Default final project, evaluation method provided.

5.3 Experimental details and results

Experiment 1: Compare mixture of experts model (MoE) with baseline model

Motivation: to see whether the basic MoE outperforms the baseline, to see whether training with oodomain training data helps

Setup:

- MoE: two experts, one trained on training set squad, the other trained on training set newsqa combined with *nat_questions*
- Baseline: the base line model is generated by fine tuning the pre-trained BERT model on all training data
- MoE with OO: same MoE model, but further trained with oodomain training data
- Baseline with OO: same baseline model, but further trained with oodomain training data

Results:

model	F1
Baseline	47.098
Baseline with OO	46.59
MoE	45.189
MoE with OO	45.50

Conclusion:

- MoE under-performed Baseline
- whether oodomain training helps: inconclusive

Experiment 2: Adjust MoE to see whether the adjustments will help

Motivation: Made some adjustments to MoE and see whether we can improve performance

Setup:

- MoE: uses three experts instead of two, gating network uses three layers instead of two layers
- Baseline: same as before

Results:

model	F1
Baseline	47.098
MoE	44.68

Conclusion: 3-expert, 3-layer network is actually worse than before.

Experiment 3:

Motivation: The MoE approach is not working, there are two possibilities, either the experts are not good enough, or the gating network is not picking the proper expert. We want to find out whether the gating network is doing its job.

Setup:

- `squad_alone`: run the expert trained using squad data as a stand alone model, apply it to oodomain validation
- `natandnews`: run the expert trained using `nat_questions` + news data as a standalone model, apply to oodomain validation
- `squad_with_untrained_moe_same_expert`: this MoE model has two experts, both are squad, and the gating network has not been trained yet. This should give the same results as `squad_alone` since no matter which expert the gating network picks, it is still a squad expert. This is essentially a sanity check that the gating network is not doing something completely crazy.
- `natandnews_with_untrained_moe_same_expert`: same idea as before, the only difference is that the two experts are set to `natandnews` instead of `squad` this time.
- `experts_with_untrained_moe`: MoE with two experts, one `squad` and one `natandnews`. Gating network untrained. We expect the result to be an average of using `squad` and `natandnews` alone.
- `squad_with_trained_moe_same_expert`: MoE with two experts, both set to `squad`, gating network trained. We are trying to see what impact training the gating network has. Since both experts are set to `squad`, the result of this model should be the same as `squad` expert, but in reality the results are different. This means training the gating network together with the experts does change the model parameters.
- `natandnews_with_trained_moe_same_expert`: same as before, except that experts are set to `natandnews` instead of `squad`.
- `experts_random_moe`: MoE with two experts, one set to `squad` and the other set to `natandnews`, the gating network randomly picks one expert each time. We wanted to compare our final gating network with this randomized gating network to see whether our gating network adds value or not.
- `experts_with_trained_moe`: MoE with two experts, one set to `squad` and the other set to `natandnews`. Gating network trained. This is the model of our focus

Results:

model	F1	EM
<code>squad_alone</code>	41.48	26.96
<code>natandnews</code>	45.79	30.37
<code>squad_with_untrained_moe_same_expert</code>	41.48	26.96
<code>natandnews_with_untrained_moe_same_expert</code>	45.79	30.37
<code>experts_with_untrained_moe</code>	43.70	28.01
<code>squad_with_trained_moe_same_expert</code>	46.50	31.41
<code>natandnews_with_trained_moe_same_expert</code>	46.50	31.41
<code>experts_random_moe</code>	46.26	30.89
<code>experts_with_trained_moe</code>	46.50	31.41

Conclusion:

- without training the gating network, the model performance will be approximately an average of the expert's performances
- training the gating network together with experts does change expert model parameters
- gating network seems to add value

Experiment 4

Motivation: As described in the approach section, we decided to use sentence embeddings to re-organize training data. This experiment is to see whether this helps. We broke the training examples into three groups according to sentence embedding similarities: group 1, 2, and 3.

