SQuAD 2.0 (QANet, Character Embeddings and Token Features, Hyperparameter Tuning)

Bernardo Casares, Eric Nielsen, Eric Redondo
{bcasares, nielsene, eredondo}@stanford.edu

Abstract

The Stanford Question Answering Dataset (SQuAD) dataset is one of the most popular benchmarks for reading comprehension and question answering tasks. This project explores several techniques to improve upon a baseline model that is designed to answer questions using phrases in given context paragraphs. We are focusing specifically on the SQuAD 2.0 dataset, in which the correct answers may or may not appear in the context paragraphs. The baseline model is based on Bidirectional Attention Flow (BiDAF), and the enhancement techniques explored include character-level embeddings, the addition of more expressive token features (e.g., part-of-speech tagging), hyperparameter tuning, and an Encoder Block architecture called QANet. The character-level embeddings and part-of-speech tagging were the most impactful improvements in terms of model performance.

1 Introduction

Training computers to understand text has gained significant popularity over the past several years due to the many applications it enables. The SQuAD 2.0 dataset is one of the most popular reading comprehension benchmarks. The dataset comprises many questions, each associated with a context paragraph that may or may not include the answer. The goal of the challenge is to train a computer to answer the questions as correctly as possible - providing a measure for how well the computer can 'understand' text. Significant research has been done over the past several years to design models to compete in this difficult challenge. In general, submissions fall into one of two categories: those that use Pre-trained Contextual Embeddings (PCE) such as ELMo and BERT, and those that do not. PCE models tend to have much higher performance than non-PCE models, however come with an added level of complexity. We chose to focus our project on enhancing existing non-PCE models.

For a baseline, we were given a Bidirectional Attention Flow (BiDAF) model (1) that used only word-level embeddings. We improved this baseline by adding a more expressive embedding layer that supplemented word-level embeddings with character-level representations and three additional token features: part-of-speech (POS), named entity recognition (NER) tags, and term frequency (TF). These new embeddings allow the model to better capture rich semantic and structural word information and to better understand word relationships. Ultimately, these updates and hyperparameter tuning resulted in a superior BiDAF model.

We also worked to improve upon a different model architecture called QANet. Prior to BERT, QANet had state-of-the-art performance for SQuAD 1.1. This led us to to believe that a successful implementation of QANet could obtain high scores in the SQuAD 2.0 dataset as well. The QANet design allows training to be parallelizable, train faster, and theoretically obtain better results as compared to BiDAF models. Although we did not obtain faster speeds or better results using the QANet, we believe that further updates could be made to improve performance.

Below, we outline a collection of related work, our approach and implementation, the experiments we conducted and their results, and a brief analysis of overall model performance.

2 Related Work

2.1 BiDAF

The baseline model is heavily inspired by the BiDAF model developed by Seo et al.(1). This 2016 paper expands on previous work utilizing attention mechanisms in machine comprehension by introducing a multi-stage hierarchical process to improve the representation of context. Additionally, a BiDAF mechanism is presented to obtain a query-aware context representation without the need for prior summarization. Using these techniques, this model achieved an F1 score of 77.3 and an EM score of 68.0 on the test set of SQuAD 1.0 dataset. The primary difference between the original model and the baseline employed in this work is that the original implementation included a character-level embedding layer.

2.2 QANet

In a 2018 paper (8), Yu1 et al. present the QANet architecture, which addresses one of the major issues with BiDAF models: their recurrent neural networks (RNNs). The authors describe the lengthy run-times of both training and inference due to the sequential (non-parallelizable) nature of RNNs. To solve this, the QANet architecture does not require RNNs. It's encoder instead consists only of convolution and self-attention, where convolution models the local interactions and self-attention models the global interactions. Due to the large computational power they had available, their model was 3x to 13x faster in training and 4x to 9x faster in inference while achieving equivalent accuracy to recurrent models. Additionally, one of the extra benefits of QANets is that the speed-up gain allows the model to be trained with much more data. The paper describes a data augmentation technique to take advantage of this benefit, however we did not include this component in our work.

