6.864 Final Project - Domain Adaptation in Question Retrieval

Aritro Biswas and Amin Manna

December 2017

1 Abstract

In this paper, we describe the Question Retrieval models that we train and evaluate on the Stack Exchange AskUbuntu dataset. One of our models is Long Short Term Memory (LSTM) network, and the other is a Convolutional Neural Network (CNN). Then, we explore the possibility of using transfer learning to solve a similar Question Retrieval task on the Stack Exchange Android dataset. We establish direct transfer as a baseline, and show that Domain Adaptation can be used to outperform that baseline.

2 Introduction

Online stack exchange forums provide a large source of question and answer pairs. We propose to leverage this data towards the task of question retrieval, i.e. to retrieve questions similar to query questions, and give higher scores to more similar questions. Such a system has practical value; for instance, a user wanting to ask a question might use this system to find a similar question that has already been answered. However, this task is also interesting from a natural language perspective. Constructing a good representation from the question body for this task is challenging, as in many instances, the body content might have lots of information specific to the user and not directly relevant to the question, making it challenging to filter out the relevant information.

In this paper, we study the problem of question retrieval in 2 settings. In the first setting, we are interested in the question retrieval task when trained on evaluated on the Ubuntu Stack Exchange Dataset. In the second setting, we are interested in using **domain adaptation** techniques to transfer a model trained off of labeled data from Ubuntu Stack Exchange and unlabeled data from Android Stack Exchange, and evaluating it on Android Stack Exchange.

3 Related work

We base our work on the question retrieval task off Lei et. al.'s work on the task [1]. Following their model, we have implemented an LSTM and a CNN to process question text to construct representations for this task. On the other hand, we extend on their work by modifying the loss function used to train the models.

Our work for domain adaptation builds off Ganin et. al.'s ideas [2]. To do this task, we introduce a separate classifier network that attempts to infer, based off the encoding produced by the encoder module, which corpus, Android or Ubuntu, the encoding corresponds to. As per their ideas, we train our text encoder with 2 separate losses, a label prediction loss measuring how well the model is able to score questions, and a domain classification loss, which measures the ability of the classifier to distinguish which corpus the corresponding question came from. The goal is to have the encoder produce representations that are invariant with respect to the dataset, allowing transfer to work better. As Ganin et. al. did, we create a 2 layer classification network for doing the domain classification task. However, we replace the RELU activation at the final layer with a sigmoid activation, and also deviate from their paper by choosing to use binary cross entropy loss for the domain classification task.

4 Model Description/Method

4.1 CNN

Our model is a neural network with the following layers:

1. A 1d Convolutional Layer with a kernel size of 4, an input dimension of 200, and output dimensions 200. It takes a sequence of 200 dimensional word embeddings, and convolves the input into a (shorter, because the kernel size is greater than one) sequence of 200 dimensional vectors.

2. A 1d Batch Normalization layer. This does not reshape it's

3. A Tanh activation function, enabling us to model non-affine transformations.

4. A pooling layer with a kernel as wide as the whole sequence, allowing us to learn fixed-dimensional encodings for variable length sentences.

4.2 LSTM

Our LSTM takes an input dimension of 200 and a hidden dimension of 200. It takes a sequence of 200 dimensional word embeddings and produces a series of 200 dimensional embeddings corresponding to the LSTM's hidden state at each time step of the sequence. The LSTM then either returns the final hidden state or the average of all the hidden states for the sequence.

4.3 Question retrieval

Given an encoder, we produce representations for each question that are averages of their title encodings and their body encodings. We then calculate the cosine similarity between question embeddings. Our loss function is the following: $\max_{p \in Q(q_i)} s(q_i, p; \theta) - s(q_i, p_i^+; \theta) + \delta(p, p_i^+)$ where we select p_i^+ to be the similar question with the least cosine similarity to the query question q_i.

As a baseline, we compare against the BM25 metrics. BM25 is a search engine for doing information retrieval. We have provided the metrics for BM25's performance on the dev and test set in the results section.

4.4 Domain Classifier and Gradient Reversal Layer

We now study the case when we wish to evaluate the model on the Android data. The Ubuntu data in this situation is labeled; the android data on the other hand is unlabeled.

Our model consists of 2 main components: a feature extractor and a domain classifier. The feature extractor is either the LSTM or the CNN mentioned earlier. It produces representations of the according Ubuntu or Android questions. On the other hand, the domain classifier tries to map the features extracted by the feature extractor for each question to a prediction on which corpus, Ubuntu or Android, the questions came from. The domain classifier attempts to minimize the corresponding domain classification loss. Simultaneously, the feature extractor attempts to maximize this loss. Having the feature extractor maximize the loss would correspond to the feature extractor producing features that were invariant with respect to the exact corpus, Android or Ubuntu, that was being used. This would enable the model to transfer better to the Android data.