Setup:

- `g1`: expert trained on training data group 1 alone, Then apply the expert to oodomain validation set

- g2: similar to g1, but trained on group 2 alone
- g3: similar to g1, but trained on group 3 alone
- moe-gdata: MoE model, gating network with three experts g1, g2, and g3

Results:

model	F1	EM
g1	43.74	26.70
g2	45.33	30.10
g3	44.09	28.01
moe-gdata	41.65	26.18

Conclusion: This was very disappointing at the time, it showed gating network did not work at all since the performance with the gating network is worse than the performance of using individual experts alone.

Experiment 5:

Motivation: Simply re-organizing training data according to sentence embeddings did not improve performance. We decided to classify training data together with validation data. Please referred to the approach section for more details.

Setup:

- g1_train: expert trained on group 1
- g2_train: expert trained on group 2
- g3_train: expert trained on group 3
- g1_val: validation group 1
- g2_val: validation group 2
- g3_val: validation group 3

Results:

model	g1_val	g2_val	g3_val
g1	45.02	40.05	43.71
g2	49.73	38.63	41.90
g3	45.65	38.23	44.78

Conclusion: g1_train suppose to have better performance on g1_val, but it is not the case here, it seems localized expert is not working

Experiment 6:

Motivation: in the experiments so far we have generated sentence embedding using data example title only. In this experiment we generated sentence embedding using context instead of title.

Setup:

- g1_train: expert trained on group 1
- g2_train: expert trained on group 2
- g3_train: expert trained on group 3
- g1_val: validation group 1
- g2_val: validation group 2
- g3_val: validation group 3

Results:

model	g1_val	g2_val	g3_val
g1	36.21	50.92	61.97
g2	36.96	48.28	60.73
g3	32.92	44.76	60.77

Conclusion: The performance is better than embedding with title only, but the correlation is totally wrong, inverse actually, expert g1 performs the best for data set g3.

Experiment 7:

Motivation: We had the following observation from experiment 6: it seems validation group 1 always has the worse performance, no matter which expert we use, and validation group 3 always has the best performance. We want to find out why this is the case.

Originally the oodomain validation data are broken into three sets: duorc, race, and relation_extraction:

duorc	18	6.5%
race	113	41%
relation	128	46%

We broke oodomain validation data into three new sets: g1, g2, and g3

We analyze the number of examples from duorc, race, and relation_extraction in each of g1, g2, and g3

group	total	duorc	race	relation
g1	93	18	66	9
g2	82	0	35	47
g3	84	0	12	71

Conclusion:

- all examples from duorc are in g1, this is probably why g1 had the worse performance
- g3 had a lot of examples from relation_extraction, which has wikipedia as its data source. Keep in mind squad and nat_questions in our training data sets also have wikipedia as data source.

Experiment 8:

Motivation: Based on our observations from experiment 7, it makes sense to train an expert on the combined set of squad and nat_questions. Then apply this model to validation set relation_extraction. We used the baseline model for validation sets duorc and race.

Setup:

- squadAndNat: expert trained on squad plus nat_questions
- baseline_5000: baseline model, with step size 5000
- baseline_5000_diff: baseline model, step size 5000, 6 epochs instead of 3, learning rate of 1e-5 instead of 3e-5
- baseline_2000: baseline model, step size 2000
- baseline_2000_diff: baseline model, step size 2000, 6 epochs instead of 3, learning rate of 1e-5 instead of 3e-5

Results:

model	duorc	race	relation_extraction
squadAndNat			67.21
baseline_5000	40.51	37.38	64.42
baseline_5000_diff	42.43	34.50	66.97
baseline_2000	42.41	35.20	66.85
baseline_2000_diff	42.65	31.38	64.48

Conclusion: squadAdnNat does well on validation set relation_extraction, race and duorc are the difficult sets to deal with.

