Harder Learning: Improving on Dynamic Co-Attention Networks for Question-Answering with Bayesian Approximations

Marcus Gomez
Computer Science
Stanford University
mvgomez@stanford.edu

Brandon Cui
Computer Science
Stanford University
bcui19@stanford.edu

Udai Baisiwala
Economics
Stanford University
udai@stanford.edu

March 21, 2017

Abstract

In this paper, we propose a novel method for building a reading comprehension system. Neural-based architectures for question answering have been improving dramatically over the past 2-4 years. In this paper, we experiment with several possible methods to improve further (Bayesian methods for learning probability distributions, dropout, Monte-Carlo sampling) and present results on the Stanford Question Answering Dataset (SQuAD).

Group name on codalab: sfua

Introduction

Being able to extract meaning from words is the fundamental goal of natural language processing. This can mean many different things - understanding commands, translating speech to text, answering questions. But most fundamentally, it means taking information in some sort of format and being able to demonstrate that a system understands it.

In this paper, we attempt to address one aspect of this problem. Specifically, we attempt to build a framework that, given a body of text and a question, can answer the question based on the provided text. In order to develop this tool, we take advantage of the SQuAD database. This is a relatively challenging version of this problem, since it involved attempting to answer questions with answers of variable length and by generating the answers rather than choosing from options. Also, the answers are crowdsourced and therefore less consistent.

Neural-based architectures for question answering have been improving dramatically over the past 2-4 years. Recent research suggests Bayesian methods of learning probability distributions instead of deterministic functions can improve model robustness and performance. Work by Gal. et. al (2016) suggests that dropout in deep networks can be used precisely for this purpose, and to date, the methodology suggested by said team has not been leveraged on the QA-task. Here, we use state of the art neural architectures and use a dropout + Monte-Carlo (MC) sampling technique to better approximate the underlying distribution.

Methodology

In this section, we discuss the major neural network tools that we use to solve this problem and discuss how we used them.

Baseline Architecture

We represent the question as a sequence of vectors $(x_1^Q, x_2^Q, ...)$ and similarly represent the corresponding passage as $(x_1^P, x_2^P, ...)$. For our baseline we utilized the framework as described in Wang

S., Jiang J. (2016) which consists of an LSTM Preprocessing Layer, a Match-LSTM Layer, and an Answer Pointer Layer. For the LSTM preprocessing layer, we fed in the vectorized paragraph and vectorized question:

$$\mathbf{H}^p = \overrightarrow{LSTM}(P), \qquad \mathbf{H}^q = \overrightarrow{LSTM}(Q)$$

thus, at a timestep t we feed in x_i^Q for the question LSTM and x_i^P for the paragraph LSTM. At the match LSTM layer we use the pre-processed matrices $\mathbf{H}^p$ and $\mathbf{H}^q$. We use the traditional word-by-word attention mechanism to get the attention vector $\overrightarrow{\alpha}_i \in \mathbb{R}^Q$. This is done as follows:

$$\overrightarrow{\mathbf{G}}_i = tanh(\mathbf{W}^q\mathbf{H}^q + (\mathbf{W}^r\overrightarrow{\mathbf{h}}^r_{i-1} + b^p) \otimes \mathbf{e}_Q)$$

$$\overrightarrow{\alpha}_i = softmax(\mathbf{w}^T\overrightarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q)$$

where we treat $\otimes e_Q$ as producing a matrix or row vector by repeating the vector or scalar Q times. From here we use the attention vector α_i in order to determine a weighted version of the question $\overrightarrow{z}_i$:

$$\overrightarrow{z}_i = \begin{bmatrix} \mathbf{h}_i^p \\ \mathbf{H}^q\overrightarrow{\alpha}_i^T \end{bmatrix}$$

we then use this vector and feed it into the forwards LSTM to get our match-LSTM:

$$\overrightarrow{\mathbf{h}}_i^r = \overrightarrow{LSTM}(\overrightarrow{z}_i, \overrightarrow{h}^r_{i-1})$$

We then proceed to do the same in the reverse direction:

$$\overleftarrow{\mathbf{G}}_i = tanh(\mathbf{W}^q\mathbf{H}^q + (\mathbf{W}^r\overleftarrow{\mathbf{h}}^r_{i-1} + b^p) \otimes \mathbf{e}_Q)$$

$$\overleftarrow{\alpha}_i = softmax(\mathbf{w}^T\overleftarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q)$$

