Question Retrieval

Kevin Akos Rankine
rankine@college.harvard.edu

May 12, 2016

1 Introduction

In this project I tackle the problem of automated question retrieval. Question answering (QA) sites and forums like StackExchange are confronted by the problem of directing users to previously answered questions that may be similar to the question they are asking. In order to do this in an automated fashion, we seek some sort of computable metric that can measure the degree of similarity between questions, such that a question and a similar question are assigned a higher score than a question and a dissimilar question. Typically our approach is, given a pair of questions, to map each question to a dense representation, and then compute the cosine similarity of the two representations. I replicate and extend on the some of the approaches and results in Lei et al. (2015).

2 Problem Description

To frame this problem more formally, given a dictionary of words $\mathcal{D}$, and a set of all possible questions $\mathcal{Q} = \{\{w_i\}_{i=1}^{n} : w_i \in \mathcal{D}\}$ consisting of sequences of word tokens, we seek to model an scoring function $s(q, p, \theta)$ with trainable parameters θ for which given a question q, a question p^+ that is similar to q and a question p^- that is not similar to q:

$$s(q, p^+, \theta) > s(q, p^-, \theta)$$

Given an annotated corpus of questions $\{(q_i, p_i^+, Q_i^-) : 1 \leq i \leq N\}$ where q_i is a question p_i^+ is a question similar to q_i and Q_i^- is a set of questions dissimilar to q_i we can optimize the parameters θ by minimizing the maximum-margin ranking loss function (a name I made up) given by

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \max_{p_i^- \in Q_i^-} \left(\max(s(q_i, p_i^-) - s(q, p^+) + \delta, 0)\right)$$

where δ is a small non-negative margin (so that the models do not simply learn the same representation for every question). We minimize the maximum loss for each training example because we want to rank the similar question above all the dissimilar questions, rather than simply ranking it higher on average. Typically we model $s(\cdot, \cdot, \theta)$ by training an encoder function $\mathbf{f}_{enc} : \mathcal{Q} \to \mathbb{R}^{d_{hid}}$ that maps questions to dense representations (of dimension d_{hid}). We then usually take the score to be the cosine similarity of the representations of the two questions so that

$$s(q, p, \theta) = \frac{\mathbf{f}_{enc}(q)^T \mathbf{f}_{enc}(p)}{|\mathbf{f}_{enc}(q)||\mathbf{f}_{enc}(p)|}$$

This allows us to use cosine similarity as way of ranking questions by their similarity to a given question. We explore various approaches to this problem of finding an encoder for which cosine similarity is a valid measure of question similarity, including simple word-level baselines and various neural architectures. We then extend on some of these results and draw some conclusions.

3 Baseline Models

3.1 BOW + tf-idf

One of the simplest approaches to this problem is to map each question to its bag of words (BOW) representation (a $|\mathcal{D}|$ dimensional vector where the i^{th} element is 1 if the i^{th} word in the dictionary is present and 0 otherwise). Under this encoding, cosine similarity corresponds to a scaled size of the intersection between the set of words in each question.

The benefit of this model is that representations and scores are easy to compute and highly interpretable. There are, however, several issues with this model. Using this representation, cosine similarity only measures the degree to which the questions have overlap in terms of the words they share, and fails to incorporate any higher level n-gram or sentence structure beyond that. Further, it weights all words equally, so that semantically uninformative words like `"the"` contribute equally to the similarity score as semantically informative words like `"u-customizer"`. The aim of most of our models is to overcome one or both of these deficits.

An extension of the bag of words approach is to replace each entry in the BOW vector with the term frequency-inverse document frequency (tf-idf) of the word it corresponds to in the question. tf-idf is a way to measure the importance of a particular word in a document. Term frequency (tf) simply corresponds to the frequency of a word in a given document. Inverse document frequency is given by:

$$IDF(w) = 1 + \log \frac{|\mathcal{D}|}{|d \in \mathcal{D} : w \in d|}$$

given a corpus of documents $\mathcal{D}$. tf-idf of a word in a question then simply consists of the product of the term frequency and the inverse document frequency (given a training corpus). The inverse document frequemcy allows us to ameliorate the aforementioned problem of all words being weighted equally when computing similarity. The relatively high frequency of some semantically uninformative words like `"the"` is offset by their low inverse document frequency.

