6.864 Project

Jose Navarro, Akshay Ravikumar
https://github.com/akshayravikumar/nlp-and-chill

1 Introduction

As internet forums like Quora and Stack Exchange increase in popularity, it becomes increasingly difficult to parse and find relevant information. This is because user-written content takes on various formats, and is filled with noisy or irrelevant information. In addition, labeled data on relevant questions is scarce. Therefore, the challenge lies in created an accurate question retrieval system based on a small labeled dataset.

In this paper, we demonstrate a machine learning model for question retrieval over various Stack Exchange forums. In particular, we generate a model that, given a Stack Exchange question, recommends similar question-answer pairs.

We start by implementing and tuning a simple CNN and LSTM model for question retrieval over the Ubuntu Stack Exchange forum. Then, we attempt to directly transfer this model to unlabeled data in the Android Stack Exchange forum. Finally, we use a domain adaption network to help bridge the gap between the two forums' distributions, and increase the accuracy of the model. Fortunately, we found that the domain adaption network significantly improves the performance of the model.

Our code is available at `https://github.com/akshayravikumar/nlp-and-chill`.

2 Related Work

We base our question retrieval model off the work of Lei et al, who also train across the Ubuntu Stack Exchange [4]. The authors start by implementing various baseline architectures such as a CNN, LSTM, and a GRU. The authors end up finding the best results using an adaptive gate decay model, known as the RCNN. In the end, the authors find results significantly better than well-known BM25 model.

We also implement transfer learning techniques developed by Gan et al. In particular, given an input set X and output space Y, assume we have trained the model on some

distribution $\mathcal{S} \in X \bigoplus Y$, and we need to transfer this knowledge to some distribution $\mathcal{T} \in X \bigoplus Y$. Assume we trained a *feature extractor* on $\mathcal{S}$ that generates a feature matrix for every input. Now,we introduce a *domain classifier* that, given a feature matrix, classifies which distribution it came from.

Now, we have two goals: we want the domain classifier to accurately distinguish the two distributions, but we want the feature extractor to force the two distributions to approach each other. Therefore, let L_s be the loss of the feature extractor, and L_d the loss of the domain extractor: it suffices to find a *saddle point* of the total loss $L_S - \lambda L_d$, where λ is a tunable hyperparameter. We achieve this by introducing a *gradient reversal layer* that backpropagates on this cumulative loss function.

3 Implementation

To solve the question retrieval task, we were given a dataset from the Ubuntu Stack Exchange forum. It included a corpus of over 160,000 questions, along with labeled data about similar and dissimilar question pairs: this information is divided into train, validation, and test sets. Then, we train a model to find an encoding for every question, and use cosine similarities to compare two different questions: therefore, given a query, and a list of candidate questions, we order the candidates by decreasing similarity from the main query.

For the domain adaption task, we are also given a corpus from the Android Stack Exchange forum consisting of approximately 50,000 questions. However, while the validation and test sets contain labeled data about positive and negative question pairs, the training data is unlabeled. Therefore, we need to transfer information gained about the Ubuntu dataset and apply it to the Android forum.

Specifically, each question consists of a title and a body, which are both padded to a maximum length of 25 and 100, respectively. When solely working with the Ubuntu dataset, we use embeddings derived from Stack Exchange, while when working with the Android dataset we use the GloVe embeddings.

To solve this problem, we implement various architectures, described below. Note that almost all of our trials run on CUDA.

4 Training the Ubuntu Data

To train a model on the Ubuntu data, we use methods described in Lei at al. Let M be a model that, given a feature matrix for a question, generates an encoding for that question.

Then, we train our model as follows: let a sample consist of a question, one similar question, and 20 dissimilar questions. We get the word embeddings for this question and pass them into M, which creates an encoding for every question: then, we compute cosine similarities between the main query and every other query, and pass them into a max-margin loss function. We configure our target such that the positive query has a higher cosine similarity than all the negative questions.

Then, for evaluation, we pass in a question and 20 "candidates" into our model, some of which are positive, and generate a ranking over those questions using M and the cosine similarities. We evaluate this ranking system using the mean average precision (MAP), mean reciprocal rank (MRR), the precision at 5, and the precision at 1. We compare these results to the BM25 baseline [1].

