Codalab username: ajaffery

Modular Sequence Attention Mix Model

Kostya Sebov
Stanford University
ksebov@stanford.edu

Ahmed Jaffery
Stanford University
ajaffery@stanford.edu

Abstract

Reading comprehension is an important area of NLP research. The SQuAD dataset is a new dataset comprising 100k question-answer pair dataset that is extracted from Wikipedia that formalizes this task. The goal is to accurately extract the span of the answer directly from the context paragraph. Answers are evaluated on an F1 and Exact Match (EM) metric. In this paper we explain our experience implementing a Tensorflow model as a part of Stanford CS224N course that strives to accomplish such task. As a derivative of 4 recently published papers and implements it uses LSTM RNNs and various attention mechanisms to learn the interaction of the query and context and then estimate the starting and ending positions of the answer span.

1 Introduction

There have been many efforts to develop a fully encompassing NLP model that can accurately understand what humans are saying, and how to respond accurately. In this aspect, reading comprehension is a critical part of Natural Language Processing. Just as in humans, in order to evaluate how well a system has understood what it has read, we must ask it questions and then grade it based on the answer it returns. This is important because every statement can be viewed as a question and answer system based on contextual references. This method could be used as a universal way of testing a machines understanding of any text.

There are potential many ways the question/answer task can be specified. In our paper we use the recently released SQuAD dataset that simplifies and formalizes the task. It is the largest Question/Answer Dataset where each sample consists of a question that has high quality answer that spans directly from the corresponding contextual document. This allows the system to learn within reasonable constraints and not rely on outside knowledge for the answer, meaning that we can much more accurately determine how well a system can understand a document. The SQuAD team maintains an international competition site where any team may submit their work and be compared against the others on their performance on a secret testing dataset based on exact match (EM) of the answers as well as F1 score.

This paper describes our attempt to build an entry to such competition in the course of Stanford cs224n course. In order to build it we drew key ideas from several recently published papers that all utilize Recurrent Neural Networks (RNN) to capture the numerical representations of the Context document and the Questions then use various techniques to recombine them into a common knowledge that is later analysed, often using RNNs as well, to generate the answer span usually by predicting the its beginning and end positions in the Context.

Codalab username: ajaffery

We will use the Tensorflow [10] framework, the SQuAD dataset for training, and the data helper files provided by the CS224N Teaching Assistant team. Being novices in the field of Machine Learning we encountered multiple obstacles, which we fought to overcome with various degrees of success. In the paper we outline our major blocks and techniques we used to remove them.

The model source code is available at the following link:

https://gitlab.com/ksebov/su-cs224n/tree/master/assignment4/code

2 Background and Related Work

2.1 SQuAD Dataset

The Stanford Question Answering Dataset is a new reading comprehension dataset consisting of questions posed by crowd workers on a set of Wikipedia articles. The answer to every question is a segment of text, or span, directly from the corresponding reading passage. With 100,000+ question-answer pairs on 500+ articles, SQuAD is significantly larger and more accurate than previous datasets.

Below is a distribution of the length (by word count) of the training dataset. The test set is hidden and not released to the public for fair evaluation reasons.

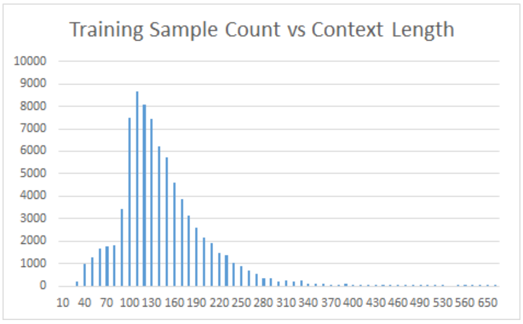


Figure 1. Context length distribution

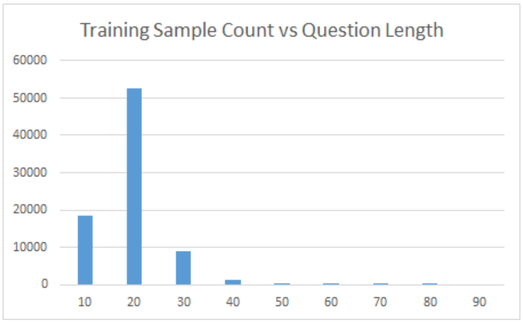


Figure 2. Question length distribution

2.2 Related work

We drew our inspiration from the following 4 papers, most of which were listed in the CS224N Assignment 4 specifications [??].

