Question-Answering System Assignment

Panagiotis Doudos

March 2023

Overview

This is an attempt to create a working question answering model from scratch. A brief overview of the data selected and the task understanding will be shown, followed by an explanation of the methods used as well as the results and main weaknesses of each model. The methodology used for modeling is based mainly on a co-attention encoder-decoder system. Finally some different possible approaches will be discussed, that were not tested due to time constraints.

Chapter 1

Data Structure and Task

1.1 Data Overview

The data selected for this task were the SQuAD 2.0 dataset, a well known dataset for question answering. The dataset includes as variables, question and text pairs, the answer in the text and the starting position of that answer. The data define as the "starting position" of the answer the position number of exact token —letter or symbol— on which the answer begins. Finally they include a text id so as to make it easier for the questions on the same text to be grouped together and an index uniquely characterizing each separate entry.

The main difference of the SQuAD 2.0 dataset to its predecessor, SQuAD 1.1, and an important factor for the analysis part, is that some of the SQuAD 2.0 question-text pairs have no answer, making it harder for any model to make accurate predictions.

1.2 Task

The general task for the SQuAD 2.0 dataset is for the model to be able to exactly match the starting and ending token position (calculated as starting position + answer length). Given the limited computational resources and time, since a deep learning model with over 1 million parameters will be created, the task for the purposes of this assignment will be simplified to only finding the starting token of each answer. The reason behind this is that adding two loss functions, one for each target token will necessitate that back propagation occurs twice per epoch, therefore increasing the necessary resources by a lot. The focus of this task in general is the setup of a working model and not output optimization.

1.3 Notes on Data Handling

Exploratory data analysis is not relevant the requested purpose of this report, but some features that played a part in how the models were built will be discussed:

Text Length The longest text in the training data consists of a bit less than 4000 tokens. Since the number of tokens will be the number of nodes in the final softmax layer of the model, an assumption will be made that no text with over 5001 tokens will be given to the model to assess.

Question and answer duplicates Some of the data entries are either identical questions on the same text receiving identical answers, or identical questions on the same text receiving different answers that vary by a small token margin. i.e "January 7, 2012", "on January 7, 2012". From those entries only one will be kept as an entry representative.

Questions with multiple answer locations Some questions have multiple correct answers in entirely different parts of the text. From those as well, only one answer entry will be kept. This is not the best practice, but time efficiency and a simpler model input is important because of all aforementioned restrictions.

Chapter 2

Co-attention Model

2.1 Original Layer Description

The model we will be trying to implement is a simplified version of the coattention encoder model for question answering, described ans one of the steps in [XZS16] with some personal touches.

The coattention model works with the following steps:

1. a question (Q) and a context (C) representation are given as input to the model
2. An Affine matrix (L) is created as $L = C^T Q$ containing information on both
3. Attention weights for the question and context representations are created as: $Att^Q = softmax(L)$ and $Att^C = softmax(L^T)$
4. An attention layer using the attention weights of each multiplied by the representation of the other is then used. This way the representation of the document based on the context and the representation of the context based on the document is produced.
5. The results are then concatenated and passed through an encoder LSTM

2.2 Differences in Implementation

In this implementation there are two main differences from the original setup. The reason these are used is because in the original paper the coencoder layer was just a simple layer but for this implementation it will be considered the entirety of the model. Those differences are:

1. We will be using an encoder-decoder setup. The representations of the models initially given to the coencoder layer will be outputs of LSTM

encoders, and the final LSTM will play the role of a decoder before the final output layer.

2. The output of the question attention weights - context representation layer will also be concatenated with the decoder output before the output layer

3. Some extra layers or concatenations here and there.

For all models —at first— the output of the model would be a dense 5002 node layer, one for each token we have assumed exists in the longest "imaginable" text plus an extra token meaning "answer not found"

2.3 Fasttext Input Coattention Model

Two slightly different models based on two different inputs were tried. The first input were Fasttext word embeddings. The concept behind selecting Fasttext was that Fasttext takes into account word parts, which is important in the sense that we are looking for a starting token. Ideally, character embeddings should have been used, but they are quite a bit more computationally demanding.

