PA4 – Question Answering on Squad

Mudit Jain
muditjai@gmail.com
muditjai@stanford.edu
SCPD student

Abstract

I created 3 NN models to implement a question answering system and measured the performance on Squad. The 1st model uses a simple bidirectional GRUs to encode the question and decode on context state, the 2nd model implements attention as suggested in the handout and the 3rd model implements ideas inspired by Multi Perspective Context matching paper, the assignment handout and few original ideas.

1 Brief Description of Problem

1.0 Overview of the task

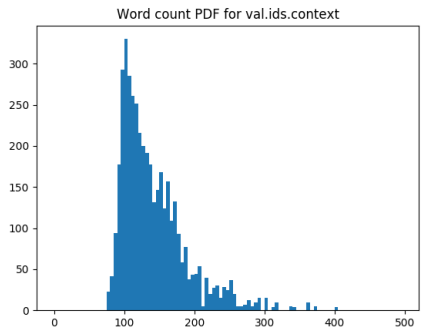
We are trying to create a machine text comprehension system. One of the ways to do so is to implement a question answering system which given a context and question, can answer the question using information from the context. We used the recently published Squad dataset[1] to train and evaluate this model.

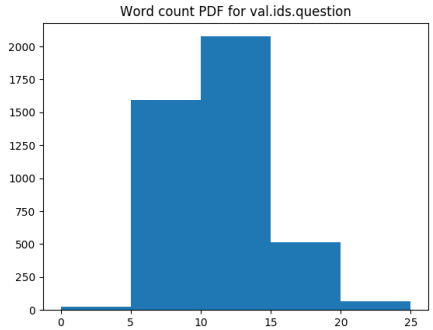
1.1 Problem formalization

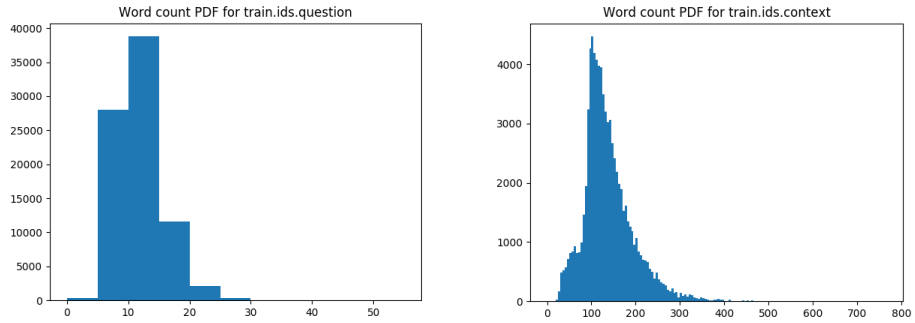
Given a question Q and a context paragraph P, produce answer span indices (a_s, a_e) where $a_s < a_e$, is from within P which correspond to the answer.

1.2 Dataset overview

The Squad dataset has 107,785 question answer pairs, of these 87K are for training and 10K for dev validation and 10K for testing. Our scripts further divide the 87K set into about 82K training and about 5K validation set.
I plotted the sentence length distribution of question and paragraphs for training and validation sets to decide the padding lengths. These are shown below.







2 Description of Models Implemented

2.0 Setup efforts before implementing models

For code setup, my initial focus was on creating a very basic working model using PA4 starter code and then try to build and improve on top of it. Modifying the starter code proved to be a huge challenge and it took almost 1 week of reading the PA3, PA4 code and tensorflow documentation to get started.

For the dataset setup, I created a very small local sample of 100 points(called TrainSample) in order see which methods lead to quick decrease in loss per epoch and take less runtime. The questions and contexts were **padded to lengths 30 and 600 from analysis of the pdf of data** even though the maximum length for question and contexts were 60 and 766 tokens respectively. Later these maximum padding lengths were **reduced to 20 and 300 to speed up the runtime.**

The embeddings used were Glove 6B embedding initially, but updated to Glove 840B for my 3rd model. The 3 models are described in the next section.

For runtime setup, I completed the Azure VM setup with few minor issues. I also learnt functionality of few commands like screen, nohup and tee to help during training on VM.