3.2 Continuous Bag of Words (CBOW)

A more novel baseline I considered is a CBOW approach, where we use a a dense representation of each word in the corpus (from running word2vec on the corpus), and then average the representations of each word in a sentence to get the representation of the sentence. In other words, given a question q we have:

$$\mathbf{f}_{enc}(q) = \frac{1}{|q|} \sum_{w_i \in q} \mathbf{W}_0 \delta(w_i)$$

Where $\mathbf{W}_0$ is the word embedding matrix (for which the i^{th} column is the embedding of the i^{th} word) and $\delta : \mathcal{D} \rightarrow \mathbb{R}^{\mathcal{D}}$ maps word to their one hot vector representations. We then compute the similarity of the representations for pairs of questions in the typical fashion. The trainable parameters of this model are then the word embeddings $\mathbf{W_0}$ themselves. Since word embeddings encode similarities between words, this allows us to match questions that have similar words, not simply questions that have the same words. Alternatively, we can fix the word embeddings, and instead learn a bilinear model where

$$\mathbf{f}_{enc}(q) = \frac{1}{|q|} \sum_{w_i \in q} \tanh(\mathbf{W}_1 \mathbf{W}_0 \delta(w_i) + \mathbf{b})$$

and $\mathbf{W_0} \in \mathbb{R}^{|\mathcal{D}| \times \mathbb{R}^{d_{hid}}}$ is fixed and $\mathbf{W_1} \in \mathbb{R}^{d_{hid} \times d_{hid}}$. The idea behind this is that the benefit of using word embeddings as features in NLP applications comes from encoding similarities between the contexts in which words appears. This allows you to "share" model parameters between different words. By directly modifying the word embeddings via the training procedure outlined above, the ability to share parameters between words is reduced. If we use a bilinear model instead, however, word embedding updates are shared between similar words without a decrease in the similarity of the representation of those words. Although this idea is intellectually appealing, it ultimately did not lead to significant differences in results. The drawbacks of these models include the fact the value of the representation decreases with the length of the sentence (since the vectors are dense summing many of them will lose the individual meaning of each contribution). They also do not incorporate higher level sentence or n-gram structure. Further, they also suffer from the problem we mentioned earlier of unimportant words being weighted equally as important words.

4 Neural Models

4.1 GRU Encoder

A more complex type of encoder I considered is a recurrent neural network. I focused on using a GRU, since Lei et al. (2015) consistently got better results with GRUs as compared to LSTMs (which was borne out by my preliminary testing). We can use a recurrent architecture to encode questions by feeding each word of the question into the RNN, and then using the hidden states as the representation. We can then either average over the hidden states or simply take the last one as the meaningful representation. Since at each step of feeding in word tokens, the GRU/LSTM gates what information it lets in, it should be able to both capture long term dependencies in the question body that have semantic importance as well as filter out irrelevant information. The GRU architecture is as below:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}^i \mathbf{x_t} + \mathbf{U}^i \mathbf{h}_{t-1} + \mathbf{b_i}\right) \\
\mathbf{r}_t &= \sigma\left(\mathbf{W}^r \mathbf{x_t} + \mathbf{U}^r \mathbf{h}_{t-1} + \mathbf{b_r}\right) \\
\mathbf{c}_t &= \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}) \\
\mathbf{h_t} &= \mathbf{i}_t \odot \mathbf{c}_t + (1 - \mathbf{i}_t) \odot \mathbf{h}_{t-1}
\end{aligned}$$

where $\mathbf{x}_t$ is the word embedding of the t^{th} word and $\mathbf{i}_t, \mathbf{r}_t$ are the input and reset gates. The benefits of this type of model is that it should be able to both capture long term dependencies in the question as well are filter out irrelevant information. The biggest drawbacks of this type of model is that they are difficult to train and interpret. Lei et al. (2015) pretrain the encoder parts of their models on a language modeling task (given a question body, model the distribution over words in the title and vice-versa) but they report that this actually yielded worse results for LSTMs/GRUs, so we skip this step. Given more time, however, I would investigated whether combining pretraining with some of the score function extensions we detail later would have led to superior results (since the authors hypothesize that the failure of pretraining for these models may stem from the nonlinearity of the learned representation).