Experiment 9:

Motivation: try various fine-tuning techniques, see whether we can improve performance

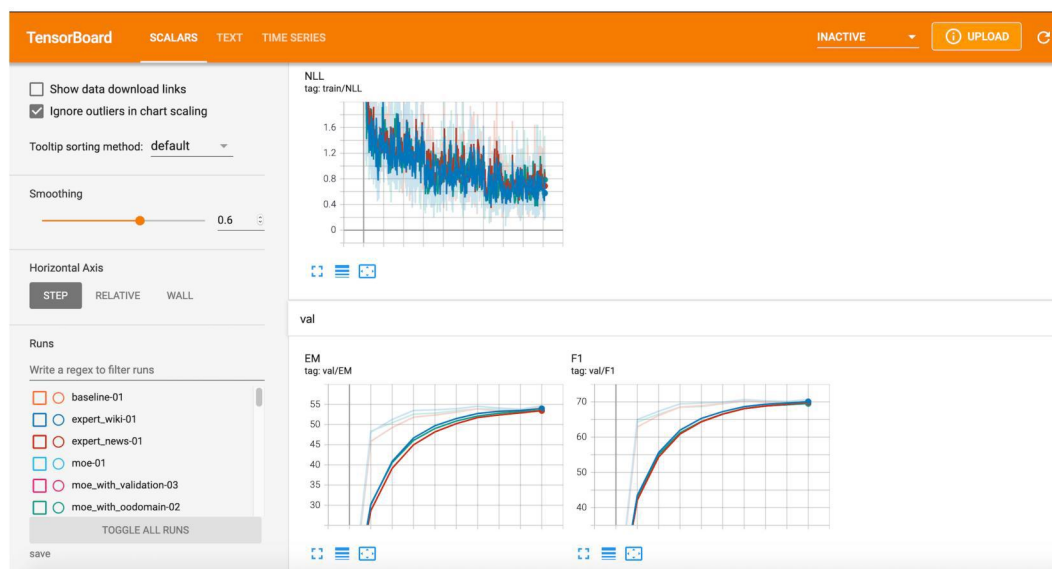
Setup:

- the pre-trained BERT model has six layers
- DBert-02: freeze the first three layers of pre-trained BERT during fine tuning
- DBert-03: freeze layer 4 and layer 5 of pre-trained BERT during fine tuning
- reinit_BERT: re-initialize parameters of layer 4 and layer 5 of pre-trained BERT
- seedBert: used a different random seed value in fine tuning.

Results:

model	duorc	race	relation_extraction
DBert-02	37.93	33.70	67.75
DBert-03	38.26	34.86	67.03
reinit_BERT	41.60	33.58	65.43
seedBert	47.20	37.11	65.89

Conclusion: Based on the experiments so far, for out of domain test cases, we decided to use model DBert-02 for test set relation_extraction, model baseline with step size 5000 for test set race, and model seedBert for test set duorc. Putting everything together, we achieved **F1 score of 59.15** on the test leaderboard. This is our best performance.



6 Analysis

MoE architecture didn't work, either due to sub-optimal local experts or failure of the gating network, or even both.

The behavior of the gating network is intriguing: in one experiment it seems to help, meaning the model with trained gating network performs better than expert model alone. But in a different model it actually made things worse. Future work is needed to further analyze behavior of gating network.

Various fine-tuning techniques help, but we don't have a good understanding as to why they help. We have some intuitions, but the results from experiments are sometimes counter-intuitive.

Interestingly, out of domain training data does not help. Our suspicion is they actually mess up our model parameters constructed during fine tuning. Maybe there are simply too few out of domain training examples.

7 Conclusion

In general, It is not easy to build model that performs well on domains it hasn't seen much.

It seems for MoE to work, we still need relevant training data to train local experts. In this project, since data set Duorc and Race are very different from the training sets, especially Duorc, which is designed to be very unique, it is very difficult to build a successful MoE system which performs well on out of domain data sets.

Data relevance still plays the most important role in our opinion. Since test set relation_extraction and training sets squad and nat_questions share the same data source(Wikipedia), an expert trained on squad+nat_questions is able to perform much better on test set relation_extraction.

Although MoE architecture didn't work in our case, it did provide us a major insight: we don't have to deal with the test sets as a whole, meaning we don't have to apply one single model to all test sets. We can divide and conquer them one by one, finding a good model for each test set, then put the predictions back together.

Another lesson we have learned is the importance of computing resource, it takes a long time to train NLP models. So more powerful computing resource definitely help a lot.

This is a super interesting project and we hope to explore other ideas mentioned by project handout to see whether we can find something that performs even better.

References