2.3 DrQA

In a 2017 paper by Chen et al. (7), the authors outline a model called DrQA that is designed to compete in the SQuAD 1.0 challenge. The model is notable due to its implementation of token embeddings, which include several token features in addition to Glove (11) word embeddings. Specifically, each token embedding includes information about the token's part-of-speech, named entity recognition tag, and normalized term frequency. At the time the paper was published, the DrQA model achieved competitive scores on the SQuAD 1.0 leaderboard - 69.5 EM and 78.8 F1 on the dev set, and 70.0 EM and 79.0 F1 on the test set. Although the authors of the paper offer little detail about exactly how the additional token features were generated and then aggregated in each embedding, our team was inspired to attempt a similar approach in our work. More expressive token embeddings should provide the model with more information to learn from.

3 Approach

Our team worked to build two types of models to compete on the non-PCE SQuAD 2.0 leader board. Our first model design kept the overall BiDAF architecture from the provided baseline, and improved its performance with tuned hyperparameters and enhanced token embeddings. These enhancements included the addition of character-level embeddings, as well as part-of-speech, named entity recognition tag, and term frequency embeddings. Our second model was designed using the QANet architecture, and also made use of the enhanced token embeddings. Initial attempts were made to instead integrate BERT embeddings, however this approach was terminated due to a number of implementation issues. It is briefly described below for completeness.

3.1 Baseline

The baseline model used for benchmark purposes was the baseline model provided by CS224N course staff. As mentioned previously, the baseline model is based on a BiDAF architecture.

3.2 Character-Level Embeddings (BiDAF)

The first improvement we made to the provided baseline was the addition of character-level embeddings. Originally, each word in the vocabulary was represented as a length 300 GLoVe embedding vector (11). Word-level embeddings are commonly used in NLP tasks and typically result in higher accuracy and lower computational costs as compared to character-level embeddings. However, models using word-level embeddings are inherently limited due to the fact that they cannot reason about words not included in the vocabulary. To address this limitation, we concatenate the word-level embeddings with length 64 character-embedding vectors.

The character-level embeddings are generated using a convolutional encoder that was built as part of a previous CS224N problem set. The first step of the encoder is a lookup of the individual length 64 character embeddings, using the character indices and embeddings provided by the baseline model. Then, the embeddings of all characters in each word are combined using a 1-dimensional convolutional network. To calculate the i^{th} output feature for the t^{th} window of the input, a convolution is done between input window $x_{reshaped[:,t:t+k1]}$ and weights $W_{[i,:,:]}$, and bias term b_i:

$$(x_{conv})_{i,t} = sum(W_{[i,:,:]} \odot x_{reshaped[:,t:t+k1]}) + b_i \tag{1}$$

The ReLU function and max-pooling are applied to the overall convolution output to produce the final embedding $x_{conv.out}$. Concatenating each of these length 64 vectors with each length 300 word-level embedding producing final embeddings that are length 364 vectors.

$$x_{conv.out} = MaxPool(ReLU(Conv1D(x_{reshaped}))) \tag{2}$$

3.3 Additional Word Features (BiDAF)

In addition to word-level and character-level embeddings for each token, we also added three additional embedding features using different attributes of each token. Part-of-speech (POS), named entity recognition (NER) tags, and normalized term frequency (TF) vectors of length 6, 5, and 1, respectively, were concatenated with each word-level (length 300) and character-level (length 64) embedding vector to produce overall token embeddings of length 376.

POS embeddings were generated using Python's $nltk$ (Natural Language Toolkit) library, which tags tokens with one of 46 possible parts-of-speech. Initially, the POS embeddings were represented as a one-hot encoding vector of length 46. However, to reduce overall embedding size (which shortens model training time), the final model represents each token's POS using a binary vector of length 6. The numerical value of each binary vector equals the index of the assigned POS tag when placed in a list of length 46. Due to time and compute constraints, the use of one-hot encoding vectors was not tested, although it is possible that this representation could lead to superior results since it would make the embeddings more expressive in terms of POS.

