Question Word Prediction

Group 16
Zakaria Nassreddine, Alicia Palmér

Abstract

While Question-Answering is a very well-studied NLP task, and for good reason, the fairly related task of Question Word Prediction remains vastly under-studied. In this paper, we attempt to address this problem by training a shallow neural network, inspired by word2vec, to predict the question word in a sentence given the rest of the question and the correct answer. Preliminary results are promising and the model achieves better performance than baseline approaches.

DD2417 Language Engineering
KTH Royal Institute of Technology
May 20, 2022

I. Introduction

Question answering (QA) is a very interesting and well-studied natural language processing (NLP) task, where given a text and a question, the task is to find the correct answer. A less well-studied, but also interesting task, is Question word prediction (QWP), that is finding a correct question word (or phrase) given the rest of the question and the correct answer. For instance, if one has a question *"What is the capital of Sweden?"* with the answer *"Stockholm"*, the QWP model will get the following question-answer pair (QA-pair) as an input:
Question: <qw> is the capital of Sweden?
Answer: Stockholm

The output for this QA-pair will then be **"What"**. The output will however not always be one word, for instance *"How many"* or *"How much"* are also valid question phrases. (The above text taken from the assignment description).

The task presented in this report is to train and evaluate a QWP model using an available QA-corpus.

A. Related work

No other paper was found where this exact problem was solved. The closest problem to this one is the general word prediction problem, where one gets given a context of a word and then needs to predict what the missing word is. This problem is one that can be used as a base for creating word embeddings in the word2vec framework, using the continuous bag of words method, cbow [1]. Next Word Prediction (NWP) is much more commonly studied and Recurrent Neural Networks (RNNs) and, more recently, Transformer architectures have been predominantly relied on to tackle this problem. Several efforts have been invested in the area, touching up on some more advanced notions like the incorporation of textual features (topics) into language models [2] or the scaling of pre-trained NWP to a federated learning scheme [3]. The difference from this method and the QWP problem is however that the position of the focus word and its corresponding context words are well defined. In classic CBoW, a window of a fixed dimension is slid over a corpus to generate fixed length sequences for each word in the corpus, such that the target word is in the middle of the sequence, surrounded by an equal number of tokens both to its left and its right [1]. Our use case is a bit different and it requires us to adopt a more flexible definition of a context that encapsulates everything in the sequence after the question word (or expression) has been removed. However, for QWP, the question (focus) word can be placed anywhere in a sentence and the context varies in size if one considers the remaining words in the question concatenated with the answer to be the context.

II. Dataset

The dataset that was used for training the question word predictor was the Stanford Question Answering Dataset (SQuAD). This dataset consists of questions proposed by crowdworkers based on different Wikipedia pages. [4] Each question is complemented with either an answer, a part of the corresponding text that could be used as the answer, or no answer. [4]

The dataset can be downloaded as a JSON file, formatted as a nested architecture where questions are sub-sectioned under associated fields the question is related to.

III. Methods

A. Assumptions about the data

When constructing the bag-of-words model in the projects, it was assumed that question phrases could at maximum consist of two words, for example "how much" or "for what". It was also assumed that the context of the question word in a question consisted of all other words in the sentence together with the answer itself. This assumption was made since the question word presumably is dictated by the word class of the answer. For example, if the answer is a proper noun, the question word could not be "when".

It was assumed that the question word will always be an interrogative word or phrase, and it is not always certain that it is located in the beginning of the question. For example, a question in the dataset could be formulated as the following:

"If I am a bird, then how can I fly?" Answer: "Using wings"

The question word here would be assumed to be "how".

Another assumption made was that only the first occurrence of an interrogative word in the data would be the question word that we want to predict. Therefore, if multiple instances of interrogative words are present in a sentence, all up until the first question word will be considered the context and the rest of the question will be removed completely.

B. Data selection

To select valuable data samples for training as well as to reduce the size of the dataset, only the questions that had a possible answer were selected. This was motivated by the idea that not using a correct answer to the question would result in the loss of a lot of information that is critical to predicting the right question word.