Along with the assignment itself, the Multi-Perspective Context Matching for Machine Comprehension byt Zhiguo Wang et al. provided general outline of a typical attention-based encoder-decoder model used in reading comprehension task.

Our encoder is based on Dynamic Coattention Networks For Question Answering by Caiming Xiong et al. [5] and utilizes Bidirectional LSTM RNN whose hidden states for Context and Question are recombined using affinity matrix.

Out decoder uses ideas from Bidirectional Attention Flow for Machine Comprehension by [1] Minjoon Seo et al. [1] that employs multiple levels of RNNs capped with linear projection and softmax classification to predict the beginning and end positions of the answer inside the context paragraph.

Finally, we borrowed an interesting idea of using learned bilinear projection matrix to implement flexible nature of the relationship between question and context words from the A Thorough

Codalab username: ajaffery

Examination of the CNN/Daily Mail Reading Comprehension Task by Danqi Chen, et al. [6]

Our training set used a database of pre-trained word-vectors generated by method suggested in the Glove: Global vectors for word representation by Jeffrey Pennington et al.[7]

3 Approach

3.3 Outline

Our model consist of the following layers:

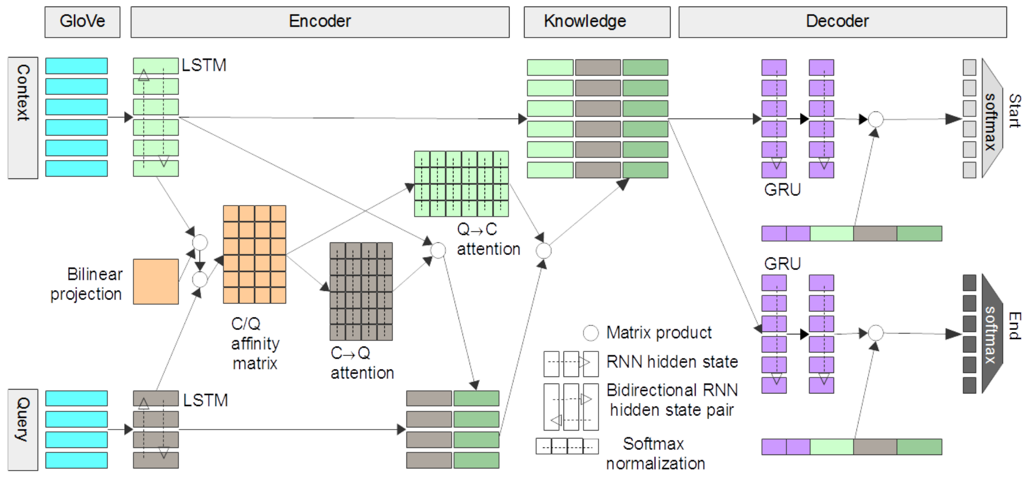


Figure 3. Answer-GRU Model

3.3.1 GLoVe Pre – Processing Layer

This layer maps each word to a vector space based on pre-trained GLoVe word vectors to obtain the word embeddings related to each word. *q_vectors* will be the word embeddings for the questions and *c__vectors* will be the embeddings for the context paragraph. Out-of-vocabulary words were initialized with random vectors.

Max_c is a user selected parameter that is based on the dataset. This is chosen by creating a histogram of context paragraph lengths, and choosing a length that covers 95% of the dataset. *Max_c* is the maximum context length our model will process.