Alongside domain classification loss, the feature extractor is trained to minimize the label prediction loss mentioned in the previous section. Training the models to optimize for both label prediction loss and domain classification loss enables the feature extractor to learn representations that are not only suitable for the label prediction task, but also transfer well to the android dataset.

We now describe the exact architecture of the domain classifier, as well as the domain classification loss that we are using. Our domain classifier takes as input a 200 dimensional question embedding and produces a scalar value between 0 and 1 which corresponds to the likelihood with which the model believes the embedding to come from the Android data. More specifically, the exact architecture of the domain classifier is as follows:

1. A linear layer with an input dimension of 200 and output dimension of 150.

2. A linear layer with an input dimension of 150 and output dimension of 1.

3. A sigmoid activation layer

To train our network, we use the binary cross entropy loss, which is

$$L_{BCE}(T, O) = \sum_{i=1}^{n} T_i log O_i + (1 - T_i) log(1 - O_i) \quad (1)$$

where T are the ground truth labels for questions, having value of 0 if the question is from Ubuntu and value 1 otherwise, and O are the domain classifier predicted probabilities for questions, and where T_i and O_i are the ground truth and predictions for question i.

We now point out an issue: the goal of the classifier is to minimize the domain classification loss, while the feature extractor attempts to maximize it. The optimal value for this loss is thus a saddle point. This means that typical optimization methods provided with deep learning libraries will not work, as they are suited to solve for global maxima or minima, not saddle points. To be able to use typical optimization methods provided by deep learning libraries, we implement a gradient reversal layer. This module applies the identity function during the forward step, but during the backpropagation step, multiplies the gradient of the output by a factor $-\lambda$, for some positive valued hyperparameter λ and passes that on to the input. We put the gradient reversal layer in between the feature extractor and the domain classifier. Because the sign of the gradient is getting changed by this layer, this enables the

classifier to be trained to minimize the domain classification loss, while simultaneously training the feature extractor to maximize that loss.

As a baseline, we compare the performance of the domain adaptation models to the performance of the Ubuntu-trained models on the android data.

5 Experiments

Question Retrieval on AskUbuntu
Direct Transfer to Android Dataset
Domain Adaptation

1. Domain adaptation with lstm 1 layer - avg pooling
2. Domain adaptation with cnn 1 layer - avg pooling
3. Domain adaptation with cnn 1 layer - max pooling
4. Ubuntu stack exchange (no domain adaptation) with cnn average pooling 1 layer
5. Ubuntu stack exchange with 3 layer lstm - avg pooling (Left pane, only 5 epochs though)
6. Ubuntu stack exchange with 1 layer lstm - max pooling
7. 1 layer LSTM - max pooling - 10 epochs, then direct transfer
8. 3 layer LSTM - max pooling - 10 epochs, then direct transfer
9. 1 layer CNN - max pooling - 10 epochs, then direct transfer
10. 1 layer CNN - avg pooling - 10 epochs, then direct transfer
11. 1 layer LSTM - avg pooling - 10 epochs, then direct transfer

6 Results

The results reported below use 20 examples per query.

6.1 Question Retrieval

Dev BM25 Scores: P@1=0.490 P@5=0.398 MAP=0.492 MRR=0.624

After training for 10 Epochs, we achieve the following results on the Ubuntu validation dataset:

Dev	Max Pool	Mean Pool
1 Layer LSTM	**P@1=0.340** **P@5=0.313** **MAP=0.402** **MRR=0.515**	**P@1=0.510** **P@5=0384** **MAP=0.477** **MRR=0.620**
1 Layer CNN	**P@1=0.445** **P@5=0.357** **MAP=0.451** **MRR=0.579**	**P@1=0.455** **P@5=0.371** **MAP=0.468** **MRR=0.589**

Test BM25 Scores: P@1=0.5 P@5=0.395 MAP=0.521 MRR=0.633

After training for 10 Epochs, we achieve the following results on the Ubuntu test dataset:

Test	Max Pool	Mean Pool
1 Layer LSTM	**P@1=0.340** **P@5=0.309** **MAP=0.399** **MRR=0.493**	**P@1=0.420** **P@5=0.366** **MAP=0.453** **MRR=0.573**
1 Layer CNN	**P@1=0.410** **P@5=0.323** **MAP=0.445** **MRR=0.564**	**P@1=0.460** **P@5=0.362** **MAP=0.478** **MRR=0.601**

6.2 Direct Transfer

Directly testing the pretrained models above on the Android validation dataset, we achieve the following results:

Dev	Max Pool	Mean Pool
1 Layer LSTM	**P@1=0.172** **P@5=0.102** **MAP=0.338** **MRR=0.340**	**P@1=0.445** **P@5=0.159** **MAP=0.585** **MRR=0.590**
1 Layer CNN	**P@1=0.280** **P@5=0.123** **MAP=0.432** **MRR=0.436**	**P@1=0.321** **P@5=0.129** **MAP=0.469** **MRR=0.472**