The model operates pretty much as described above. Two separate inputs are inserted to the model, embedded, then passed through one LSTM encoder each. Those are considered the question and text represntations which are used to define the affine matrix L. From there, two sets of attention weights are created, each multiplied to the representation of the opposite type through a Luong Attention layer. The coattention question output is multiplied once again by the context factors —a step performed in some implementations—. They are then concatenated and passed through the decoder. The output of the decoder is then concatenated with the output of the question attention weights - context representation layer before being put through the output layer. Below is an outline of this first model.

2.4 BERT Input Coattention Model

For the second attempt, we changed the input to the model to be the pooled output of the final layer of a small BERT with 6 blocks, 128 layers per block and 2 Attention heads. These sentence embeddings are considered the question and context representations which are then passed through the model. The layer in which the coattention question output is multiplied a second time by the context factors was also removed as it seemed redundant. Below is an outline of that model as well

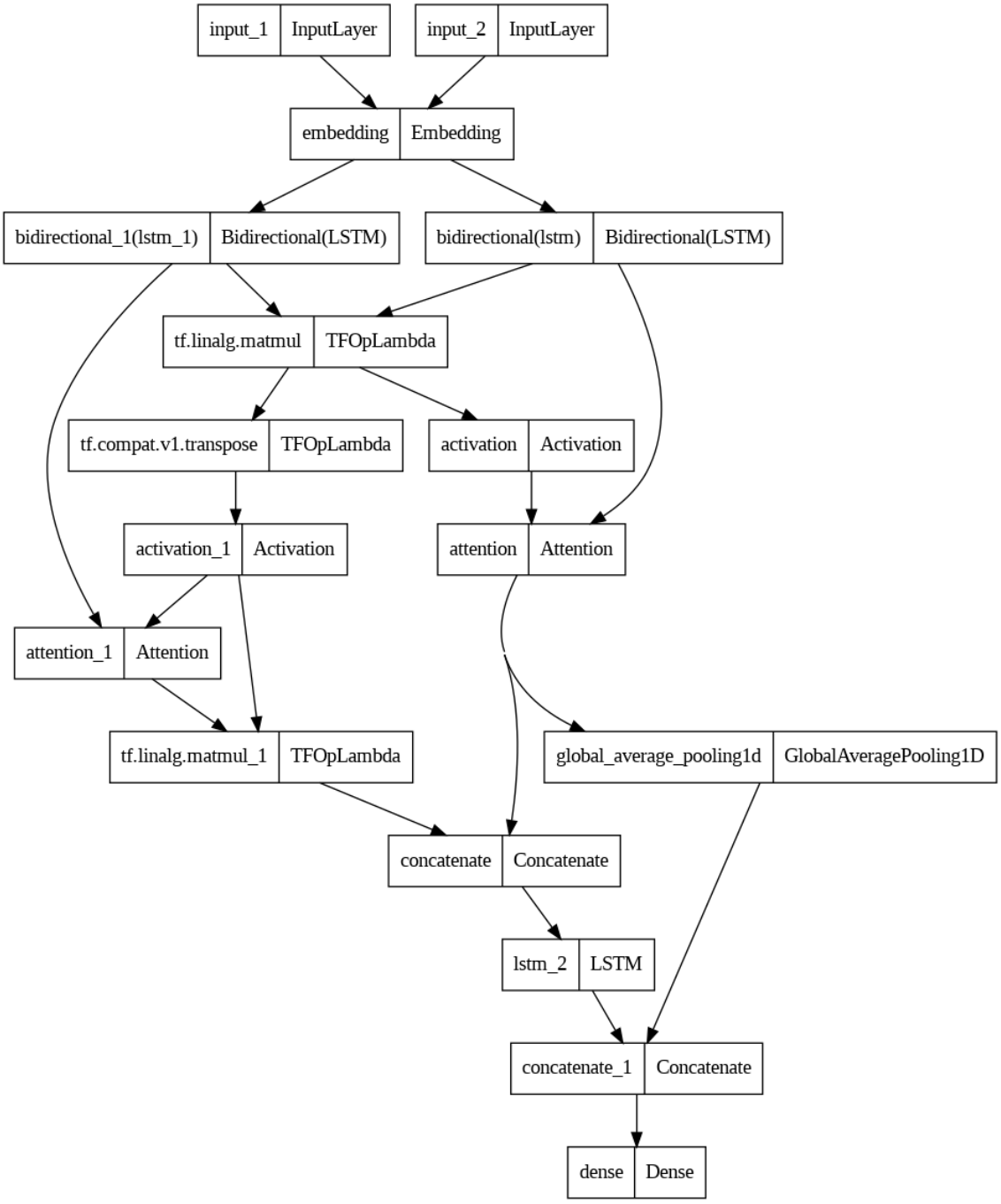

Figure 2.1: An overview of the first model

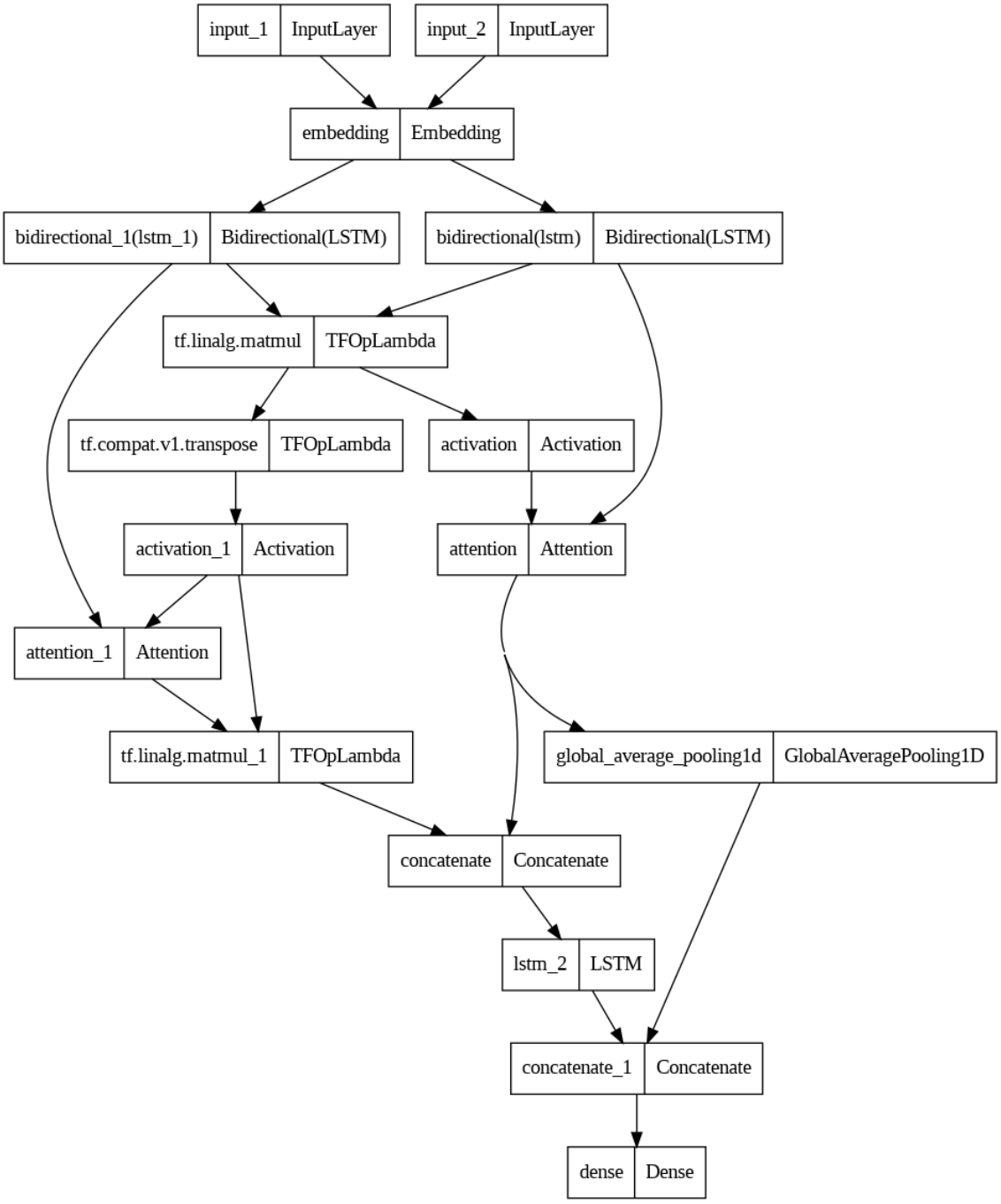