Let $c_{len} = max\_c$ and $q_{len} = max\_q$

$$q_{vectors} s \in R^{d\, x\, c_{len}} \ ; \ c_{vectors} s \in R^{d\, x\, q_{len}}$$

3.3.2 Context Embedding Layer

This layer is designed to model the temporal interactions between words. For the baseline, we run a Bidirectional LSTM on both the $q_{vectors}$ and $c_{vectors}$ separately in order to read the context paragraph and question both forwards and backwards. We then concatenate the forward and backwards outputs from each time step and obtain D from the $c_{vectors}$ and Q from the $q_{vectors}$.

$$D = BiLSTM(c_{vectors}) \qquad Q = BiLSTM(c_{vectors})$$
$$D \in R^{2d\, x\, c_{len}} \qquad Q \in R^{2d\, x\, q_{len}}$$

Codalab username: ajaffery

3.3.3 Attention Layer

The attention layer is meant to link the context and question paragraph together in a meaningful way. Similar to the co-attention calculation from C. Xiong et al. [5] and Seo et al. [1] we combine the context and question paragraph.

The inputs to the layer are outputs from the context embedding layer, D and C. Here we calculate our affinity matrix L which scores the correlation between every pair of context words and question words.

$$L = D^T \cdot Q \qquad L \in R^{q_{len} x\, c_{len}}$$

We then normalize this by taking the softmax of L with respect to both Q and D separately to use for our attention matrix calculation.

$$A_Q = softmax(L)\ w.r.t\ Q \in R^{c_{len} x\, q_{len}}$$
$$A_{DT} = softmax(L)\ w.r.t\ D \in R^{q_{len} x\, c_{len}}$$

Using A_Q we calculate the attention of the context paragraph with respect to each word in the question. Similarly using A_{DT} we calculate the attention of the question with respect to each word in the context paragraph.

$$C_Q = D \cdot A_Q \qquad Q = concat(Q, C_Q) \qquad C_D = A_{DT} \cdot Q$$

3.3.4 Knowledge Layer

After computing the attention we calculate the final encoding: $encoding = concat(D, C_D)$
This allows us to correlate our attention matrix with the temporal data of the context paragraph and provides an encoding for choosing which span in the context could be the best answer. This encoding is the co-attention similar to C. Xiong et al. [5].

3.3.5 Decoder Layer

The decoder predicts a start and end answer index separately using 2 layers of GRU RNN, followed by linear projection and softmax normalization. For $pos \in \{start, end\}$

$$d_{pos} = GRU(encoding)$$
$$pred_{pos} = b_{pos} + d \cdot U_{pos}$$
$$prob_{pos} = (softmax(pred_{pos}))$$
$$loss_{pos} = CE(prob_{pos},\ onehot(answer_{pos}))$$

Here both b_h and U_h are trainable vectors so the model can better learn sub phrases based on the GRU output, CE is a cross-entropy metrics used to calculate the loss. We train the network to minimize:

$$loss_{total} = loss_{start} + loss_{end}$$

3.3.6 Prediction Layer

To generate resulting span we first attempt to use individual argmax:

$$result_{pos} = argmax(prob_{pos})$$

In cases where $result_{start} > result_{end}$, the resulting we assume the result is one-word span at a position:

$$result_{start} = result_{end} = argmax(prob_{start} \cdot prob_{end})$$

Codalab username: ajaffery

4 Experiments

4.1 Construction

Building ML model of such complexity was a first-time experience for both of us so the process proved to be quite challenging.

We used a starter code that tokenized context and question texts and converted it into files containing lists of indices into a vocabulary of words corresponding to the above GloVe set. Loading these files into Python variables was easy although in the process we discovered a bug that generated many invalid ground truth answer spans where the start followed the end position.

The next step was the first major challenge-- implementing a basic Tensorflow model. Since we both lacked practical experience, it took about a week to connect the dots and build trainable graph from input placeholders to the loss and optimizer ops.

The first iteration used different classifier in the decoder. As suggested from the Assignment we classified each word by a linear perceptron whether the word belonged to the answer span or not. Unfortunately the model was quickly degenerating to conclude that there is no answer at all.

Professor Socher suggested we greatly simplify the model to remove complex attention and replace it with simple 1-D weight modulation computed by cross-product value of the final hidden state of question RNN and each given context state. Decoder was also replaced with linear projection layer, followed by relu activation and 2 softmax classifiers. This produced first model that could learn to overfit on small dataset.