For theoretical foundations, I read the Match LSTM[2], Multi Perspective Matching[3] and Pointer Net[4] papers to understand common approaches taken. I tried to read the ReasoNet[5] paper, but it was based on reinforcement learning and too difficult to understand.

2.1 Model 1 - Simple Bidirectional encoder-decoder GRU

Since this is a text comprehension problem, using RNNs made most sense for encoding/decoding. As a first cut, I implemented an architecture similar to the encoder-decoder network used for machine translation tasks

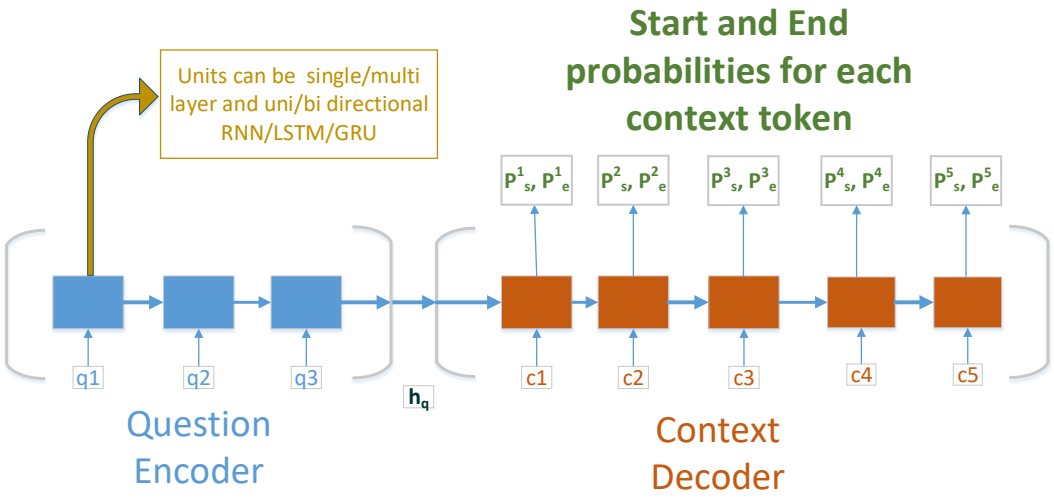
The steps are for this model are

1. First the question and context are convert to dense Glove embedding representation.

2. A RNN encodes the question to generate a final hidden state vector h_q.

3. This h_q is then fed as an initial state for the decoder RNN that decodes over the context and produces 2 output vectors which are considered to be unnormalized probability distributions for start and end indexes.

4. Loss is computed as cross entropy softmax for each of the 2 distributions.



As mentioned before, for 300 output states and with random initialization of weights and therefore predictions, we **expect the loss to be 5.7 + 5.7 = 11.4 initially**. This was used to as a sanity check to make sure the network is initializing correctly.

I started with RNN as cell units to make sure the networks works without errors. Then I tried LSTM and GRU and observed that GRUs are faster and converge quicker than LSTM on the TrainSample(set of 100 training points). This influenced my decision to primarily use GRUs in subsequent networks too. I then tried bidirectional GRUs and found them to be even better.

Problems and learnings

After verification of correctness on local machine, I trained this network on Azure VM after increasing state size to 512. However, there were several problems

1. The runtime was slow - I fixed the runtime issue by reducing the context(output) size to 300 and question size to 20 from initial values 600 and 30. This was done based on analysis of PDF of sentence lengths, since most context sentences were shorter than 300 words and most questions were shorter than 20 words.

2. The network was returning NaN losses after training on about 20K samples – I fixed this by clipping the label indices to 300-1=299 since **sparse softmax function will return NaN** if label index is beyond the probability vector length.