C. Sample construction

The questions and corresponding answers were first extracted from the raw JSON file. All question marks were removed from the questions.

Since the samples were going to be fed into a neural network, they all needed to have the same number of features. This dimension was set according to the longest question-answer pair in the dataset, determined by the maximum number of words in both the question and the answer together.

In order to separate the words belonging to the question and the ones belonging to the answer, padding was added in the form of strings filled with zeros right between the question and the answer segments. At least one zero was added, this edge case corresponds to the longest question-answer pair sample in the data set, which sets the input layer dimension of the network to the size of the longest pair plus one. All other pairs were then concatenated with the right number of zeros in between in order to reach the target dimension.

In this concatenated string, the question word was determined as the first occurrence of any question word or phrase

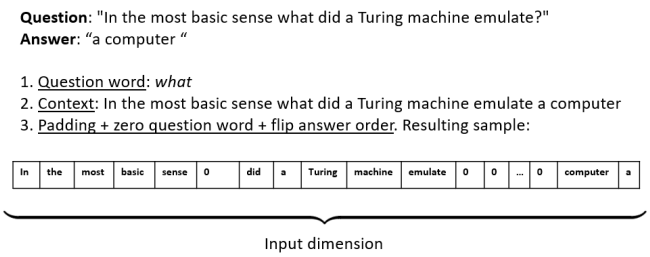

Figure 1: The process of creating a sample from a question-answer pair in the raw data

found. This word was then replaced with a zero. If multiple question words were found, then everything following that word and that word itself were also set to zero. This was done both for the question and the answer parts of the context. Finally, the word order of the answer was reversed, i.e the first word of the answer appears last in the feature vector.

In Figure 1, a simple illustration exemplifies this process.

All the data processing has been performed using pandas, numpy, nltk and the standard json library for python.

IV. Implementation

A. Architecture

The intuition behind our approach was to experiment with the use of word2vec-like architectures as actual classifiers, rather than leveraging their classification prowess as a proxy to build compact word representations. Given the nature of the problem, we intuitively chose a CBoW (Continuous Bag of Words) architecture, as in we decided to use the context (the surrounding words) as input to predict the focus (the target word). After tokenising the input data to construct the corpus, we build a vocabulary out of the unique words that occur with a frequency exceeding a predefined threshold. This allows us to map words to unique indices, and by building a reverse dictionary, we can map from indices back to words which is needed at inference time. As for any word2vec implementation, our model is fairly simple and consists of two layers: an embedding layer takes word IDs and returns embeddings of a fixed dimension (another hyperparameter), it can be thought of as a lookup table with learnable weights, after which comes a linear (dense) layer whose dimensionality is decided by the number of classes in our multi-label classification problem where each question word is a class.

B. Class imbalance

Perhaps the most pressing issue with the data set is the strikingly uneven distribution of classes amongst the training samples, as shown in figure 2.

Not addressing this problem is detrimental to the performance of our model. This can be approached in several ways, like subsampling and oversampling or the use of generative methods to create synthetic data points from the underrepresented classes, but our method of choice was to introduce sample weighting in the loss function. The idea is to weigh the loss computed for different samples differently based on