Now, let $\overrightarrow{\mathbf{H}}^r$ represent the following concatenation of vectors $[\overrightarrow{\mathbf{h}}^r_1, \overrightarrow{\mathbf{h}}^r_2, \cdots, \overrightarrow{\mathbf{h}}^r_P]$ and let $\overleftarrow{\mathbf{H}}^r$ represent the concatenation of the vectors $[\overleftarrow{\mathbf{h}}^r_1, \overleftarrow{\mathbf{h}}^r_2, \cdots, \overleftarrow{\mathbf{h}}^r_P]$. Now, let us define the matrix $\mathbf{H}^\tau$ to be the concatenation of these two matrices:

$$\mathbf{H}^\tau = \begin{bmatrix} \overrightarrow{\mathbf{H}}^r \\ \overleftarrow{\mathbf{H}}^r \end{bmatrix}$$

Lastly, our model utilizes the boundary model as described again in Wang S., Jiang J. (2016). For this we are only trying to predict two indices, a_s and a_e, or the start and end indices. In order to generate such a probability, we begin by trying to generate the kth token. We use an attention mechanism which is represented as the following:

$$\mathbf{F}_k = tanh(\mathbf{V}\mathbf{H}^\tau + (\mathbf{W}^a h^a_{k-1} + \mathbf{b}^a) \otimes \mathbf{e}_{(P+1)})$$

$$\beta_k = softmax(\mathbf{v}^T\mathbf{F}_k + c \otimes \mathbf{e}_{P+1}$$

Now, for the answer the hidden state given the hidden state at the $k-1th$ position, we run it through the following LSTM:

$$\mathbf{h}^a_k = \overrightarrow{LSTM}(\mathbf{H}^\tau\beta_k^T, \mathbf{h}^a_{k-1})$$

Now, the probability of the kth token is:

$$p(a_k = j|a_1, a_2, \cdots a_{k-1}, \mathbf{H}^{tau}) = \beta_{k,j}$$

and with the boundary model we are trying to maximize the probability of:

$$p(\mathbf{a}|\mathbf{H}^\tau) = p(a_s|\mathbf{H}^\tau)p(a_e|a_s, \mathbf{H}^\tau)$$

while trying to minimize the loss function over the N training examples:

$$-\sum_{n=1}^{N} logp(\mathbf{a}_n|\mathbf{P}_n, \mathbf{Q}_n)$$

Dynamic Co-Attention Architecture

As an improvement over the baseline model, we pull from the work of Xiong et. al (2017). For the encoder layer here, we again use an LSTM; this time however, we use the same LSTM such that we map both the question and the passage into the same representation space. In particular, we have

$$q_t = \overrightarrow{LSTM}_{shared}(x_t^Q, q_{t-1}), d_t = \overrightarrow{LSTM}_{shared}(x_t^D, d_{t-1})$$

We then define $D = [d_1, d_2, ...d_m]$ and $Q = tanh(W^Q[q_1, q_2, ..] + b^Q)$ as our final encoded representations. Note we apply the non-linear transformation to the raw encoder output to allow variation between the encoding spaces, in accordance with Xiong et. al (2017). Given the time and computation constraints of the project, we diverge from their implementation here and opt to not include sentinel vectors – forcing the model to always attend to some component.

For the attention mechanism, we draw also draw heavily on the work of Xiong et. al (2017), as well as the earlier work of Lu et. al (2017). We define an affinity matrix $L = D^T Q$, and then normalize row-wise to get passage-wide attention per word in the question (called, A^Q), and normalize column-wise to get question-wide attention per word in the passage (called, A^D). We then apply the attentions to the passage and document to get "summaries" (i.e. to learn what components to specifically attend to), and further apply the per-word documentation attention to the question summary and concatenate it to get a fully co-dependent passage-question representation. In matrix algebra, we have

$$C^Q = DA^Q M = QA^D \rightarrow C^D = [M\ C^Q]$$

Finally, to encode the time information, we as with the baseline, we define forward and backward LSTMs;

$$\overrightarrow{h_t^r} = \overrightarrow{LSTM}([d_t; c_t^D], \overrightarrow{h_{t-1}^r})$$
$$\overleftarrow{h_t^r} = \overleftarrow{LSTM}([d_t; c_t^D], \overleftarrow{h_{t-1}^r})$$

With $\overrightarrow{H^r} = [\overrightarrow{h_1^r}\overrightarrow{h_2^r}....]$ and $\overleftarrow{H^r} = [\overleftarrow{h_1^r}\overleftarrow{h_2^r}....]$, we once again define $H^\gamma = [\overrightarrow{H^r}^T; \overleftarrow{H^r}^T]^T$.