Figure 2.2: An overview of the second model

Chapter 3

Model training and results

3.1 Training, failed results and diagnosis

The models were trained for 5 epochs each, with Adamw as an optimizer and learning rates of 10^{-5} and 10^{-3} for the first and second model respectively. Unfortunately due to hardware limitations no tuning could be performed, but some ideas under the hypothesis of more available resources will be made in a later chapter. The metric that SQuAD traditionally works with is f1-score, but because only the start of each answer was taken into account in our case, accuracy was deemed a satisfactory metric, seeing how we mostly care about how many of the starting tokens we correctly identified.

Unfortunately no case can be made for the first two models. In their only five training epochs they only classified examples to the majority class, that being the "no answer" 5001 class, consisting of about 33% of the training dataset (after train test split). Both models ended up on the same page, returning logits almost identical for every single example after only five epochs.

To diagnose this issue, two hypotheses are the most common. first of all, a complicated model such as this needs more than 5 epochs to even grasp the data structure and secondly, that the data fed to the model, at least at first should be a bit less imbalanced in the available classes. To counter those, we decided to take two measures and retrain the second model on a new hypothesis.

- With only 5 epochs, it is indeed impossible for the model to grasp the data structure, no matter the complexity, since this is not a simple task. Therefore to simplify the problem, we split the context in 10 token "bins" and tried to guess in which bin each example falls. This not only makes the task of the model 10 times easier, but a case can be made that falling 10 tokens off the starting token is not such a big deal, seeing how any answer usually requires context to be understood.
- We undersampled the two majority classes, being class 0 at around 1500 occurrences and class 5001 at around 24000 occurrences, to 250 entries each, so we can see how the models react to more balanced data.

3.2 Second Trial

The output of the second training of the model was much more hopeful. The resulting accuracy on the train and validation sets came at an abysmal 2%, however it showed the trends of a model actually being trained. A fairly steady downward trend on the loss function shows improvement, but most importantly the logits show large differences between the —even wrongly— classified examples. The model seemed to start with once again classifying everything to one class, but now as epochs go by, it spreads and starts finding more bins to classify entries to. This could possibly mean that given enough epochs to train, which depending on the model could be hundreds or thousands, the model can understand the task fairly well.