Directly testing the pretrained models above on the Android test dataset, we achieve the following results:

Test	Max Pool	Mean Pool
1 Layer LSTM	**P@1=0.183** **P@5=0.093** **MAP=0.328** **MRR=0.333**	**P@1=0.478** **P@5=0.156** **MAP=0.603** **MRR=0.608**
1 Layer CNN	**P@1=0.257** **P@5=0.125** **MAP=0.423** **MRR=0.427**	**P@1=0.359** **P@5=0.137** **MAP=0.499** **MRR=0.502**

6.3 Domain Adaptation

After training for only 1 epoch to tune the value of λ:

Dev	1 Layer CNN, Mean Pool	1 Layer LSTM, Mean Pool
λ=100	P@1=0.130 P@5=0.073 MAP=0.264 MRR=0.267 AUC=0.065	P@1=0.080 P@5=0.059 MAP=0.215 MRR=0.216 AUC=0.016
λ =1e-7	P@1=0.341 P@5=0.127 MAP=0.479 MRR=0.481 AUC=0.194	P@1=0.417 P@5=0.154 MAP=0.563 MRR=0.567 AUC=0.331
λ=1e-5	P@1=0.336 P@5=0.125 MAP=0.473 MRR=0.476 AUC=0.168	P@1=0.429 P@5=0.157 MAP=0.572 MRR=0.575 AUC=0.309
λ=1e-3	P@1=0.323 P@5=0.125 MAP=0.463 MRR=0.466 AUC=0.197	P@1=0.423 P@5=0.158 MAP=0.570 MRR=0.575 AUC=0.168

The result is that we select 1e-5 for the LSTM, and 1e-7 for the CNN, as reasonable values of λ. After 10 epochs:

Dev	λ = 1e-5	λ = 1e-7
1 Layer LSTM, Mean Pool	P@1=0.453 P@5=0.160 MAP=0.594 MRR=0.598 AUC=0.214	
1 Layer LSTM, Max Pool	P@1=0.185 P@5=0.101 MAP=0.341 MRR=0.343 AUC=0.191	
1 Layer CNN, Mean Pool		P@1=0.373 P@5=0.139 MAP=0.510 MRR=0.513 AUC=0.174
1 Layer CNN, Max Pool		P@1=0.238 P@5=0.119 MAP=0.398 MRR=0.402 AUC=0.796

The best domain adaptation CNN-based model is the 1 layer CNN with mean pooling, and λ =1e-7. Evaluating this model on the android test dataset, we get: **P@1=0.382 P@5=0.141 MAP=0.524 MRR=0.527 AUC=0.175**.

The best domain adaptation LSTM-based model is the 1 layer LSTM with mean pooling, and λ =1e-5. Evaluating this model on the android test dataset, we get: **P@1=0.483 P@5=0.156 MAP=0.604 MRR=0.611 AUC=0.209**.

7 Conclusion

When we compare our best CNN-based models for Direct Transfer and Domain Adaptation, we see that the Domain Adaptation strategy outperforms Direct Transfer by 6.4% in P@1, 2.9% in P@5, 5.0% in MAP, and 5.0% in MRR.

On the otherhand, the LSTM-based model with domain adaptation showed no significant improvement over direct transfer (only 1%). This is likely explained by analyzing model complexity. A more complex LSTM model, with more layers, might have been able to benefit more from domain adaptation, and the same can probably be said of the CNN model.

8 Contributions

Aritro: Did LSTM implementation, both average and max pooling. Helped Amin refactor the data loader to make it use Android data. Implemented training/validation script for question retrieval task. Implemented metrics to evaluate models and BM25 baseline for question retrieval task. Constructed android training data for the domain adaptation task. Designed the loss function for the domain classifier and the activation units for the domain classifier to be sigmoid.
Amin: Wrote the data loading code and implemented the CNN. Wrote the question retrieval task's loss function. Wrote training/validation script for the domain adaptation task. Wrote a preprocessing script for combining the positive and the negative examples from the android training data into one file that has the same format as AskUbuntu's train_random. Re-factored Aritro's metrics calculation code and imported it into the domain adaptation script. Implemented the Domain Classifier and we jointly implemented the gradient reversal layer.

References

[1] Semi-supervised Question Retrieval with Gated Convolutions. Lei, T.; Joshi, H.; Barzilay, R.; Jaakkola, T.; Tymoshenko, K.; Moschitti, A.; Marquez, L

[2] Unsupervised Domain Adaptation by Backpropagation. Ganin, Y.; Lempitsky, V.

[3] Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. Now Publishers Inc..