For the decoder layer, we use the same decoder described in the baseline.

Learning Probability Distributions

The main novel approach we take in this study is to change the objective that we are learning during training. With standard neural architectures, the goal is to learn some deterministic function f that adequately describes the parameter space and maps inputs x to outputs y. Here, instead of learning a deterministic function, we attempt to learn some probability distribution q, in alignment with a recent study by Gal et. al (2016). Specifically, if the parameters of our architecture are θ, then we wish to learn some function q such that $q(y^*|x^*) = \int p(y^*|x^*, \theta)q(\theta)d\theta$ for a learned probability density function p. Here then, we wish to estimate both the mean $E_q[y^*]$ and the variance $Var_q[y^*]$. The work of Gal et. al (2016), gives a full proof demonstrating that dropout models are excellent approximations of these distributions; for sake of clarity and conciseness here, it suffices to note that the above quantities of interest can be computed empirically by taking a sufficiently large number of stochastic forward passes through the network and computing the sample mean and sample variance (in a Monte-Carlo style approach). Importantly, this means the dropout rate is non-zero at test time.

Voting Methods

Given the prediction task at hand, the concept of "averaging" to compute the Monte-Carlo approximation of the mean and variance of the distribution is ill-defined. Here, we propose a few basic "averaging" mechanisms, which we test and compare efficacy of in the results. Given a set of N samples $T = \{(a_{s_i}, a_{e_i})\}_{i=1}^N$

MostCommon

Here, we simply take the $(a_s, a_e) = mode(T)$

L-R-Mode

Instead of considering the pairs together, we separate the start and end token predictions into sets T_s and T_e. Then, we take $a_s = mode(T_s)$ and $a_e = mode(T_e)$

MinBound

Here, we take the minimum span of the answer predicted in the samples; we take $a_s = max(T_s)$ and $a_e = min(T_e)$

MaxBound

Here, we take the maximum span of the answer predicted in the samples; we take $a_s = min(T_s)$ and $a_e = min(T_e)$

Dataset

Using the Stanford Question Answering dataset (SQuAD) recently released by Rajpurkar et al. (2016) is a diverse hand annotated dataset that is significantly larger than previous hand annotated datasets. For this task we trained on a predetermined training dataset that consisted of over 80,000 training examples and we used a provided development set of 5% of the training set or approximately 5,000 examples for tuning hyperparameters. In experiments, for paragraphs with length shorter than the input length, we would zero pad the paragraphs, while for paragraphs longer than the input length we would truncate the paragraphs to the given length. Similarly, for questions longer than the input length, we would zero pad, and for questions longer than the input length we would truncate.