4.2 CNN

I also implemented a convolutional neural network for this task. In this model we apply a convolution of a certain width over the input sequence of word embedding vectors followed by a nonlinearity and some form of pooling. The aim is to be able to extract relevant features from small windows of words that capture relevant semantic cues. The CNN (for width k) consists of a set of filter matrices $\{\mathbf{W_i}\}_{i=1}^{k}$ such that each point in the sequence is mapped to a feature vector $\mathbf{h}_t$ such that

$$\mathbf{c}_t = \sum_{i=1}^{k} \mathbf{W}_i \mathbf{x}_{t-k+i}$$
$$\mathbf{h}_t = \tanh(\mathbf{c}_t + \mathbf{b})$$

We then aggregate the state vectors produced at each point in the sequence using max-pooling or mean-pooling. The advantage of this model is that it is easier to train than the RNN, is more interpretable in terms of understanding of what types of information each dimension of the hidden states represent, and should be able to capture higher order cues relative to the word level models and filter out irrelevant information. The downsides are that it cannot capture any sort of long term dependencies beyond the width of its filters and is still much harder to train than the simpler models.

5 Experimental Models

5.1 Weighted CBOW

One extension I undertook is modifying the averaging operation present in my CBOW model to be weighted depending on the word. In other words I sought to extend the model such that we have

$$\mathbf{f}_{enc}(q) = \left(\frac{1}{\sum_{w_i \in q} \lambda_{w_i}} \right) \sum_{w_i \in q} \lambda_{w_i} \mathbf{W}_0 \delta(w_i)$$

$$\lambda_{w_i} \geq 0$$

where $\{\lambda_{w_i}\}_{w_i=1}^{|\mathcal{D}|}$ are trainable parameters of the model and $\mathbf{W}_0$ is fixed. Alternatively, I considered trying to parameterize these values using an MLP (multilayered perceptron) that maps word embeddings to weights. In the same sense as for the bilinear version of the CBOW model this allows us to share parameters between words. This would look like:

$$\lambda_{w_i} = \sigma(\mathbf{W_2}\tanh(\mathbf{W}_1\mathbf{x}_{w_i} + \mathbf{b}_1) + \mathbf{b}_2)$$

where $\mathbf{x}_{w_i}$ is the word embedding of the word w_i. Either of these approaches should allow us to counteract the problem of the average of a large number of word vectors being less meaningful by giving low weights to irrelevant words and high weights to relevant ones. The weights are analogous to the gates in a GRU/LSTM. The weights should also act analogously to IDF in tf-idf, in that either scheme should learn to up-weight semantically relevant words and down-weight irrelevant ones. Making these parameters trainable is analogous to how dos Santos et al. (2015) modify the BOW component of their model by replacing inverse-document frequency with learned weights.

5.2 Non-Cosine Scores

Another idea I had was to look beyond simply using cosine similarity on the output of the GRU. Instead I considered either first applying another MLP to the output of the GRU, and then using cosine similarity on that representation, or simply using an MLP to map directly to some sort of similarity score. The idea behind both of these is that since GRU state is highly nonlinear it is possible that directly comparing entry by entry (which is essentially what cosine similarity does) does not necessarily correspond to the best measure of similarity in the underlying questions.

