CS224N Final Report

Dirk Hofland
M.S. Symbolic Systems
Stanford University
dhofland@stanford.edu

Allen Zhao
M.S. Symbolic Systems
Stanford University
arzhao@stanford.edu

Abstract

As part of the Stanford course CS 224N (Natural Language Processing with Deep Learning), we compete with our peers in the Stanford Question Answering Dataset (SQuAD) challenge, which aims to build a neural network capable of answering questions about Wikipedia articles. We cover the techniques that we utilized and lessons gained while building out a full neural network architecture for this task.

1 Introduction

1.1 Motivation

In the past decade, the advent of neural networks has led to breathtaking breakthroughs in machine translation, vision, and many other previously inaccessible fields. With dozens of highly motivating papers coming out every year on deep learning, the field of machine translation has moved far enough such that it required a new challenge to serve as a benchmark for future efforts.

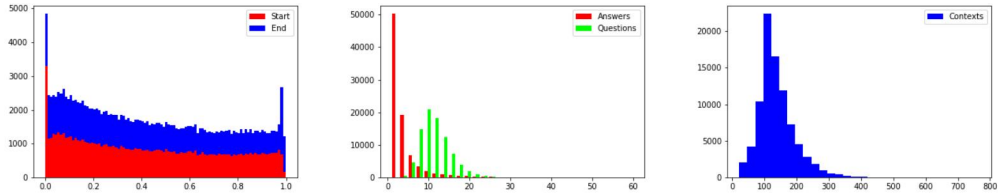
The Stanford Question Answering Dataset became that challenge, providing a labeled set of data to train on and a leaderboard to encourage competition. The dataset encompasses over 100,000 pairs of questions and answers that can be found within more than 500 Wikipedia articles, making this one of the most comprehensive corpuses available for supervised natural language processing research. For further reading, please refer to the SQuAD website here. We chose to tackle this challenge, presented as part of the CS 224N default final project, as the preformed dataset offered a chance for us to direct all of our attention to model-building.

1.2 Approach

The initial work started with a preliminary data analysis in order to better understand the SQuAD dataset. Once completed, the team subsequently focused efforts on improving the baseline model's attention implementation. We did so by reproducing several of the most successful recent attention-based models, where we paid particularly emphasis on memory- and time-efficient architectures, given the significant computational restrictions caused by the size of the SQuAD dataset and the limited resources available. We went on to incrementally test various minor architectural adjustments and features in order to finalize the model. Once this was completed, we optimized our model's performance by improving its prediction mechanism and conducting an extensive hyper-parameter search to determine final run parameters.

1.3 Preliminary Data Analysis

Basic data analysis was conducted by plotting the statistics that were most immediately relevant: the answer start- and end-positions, along with the answer, question, and context lengths. The results are collected in Figure 1.



(a) Answer Start/End Positions (b) Answer/Question Lengths (c) Context Lengths

Figure 1: Counts of Various SQuAD Train Input Lengths/Positions

We immediately noticed that the context length distribution has a long but thin tail. Accommodating this entire tail would be very expensive in terms of both computation-time and memory. The default context length for the baseline model was 600, but we quickly determined that this value was excessively high. Out of the 86326 samples in the training set, only 19 samples (0.02%) had context lengths greater than 600, while only 34 contexts (0.04%) were longer than 500, with this figure rising to 182 (0.21%) for contexts larger than 400. To save memory and time, all architectures were therefore developed and tested with a context length of 400. The default question length of 30 was kept, however, since at this value only 58 answers (0.07%) would not be fully captured and this is not a large memory sink. Finally, while we were intrigued by the clearly visible concentration of answer start- and end-positions at context edges, due to time-constraints we were unfortunately forced to forego further analysis in favour of developing our model.

2 Related Work

As we implemented every major form of attention mentioned in the CS 224N project handout, all of the following papers were supremely relevant to our project and the architectures we built. Full citations are available in the References section at the very end of this paper.