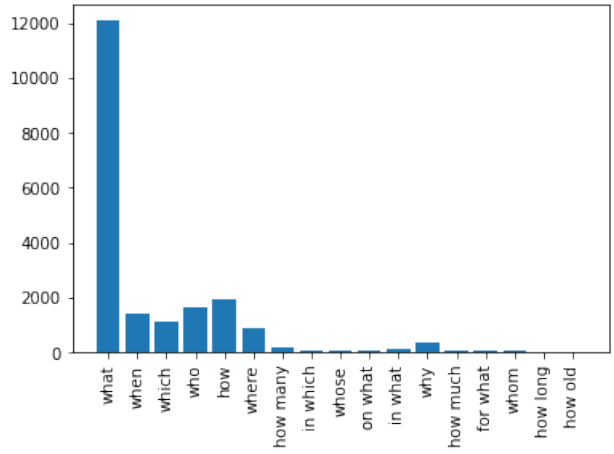

Figure 2: Training set distribution of number of samples per class

the how represented the class they belong to is, by assigning weights that are inversely proportional to the frequency of the class in the training corpus. The weights corresponding to each class were computed using the *compute_class_weight* function provided by Scikit-Learn, making use of the "balanced" heuristic that is inspired by Logistic Regression in Rare Events Data, King, Zen, 2001.

C. Pre-trained embeddings

Given the limited time and computational resources we had at hand, we decided to make use of pre-trained word vectors to have some head start instead of random initialisation, by adhering to the Transfer Learning paradigm to be able to reach satisfactory performances faster. To this end, we used the 6B version of GloVe embeddings [5] to initalise the hidden layer of our network. While this remains better than starting with a clean slate, pre-trained representations aren't domain or task specific so one can achieve better performance by fine-tuning the parameters on the data set of interest, which is what we did by enabling the error to backpropagate through the embedding layer during training, rather than keeping it as a fixed lookup table. The size of the embeddings has an impact on the model's performance, but the gain in accuracy comes at a high computation cost. We followed the general guidelines by the original authors and experimented with 50, 100 and 300 dimensions.

D. Size of the vocabulary

The size of the vocabulary is determined by the threshold that we set to filter out rare words. The higher this threshold is, the smaller the vocabulary. A larger vocabulary accounts for more words but increases the complexity of the model and the training time. We exprimented with a threshold valued at 50 and then 10.

E. Softmax Layer (or lack thereof)

Since we're dealing with a multi-class problem, a Softmax activation function comes as a rather natural choice. But

since we're using the *torch.nn.CrossEntropyLoss* function that expects logits and applies Softmax on its own, we omitted the Softmax layer from our architecture. During our first rounds of training on small values for our hyperparameters to reduce the model complexity, we noticed that sometimes the network would predict non-question words, which is what led us to restrict the predictions to only the classes we wanted to predict. We had initially decided against this to be able to add an evaluation metric that reflects how often the network predicts a question word, regardless of whether or not it is the right one. But we ended up restricting our output layer to a dimensionality that matches the number of classes of interest.

F. Training

We built our model using PyTorch [6], initialised its embedding layer using GloVe embeddings (cf. section IV-C) and trained it using Stochastic Gradient Descent with a weighted (cf. section IV-B) Cross Entropy loss function. In total, 85142 samples were used for training, corresponding to about 80% of the available data.

G. Testing

We pre-processed the raw data and created the padded sample vectors for our test samples, according to the description presented in III.

V. Results

The question words together with their number of occurrences in the test data as well as the frequency at which they were predicted when using the model on the test samples at inference time is presented in Table I.

Question word	Freq. in test data	Freq. in predictions
what	12099	16020
when	1386	989
which	1116	0
who	1657	0
how	1959	1627
how many	157	125
in which	83	0
whose	65	0
on what	50	0
in what	120	0
why	375	102
how much	40	1
for what	58	0
whom	42	0
how long	6	0
how old	3	0
where	872	168

Table I: Frequencies of question word labels and classifier predictions for a model trained on a vocabulary with word frequency larger than 50, 300-dimensional word embeddings for 50 epochs.

A. Resulting accuracy

When evaluating the question word classifier (vocabulary with word frequency larger than 50 & 300-dimensional word embeddings) on the test set containing 20088 samples, the resulting accuracy was calculated to **69.36%**

When using a dummy classifier predicting only "what", the resulting accuracy was calculated to 57.17%. Using a classifier that randomly chose any class had an accuracy of 35.70%.

VI. Discussion

The resulting accuracy of the final classifier was found satisfactory for a 17-class prediction problem, especially taking into account that the data set is very biased towards one class and a dummy classifier only predicting that majority class performed about 12 percentage points worse. It also performed remarkably better than a random classifier.