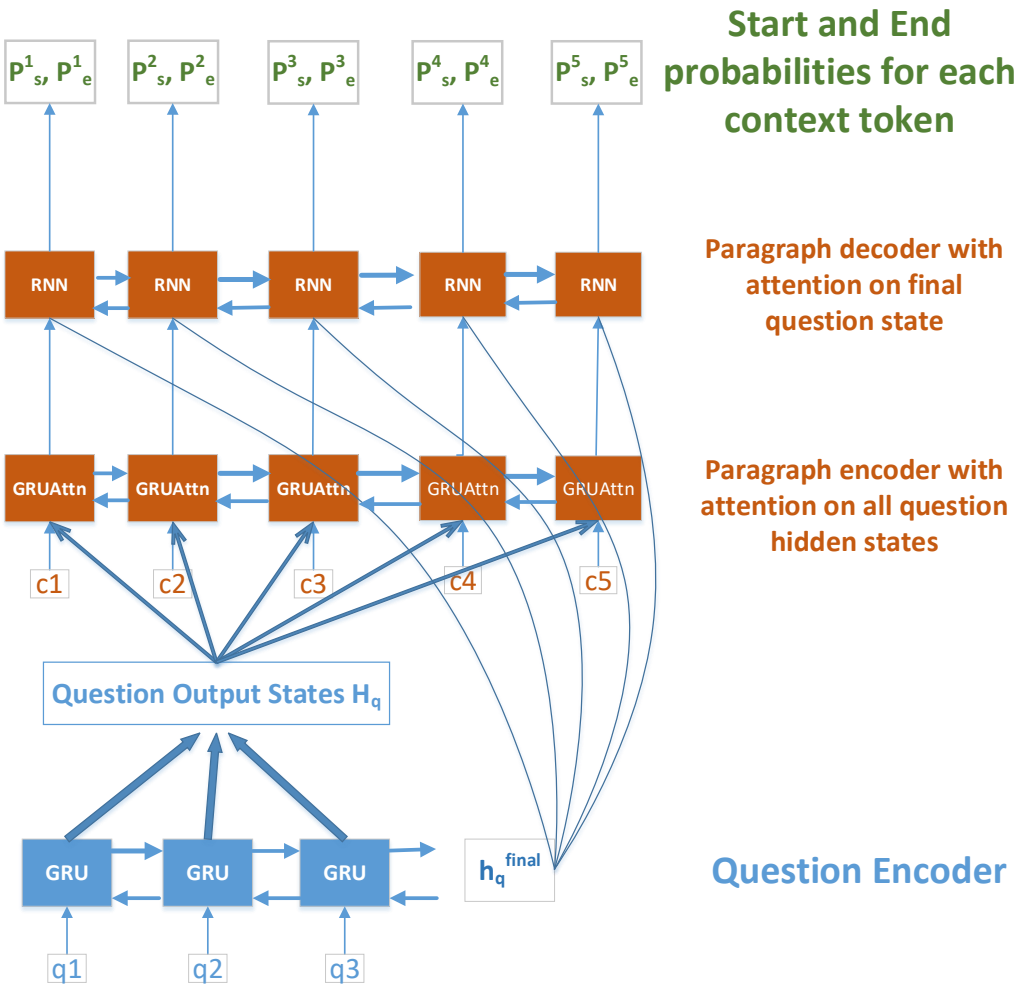
Another issue which was more subtle and took me almost 1 day to understand was effect of learning rate. The network would run fine locally, but give NaNs on full data. I assumed its due to a data issue, but after lot of debugging **the problem was that I had to decrease the learning rate from 0.01 to 0.001.** I later tried different learning rate annealing methods for other models.

Performance

The performance of this model was pretty bad due to its simiplicity. Here are the numbers

	Sanity Check Set		Dev Set		Test Set	
	F1	**EM**	**F1**	**EM**	**F1**	**EM**
Simple Encoder Network	3.34	0.37	5.20	0.64	4.89	0.63

2.2 Model 2 - Attention based model



This model was based on the assignment handout. Here're the steps for this model.

1. The question tokens are converted to dense Glove embedding representation and encoded using a bidirectional GRU for all question tokens. Let the concatenated output/hidden states of question be H_q and concatenated final state be h_q^{final}

2. Then the paragraph is converted to dense representation and for each word of paragraph, it is passed through a bidirectional GRU cell to obtain h_p

3. Then similarity values(ie attention coefficients) are calculated between h_p and H_q by calculating their dot products. The attention coefficients are passed through a softmax to conver them into probabilities. Then a weighted average of H_q with these attention coefficients is calculated to get a new vector c_p ie a new question context vector for the paragraph hidden state h_p . Then I calculate their linear combination ie $W_1h_p + W_2c_p + b$ and return that as GRU output. This gives us a mixed question-paragraph representation h_{qp} with hopefully a focus on important states of the paragraph wrt question. This part of code was very similar to GRUCellWithAttn as provided in the PA4 helper pdf.