Now, it remains to choose the best feature extractor M: we consider a CNN and LSTM, described below.

4.1 Vanilla CNN

First, we tried a simple convolutional neural network architecture. Our model consists of a one-dimensional convolutional layer, along with a mean pooling layer. Because we pad the title and body, a large portion of the embeddings are zeroed: to mitigate the adverse effects of this, we ensure that the mean pooling layer ignores the padding when computing the average.

Finally, given the embeddings from the mean pooling layer, we compute cosine similarities and pass them into the max-margin loss function.

We performed hyperparameter tuning and was able to exceed baselines found in the paper with the following parameters:

- Learning rate: 0.01
- Hidden size: 300
- Window size: 3

See Figure 4.1 for the accuracy metrics with thesse hyperparameters.

4.2 Vanilla LSTM

We also tried a similar long short-term memory (LSTM) model to extract an encoding from every question. Our model simply consists of a LSTM with one layer, along with

	MAP	MRR	P@1	P@5
Dev	0.5457	0.6549	0.5026	0.4433
Test	0.5477	0.6817	0.5215	0.41823

Fig. 4.1: The optimal accuracy metrics for the CNN.

	MAP	MRR	P@1	P@5
Dev	0.5366	0.6737	0.5502	0.4328
Test	0.5568	0.6755	0.5437	0.4333

Fig. 4.2: The optimal accuracy metrics for the LSTM.

a mean pooling layer and tanh activation. As before, we make sure to ignore padding when computing averages. Finally, we also consider adding a dropout layer to increase the accuracy of the model.

We performed hyperparameter tuning with the following parameters: (1) No dropout, (2) Dropout $p = 0.5$, (3) Dropout $p = 0.2$, (4) hidden-size=240, and (5) hidden-size=300.

We our best performance with the following hyper-parameters: (1) Dropout $p = 0.2$, hidden-size=300, Adam learning rate=0.001 . The accuracy metrics for this hyperparameter set are summarized in Figure 4.2.

5 Transfer Learning

Now, given the unlabeled Android dataset, we wish to use the Ubuntu data to generate an accurate model. We evaluate our ranking system using the area under curve (AUC) metric which evaluates the accuracy of a binary classifier under various discrimination thresholds.

5.1 Direct Transfer

First, we attempt direct transfer; i.e., we simply train the model on the Ubuntu dataset and evaluate it on the Android dataset using the AUC metric. We use both the CNN and LSTM models described above.

First, we run the CNN with the optimal hyperparameters described above: in the end, we get a validation AUC score of 0.5503 and a test AUC score of 0.5506.

When we run the LSTM with the optimal hyperparameters described above, we get a validation AUC score of 0.5875. Because the CNN has sufficient accuracy (according to Piazza, a score of 0.55 to 0.6 was reasonable for direct transfer), we proceed to the next

λ	Maximum Val. AUC
10^{-1}	0.5612
10^{-2}	0.6694
10^{-3}	0.6366

Fig. 5.1: The maximum validation AUC(0.05) score for various values of λ

η	Maximum Val. AUC
10^{-3}	0.4121
10^{-4}	0.7239
10^{-5}	0.5431

Fig. 5.2: The maximum validation AUC(0.05) score for various values of η

part. Because the two models have comparable perforamnce, and the CNN is much faster, we focus on the CNN when hyperparameter tuning the domain adaptation model.

5.2 Domain Adaptation Model

Now, we attempt the domain adaption techniques described in Ganin et al. to improve the accuracy of this model. Specifically, our domain classifier model consists of the following layers: the input is fully connected to a size-300 layer, then a size-150 layer, then a size-2 layer. We have RELU activations in between, and we use a cross entropy loss to evaluate the performance of the domain classifier, which combines log softmax and NLL loss into a single function.

To train this model, we let our "total loss" be $L_s - \lambda L_d$, and we backpropagate this value as opposed to backpropagating either value separately. In addition, we use two Adam optimizers: the first trains the feature extractor with a positive learning rate, while the second trains the domain classifier with a negative learning rate.