The first idea posits that, given a hidden representation $\mathbf{h_t}$ from the GRU encoder, we instead map this to a second hidden representation $\mathbf{h_t}^*$ given by

$$\mathbf{h_t}^* = \tanh(\mathbf{W}_1\mathbf{h_t} + \mathbf{b}_1)$$

or

$$\mathbf{h_t}^* = \tanh(\mathbf{W_2}\tanh(\mathbf{W}_1\mathbf{h_t} + \mathbf{b}_1) + \mathbf{b}_2)$$

or a similar transformation (and then compute the cosine similarity on that representation). In the second idea we would replace $\mathbf{h_t}^*$ with a score we are trying to predict instead.

I also sought to experiment with modifying the means by which question bodies are incorporated into the encoders. In Lei et al. (2015) they simply run the encoder on the question title and body separately and then average the representations. An immediate modification I made in order to get decent results was to run separate encoders for question and title. This worked a lot better for me. Intuitively this makes sense because the question body is noisier than the title, so the encoder should be much more selective regarding what information it incorporates. Another modification on this scheme that I considered was to apply cosine similarity to the title and body state representations separately, and then learn a parameter β that modulates the tradeoff between the score from the body and the score from the title. This parameter could either be fixed at test time, or it could be parameterized by an MLP that takes in as input the hidden states:

$$s(q, p, \theta) = \beta * s_{body}(q, p, \theta) + (1 - \beta)s_{title}(q, p, \theta)$$
$$\beta = \sigma(\mathbf{W_2}\tanh(\mathbf{W}_1[\mathbf{h}_q^{(title)}, \mathbf{h}_p^{(title)}, \mathbf{h}_q^{(body)}, \mathbf{h}_p^{(body)}] + \mathbf{b}_1) + \mathbf{b}_2)$$

Due to a lack of time I did not investigate this last extension.

6 Data + Results

6.1 Data

The dataset I used for this project comes from the AskUbuntu dataset used by dos Santos et al. (2015) and modified by Lei et al. (2015). It consists of a $167,765$ question corpus with $12,584$ training, 200 dev, and 200 test questions. The corpus itself contains both the question titles and the question bodies. The titles tend to be fairly short (average ≈ 6, max $= 34$), while the bodies tend to be rather long (average ≈ 60, max ≈ 6000).

The training data is structured as follows: each line consists of the ID of a question, the IDs of one or more similar questions, and the IDs of 100 dissimilar questions. The dev and test datasets are structured similarly, with the ID of a question, the IDs of one or more questions that match it, and the IDs of 20 questions that may or may not match it. The original dataset was based on annotations by users which had high precision but very low recall, so in order to make the datasets more representative Lei et al. (2015) used standard IR systems to retrieve candidate questions for each question, and then used two experts to annotate these as similar and dissimilar.

6.2 Metrics

The metrics I focused on for this project were Mean Reciprocal Rank (MRR) and Precision at 1 (P@1). MRR is defined as the average over all queries of the reciprocal of the rank of first similar question. P@1 is the fraction of all queries where the first question was similar. Other metrics include the more general Precision at n (P@n), which is given by the average over all queries of fraction of the top n items retrieved that were relevant. Mean Average Precision (MAP) is the average over queries of a weighted average of P@n for all n.

6.3 Implementation Details

I implemented the models in Lua/Torch. Other than BOW/TF-IDF I trained essentially all of these models using a single framework. I followed the setup of Lei et al. (2015). I experimented with both SGD and Adam for optimization, and found that while SGD did well with my CBOW model, we really needed Adam to get the neural models to work. During training we sample 20 negative examples for each positive example and minimize the loss using Adam and small batches of 21 training examples each. Batch sizes of 8 and 16 worked pretty well.

I tuned learning rates using values in {5e-3, 1e-3, 3e-4, 1e-5 }. I also experimented with dropout, with values in their suggested range of $\{0.1, 0.2, 0.3\}$. I also tried lower values when not using the final state as the representation , because I hypothesized that high dropout would lead to the final state not being able to incorporate information from early in a long sequence during training. In other words, it would bias training towards the ends of long sequences of word tokens. This was borne out in my experimentation, and incorporating question bodies (which tend to be long) into the GRU model worked much better with low dropout. I also experimented with the margin, and found that low margins between 1e-2 and 5e-2 tended to be good because they prevented overfitting. Using a margin of 0 turned out to get terrible performance precisely as

hypothesized - the models tended to learn a fairly uniform representation (which achieves losses around 0 for this loss function).