4. Then decoder takes in h_{qp} and the concatenated question presentation h_q^{final} and calculates a dot product to get **pointer-net like similarity** between question representation h_q^{final} and every h_{qp}. The paragraph states are multiplied with these similarity values and a 2 output state

bidirectional RNN is run over them. The 2 states of both forward and backward RNN outputs are added in a linear way to get the final start and end probability distributions.

5. These start and end probability distributions are then passed through a softmax cross entropy operation to get the loss.

Problems and learnings

1. This model had a steep learning curve and took a lot of debugging time to implement correctly – I learnt about how to do various **tensor manipulations in 2 and 3 dimensions**.

2. The network used to diverge on losses and sometimes give NaNs – I learnt about the importance of keeping the learning rate low and reducing it over time and **observing the gradient norms and clipping them to avoid gradient explosion.** I also **tried piecewise constant learning rate decay.**

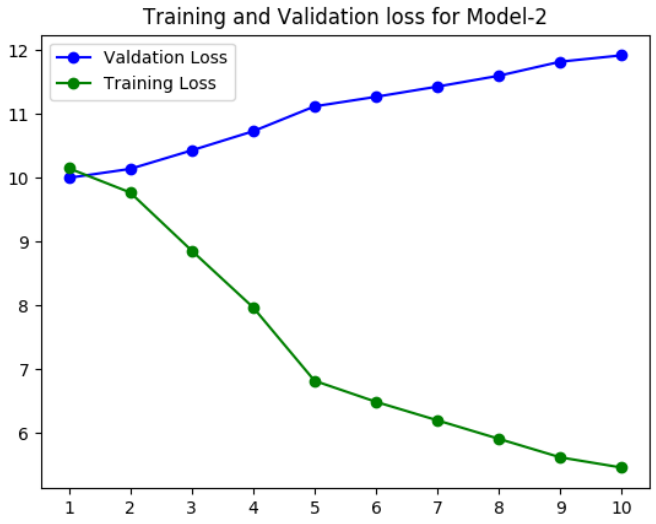
3. Each epoch took 2.5 hours to run with multiple restarts needed due to various issues.

4. The loss on training set decreased promisingly, but the final result on dev set and test set was disappointing as it most likely **did overfitting on training data**.

However, it was a strong learning experience for me and made me much more confident about future implementations.