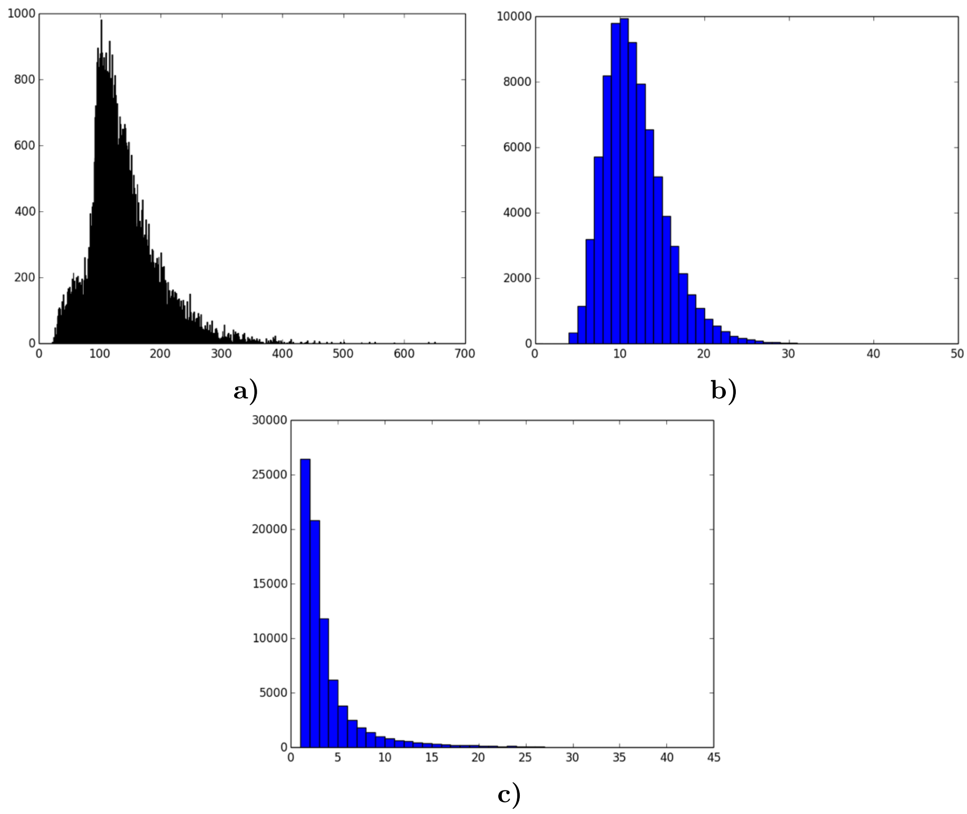


Figure 1: Dataset stats for **a)** paragraph **b)** question and **c)** answer lengths

Based on the plots in figure 1, the mode of the paragraph lengths is around 200, the question lengths is 15, and the question answers is 1. Additionally, there are few paragraphs with a length greater than 300 words and few answers that fall beyond 15 words. Thus, we choose to truncate paragraph lengths at 200, we will only truncate a small portion of the dataset and by truncating paragraph lengths of 300 we will truncate almost no parts of the dataset.

Lastly, when training and testing the model, we utilized pre-trained 100 or 300 dimension GloVe vectors.

Results and Analysis

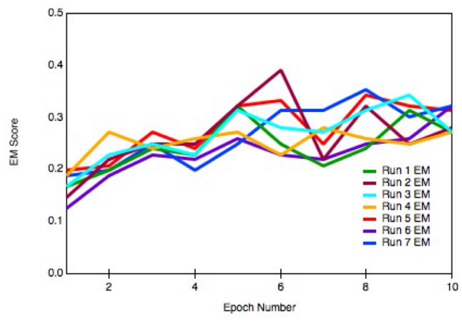
Baseline Model

The parameters used in this set of experiments are detailed below (Table 1):

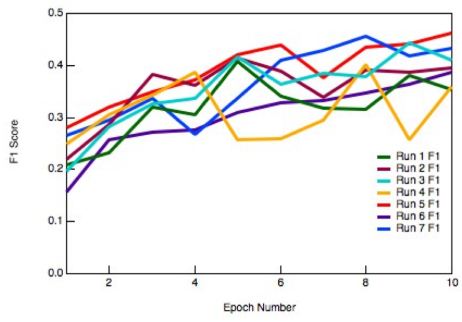
Experiment Number	h	P	A	d	F1	EM
1	150	200	16	100	0.365	0.257
2	150	200	16	300	0.353	0.291
3	150	200	31	100	0.3	0.177
4	150	200	31	300	0.289	0.193
5	300	200	16	100	0.367	0.254
6	300	200	31	100	0.391	0.281
7	150	250	31	100	0.384	0.269

Table 1: hyperparameter tuning used for baseline

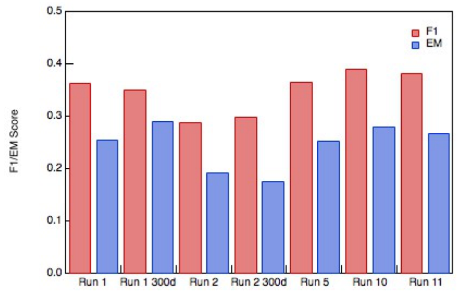
here h represents the hidden size, P is the paragraph length, A is the answer length, and d is the size of the word vector embeddings used. The F1 and EM scores were obtained after fully training our model and then running the model on the development set.



(a)



(b)



(c)

Figure 2: The **a)** F1 and **b)** EM scores over epochs plus **c)** the final F1 and EM scores on the dev set for the baseline model

For our baseline model in order to show that our model was learning, at the end of every training epoch, we ran our model across a small fixed subsample of the development set to get an F1 and an EM score. Generally, as the number of epochs increased, the F1 and EM scores would increase, indicating that the model was learning. Additionally, we noticed that as we increase the hidden layer, we get a small boost in performance for the final F1 and EM scores. Additionally, we noticed that by constraining our answer length we were able to increase the F1 and EM scores. Overall, we were able to achieve an optimal F1 of 0.391 and an optimal EM of 0.281 for the baseline model.