Margin	Maximum Val. AUC
0.05	0.6046
0.1	0.6638
0.5	0.6361

Fig. 5.3: The maximum validation AUC(0.05) score for various margin values.

For our baselines, we ran a model where we simply perform cosine similarities over GloVe embeddings: this results in an AUC(0.05) score of 0.1272. In addition, it was reported on Piazza that the BM25 numbers are approximately 0.68, so we use this as another baseline metric when hyperparmeter tuning.

5.2.1 Using the CNN

For this section, we use the CNN as our feature extractor. For hyperparameter tuning, we focus on tuning λ using learning rates of 10^{-2}, which dictates the tradeoff between minimizing the loss of the feature extractor and maximizing the domain classifier loss. Our results are summarized in Figure 5.1: altogether, we found that $\lambda = 10^{-2}$ led to the optimal AUC score.

In addition, we independently focus on tuning the learning rates using a fixed $\lambda = 0.01$: we use the same learning rate for both the domain classifier and feature extractor, although the former is negative. Note that to save time, we only train on 1/3 of the data, but we evaluate on the entire validation set. These are summarized in Figure 5.1. We find that a learning rate of 10^{-4} produces optimal results.

Now, fixing the learning rate and λ, we tune the margin value for the maximum-margin loss function. These results are summarized in Figure 5.3: we find that a margin value of 0.1 produces the optimal AUC values. Again, we train on 1/3 of the data for the sake of efficiency, but evaluate on the entire validation set.

Now, we use the optimal parameters as described above: a margin of 0.1, $\lambda = 0.01$, and a learning rate of 10^{-4}. This leads to an AUC score 0.7292 for the validation set and 0.7145 for the test set, which is the largest value we found. Because this exceeds the Piazza BM25 baseline of approximately 0.68, we are satisfied with our choice of hyperparameters.

5.2.2 Using the LSTM

Now, we train the domain adaptation model using the LSTM. Because the CNN gave performance values that exceeded our expectations, we did not tune our domain adaption network on the LSTM as much. However, running the LSTM with the same hyperparameters as the CNN resulted in a validation accuracy of 0.6777, which is comparable to the BM25 baseline numbers, and not as accurate as the tuned CNN model.

6 Potential Extensions

We spent some time analyzing other methods of domain transfer. In particular, we found the paper about Adversarial Discriminative Domain Adaptation (ADDA) by Tzeng et al. interesting [5]. The paper seeks to combine the best of Ganin et al., generative adversarial networks, and discriminative modeling into one framework.

Another paper we looked at was Shen et al., who extend Ganin et al. by adding a *domain critic*, whose job is to compute and minimize the Wasserstein distance between the source and target distributions [6]. The loss metric now includes an additive term for the domain critic, in addition to the feature extractor and domain classifier.

We worked on implementing the former model for some time, but unfortunately we weren't successful.

7 Conclusion

In conclusion, we created a model that, given a question, can recommend similar questions from a database of Stack Exchange questions. We started by implementing and tuning a simple CNN and LSTM model on a labeled dataset, then augmenting that with domain adaption techniques to extend to an unlabeled dataset. We were able to get results that exceeded that of the BM25 baseline, which demonstrates that these methods are effective.

8 Acknowledgments

We'd like to acknowledge Professor Regina Barzilay, along with the rest of the course staff, for making 6.864 a fun, interesting class. Also thanks to based Darsh for all the help on Piazza.

References

[1] Stephen Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond.

[2] Yaroslav Ganin, Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation. *arXiv preprint arXiv:1409.7495*.

[3] Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Aspect-augmented Adversarial Networks for Domain Adaptation. *arXiv preprint arXiv:1701.00188*.

[4] Tao Lei, Hrishikesh Joshi, Regina, Barzilay, and Tommi Jaakkola. Semi-supervised Question Retrieval with Gated Convolutions.

[5] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial Discriminative Domain Adaptation. *arXiv preprint arXiv:1702.05464*.

[6] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein Distance Guided Representation Learning for Domain Adaptation. *arXiv preprint arXiv:1707.01217*.