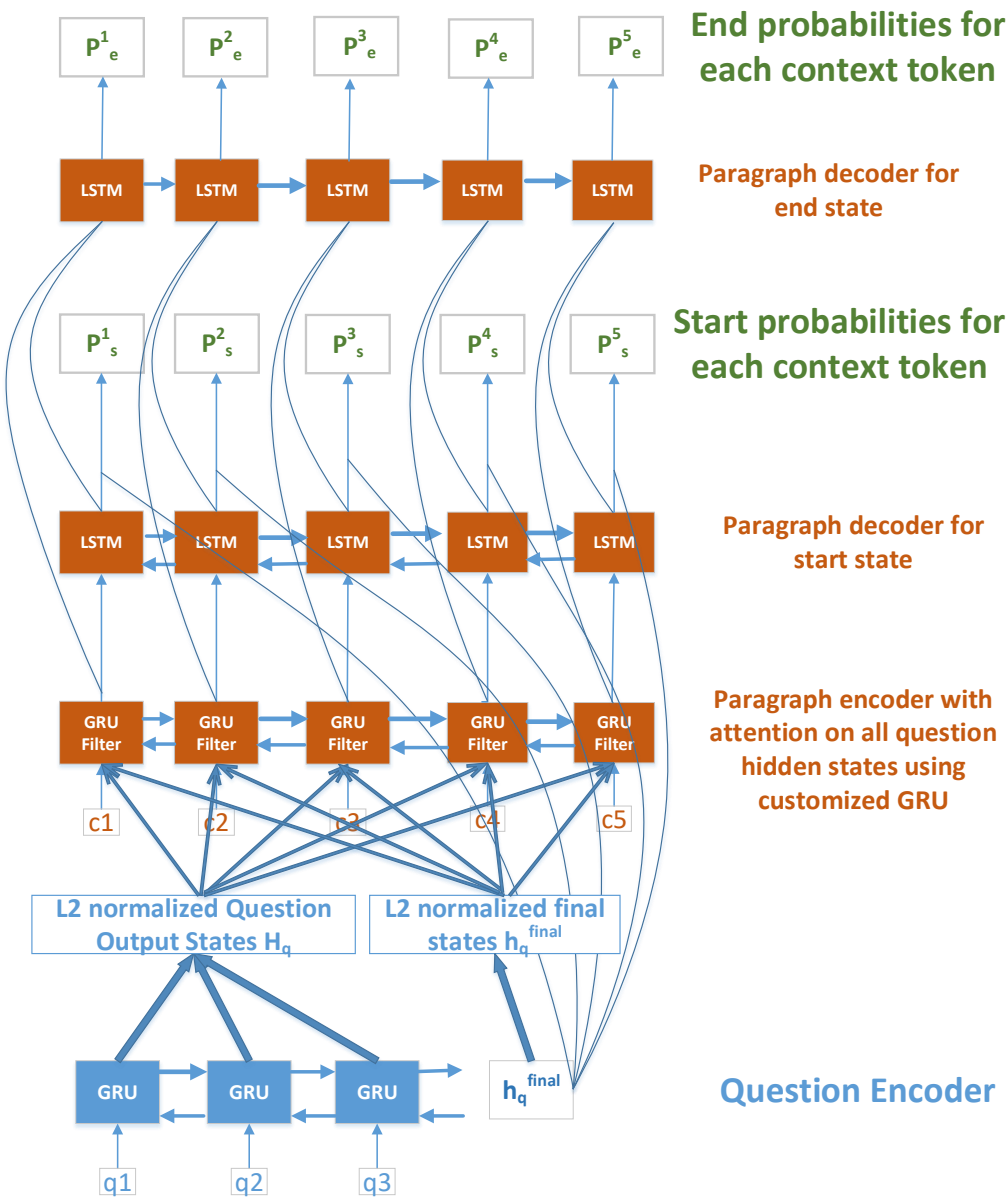
Performance

This model showed increase in validation loss while training loss decreased. And the performance of this model was even worse than model 1 on sanity check set and I didn't evaluate it further on dev set. Since the performance was so bad, I didn't try dropout etc.



	Sanity Check Set	
	F1	**EM**
Attention based model epoch 4	2.33	0.12
Attention based model epoch 8	2.27	0.12

2.3 Custom attention and filtering based model



This model was based on experience of training previous models and reading the Match LSTM and Multi perspective matching papers. I implemented it on Sunday and so kept it simple. But in part due to Azure credit expiry, the training is still ongoing. This model has **15.5 million parameters**.

The model was based on following observations

1. GRUs are very good at encoding the questions and paragraphs.
2. **Adding question state at different stages seems to help** in learning.
3. Inspired from filtering step of Multi perspective matching paper, this network find the maximum and average similarity of paragraph with normalized question output

states and also with normalized question final state.

4. Using a better embedding helps. So I **switched to Glove 840B 300dim** cased embeddings in this model.

5. The **final decoder should carry more state information than 1** and so I switched to LSTM from RNN/GRU since LSTM supports both a high dimensional cell state and a projection to size 1 in output. This was a big weakness of model 2.

The steps in this model are

1. The questions are encoded into dense representation using Glove 840B embeddings and bidirectional GRUs. This step is same as before.

2. The paragraph encoding uses ideas on filtering from Multi perspective matching paper. Instead of model 2 approach of calculating a single context vector by applying attention on all question output states H_q, we do multiple calculations. First we normalize the H_q and question final state h_q^{final}. Next we calculate dot product (cosine) similarity(ie attention coefficients) α between paragraph hidden state h_p and H_q and also between h_p and question final state h_q^{final}.

$$\alpha = \text{cosine}(h_p\ H_q)$$

3. Next we multiply h_p multiply with $\max(\alpha)$ to account for a question word having strong affinity for h_p, we also multiply with $\text{mean}(\alpha)$ to account for multiple question words having high affinity with h_p. In addition, we calculate cosine similarity between h_p and h_q^{final}.

$$h_p^{max\_sim} = h_p * \max(\alpha)$$
$$h_p^{mean\_sim} = h_p * \text{mean}(\alpha)$$
$$h_p^{question\_sim} = h_p * \text{cosine}(h_p, h_q^{final})$$

Next we calculate a linear combination of $W[h_p^{max\_sim}\ \ h_p^{mean\_sim}\ \ h_p^{question\_sim}\ h_q^{final}] + b$ to get our output representation h_{qp} combining both question and paragraph.
I created a new GRU inherited cell called GRUWithFilter for this calculation.