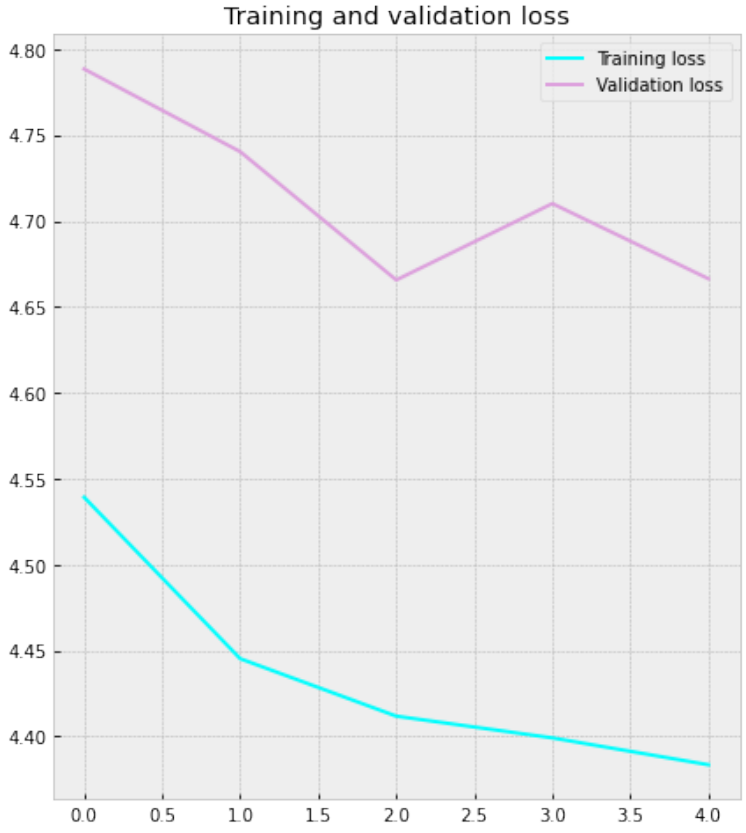

Figure 3.1: Loss curves for the training of the second trial model.

It was also very interesting, that taking Euclidean distance between output vectors of entries categorized in the same class we got a very large variety of distances for class 0, including a distance of around 0.24 (very large for the latent space of the output vectors) to quite less than 0.1

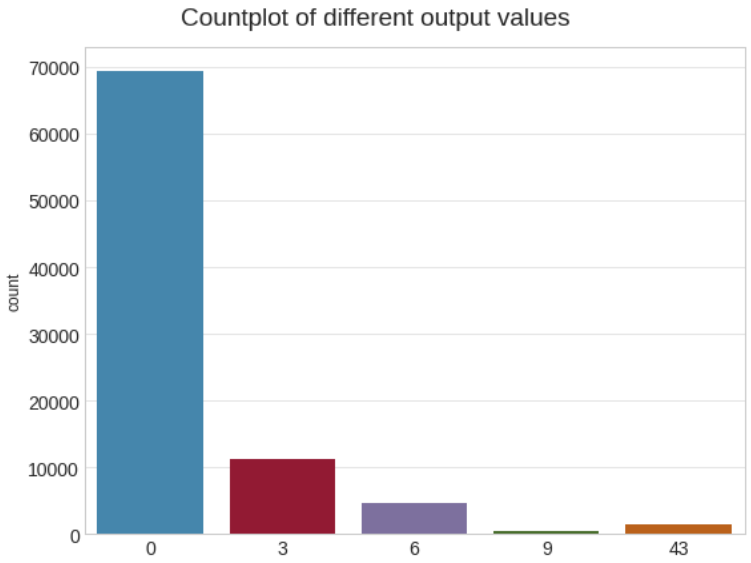

Figure 3.2: The different outputs returned by the model on the training set.

Chapter 4

Assuming Resources, Tuning and Ideas

In conclusion, a short description of things that could be done differently should we have available resources. Tuning would be ideal, but even with more resources, tuning the entirety of such a model is wasteful. Learning rate seems to be the most important factor in such cases, and possibly a few dropout layers, depending on how many epochs we would have the luxury of training the model. Secondly, possibly reinforcing this encoder-decoder block with another similar one to try and make the model more conscious to small quirks of the text. Using character embeddings would obviously be another big plus, given that they are closer to the nature of the problem.

Overall, almost everything could be improved given enough resources, and hopefully this model would be able to learn if given enough epochs, making a solution to the problem.