Adding full attention including affinity matrices proved to be relatively simple so the encoder was completed soon. However, expanding decoder proved to be the single biggest problem we were eventually unable to overcome. Essentially, any layer of RNNs added in the decoder caused the model to quickly stop learning and settle on high loss value even on miniscule data sets of 100 or 200. Since this is usually an indication of a bug in the model most of the efforts were henceforth spent on debugging the model.

4.2 Debugging

In order to troubleshoot model's underfitting we used the following approaches:

- Verified our variable-length softmax and matrix product calculation were correct. Actually found and fixed a series of bugs.
- Verified that Encoder's RNNs and Attention matrices contain correct data through careful examination of intermediary model outputs, as well as Tensorboard summaries
- Verified gradient magnitude is within appropriate range (1..2) but implemented gradient clipping just in case.
- Varied learning rate, implemented exponential learning rate annealing, even though Adam optimizer is supposed to be self-adapting.
- Implemented concrete LSTM hidden state initialization using Xavier initializer
- Various ways to initialize bilinear projection matrix used in the attention layer
- Added/removed layers in Decoder, changed types (GRU and LSTM), uni- and bidirectional RNNs

Unfortunately, we couldn't get to the bottom of the issue and remove this fundamental problem.

4.3 Model Modularity

Because of various debugging scenarios we ended up implementing a toolkit that allows easy re-configuration of the model. The following configuration parameters are available as command line options for qa_model.py:

Codalab username: ajaffery

- Cross Id Bias – coefficient for attention matrix initialization
- Concatenate Encoding – combine encoding with LSTM decoding
- Attention Components
  - AD – Use context relevant questions words in encoding
  - AQ – Use question relevant context words in encoding
- Answer Start Decoder – Decoding RNN for start index prediction
- Answer End Decoder - Decoding RNN for end index prediction
- Decoder Start Layers – Number of RNN layers for start decoder
- Decoder End Layers – Number of RNN layers for end decoder
- Encoder RNN – RNN encoder

RNN options include GRU, Bi-GRU, LSTM, Bi-LSTM

4.4 Parameter Selection

Available Hyper parameters:

- Embedding Size – GLoVE embedding vector size
- RNN Size – size of each RNN layer
- Max Q – Max context length
- Max C – Max question length
- Optimizer – Adam or SGD optimizer
- Learning Rate – the step size for the optimizer

4.5 Evaluation Methods

We evaluate our model by measuring the decreasing of the loss and using the industry standard F1 and EM score.

4.6 Experiment Trials

For all our experiments we used the Adam optimizer as it performs significantly better than a basic SGD optimizer. D. Kingma et al.[7]. We also utilized exponential learning rate.

All parameters except the bilinear projection matrix were initialized with standard Xavier initializer based on parameters' dimensionality. In order to speed-up initial attention learning the projection matrix was initialized with a weighted sum of identity matrix controlled by a hyperparameter and random uniform noise ranging from -0.01 and 0.01.

4.6.1 Baseline

For the baseline we used simple linear projection with softmax normalization and cross-entropy loss to predict end position similar to (but separate from) the prediction of the start position. Here are some highlights to show exaggerated effects of hyper parameter selection..

Trial	**Max_Q**	**Max_C**	**Learning Rate**	**RNN Size**	**Embedding Size**	**Epochs**	**Training Loss**
1	200	100	.001	100	100	5	11.42
4	350	100	.001	200	300	5	4.26
5	300	80	.01	100	300	5	3.83

Table 1. Baseline-Concatenation Performance

The baseline would often plateau learning after about 4 epochs and the loss would stay above 3%. The two largest performance increases occurred from increasing the embedding size and decreasing the learning rate. We also had a version of the baseline where we did not concatenate the final decoding with the encoding. That version performed very poorly.

Codalab username: ajaffery

4.6.2 Final model

Trial	Max_Q	Max_C	Learning Rate	RNN Size	Embedding Size	Epochs	Training Loss
1	200	100	.001	200	300	7	3.83
2	350	100	.001	200	300	5	4.22
3	300	80	.001	100	300	10	2.52

Table 2. Answer-GRU Table

Running with an RNN size of 200 significantly slowed down training with minimal improvements in training. Reducing the Max_q to 300 and the RNN size to 200 allowed the network to train 10 epochs in the same amount of time as trial 2 took 5 epochs.