1. 'SQuAD: 100,000+ Questions for Machine Comprehension of Text' by Rajpurkar et al. [Perhaps the most relevant work as it provides the foundational data and motivation for our work here]
2. Bidirectional attention flow for machine comprehension by Seo et al. [Used for BiDAF implementation and character-level CNN-based embeddings]
3. Dynamic coattention networks for question answering by Xiong et al. [Used for the coattention implementation]
4. R-Net: Machine Reading Comprehension with Self-Matching Networks by Microsoft Research Asia [Used for the R-Net and self-attention impementations]
5. Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. [Used for the (failed) answer pointer]
6. Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. [Used for the (successful) smarter / conditional span selection]

3 Approach

As previously mentioned, we spent most of our time implementing some of the most successful recent attention models. We considered 5 models: Bidirectional Attention, Coattention, R-Net (full), and Self Attention. Of these, bidirectional attention performed similarly to baseline, coattention worked somewhat well but trained too slowly, R-Net trained incredibly sluggishly, and self attention worked very well. We spent much of our time on attempting to reproduce the complete R-Net architecture but unfortunately the latter was significantly inhibited by the many ambiguities in the original paper. We ultimately selected the Self Attention model due to its speed and high performance, but

it was particularly instructive to see how each network functioned. We discuss each of the unused implementations below, where we analyze some of the strengths and weaknesses of the competing architectures. Afterwards, we describe our chosen attention architecture and discuss some of the considerations that informed our decision.

3.1 Bidirectional attention

This was our first model - a relatively simple implementation inspired by the project handout and original paper, although we replaced the bidirectional LSTM in the paper with GRU's for faster training. We also decided to apply dropout to the final values, contrary to the baseline model. Interestingly enough, our first model underperformed the baseline model significantly, which we believed indicated a a regularization issue. Once we reduced the dropout rate from 0.15 to 0.05, our resulting scores were better, but still very underwhelming - in fact, we achieved no significant improvement over baseline. While this could have in part been a function of our switch to GRU's, we decided that the results were not impressive enough to warrant further exploration.

Strengths: fast and computationally efficient.

Weakness: low performance (or faulty implementation/didn't fit with our hyper parameters).

3.2 Coattention

After moving on to coattention, we found that this model (and our issues with its implementation) were much more complex. We've listed the numerous major issues that we ran into with this implementation below, in the interest of informing future readers.

1. For one, we struggled with concatenating vectors to batches of unknown size in computationally efficient ways, given that we must add an extra bidirectional-LSTM to the model. Given the huge number of parameters that this entails, we constantly ran out of memory. We wanted to preserve batch size as we found training went more smoothly with a high batch size but were forced to cut other parameters - we cut down to 50 word embedding size (down from a default of 100) and hidden size of 150 (down from 200), in addition to the restriction of context lengths to 400.
2. We found that if we did not mask the sentinel vectors, then the gradient norm would regularly hit infinity. We fixed this by masking the sentinel vectors, but this definitely caught us by surprise.
3. After masking the sentinel vectors, we realized that we were masking the same dimension when computing both the context-to-question and question-to-context attention. This was a simpler fix than above but was difficult to find.

Fixing these issues brought us to a functioning implementation, but the model took forever to train since there were so many parameters. Even after cutting down the LSTM version (with 9,109,602 parameters) to a GRU-based model (with no discernible drop in accuracy), we still had to deal with 6,948,402 parameters. In the end, we gave up coattention because of the bloat involved and the slow pace of training involved. One improvement that we considered implementing was integrating the bidirectional layer outputs into our larger model instead of just having them compute START and END, but we moved on before this was done.

Strengths: comprehensive attention of both question to context and context to question, multi-layered, and offered decent performance given time.

Weaknesses: memory-inefficient, requires complex solutions to deal with the sentinel vectors, and slow.

