Question Answering using simple nn-model with attention

Danit Widder (ID. 036806750)

Submitted as a final project report for Natural Language Processing, IDC, 2019

1 Introduction

Question Answering is a reading comprehension task, consisting of answering a query about a given context paragraph, where the answer is a span within the context paragraph. It requires modeling of the given paragraph as well as the query and interactions between them, in order to output the relevant answer found in the context paragraph. In this project I've built and trained a simplified neural-network model based on BiDAF model suggested in the paper "Bidirectional Attention Flow for Machine Comprehension". The model was simplified by omitting the character level embedding Layer and two-layer Highway Network, using only a simple word embedding layer, as well as using a simplified output layer.

The model uses a modular attention layer, in order to compare several attention schemes' results for this task. I've compared a dot-product attention, bi-linear attention and bi-directional attention flow (BiDAF) used by my module and was able to get rather good results on SQuAD v1.1 dataset, given the model's simplicity.

1.1 Related Works

The state of the art models for this task are complex ensemble models that can overdo human's performance. The current leading models for SQuAD v1.1 and v2.0 are models that are based on BERT and XLNet by Google.

In this project I've focused on being able to receive decent results with a simplified model I've built and trained on my laptop, comparing various attention schemes, and didn't aim to reach state of the art's result.

2 Solution

2.1 General approach

The general approach to the problem was to encode the context and the query separately, and then to combine them using an attention layer, allowing context to query and query to context attention, depending on the chosen attention scheme. The encoded context and attention output are then combined by concatenation and are passed to an output layer that predicts the start and end offset of the answer span in the given context.

Preprocessing: Context and queries are tokenized to words using spaCy word tokenizer. The vocabulary is built from words seen on the train dataset and any unseen word is considered as out of vocabulary.

Embeddings: All word tokens are represented by GloVe-300 word vectors. The model uses the pre-trained embeddings as input. Word Vectors can tuned while training if this option is chosen (default is False).

RNN Encoder: The model uses a bi-directional LSTM in order to encode the context and the query. The hidden size used is configurable and was set to 100 (due to RAM limitation).

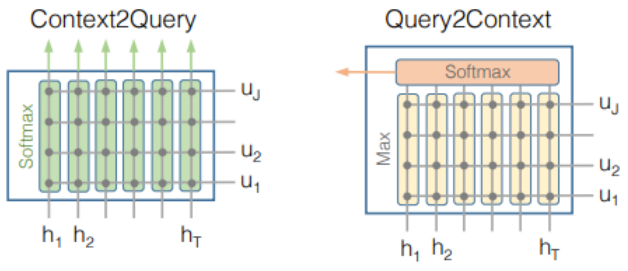
Attention:
Let the batched contexts hidden states matrix be C, the batched queries hidden states matrix be Q and the query hidden dimension be d_Q.
I've compared 3 different attention schemes:

- Dot Product:

  The attention implemented is a scaled dot-product. attention scores are given by $\frac{CQ^T}{\sqrt{d_Q}}$ and are multiplied with Q to get the attention outputs.

- Bilinear: attention scores are given by $\frac{C(W_aQ)^T}{\sqrt{d_Q}}$ where W_a is a trainable weight matrix in the attention layer, and are multiplied with Q to get the attention outputs.

- BiDAF: The attention scheme described in the paper "Bidirectional Attention Flow for Machine Comprehension". The main idea of this attention scheme is that attention should flow both ways: from context to query and from query to context. This attention computes both Context2Query and Query2Context while the previous attention schemes computes only Context2Query attention. The implementation for this attention scheme is taken from Stanford's cs224n class example code.

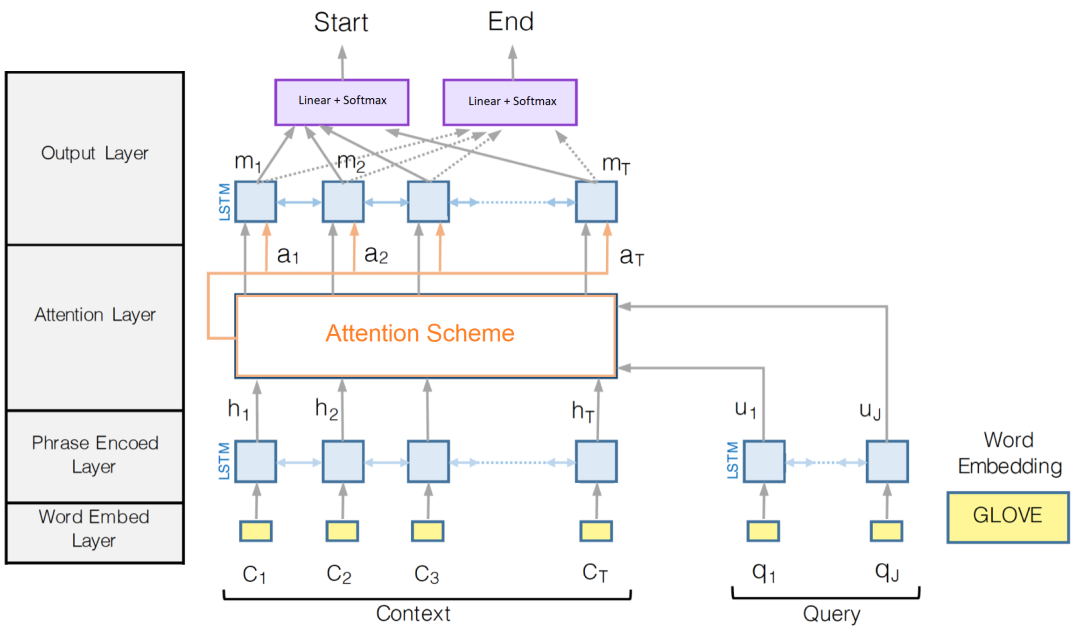


The above image demonstrate the Context2Query and Query2Context attention, where h_i stands for context hidden states and u_i stands for query hidden states.