4.7 Figures and Tables

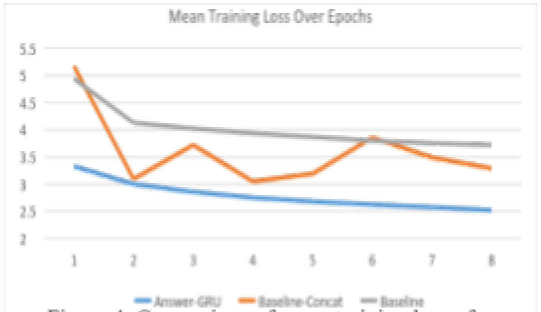


Figure 4. Comparison of mean training loss of Baseline vs Answers-GRU

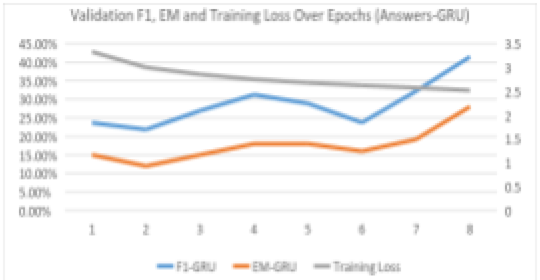


Figure 5. Validation F1/EM vs training loss over epochs

Model	Data	F1	EM	Loss	Epochs
Baseline-Concatenation	Dev	25.615	16.263	3.03	6
Answer-GRU	Dev	26.126	16.831	2.57	7
Answer-GRU	Test	26.684	17.098	2.57	7
DCN [5]	Dev	75.6	65.4	N/A	N/A

Table 3: SQuAD Evaluation Scores

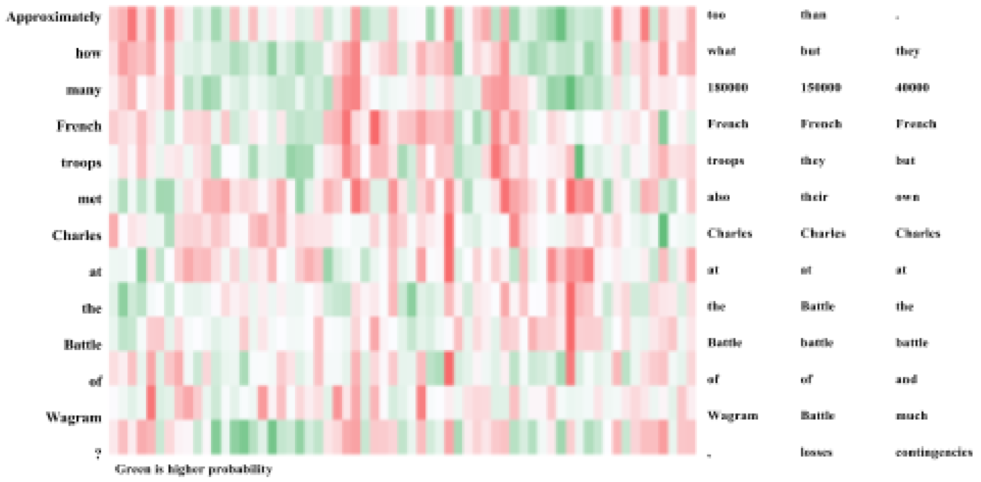
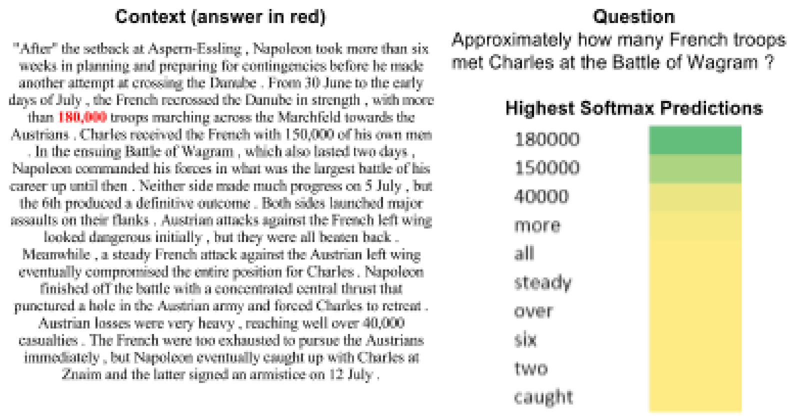


Figure 6. Softmax predictions of context v.s. question. Top 3 context words are shown.