Dynamic Model

Experiment Number	h	P	A	d	F1	EM
1	150	200	16	100	0.542	0.402
2	300	300	16	100	0.558	0.406
3	150	200	16	300	0.525	0.379
4	300	200	31	300	0.532	0.384
5	300	200	16	300	0.524	0.363
6	150	200	31	100	0.521	0.378
7	150	200	16	100	0.529	0.387

Table 2: hyperparameter and corresponding F1 and EM scores

here h represents the hidden size, P is the paragraph length, A is the answer length, and d is the size of the word vector embeddings used. Additionally, the F1 and EM scores are obtained from running the entire development set on the fully trained model.

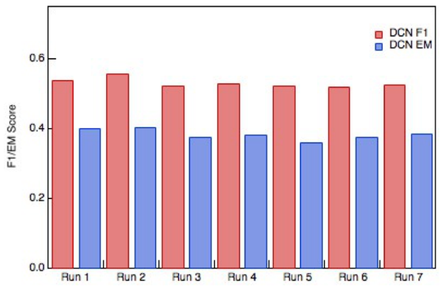


Figure 3: The Final F1 and EM scores on the dev set for the dynamic model

Comparing the baseline to the dynamic model, we notice that by using dynamic encoding we are able to achieve significantly higher F1 and EM scores, generally 0.15 higher for both. Based upon the hyperparameter tuning, the optimal result of an F1 of 0.558 and an EM of 0.406 was achieved on the 2nd experiment with an embedding size of 300, a paragraph length of 300, an answer length of 16, and glove vectors of 100.

Dynamic Bayesian Model

Due to the short nature of this research endeavor, we were forced to do iterative hyperparameter selection instead of choosing all parameters at once; inherently, these results might be biased, and we make note of this more concretely in the discussion.

Choosing the best hyperparameters from the DCN model (hidden size = 150, glove vector size = 300, paragraph length = 200, and answer length = 16), we began the crux of our experiments. We added dropout to the end of the encoder and attention layers of our model. We then trained these dropout models with different keep probabilities; for a given model, we could then choose a voting strategy and the number of Monte-Carlo (MC) samples to draw. We ran 4 different keep probabilities, 4 different voting strategies, and 4 different MC-sample rates, yielding a total of 64 experiments. The full results are shown in Appendix A, but here we just present the main results.

First, we run a comparison of our voting strategies (Figure 4); across all keep probabilities, for more than 2 samples drawn, the MinBound and MaxBound voting strategies perform very poorly and actually perform worse as the number of samples drawn increases. This makes sense since as the number of samples increases, MaxBound will continue to select larger and larger selections of the passage and MinBound will at best select one word as the entire answer, which in many cases is insufficient (as we saw in the earlier frequency distribution plots). In general, the MostCommon strategy performed the most effectively, with L-R-Mode having similar efficacy; with both of these two models, as expected, increased number of samples improves performance non-trivially.

Next, for this voting method, we considered a variety of dropout probabilities and sampling rates; the results of this experiment for the MostCommon voting strategy are summarized in Table 3 and Figure 5; in general, lower dropout probability and larger number of samples lead to higher efficacy, which makes sense, since lower dropout probability means that the samples will not

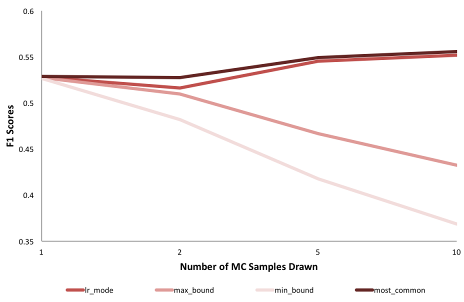


Figure 4: Relative performance of voting methods (dropout probability 0.2)

diverge far from the population mean of the underlying distribution, and as the number of samples increases, the empirical mean approaches the population mean by the Law of Large Numbers.