4. For decoding, the above h_{qp} was **concatenated with h_q^{final}** and fed as an input into a bidirectional LSTM. The output of LSTM was sent through linear combination with h_q^{final} to get the start probabilities P_s for each paragraph state and cell state C_s.
h_q^{final} is used repeatedly in different stages since it keeps the focus on question and was also empirically improving performance on small TrainSample set.
The reason for using LSTM instead of RNN/GRU was that it **allow higher state(cell) size with unit output size**(using output projection). Using **state size of 1 was a key weakness of model 2.**

5. The state C_s from previous LSTM was concatenated with h_{qp} and passed through a unidirectional LSTM to get the end probabilities P_e

Problems and learnings
1. The first version again had NaN values – I discovered that it was due to a divide by 0 error in normalization code of step 1. Debugging this made me more confident in using tf.Print() and interpreting actual tensor values on individual examples.

2. High gradient values - Learning rate was kept at 0.0001 initially, but by 3rd epoch I started seeing higher gradient norms and had to reset the LR to 0.00005 and resume training.

2. Insufficient time - This model was implemented on Sunday and started on Sunday night, but Azure credits expired that night and the model could only be trained till 3rd epoch the next day.

Performance

	Sanity Check Set		Dev Set		Test Set	
	F1	**EM**	**F1**	**EM**	**F1**	**EM**
GRU filter attn epoch 1	18.90	11.23	17.65	8.24	17.36	7.89
GRU filter attn epoch 2	25.88	18.39	24.39	13.3	--	--
GRU filter attn epoch 3	28.04	20.61	28.87	17.38	--	--

Analysis of types of error

High level comments – Although the model performance is below the expectation of 60/50 on F1/EM, it still does learn quite a few patterns.

Strengths –

1. It is able to understand the nature of response eg "Who" questions need a person as response, "When" questions need a time, "How much" needs a quantity etc. Also the **answers are mostly coherent** even when it makes a mistake. Eg

Context – In 1900, Tesla was granted patents …… radio transmission in 1901 ….

Question – When did Tesla attain his electrical transitter patent?

Model Answer – 1901

Actual Answer - 1900

2. For simple questions it is able to answer correctly. Eg.

Context - In October 1529 , Philip I , Landgrave of Hesse , convoked an assembly of German and Swiss theologians at the Marburg Colloquy …..

Question - Who was Philip I ?

Model Answer – Landgrave of Hesse

Actual Answer - Landgrave of Hesse

3. It is able to answer few difficult questions also Eg

Context - NASA immediately convened an accident review board , overseen by both houses of Congress . While the determination of responsibility for the accident was complex , the review board concluded that " deficiencies existed in Command Module design….

Question - Who kept tabs on the accident review board that NASA created ?

Model Answer –both houses of Congress

Actual Answer - both houses of Congress

Weaknesses -

Some weaknesses are -

1. Picking an incorrect entity as the answer as seen in example before.

2. Predicting start index after end index, for questions with longer answers. This happens in many instances and can perhaps be fixed by enforcing harder constraints through the loss function.

3. Getting confused by more difficult questions. Eg

Context – The best , worst and average case complexity refer to three different ways of measuring the time complexity (or any other complexity measure) of different inputs of the same size . Since some inputs of size n may be faster to solve than others , we define the following complexities :

Question – Case complexity <unk> provide variable probabilities of what general measure ?

Model Answer – different inputs of the same size

Actual Answer – time complexity

Acknowledgments

I would like to thank the instructors and hard working TAs of CS224n for solving so many doubts on piazza quickly. Piazza has been a great source of learning for this course.

References

[1] Squad dataset Rajpuarkar et al.

[2] Machine comprehension using match-lstm and answer pointer. Wang et al.

[3] Multi perspective matching for machine comprehension. Wang et al.

[4] Pointer Net – Vinayals et al.

[5] Reasonet – Learning to stop reading in machine comprehension. Shen et al.