3.3 R-Net

We ran into a similar problem with R-Net as in coattention: memory. The authors of the R-Net paper lay out an architecture that's breathtakingly memory-intensive and we were initially forced to drop our batch size to 30 to even run the model (along with many other hyperparameter reductions) on an Azure NV6 with 12 GB of GPU memory. This crippled our training, so we devised a clever

solution to avoid allocating the largest tensor - a rank 4 matrix used in both self-attention and output layer calculations. For example, one of the original equations involved in attention is as follows: $s_j^t = V^T tanh(W_v^P v_j^P + W_v^P v_t^P)$. In this example, tanh is applied to the sum of large tensors that must be broadcast into the right shape and only subsequently reduced to a rank 4 tensor by taking the dot-product with V; this requires the allocation of a very large block of memory. We avoid this memory-intensive operation by exploiting the below identities (where a and b represent each product $W_v^P v_j^P, W_v^P v_t^P$) such that we never have to apply tanh() to a sum of 2 different-sized (and thus broadcasted) tensors.

$$
\begin{aligned}
\tanh(a+b) &= \frac{e^{a+b} - e^{-a-b}}{e^{a+b} + e^{-a-b}} \\
&= \frac{\tanh(a) + \tanh(b)}{1 + \tanh(a)\tanh(b)} \\
(V^2)^T \tanh(a+b) &= \frac{\langle V^2, \tanh(a)\rangle + \langle V^2, \tanh(b)\rangle}{(V^2)^T(1 + \tanh(a)\tanh(b))} \\
&= \frac{\langle V^2, \tanh(a)\rangle + \langle V^2, \tanh(b)\rangle}{\sum_i \left(V_i + V_i \tanh(a)_i V_i \tanh(b)_i\right)} \\
&= \frac{\langle V^2, \tanh(a)\rangle + \langle V^2, \tanh(b)\rangle}{\sum_i V_i + \langle V, \tanh(a)\rangle \langle V_i \tanh(b)\rangle}
\end{aligned}
$$

This allowed us to raise batch size significantly. However, this model still trained very sluggishly due to the still-high number of parameters. As such, we extracted only the self-attention portion for our final model.

Strengths: Self attention has great performance and R-Net is one of the best-performing models

Weaknesses: Calculations involve many rank4 tensor, the paper is incredibly ambiguous, the model requires a custom GRU implementation for question attention, memory inefficient, and was slow/trained very sluggishly (likely due to massive memory requirements)

3.4 Final Architecture (Self Attention)

Strengths: Simple implementation, modular (could be attached to e.g. coattention), good performance, relatively memory-efficient, allows for multi-layered attention

Weaknesses: It's not full R-Net

Our final architecture is primarily reliant on self attention, first presented in R-Net, and character-level embeddings. Please see Figure 2 below for a high level visual overview of the model.

As with the baseline model, we first process the data into a set of pretrained, untrainable word embeddings from GloVe. We then process the characters within each word, generating a set of character embeddings from a dictionary that maps alphabetical character (plus periods and commas) to an embedding vector. This set of embeddings is then fed into a 1-dimensional convolutional neural network and max-pooled to obtain a set of embeddings for a given word. We added this step for multiple reasons: 1) this offers information on words that would otherwise be complete unknowns to the model, and 2) this acts as an input feature that gives the neural network a chance to train on lower-level features (such as the similarity between surprised and unsurprised) that the GloVe embeddings lack.

The question and context embeddings are each then fed into independent bidirectional LSTM networks that encode them. These were originally GRU networks in the baseline code, but we found that LSTM's (which are best suited for long sequences) worked better on the input, where context could be 500 words long. Finally, dropout is applied here for regularization.

We multiply the new hidden representations of the question and context, then apply a masked softmax to the result. This provides us with a basic form of attention, which we then feed into our self attention layer. For this next layer, we modified the self attention equations given in the R-Net paper