Dropout Probability	**1 MC Sample**	**2 MC Samples**	**5 MC Samples**	**10 MC Samples**
0.2	0.529	0.528	0.549	0.555
0.3	0.508	0.507	0.536	0.546
0.4	0.481	0.481	0.516	0.530
0.5	0.441	0.445	0.484	0.501

Table 3: F1 scores for varied drop probabilities and MC-sample rates

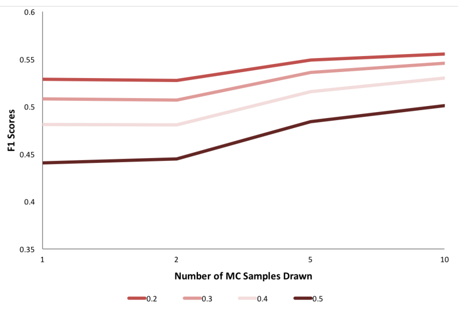


Figure 5: Relative Sampling Performance of Dropout Rate over Iterations (using most common sampling methodology)

One of the most important reason for increasing the number of samples to reduce model uncertainty; in particular, for low number of samples, these methodologies may not be as effective and the models may have relatively high variance in their belief. In Figure 6, for each of the MC-sample rates, we perform the experiment on the dev set 10 times and compute both the mean and variance in F1 score. Of note here is that, as the number of MC samples increases, the variance (shown in the grey region surrounding the trendline) in the F1 score falls/compresses – meaning that, as we sample more, our estimate is in fact getting more confident / stronger, which is the behavior that we both want and expect from a valid probability distribution.

We extended these experiments by trying different sampling methods, sampling rates, and dropout rates. The full results of those experiments are presented in Appendix A.

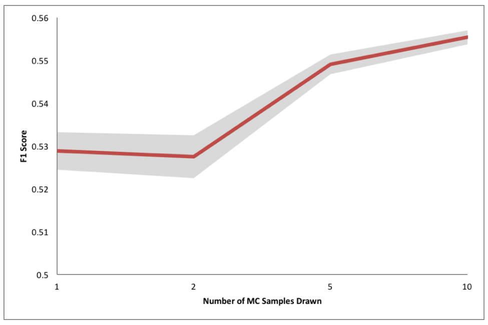


Figure 6: Variance (shaded region) in F1 Accuracy with Increased MC-samples

Final Model

Parameters

For our submitted model to CodaLabs, we had trained used a hidden-size of 150, paragraph length of 200, answer length of 16, glove vector size of 300, dropout probability of 0.2, and a MC-sample size of 50. This model on the dev set achieved an F1 of 62.106 and an EM of 48.524, while on the test set achieved an F1 of 61.234 and an EM of 48.054.

Error Analysis

While our model performed fairly well on the dev set, we still want to look at some of the ways in which it did badly. We note, for instance, that performance was strongly correlated with length of predicted answer - the shorter our predicted answer was, the more likely it was correct. This suggests that our model is better at answering simple questions, since they would be more likely to have one or two word answers. The model performs quite badly when it predicts long answers. The fact that we use an LSTM and an attention model would help to improve our longer answers, but apparently not enough to overcome the fact that short answers are intrinsically easier to get right.

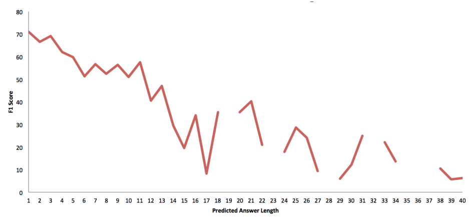


Figure 7: F1 Performance of our model vs Length of Predicted Answer (gaps represent non-existent lengths

Future Direction

At the moment, we have trained an ensemble of four models and on the dev set partitioned from train, we achieved an F1 of 64.76 and an EM of 50.84. Unfortunately at the moment, we have been unsuccessful in uploading our code to CodaLabs for submission, and as a result we would like to further test our ensemble model.

Beyond this, we would be interested in continuing to address specific weaknesses in our model. For instance, we note that our model performs badly on longer words. One thought we had here is to attempt to initially classify questions into one with long or short answers and then train those models separately. The motivation with that is that questions with short answers are likely to be identification questions while those with long answers are likely to be more involved. It would be interesting to see if training separate models for those question types could lead to a better overall model.

Bibliography

Gal Yarin, Ghahramani Zoubin. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *arXiv preprint arXiv:1506.02142,* 2016.