I experimented with both using the question bodies and not (although I was able to get results pretty close to those in Lei et al. (2015), I was not really able to successfully incorporate question bodies in a way that improved the scores, possibly because I wasn't using enough of the text - they clipped the body at about 100, while I had to clip it at about 40 because my machines crapped out after that). I also experimented with gradient renormalization above a certain threshold, which helped improve the performance of the models when incorporating the question bodies.

Since pretraining the GRUs/LSTMs on the entire corpus by using an encoder-decoder framework to do language modeling over the titles given the titles of similar questions yielded worse results for Lei et al. (2015) than no pretraining I did not incorporate it.

6.4 Results

Model	Dev MRR	Dev P@1
BOW	34.2	0.005
TF-IDF	66.7	56.1
CBOW	66.8	52.4
WEIGHTED CBOW	65.5	52.4
GRU ($d_{hid} = 280$, LAST STATE)	71.1	59.2
CNN($d_{hid} = 700, k = 3$, MEAN POOLING)	69.9	58.1

Table 1: Results for various models.

BOW performs horribly, as expected. TF-IDF and CBOW achieve similar scores to each other, and are roughly in line with the baselines from Lei et al. (2015). Unfortunately the weighted CBOW model performs worse than the CBOW, but I didn't really have a lot of time to tune it since I spent a great deal of time getting the neural models up to par. I think that given a bit more time spent tuning the parameters/implementation I would have probably been able to achieve better results using it than the baselines.

My best results were achieved by the GRU encoder. This achieved both the highest MRR and the highest P@1. What probably kept the GRU's score from increasing a couple percentage points to the performance of the best GRU model from Lei et al. (2015) was that I wasn't able to successfully incorporate enough of the question body. This is possibly either due to having parameters slightly off, or simply because I used shorter question bodies (due to time + resource constraints). CNN performance is roughly in line with what was achieved in Lei et al. (2015). I also attempted to implement a non-cosine MLP-based similarity function but this consistently lead to the local minimum of uniform representation. On the other hand, adding another MLP layer after the RNN did not lead to the pathological local minimum, but it did not improve scores notably. Unfortunately I spent a great deal of time getting the models from the paper to work, so I wasn't able to completely implement the score tradeoff concept I detailed in section 5.

7 Conclusion

In this project I dealt with the task of automated question retrieval. I was able to replicate some results from Lei et al. (2015) and come close to others. I also came up with several ways to extend the models present in that paper that incorporated some ideas from dos Santos et al. (2015). In terms of shortcomings, I would have liked to have been able to more successfully use the question bodies exactly as they did in the paper, and I think that would have closed the small gap in performance that I observed in the neural models. In terms of challenges, I spent a lot of time debugging stupid errors and waiting for GPU time. The most unfortunate error I made was at one point inadvertently not setting the word embeddings (I had `LT.weights = embeddings` when I needed `LT.weight:copy(embeddings)`), which caused me to mindlessly try to tune hyperparameters for half a week. In terms of successes, I did learn a considerable amount about how to read and implement these types of papers and how to set up an experimental framework within which to train and validate many different types of models.

My code is located at `https://github.com/kevinrankine/cs287-final`

References

dos Santos, C., Barbosa, L., Bogdanova, D., and Zadrozny, B. (2015). Learning hybrid representations to retrieve semantically equivalent questions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 694–699, Beijing, China. Association for Computational Linguistics.

Lei, T., Joshi, H., Barzilay, R., Jaakkola, T. S., Tymoshenko, K., Moschitti, A., and i Villodre, L. M. (2015). Denoising bodies to titles: Retrieving similar questions with recurrent convolutional models. *CoRR*, abs/1512.05726.