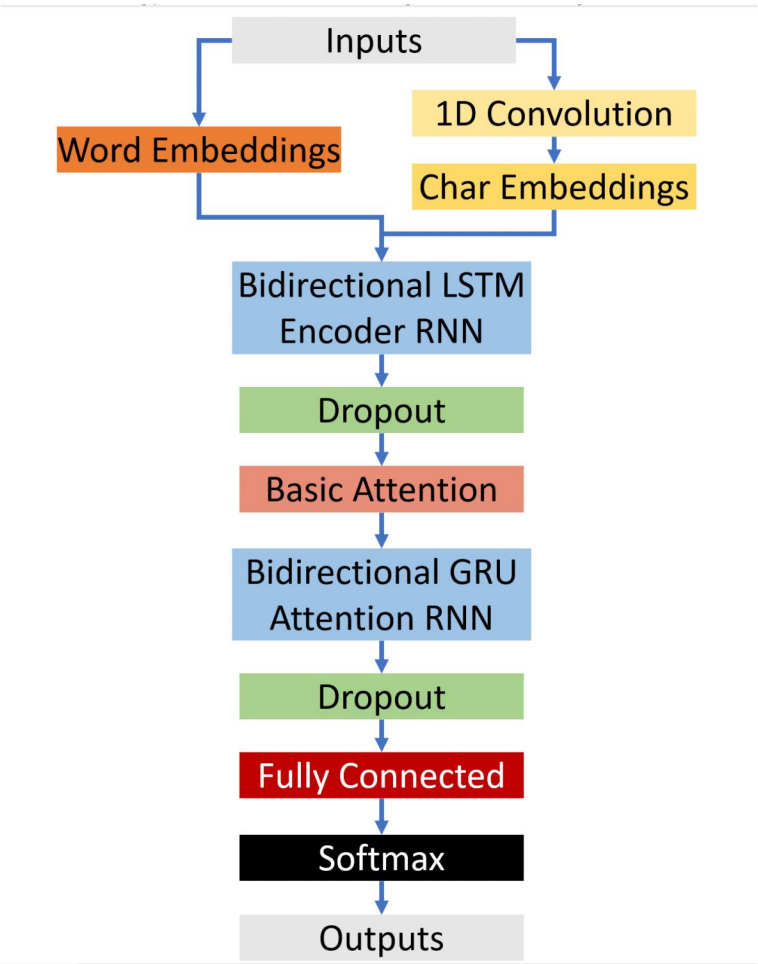


Figure 2: Model Architecture

as we removed the rest of the infrastructure. We thus use the following equations (note that just the first is changed) instead:

$$s_j^t = V^T tanh(W_v^1 v + W_v^2 v)$$
$$a_i^t = exp(s_i^t) / \sum_{j=1}^{n} exp(s_j^t)$$
$$c_t = \sum_{i=1}^{n} a_i^t v_i$$

Should a side-by-side comparison be done, the reader will notice that only the first equation has actually been changed; we multiply each attention output by a different set of weights before expanding them in alternate dimensions (W_v^2 is expanded in a different dimension; read the paper for an idea of what they use it for). Also of note is the fact that we use the memory efficient processed described in the R-Net section above. Once we have c_t, we concatenate the attention output with it and feed the result into a bidirectional GRU network. Finally, we apply dropout to the resulting output and run it through a fully connected layer for our final attention output.

Once we have our attention and hidden context/question encodings, we multiply and combine them in a softmax layer to get our distributions. We then find the argmax over the start distribution to get our starting prediction, then take the argmax over the next 15 words to find our end prediction. Finally, we obtain the loss by summing up the cross-entropy loss over the start and end positions within each predicted/ground truth answer. To train, we rely on the Adam optimizer with default parameters.

4 Experiments

As iterated before, we tested on the Stanford Question Answering Dataset (SQuAD). As we participated in the default project, we used CodaLab to compete with other teams on a secret test set.

4.1 Model Tests and Configurations

We realized early on that there would not be much hyperparameter optimization beyond regularization rates/locations, learning rates, and hidden layer sizes. Thus, we focused primarily on these when performing various searches while testing slight modifications to our architecture. Once we found parameters we were satisfied with, we continuously ran the model for up to 15,000 iterations or until the dev loss rose significantly; this allowed us to cull only the best development score possible from each run.