Modeling and Output Layer: The output layer consists of two fully connected linear layers: one for predicting the start offset and one for predicting the end offset. The offset probabilities are then given by performing (masked) softmax on the linear layers' outputs, and then the start and end offsets are determined by considering the likeliest pair of [start,end] offsets within the maximal answer span length (provided argument).

Adding a modeling LSTM network before the output layer, so that its input is the combined context-attention, and its output is the input for the two offset prediction fully connected linear layers, improves the results. I use a 2 layers bi-LSTM as BiDAF's modeling layer does.

Model Architecture Diagram:



2.2 Design

The implementation uses PyTorch neural network library. The model.py file contains the suggested model's code, and layers.py contains the implementation for all sub layers and attention schemes.

At first I've implemented the model and training routine in a sequential way that doesn't support batching. It was too slow and couldn't be trained in a reasonable time, even when running on GPU. In order to be able to train this model efficiently I had to change the model and its layers so that they could handle batching. The batch training and evaluation code is based on the code provided by Stanford's cs224n class default SQuAD project, with several modifications and adjustments to fit with my model.

The training of the model was performed with batch size of 16 (configurable) due to RAM limitations, and best result was received when the model is optimized with Adadelta optimizer algorithm with starting learning rate of 0.5 (configurable). Gradient clipping is performed in order to avoid numerical overflow (exploding gradients). The model is evaluated every configurable number of steps, and the best performing model is saved. The metric to determine best checkpoint is configurable and set to F1 by default (see next section). Training is run in parallel on several GPUs if exists or on CPU if there is no GPU device.

3 Experimental results

SQuAD provides train and dev datasets, so that model training is done only on the train dataset while model evaluation is done on the dev set. Each context-query pair was given three ground truth answers The measurement metrics commonly used for this task are:
Exact Match (EM): measures the percentage of predictions that match any one of the ground truth answers exactly. Exact Match is a binary measure (i.e. true/false) of whether the model output matches the ground truth answer exactly. For example, if the model answered a question with 'Einstein' but the ground truth answer was 'Albert Einstein', then the EM score is 0 for that example. This is a fairly strict metric.
F1: measures the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens, and compute their F1 score (the harmonic mean of the precision and recall). We take the maximum F1 over all of the ground truth answers for a given question, and then average over all of the questions. F1 is a less strict metric: In the 'Einstein' example, the system would have 100% precision (its answer is a subset of the ground truth answer) and 50% recall (it only included one out of the two words in the ground truth output),thus a F1 score of $2 \cdot prediction \cdot recall/(precision + recall) = 2 \cdot 50 \cdot 100/(100 + 50) = 66.67\%$

	EM	F1
Dot-Product Attention:	65.31	75.00
Bilinear Attention:	66.33	76.28
BiDAF Attention:	68.57	78.56
BiDAF (single) Model:	68.0	77.3
Human Performance:	82.3	91.2

The table presents the results received with Dot-Product, Bilinear and BiDAF attention schemes used with my model, compared to BiDAF model results for single model presented in the BiDAF paper, and to Human Performance on SQuAD 1.1 dataset.

4 Discussion

The main insight is that the attention layer in most Question Answering models is the combining layer of context and query and thus has a very large impact on model's results. Attention schemes which were used for sequence-to-sequence tasks can also be applied for this task, as well as attention schemes which were designed proprietorially for this task. The conclusion from the results is that in this case the proprietary designed attention (BiDAF) has achieved better results then other attention schemes.

Another result of this project is that I was able to achieve similar results on SQuAD 1.1 as were presented on BiDAF paper, using the BiDAF attention layer with my simplified model.

This project has been my first encounter to neural-network model building and training, and I was surprised to see that it is possible to receive, even if only average, results by constructing a quite simple model that I was able to train on a limited resources laptop (but that has a GPU).

During this process I've learnt about the Question Answering task, as well as its more challenging version of open-domain Question Answering, about construction of a neural network model, using of attention, and how to implement and train a neural network using PyTorch.

5 Code

The code is uploaded to: https://github.com/danitwid/QuestionAnswering
The data files are too large to upload but can be obtained by running setup.py (takes a while to download GLoVe embeddings and run SQuAD pre-processing)
I've included some train and test logs I ran.

References

SQuAD: https://rajpurkar.github.io/SQuAD-explorer/
BiDAF: https://arxiv.org/abs/1611.01603
Stanford cs224n: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture10-QA.pdf
https://towardsdatascience.com/nlp-building-a-question-answering-model-ed0529a68c54
Attention:
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
https://arxiv.org/pdf/1508.04025.pdf
https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html