NER tag embeddings were generated using Python's $spacy$ library, which tags tokens with one of 19 possible NER designations. These designations correspond to a series of descriptive categories (e.g., person names, organizations, locations, times, money). Similar to the POS embeddings, NER tags were initially represented as one-hot encoding vectors, but were shortened to binary vectors of length 5. Of note is the limitation of $spacy$ (and other comparable libraries) in tagging many tokens with NER tags. Of the 88,714 total words in the model's vocabulary, only 14,696 (16.6%) could be associated with NER tags. Future model enhancements could include the ensembling of multiple NER tagging tools to provide greater coverage.

Finally, the normalized term frequency of each token added a single additional embedding value. The frequencies were computed using all phrases in the training set (both context paragraphs and questions). The number of times each unique token appeared were first counted; then the counts were normalized using the largest value to restrict values to the range of 0 to 1. Though not attempted in this project, it is possible that scaling the term frequencies or representing them in another way could lead to better model performance, as it could allow the frequencies to play a more significant role in each token embedding.

3.4 Hyperparameter Tuning and Additional Updates (BiDAF)

Based on initial experimentation with the BiDAF models, several hyperparameters were tuned via experimentation in an attempt to combat noted weaknesses. In an effort to make the model more expressive, the hidden size was increased from 100 to 128. To prevent overfitting, the drop probability was increased from 0.2 to 0.3 and L2 weight decay was added with parameter $\lambda = 0.001$. Finally, to improve the model's convergence, the Adam optimizer (9) was substituted for Adagrad (10), due to the fact that it generally does better modulation of the learning rate and makes use of momentum in the gradient.

3.5 QANet

Before QANet, most question answering models were based on RNNs with attention. However, RNNs have a major downside, they are not parallelizable and slow. In an attempt to solve this, while trying to maintain a similar performance, our group added on top of the QANet architecture described in (8).

The high level structure of the model consists of an embedding layer, an embedding encoder layer, a context-query attention layer, a model encoder layer and an output layer. A visual description can be seen in Figure 1.

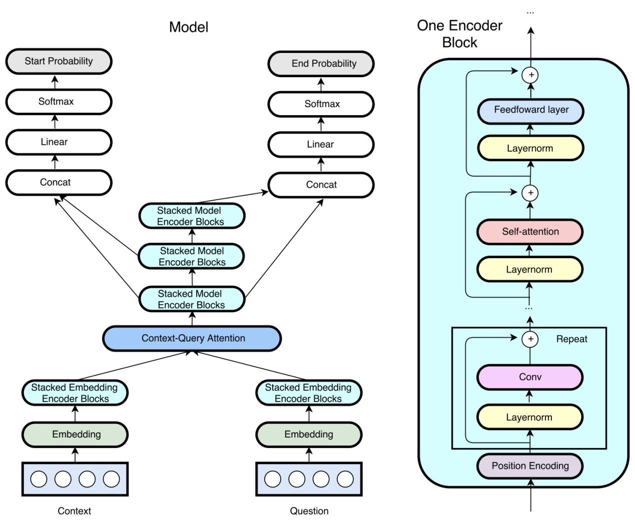


Figure 1: A visual description of the QANet Architecture

We followed the implementation from the paper as closely as possible, with the exception of the embedding layer. We also used several functions of existing code from public sources, specifically two repositories containing implementations of transformers (from which QANet adapts many ideas): jadore801120/attention-is-all-you-need-pytorch (13) and SamLynnEvans/Transformer (14).

The embedding layer from our most successful BiDAF model was used, meaning that embeddings included word-level and character-level representations, in addition to three additional token features. For the encoder blocks (both the stacked embedding and stacked model blocks in Figure 1) we used existing positional encoding code from (14) and self-attention and feed-forward code from (13). The repeated convolutional sub-block was coded by our team. The context-query attention block code from taken from the class-provided BiDAFAttention function, while the three layers in each output block were coded by our team.

3.6 BERT

Our first attempt for the project was trying to modify an existing Bert implementation and adapt it to the baseline model.