1. We first invested a significant amount of time in regularization. Without dropout, our model overfit to an absurd degree (.2-.3 difference in F1 scores, maxing out at .84 on the train set compared to less than .5 on the dev set). Since the modeling power so clearly outstripped regularization, we focused much of our attention on tuning dropout rates and location. L2 regularization was also attempted, but we were wary of using it because it has the potential to suppress finer model parameters; our limited experimenting with it resulted in models that trained poorly in comparison to dropout-based models.

   Our baseline model started out with a dropout layer in the encoder and basic attention layers but still overfit. We thus added a dropout node within our self attention layer. This worked better than baseline; however our testing revealed that the best results came when we removed the basic attention dropout node and relied on a single dropout layer after self attention. With one 15% dropout layer in each of the encoder and attention networks, we consistently achieved a maximum F1 score increase of (see the Evaluation subsection below for more details) around .02 over having 2 attention dropout layers. This is despite a train F1 score of around 0.83, which indicates a large degree of overfitting. Interestingly enough, after over a dozen runs varying the 2 dropout values (from 8% to 20%) we found that 15% dropout still created the best models. Additionally, we tried varying the dropout rates of the layers (e.g. 10% on the encoder network and 15% for the attention output), but these tests all produced worse scores.

2. Learning rates were the next parameters that we optimized for. We generated 7 learning rates from a uniformly random distribution between .0002 and .002; the best rate after 7 epochs (picked for time constraints) still ended up being the baseline .001 learning rate model. It struck a good balance between efficient loss reduction and stability, as higher learning rates had a higher chance of derailing during training while lower learning rates could cause the model to get stuck in a suboptimal local minimum early or just train too slowly. We actually ran over a dozen tests on learning rates from a previous architecture but had to settle for 5 tests for the current model when we made a last minute change.

3. Our last major set of hyperparameter searches involved the hidden layer sizes for both our character embeddings and bidirectional networks. We searched from 200 to 300 for the hidden bidirectional network sizes and found that 200 was actually perfect for our needs - higher hidden sizes required higher regularization rates, which reduced our dev F1 scores. Thus, for a dropout rate of 15%, 200 was a good hidden size. For our character embeddings, we initially trained on a convolutional neural network feature size of 50, but we found that 100 worked better. Unfortunately, we were unable to test these as much as preferred due to the aforementioned architecture change.

4. Other tests: We did quite a bit of testing with the Adadelta optimizer but stayed with Adam as it always performed better. Similarly, once we had our final architecture we tried replacing various activation functions in the fully connected/GRU layers, such as SELU (which is backed by a fantastic paper, by the way), but this gave us no increase in performance. We also tried averaging and max pooling our attention/encoder outputs instead of concatenating them, but we also obtained subpar performance with this. Additionally, we attempted to stack LSTM and GRU layers on top of the one within the attention layer, but this caused the model to expand massively (in terms of parameters) and made it very difficult to train;

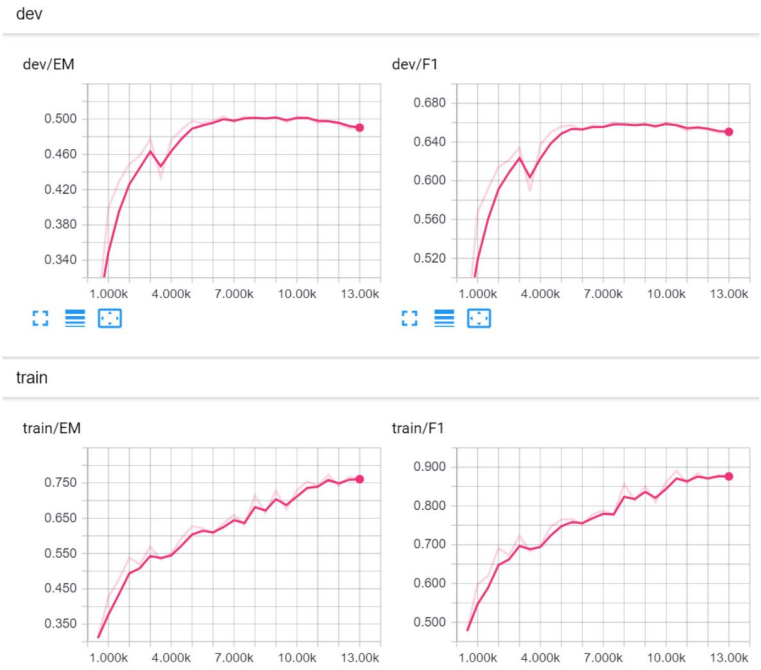


Figure 3: Model Results (x-axis represents iterations, y-axis represents score)