Codalab username: ajaffery



Context (answer in red)

"After" the setback at Aspern-Essling , Napoleon took more than six weeks in planning and preparing for contingencies before he made another attempt at crossing the Danube . From 30 June to the early days of July , the French recrossed the Danube in strength , with more than 180,000 troops marching across the Marchfeld towards the Austrians . Charles received the French with 150,000 of his own men . In the ensuing Battle of Wagram , which also lasted two days , Napoleon commanded his forces in what was the largest battle of his career up until then . Neither side made much progress on 5 July , but the 6th produced a definitive outcome . Both sides launched major assaults on their flanks . Austrian attacks against the French left wing looked dangerous initially , but they were all beaten back . Meanwhile , a steady French attack against the Austrian left wing eventually compromised the entire position for Charles . Napoleon finished off the battle with a concentrated central thrust that punctured a hole in the Austrian army and forced Charles to retreat . Austrian losses were very heavy , reaching well over 40,000 casualties . The French were too exhausted to pursue the Austrians immediately , but Napoleon eventually caught up with Charles at Znaim and the latter signed an armistice on 12 July .

Question

Approximately how many French troops met Charles at the Battle of Wagram ?

Figure 7. Final softmax to determine answer

5 Conclusion

Using Tensorflow and building even basic models proved to be more difficult than expected. With little experience in this framework, debugging was a challenge. Due to these setback we were not able to build heavily upon the basic foundations of our model and utilize the full modularity that had been implemented. Adding RNN layers to the decoder significantly increased training time. Using only an RNN for start and end span classification prevented overfitting to a small dev set. Our model was slower to train than many others, however we were able to see the training loss slowly go down over time which means that we did not reach a convergence point necessarily.

It was a somewhat painful but nevertheless extremely valuable experience. We're excited to realize that after these 2 weeks we are capable of much better understanding of tools and methods of Machine Learning in general and NLP in particular. On the other hand, there is definitely a lot more to be learned and we are both eagerly looking forward to mastering this knowledge.

6 Contributions

- Kostya Sebov – Created working Tensorflow model and experimented with many different implementations. Debugged code and made the model highly modular. Managed code merges. Finalized poster and paper. Spent around 180 hours.
- Ahmed Jaffery – Created saving/reload, tensor-board, and evaluation/validation functionality for the model. Attempted to implement the provided baseline. Filled out qa_answer.py script for submission. Handled Codalab submissions. Created first pass of poster and wrote the paper. Spent around 160 hours.
- We both ran the training on our separate Azure instances as well as my personal desktop. This allowed us to test out more models and better tune hyper parameters and eventually find something that worked for us.

Codalab username: ajaffery

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional Attention Flow for Machine Comprehension. *arXiv eprint arXiv:1611.01603.*

[2] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. *arXiv eprint arXiv:1506.03340.*

[3] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. 2016. Multi-Perspective Context Matching for Machine Comprehension. *arXiv eprint arXiv:1612.04211.*

[4] Shuohang Wang, and Jing Jiang. 2016. Machine Comprehension Using Match-LSTM and Answer Pointer. *arXiv eprint arXiv:1608.07905.*

[5] Caiming Xiong, Victor Zhong, and Richard Socher. 2016. Dynamic Coattention Networks For Question Answering. *arXiv eprint arXiv:1611.01604*

[6] Danqi Chen, Jason Bolton, Christopher D. Manning. 2016. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. *arXiv eprint arXiv:1606.02858*

[7] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *In EMNLP, 2014.*

[8] Christopher D Manning, and Richard Socher. 2017. CS 224N: Assignment #4: Reading Comprehension *Stanford CS224N*

[9] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[10] Google Research. 2017. Tensorflow. *www.tensorflow.org*.