From the resulting frequency table, Table I, it is possible to see that the bias also is reflected in the statistics of the predicted labels. From this it is possible to reason that even larger weights could have been used for the less frequently appearing question words.

The different assumptions about the data may have affected this classification accuracy since, for example, if multiple question words appear in the same question-answer pair, then it is possible that it is actually the second question word that is the one that one wants to predict. For example, "When Queen Elisabeth was a young girl, what was her nanny called?", the question word would in this sentence actually be "what", yet it gets the label "when" as this is the first interrogative word that appears. One could reason that given the task description, only questions whose question word or phrase appears only at the beginning of the sentence should have been extracted from the data set in order to avoid this problem. It would however have reduced the number of training samples, removing also the samples for which the assumption that the first interrogative word being being the question word would have been valid for.

Another assumption that could have an impact on the classification accuracy is the choice of question words or phrases to predict, since these were not specifically stated in the problem description. If one was to change the definition of question words, for example only including "what" as a question word, and not including "in what", "for what" and "on what" as separate question word phrases, the number of classes would be reduced, which potentially could make it easier for the neural network to find patterns in the data related to the word what. It could however also be better to add more question phrases since for example questions beginning with "What could..." and "How did..." could potentially also be reformulated to begin with a combination of the two examples; "What did..." and "How could...". In this case, the same question words would have different contexts, and therefore potentially make it more difficult to find relevant connections in the training data.

A. Potential improvements of the presented solution

One potential modification that could have been done when creating the data samples from the raw data would be to use the

word class tags instead of the words themselves. This choice could be motivated by the fact that the type of question the answer consists of gives sufficient information for what the question word would need to be. As illustrated earlier, the question word ”when” could not be matched with an answer word of class proper noun or name for example. Since we in this case do not care about the context of the question or the factual information given in the answer, it would probably also make the training of the neural network better since this method would reduce the number of words in the vocabulary. In theory it could however have the opposite effect as well. For example, the words ”could” and ”did” are both verbs in past tense, but questions starting with ”What could...” and ”How did...” would have different question words. Therefore, information could also potentially be lost using conversion to word tags instead of words. This method could be interesting to look at if one was to further develop a QWP system.

Another possible improvement that could be evaluated would be to amplify the impact of the answer in the context. For example, the two questions answer pairs ”Whose apple is this?” ”It is Zakaria’s” and ”Which apple is this?” ”It is the one Alicia picked” only differ in the question word and the answer, and therefore the prediction of the question could benefit from having the answer part of the context weighted more heavily than the question part.

VII. Conclusions

The resulting classifier got a satisfactory accuracy that performed better than both a most-frequent-class predictor and a random predictor. These first preliminary results are encouraging and suggest that this work could be built upon and taken further by, for instance, replacing the words with their POS tags, building a better and more representative data set that is designed speficically for this use case and altering the architecture of the network by replacing the simple fully connected layers with an architecture that better suits this context like Recurrent Neural Networks to capture the contexts of sequences, and notably the more robust variants like Long Short Term Memory networks and Gated Recurrent Unit networks.

References

[1] D. Monia. (2021) Word2vec: Cbow vs skip-gram. Accessed: 19-05-2022. [Online]. Available: https://medium.com/mlearning-ai/word2vec-cbow-and-skip-gram-55d23e64d8b6

[2] S. Ghosh, O. Vinyals, B. Strope, S. Roy, T. Dean, and L. Heck, “Contextual lstm (clstm) models for large scale nlp tasks,” 2016. [Online]. Available: https://arxiv.org/abs/1602.06291

[3] J. Stremmel and A. Singh, “Pretraining federated text models for next word prediction,” 2020. [Online]. Available: https://arxiv.org/abs/2005.04828

[4] “The stanford question answering dataset.” [Online]. Available: https://rajpurkar.github.io/SQuAD-explorer/

[5] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf