Deep Learning for the Soft Cutoff Problem

Miles Saffran
Yale University
miles.saffran@yale.edu

Dragomir Radev
Yale University
dragomir.radev@yale.edu

Abstract

The MATERIAL project at Yale seeks to construct a reliable search engine for English queries to return results in low-resource languages. This engine must make some determination about the proper number of documents to return specific to each query. In this paper, I put forward a method for determining document cutoff such that relevant documents are returned and irrelevant documents are left out. To do this, I construct a neural network that trains on AQWV scores from known query/document pairings in order to predict optimal cutoffs on test data. The network deduces cutoff scores based on input features like document score, document length, and query word embeddings. Over a series of trial runs, I found that the optimal network employs two hidden layers of different sizes. Ultimately, though cross-lingual training proved difficult, the network improved on previous AQWV score estimations when trained and tested on different sets in the same language.

1 Credits

This project was developed with assistance from Javid Dadashkarimi and Jungo Kasai, who both helped me find the necessary introductory code for my project and aided me in revising my work. The word embedding feature was developed by Caitlin Westerfield; the code from her portion is noted in the comments of my program.

2 Introduction and Problem Description

The MATERIAL (Machine Translation for English Retrieval of Information in Any Language) project is a research program, commissioned by IARPA, to create an engine capable of retrieving content in low-resource languages using English queries. Yale is part of a joint effort that spans multiple universities, including Columbia, Cambridge, and the University of Maryland, to develop a robust engine that fulfills the requirements of the project. The task is quite large and contains a vast array of problems, including sentence matching, paraphrase identification, and question answer tasks. The scope of the project is described in more detail here[1].

As part of this project, my research deals with a subsection of information retrieval known as document cutoff. Other researchers working on the project have calculated which documents are most closely associated with queries, using an Indri document scoring system. Indri assigns a probability value to each document through language modeling of both the query and document, providing scores that describe the likelihood of a document being relevant to a particular search phrase.

However, though Indri tells us which documents fit the query, it does not tell us how many documents should actually be returned. An important component of a proper MATERIAL engine is its ability to discern which results are relevant enough to be included. As a result, though there might be 100 documents associated with a certain query, users are presumably not interested in every particular document. Though training data provided by IARPA tells us the optimal cutoff for a certain subset of queries, the project will be evaluated on private data whose AQWV scores remain hidden (more on this later in the report). Therefore, a proper document

[1] https://github.com/dadashkarimi/Y-Flow

cutoff system must use some metric that maps features of queries and documents onto the correct number of documents to return.

3 Data sets used

In this section, I will describe the data sets I used as well as some of the programs I wrote to clean them for the neural network. Generally speaking, the only training data I used was the files available to me as part of the MATERIAL project. Starting from the home Y-Flow directory, the /data folder contained examples in Swahili, French, and English which all became viable data for the model. Take, for example, the sw folder (which refers to Swahili). Inside, there were several sub-folders containing data for English queries to Swahili documents, English queries to English documents, etc. Each of these subfolders had several important programs written by other programmers working on MATERIAL that I will detail below.

First, result/result.file contained a list of queries, the top 20 documents associated with each query, and the Indri score of each pair. Second, query_list contained a list of query to word mappings that would become useful later on in my analysis. Third, the docs/ folder contained all of the documents associated with each query; I would use features of the documents as inputs to the network later on. Finally, svm.py calculated the relevant AQWV score which I used as the correct labels for my training data (more on this in the next section).

Later in my evaluation, I added word embeddings of the query as features of the neural network. This code was written by Caitlin Westerfield, and sources the GloVe word embedding files available on the tangra server.

Overall, though there was a plethora of documents, actual training data was fairly hard to come by. In the French collection I was able to utilize around 150 training examples, and in the English collection only 64. Though this was enough for my purposes, the model would likely achieve better accuracy with more data to train on.

4 AQWV and Information Retrieval

4.1 AQWV

Actual Query Weighted Value, or AQWV, is the metric used by IARPA to grade the performance of each MATERIAL search engine [2] . Unlike a search engine one would use for commercial purposes, AQWV imposes a penalty for false positives: documents that were returned that should not have been. The following formulae explain how AQWV works (all taken from the previously cited paper):

$$AQWV = 1 - (P_{Miss} + \beta P_{FA})$$

P_{Miss} is equivalent to the probability of a missed detection error, while P_{FA} is equivalent to the probability of a false alarm error. The former penalizes the system for missing a document that is relevant to the query; the latter penalizes the system for including a document irrelevant to the query. The β is further defined as:

$$\beta = \frac{C}{V}(\frac{1}{P_{Relevant}} - 1)$$

C is the cost of an incorrect detection while V is the value of a correct detection. V, of course, is set to 1 while the value of C will vary depending on MATERIAL's needs. $P_{Relevant}$, meanwhile, is the estimate of any document's relevance, again set by IARPA to some value. The linked paper dives into greater detail about the exact calculation behind each of these figures, but intuitively AQWV is a useful metric for determining cutoff. Thus, when generating training data, I used svm.py to determine the cutoff point that had the highest AQWV as my gold standard.

5 Approach and Evaluation

As a broad overview, in order to solve this problem, I decided to construct a neural network that would train on the data that we were given and successfully predict the optimal cutoff on any test data that IARPA had. In order to create a network with predictive capacity, I needed to determine which features of the query/document pairings I had were informative. Ideally, the network would learn parameters that abstracted certain information about these features and could calculate a cutoff. Initially, I used features like Indri score that were already provided to me, but throughout the process I added more features to the benefit of the model. I wrote the network in TensorFlow, as it was the library I was most familiar with from coursework. In the remainder of this section, I will blend a discussion of my

[2] https://tinyurl.com/y98ds2zl

approach with an examination of the results at each step.

5.1 Cleaning the data

In order to pass any meaningful information into the neural network, I had to first clean up the data generated by other programs. **clean_data.py** performs the majority of this work. First, I stripped out the query identification numbers and stored them as keys in a defaultdict. As values, I stored a list of the Indri scores of the 20 documents most closely associated with each query. Next, I computed the length of each document that was associated with the query. In order to avoid issues with wonky gradients and weights, I normalized each document length by dividing it by the length of the longest document in each language's collection. I added these features together into a vector. Meanwhile, for each query, I found the optimal cutoff by highest AQWV (as described in the previous section), and kept it in a separate dictionary. Finally, I combined each (vector, label) pair into a tuple that I stored by query.

5.2 Building the simple neural network

Having arranged the data, I then set out to construct a simple neural network in the file **simple_nn.py**. The neural network was fairly straightforward: the input layer was the 40-feature vector mentioned in the previous section. This was fed into a hidden layer with a ReLU activation function, which in turn was fed into the output layer. Figuring out how to construct an output layer that would cohere with the data was somewhat tricky. Eventually, I decided to convert each cutoff label into a one-hot vector that was on in the array location of the cutoff number and off everywhere else. This allowed me to use the sigmoid activation function for the output layer, as my predictions were now assigned probabilities normalized to 1 for each cutoff position. Then, I used the cross-entropy function to calculate the loss between my prediction and the actual output, which backpropagated and updated the weights to minimize loss.

5.3 Adjusting hyperparameters

Initially, I set the hidden layer size to 30, and the learning rate to 0.01. After loss failed to significantly decrease, I adjusted the code to allow for a training of hyperparameters. I iterated

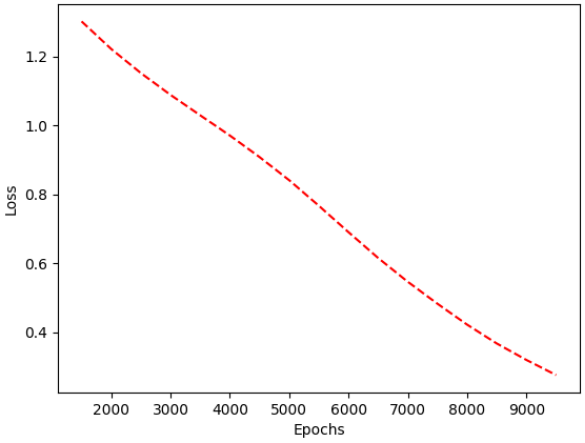

Figure 1: Training loss over epochs

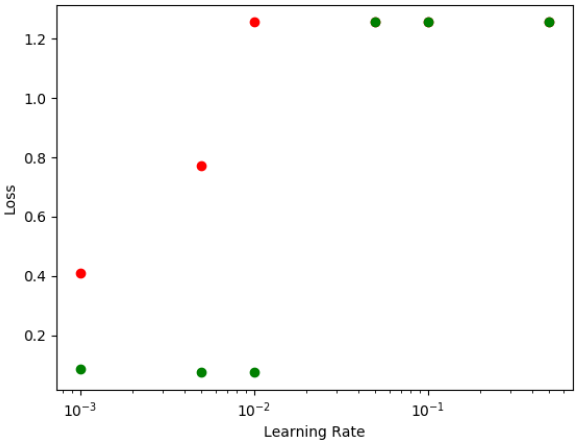

Figure 2: English loss with different learning rates

through a series of hidden layer sizes and learning rates, recording the results of each combination and saving the weights with the lowest loss in a separate folder. I recorded the results of training on both English and French analysis sets on 10000 epcohs. As you can see in Figure 1, loss dropped quite steadily over each epoch. However, there were still some issues with the network. Though loss was dropping as expected during certain trials, for the most part, it would converge rather early and stay still. Figure 2 graphs the loss as a function of the learning rate. The red points are from the model with a hidden layer size of 5 and the green points are from the model with a hidden layer size of 30. Despite the structural difference in the network's construction, we can see that as the learning rate goes up, the loss gets worse and tends to settle in the same place; many of the red dots are hidden by the green dots as they converge to the same loss.

Upon investigation, it turned out that when the

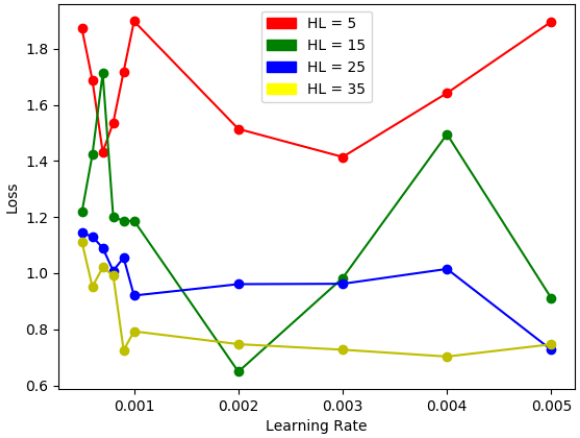

Figure 3: French loss with different learning rates and different hidden layers

machine was converging, it was doing so on the prediction that every document cutoff should be 1. Most document cutoff numbers are quite low for maximum AQWV; as a result, the model was not learning anything special about the input features but just setting the document cutoff to 1 each time. Though this yielded about 40% accuracy, it is not at all predictive of any important feature.

As you can see in Figure 2, when the learning rate went up, the model converged to this sub-optimal loss, implying that smaller learning rates would find better minima. When I made the learning rates smaller and graphed loss over this new set, I got the result in Figure 3. Generally, more hidden layers seemed to help loss converge to its lowest values over the course of more epochs. This makes sense, given that more parameters in the hidden layer necessitates more training in order to accurately encapsulate the input features. Having measured the models with the lowest loss, I also evaluated how this translated to training accuracy (how often the predictions synced up with the actual label). These results are all saved in the **test_results** folder. For the top 3 models, the results were:

Hidden Layer Size: 20, Learning Rate: 0.002, Loss: 0.799725711346, Acc: 0.7208122015
Hidden Layer Size: 35, Learning Rate: 0.004, Loss: 0.702716350555, Acc: 0.715736031532
Hidden Layer Size: 20, Learning Rate: 0.004, Loss: 0.978114545345, Acc: 0.644670069218

The best performing model had a hidden layer of size 20 with an accuracy of 72% on the French collection. While this was fine, the data seemed as though it was prone to overfitting issues. When testing the weights on a different set of French queries, for example, the best performing model yielded 30% accuracy. This disparity implied that the neural network was learning features particular to the specific input as opposed to more generalizable information.

5.4 Adding word embeddings and deep learning

To see whether the performance of the model could be amplified, I made two important changes. First, I added word embeddings thanks to a feature written by Caitlin Westerfield. To my **clean_data.py** file, I added functions that looked up each (query, word) pairing in a query file particular to the collection. Then, I used the GloVe word embedding system to get a feature vector of length 50 that encapsulates a learned vector space representation of the word [3]. This instantiated a 90-feature vector that contained Indri score, document length, and query embedding.

The other change I made to the network was to add another hidden layer. Instead of being fed through one hidden layer and then immediately to the output, I created another layer above the current hidden layer of fixed size 5 with a sigmoid activation function. My reasoning was that a more highly compressed hidden layer would actually learn abstract properties about the data, avoiding the overfitting problem from earlier and reducing components of individual variance. I still kept the training on the optimal hyperparameters in the first layer so that I could find the ideal model for training. I saved this code into **complex_nn.py**.

5.5 Performance of complex_nn.py

Overall, the complex neural network performed vastly better than the simple neural network. In Figure 4, you can see the graph of loss over epochs as it converges significantly below the loss value of the simple network Using the best model gained from training on ANALYSIS-EN, we can use our complex model to predict DEV-EN. The model does quite well, predicting with 56% accuracy what the optimal cutoff point is on a dataset it has never seen before. Figure 5 shows both the accuracy of the system as well as the model's predictions and the actual cutoff labels.

[3] http://www.aclweb.org/anthology/D14-1162

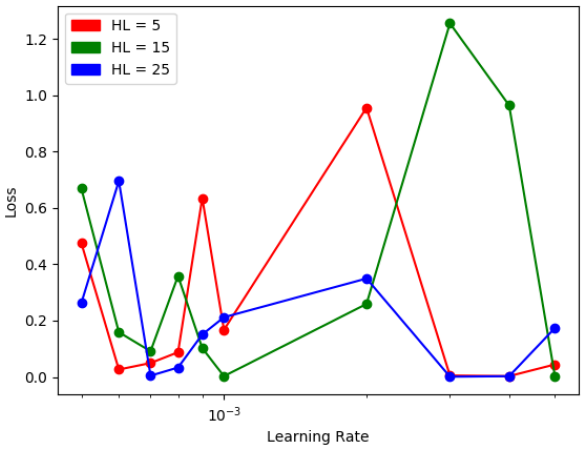

Figure 4: English loss with the complex model

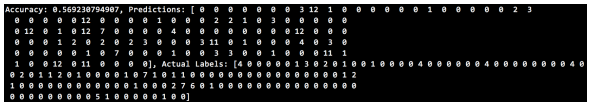

Figure 5: Performance of complex model trained on ANALYSIS tested on DEV

6 Final Results

For a final determination of the model, I trained the complex model on one Swahili dataset and tested it on another. Below is a picture of my prediction versus the correct label:

I wrote a program **generate_results.py** to create a new **result.file** that could be tested in the trec_eval system. When trained on the Swahili ANALYSIS-EN data set and tested on the Swahili DEV-EN, the overall AQWV score of the neural network was 0.1, better than alternative methods that had been used previously to predict cutoff and close to the optimal score of .1408. However, when trained on Tagalog ANALYSIS-EN and tested on the DEV-EN, the model scored .151 on the overall AQWV score compared to an optimal .3575.

7 Conclusion

Ultimately, the model was fairly successful in predicting cutoff even for data sets it had never seen before, though it has high variance across languages. With that being said, there are some large problems with the model. If the query does not return at least 20 documents, then the model cannot predict where the optimal cutoff is since its weights have been trained on a specific type of input. Furthermore, performance varies wildly based on the training and test language, making it difficult to predict how well it would do in a closed-off setting on a completely new language.

```
map                     all 0.1045
aqwv                    all 0.1000
```

Figure 6: Output from the trec_eval system on my generated result file

7.1 Future Work

One problem my model suffered from was a dearth of training examples. In the future, finding more data for the model to evaluate on could help it avoid overfitting and learn more about different types of input. Further, the model uses a one-hot vector for label identification as opposed to computing a number. If the optimal cutoff point is past 20 (which is somewhat unlikely), then the model would not be able to detect it. Finally, it may prove that training the model on one language yields the best results for testing on all other languages. Alternatively, it may be better to save the weights of the best performing model for each language and try each one of them on a surprise language. More testing is needed to see whether the model can be optimized to achieve the best possible AQWV.

8 References

1. Jeffrey Pennington, Richard Socher, Christopher D. Manning. 2014. *GloVe: Global Vectors for Word Representation*