Homework 8: Question Answering with SQuAD

Due April 23, 2021 (11:59 PM)

The link to this homework's colab is: https://github.com/dbamman/nlp21/blob/main/HW8/HW_8.ipynb

1 Introduction

For our last homework of the semester, we will be implementing a model for Question Answering. QA draws on many of the areas we've explored throughout the semester, including syntactic and semantic representations, parts of speech, named entity recognition, and information retrieval.

One of the largest and most commonly used datasets used in Question Answering research is the Stanford Question Answering Dataset (abbreviated to SQuAD). In SQuAD, QA is formulated as a recognition problem by making the simplifying assumption that the answer to any given question must be available as a continuous span of text in a given paragraph. This paragraph (on the right in the Figure 1) is referred to in our QA model as the "Context". Each datapoint in SQuAD contains a question, and answer, and a context; given a question Q and a context passage P, by training a model to predict the start and end indices of the words in the Context that correspond to the answer, we can train a QA model using supervised learning. The training data X consists of the questions and context, and the training labels y consist of the answers. Below is one example of a datapoint drawn from SQuAD:

Question: Other than San Bernardino, what is the name of the other city that maintains the districts including University Town?

Answer: Downtown Riverside

Context: The San Bernardino-Riverside area maintains the business districts of Downtown San Bernardino, Hospitality Business/Financial Centre, University Town which are in San Bernardino and **Downtown Riverside**.

We will be implementing a Neural Network model for QA based on the paper Reading Wikipedia to Answer Open-Domain Questions (Chen et al., 2017). For the above example, this model is jointly trained to maximize the probability that "Downtown" (the token in position 29 of 31 tokens in the context) is the *start* of the answer and "Riverside" (the token in position 30) is the *end* token of the answer. This model uses an LSTM recurrent neural network to encode the Question, another LSTM to encode the Context, two Attention mechanisms, and finally two classification heads, one to predict which token in the Context is the *start* of the answer and one to predict which token is the *end* of the answer. The full model architecture is shown in Figure 1 below (reproduced from the lecture slides).

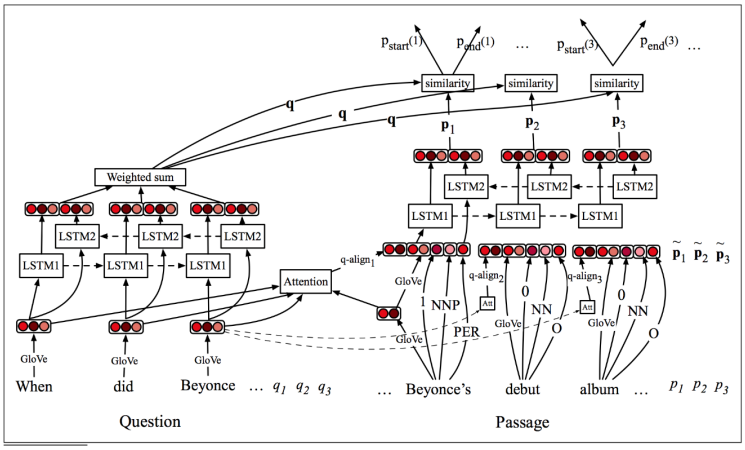

Figure 1: QA model architecture from Chen et al. (2017)

2 Adding Part of Speech and Named Entity features to the model

One interesting aspect of this model is the variety of features in use. As shown in Figure 1, we use pretrained GloVe word embeddings to featurize the words in both the Question and the Context. These are used as input features for the encoder LSTMs and also for the Attention mechanism that computes a linear combination of the question word embeddings as an additional input to the Context encoder LSTM (on the right). In addition to these embedding features, each word in the Context passage has one extra feature indicating its part of speech (e.g. NNP), and another feature indicating its entity type (e.g. PER).

The NeuralQA class provided in the Colab contains Pytorch code for this model. Your task is to complete the implementation of the NeuralQA model by adding embeddings for Part of Speech (POS) and Named Entity Recognition (NER) features. Similar to the positional embeddings we used in Homework 7, these POS and NER features are categorical rather than continous variables. You will incorporate these in the model by embedding each of these categorical features into 16-dimensional vectors that can be trained end-to-end. These two feature vectors should then be concatenated with the other GloVe-based features used to represent the Context as shown in Figure 1.

After completing Deliverable 1.1 and 1.2, you will be able to train your network to answer questions. We evaluate the model using an accuracy metric called 'Exact Match'. This metric tests whether the span predicted by the model exactly matches the labeled answer. This means that both the *answer start* and *answer end* tokens need to be correctly predicted in order to count as a correct answer.

Because Question Answering is a challenging task that requires reasoning not just about the syntax and semantics of text, but about any entities (e.g. The Empire State Building or California) that might appear in the data, we need a fairly large amount of data to train this model, and we use a larger, case-sensitive vocabulary of pre-trained GloVe vectors. The training code in the Colab notebook might take 10 minutes or more to run, so keep that in mind when working on your assignment!

2.1 Deliverable 1.1: Complete the __init__() method

For this deliverable, initialize two embedding matrices in the __init__() method: one for the POS embedding and one for the NER embedding. These matrices should be initialized from scratch and should update during training, and the size of each embedding should be 16. The embedding matrices should be created with space to encode, respectively, the part-of-speech tags provided in the 'pos_tag_list' variable, and the NER tags provided in the 'ner_tag_list' variable. Hint: (because we have variable length sequences that need to be padded for training in Pytorch, we also need to make

space for a special PAD token these embedding matrices).

2.2 Deliverable 1.2: Complete the forward() method

In the forward() method, fill in the code to access the appropriate embeddings for the given POS and NER features. Then, combine these embeddings with the rest of the Context representation that has already been computed.

3 Deliverable 2: Ask the model some questions

Now that we have a trained model, let's try answering some questions of your own. Choose a paragraph from any page on Wikipedia (or write your own if you are feeling adventurous), copy that paragraph into the Colab (make sure it's less than 150 words, since we restricted our training data to short paragraphs to make the training time more manageable), and write a question that can be answered by highlighting a span from that paragraph. Try to find one paragraph/question pair that the model is able to answer, and another paragraph/question pair that the model is not able to answer.

In 200 words or less, give a quick summary of what kinds of things you think the QA model has learned, and what kinds of things it has failed to learn. Along with this writeup, copy your two paragraph/question pairs and their associated 'top answers' printed out by your model into a PDF file to turn in on Gradescope.

4 How to Submit

1. Download your Colab notebook as an .ipynb file (File $\rightarrow$ Download .ipynb)
2. Submit **HW_8.ipynb** to the HW8: Code (ipynb) assignment on Gradescope. The file must be named **HW_8.ipynb** for the Gradescope autograder.
3. Create a PDF file called **HW_8_Writeup.pdf** containing your writeup of Deliverable 2. Submit this PDF to the HW8: Writeup(PDF) assignment on Gradescope.