we abandoned this after juggling memory issues and highly unstable loss landscapes. Finally, we implemented many different output layers - the DrQA output, Conditioned end prediction, and the R-Net output. Unfortunately, none of these outperformed the simple softmax output layer.

4.2 Evaluation

For evaluation, we rely on F1 and EM score. F1 score is the harmonic mean of precision and recall, while EM score represents 'Exact Match' - it is 1 for a sample only if the prediction matches the ground truth exactly, 0 otherwise. However, there are 3 'ground truths' provided in the dataset - the evaluation metric takes the maximum score for each sample over each of these 3 to ideally mitigate errors in the labeling process.

4.3 Results

We achieved an F1 score of 71.107 and an EM score of 60.097 on the SQuAD test set; these were slightly below our F1/EM scores on the dev set as expected. The run below is the one we settled on for our submission to the test leaderboard. Note the increasing overfit after about 7k iterations, when the dev F1/EM scores begin to drop as train F1/EM scores rise. Our submitted model used the checkpoint from 7k iterations to avoid the overfit.

While performing qualitative error analysis, we noticed issues with many similarly-placed (or named) entities within the context. Below is perhaps the best example of this failure that we've found, which seems to get confused from all the 'downtowns' present. This could potentially be helped by (for example) adding an input feature that indicates flipped semantic meaning when 'not' is in front of a word (including the question).

```
Context: southern california is home to many major business districts .
central business districts ( cbd ) include downtown los angeles , downtown
san diego , downtown san bernardino , downtown bakersfield , south coast
metro and downtown riverside .
    QUESTION: what is the only district in the cbd to not have " downtown "
              in it 's name ?
```

```
TRUE ANSWER: south coast metro
PREDICTED ANSWER: downtown los angeles , downtown san diego ,
                  downtown san bernardino
```

All in all, the neural network performed quite accurately. We also noticed errors in the labeling as we parsed through examples, but those were generally rare.

5 Conclusions

This project was an incredibly educational experience for us and we're glad we took the approaches we did. Covering all of the various forms of attention (and reading all of the associated papers) broadened our view into the field in ways we never would have anticipated. An especially interesting learning exercise came from delving into the stark differences between BiDAF and R-Net - the two models are vastly different, yet both perform extremely well on the SQuAD challenge and develop intuitive theories. The process of attaching a convolutional neural network was also useful, as coverage of the integration of recurrent and convolutional networks was rare in class assignments.

There were definitely some features we were unable to pursue due to time constraints. For one, we were never able to tune either BiDAF or R-Net to work particularly well. While R-Net simply had too many parameters to reasonably train (even after we apply our fancy memory-saving algorithm and weight-sharing) within our timeframe, BiDAF is something we would definitely like to look at again for future work. We also would have liked to develop an ensembling method instead of the single model currently being used.

The sheer amount of knowledge and experience that we gained over the course of this project is astounding. The numerous influential papers (all published between 2016-2018 and all making a large impact on deep language modeling), the experience of training neural networks, and general atmosphere of friendly competition with our peers are all valuable kernels that we've taken away from this project. We would like to thank the teaching staff of CS 224N for their fantastic instruction and look forward to bringing our newfound experiences to other fields.

6 References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.

[2] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603, 2016.

[3] Microsoft Research Asia. R-Net: Machine Reading Comprehension with Self-Matching Networks. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf, 2017.

[4] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.

[5] Self-Normalizing Neural Networks by Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter. https://arxiv.org/pdf/1706.02515.pdf

[6] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.

[7] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.