Abstract

In this assignment, a deep learning architecture was created for use on a reading comprehension task. The specific task was finding the answers to queries in a paragraph of contextual information, with the dataset used being the Stanford Question Answering Dataset. The centerpiece of the architecture were bidirectional LSTMs that were used to encode the question and context representations, which were combined into a context vector, which when weighted was used to predict the beginning and end of the answer spans. Owing to limitations in time and personnel, many means remain to potential improve the predictive power of the model, which obtained an F1 score of 5.165 on the dev set, and these potential improvements are described in the paper below.

1 Introduction

1.1 Dataset

The Stanford Question Answering Dataset (SQuAD) has 100k triplets consisting of questions, answers, and a context paragraph in which the answer may be found. The context paragraphs come largely from Wikipedia, and the questions and answers were extracted by humans, and human accuracy at the task remains unsurpassed although recent models have shown impressive results at this and other reading comprehension challenges.

1.2 Implications for the Model

Unlike named entity recognition which requires sorting into a few predefined categories, question answering is much more open-ended, with any of the words in the context paragraph possibly being the beginning of the answer and the
The answer can simply be the beginning word or stretch to the end of the context paragraph. Determining the answer span therefore requires the model to learn the complex interplay between words and ideas in the query and in the context paragraph in their entirety, and keep in perspective important features as it searches the context paragraph. This need, to remember and prioritize is a natural fit for LSTMs, and so they were the main components of the model implemented.

2 The Current Model: The Sequence Attention Mix Model

The first decision made was how to take in the data. Initially a batch size of 10 question context paragraph sets was used, which was shrunk down to 5 when it was suspected that memory issues might be behind the failure of a submission that was successful on the development set on the test set, though that was ultimately unsuccessful. In each batch, the questions and context paragraphs were padded with zeros to the maximum length of a question or paragraph in the batch respectively. The now padded questions and paragraphs were then fed into to be encoded, which was done by LSTMs. These LSTMs were bidirectional in order to take into account both the forward and backward context of each part of the questions and paragraphs. After being encoded by the LSTMs, the respective outputs were then combined to create the context vector.

To decode our representation and extract the answer span, the context vector was then combined with the paragraph representation. Then using trained weights, the paragraph representation concatenated with the context vector was classified as to the probability of being the start point of the answer span, and using a different set of weights the various points were estimated as to the likelihood of them being the ending point of the answer span. The training of these weights and other parameters was done using an Adam optimizer, with loss being computed via softmax versus one-hot vectors that consisted of the actual start and stop of the answer spans within a paragraph.

3 Potential Upgrades to the Model

There were a number of potential upgrades that were not implemented due to lack of additional time or personal required to deal with debugging, segmentation faults, memory issues or other complications associated with more sophisticated models before the project deadline, though not necessarily for lack of effort. The first and foremost priority would be to add another bidirectional LSTM to the model, this time in the decoder. Using weights has limitations because of their inability to grasp connections and context throughout the paragraph. In fact, occasionally they would predict the end of the answer span at point previous to where weights had predicted the beginning. Although code was written to reverse them in those eventualities, they illustrated the need for
a decoding mechanism that can take into account the whole context paragraph.

There’s also no reason do stop at adding just one mechanism to the decoding. Altogether alternative models could be added as well, with a vote being taken of the various members of the ensemble to make the prediction of the answer spam. Other researchers have gotten increased accuracy even with highly effective models by using an ensemble.

In addition to adding complexities of the architecture, there are simpler ways to boost performance. One of the very simplest would be using additional training time to go over the entire training set. Care would be needed to prevent overfitting to the training data, so likely dropout would need to be added to the training.

Another option would be to take more time tune the learning rate, hyperparameters, and to explore other optimizers than the Adam Optimizer. In spite of many other obstacles, exploding gradients did not noticeably occur in the span of the lifetime of this model. However, if they do occur after some alterations, they could be taken into account with gradient clipping.

Additionally, if memory issues were not a problem, additional input information could be used. In the bidirectional attention flow paper, researchers included not just word embeddings but character embeddings for every character. Less drastically, in the current model the default GLoVe embedding was used, 100.npz, but a larger size could be used to improve performance if run-time and memory issues could be resolved.

Acknowledgments

This project was made possible thanks to generosity of Microsoft which donated time on Azure GPUs for training, and the faculty and course staff of CS224n, without whose assistance it would not have possible.

References