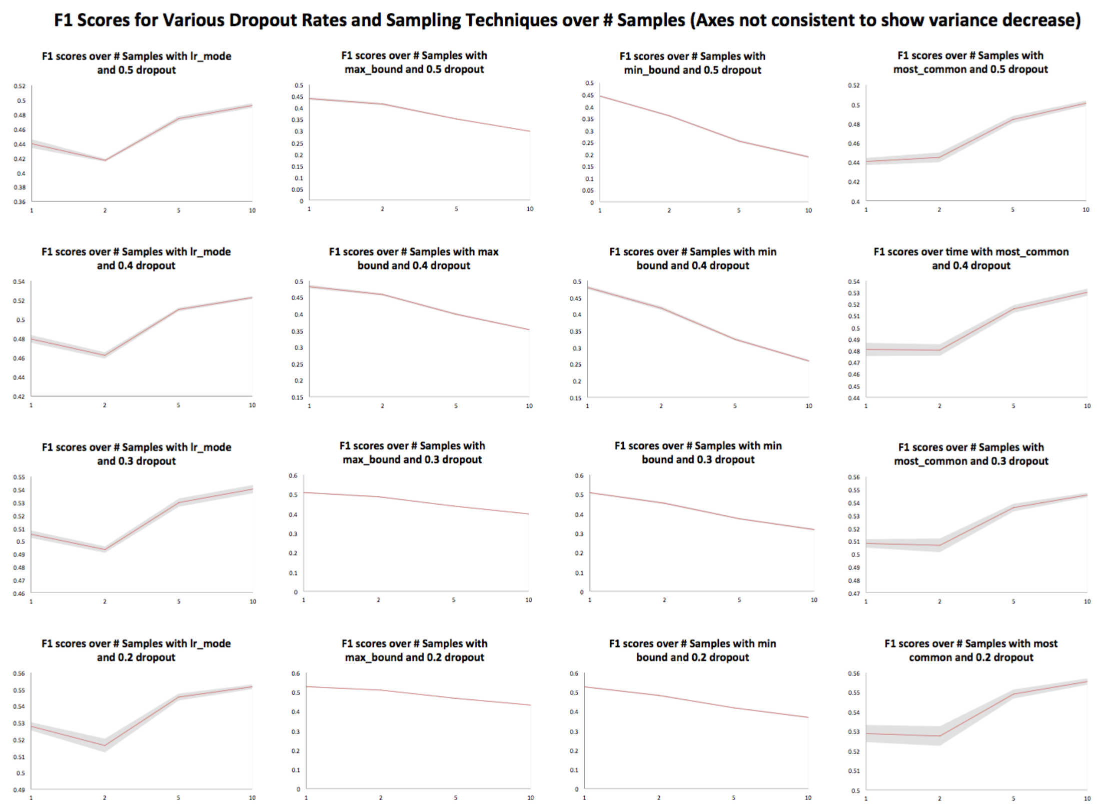
Lu Jiasen, Yang Jianwei, Batra Dhruv, Parikh Devi. Hierarchical Question-Image Co-Attention for Visual Question Answering. *arXiv preprint arXiv:1606.00061,* 2017.

Shuohang Wang, Jing Jiang. Machine Comprehension Using Match-LSTM and Answer Pointer. *arXiv:1608.07905*, 2017.

Xiong Caiming, Zhong Victor, Socher Richard. Dynamic Coattention Networks for Question Answering. In *ICLR*, 2017.

Appendix A

In the below figure, we provide the detailed results of our experiments with number of samples, sampling methods and dropout rates.


F1 Scores for Various Dropout Rates and Sampling Techniques over # Samples (Axes not consistent to show variance decrease)
F1 scores over # Samples with lr_mode and 0.5 dropout
F1 scores over # Samples with max_bound and 0.5 dropout
F1 scores over # Samples with min_bound and 0.5 dropout
F1 scores over # Samples with most_common and 0.5 dropout
F1 scores over # Samples with lr_mode and 0.4 dropout
F1 scores over # Samples with max bound and 0.4 dropout
F1 scores over # Samples with min bound and 0.4 dropout
F1 scores over time with most_common and 0.4 dropout
F1 scores over # Samples with lr_mode and 0.3 dropout
F1 scores over # Samples with max_bound and 0.3 dropout
F1 scores over # Samples with min bound and 0.3 dropout
F1 scores over # Samples with most_common and 0.3 dropout
F1 scores over # Samples with lr_mode and 0.2 dropout
F1 scores over # Samples with max_bound and 0.2 dropout
F1 scores over # Samples with min bound and 0.2 dropout
F1 scores over # Samples with most common and 0.2 dropout