We started from the following github repository "huggingface/pytorch-pretrained-BERT", and worked to extract the bert embeddings for our project. However, after several hours of work, we decided that it was going to be easier to just run Bert from the hugginface repository. We run into memory errors, decreased the batch size to 8, and changed the gradient accumulation steps to 4. However, doing this approach did not work well for us, and we got very bad results (EM: 0.099, F1: 3.869).

Although our initial results could likely be improved by fine-tuning the non-embedding layers for the SQuAD 2.0 dataset, we realized it was going to be very challenging to add additional improvements on top of BERT. Given our experience and the advice of course TAs, our team decided not to integrate BERT with the provided BiDAF baseline, and to stop any further work with BERT.

4 Experiments

4.1 Data

The dataset for the project is SQuAD 2.0 (2). The train and dev splits used for the project are sourced from the official SQuAD 2.0 train set, while the test split is sourced from the official dev set.

4.2 Evaluation Method

The primary methods for model evaluation are EM and F1 scores, the official SQuAD 2.0 evaluation metrics. Results will be compared to those of other students on the leaderboard. Since the final model design does not use BERT embeddings, we are competing in the non-PCE division.

4.3 Experimental Details

For all experiments in the subsections below, the default train and dev sets were used. The BiDAF experiments built on top of or altered the provided baseline model. Unless otherwise noted, a learning rate of .001 was used, and the model was trained for 30 epochs.

4.3.1 BiDAF Model Experiments

The initial experiment utilized the provided BiDAF model with no changes made to the default architecture or parameters in order to reproduce the expected baseline performance. The learning rate used was 0.5.

Subsequent experiments revolved around the addition of new word and/or character embedding information. First, experiments were run with the addition of character embeddings. These experiments incorporated the baseline model with character-level embeddings and were trained with the Adam optimizer. Next, experiments were done to gauge the effectiveness of character-level embeddings supplemented with additional embedding features: part-of-speech (POS), named entity recognition (NER), and term frequency (TF). Each of these experiments was run with the BiDAF model and the Adam optimizer.

Final BiDAF experiments with varying hyperparameters were run on a model using full token embeddings (combining word and character embeddings with all three additional token features). The version using the hyperparameter configuration leading to the best EM and F1 scores is included in the results below. Specifically, this model using a learning rate of 0.001, L2 weight decay with $\lambda = 0.001$, drop probability of 0.3, a hidden layer size of 128, and the Adam optimizer. It was trained for 35 epochs.

4.3.2 QANet Model Experiments

Due to time and computing constraints after the QANet architecture was successfully implemented, a single experiment was conducted using the model. For this experiment, the embedding layer of highest performing BiDAF model was used; token embeddings comprised word-level and character-level representations, as well as POS, NER tags, and normalized TF vectors. The hyperparameters used were largely taken from the original QANet paper (8). The learning rate increased logarithmically from 0 to 0.001 during the first 1000 training steps, then remained constant. Normalization was implemented using L2 weight decay with parameter $\lambda = 1 * 10^{-7}$, and dropout layers with drop rates of 0.1. Differing from the paper, we chose a batch size of 8 and a hidden size of 96 for better performance on the hardware used. All other hyperparameters matched those described in the original QANet paper.

The experiment was run on an NV12 virtual machine for a total of 25 epochs, each including 129,941 iterations. Each epoch took approximately 1.5 hours to complete. The number of epochs trained was fewer than the standard 30 due to model's lengthy run time.

4.4 Results

4.4.1 BiDAF Models

Table 1 and Figure 2 provide a summary of our final BiDAF model results. The model using full embeddings (word, character, POS, NER tag, and TF) with tuned hyperparamters performed the best out of the 7 models tested. It achieved an EM score of 60.07 and an F1 score of 63.19 on the dev set. Compared to the baseline model, this is an EM score increase of 3.66 and an F1 score increase of 3.30. The training loss plot in Figure 2 shows that the tuned model maintained the highest training loss out of any of the models. This suggests that the other models overfit the training set, and that the tuned model was better able to generalize, therefore performing better on the dev set.

Also of note, it appears that adding part-of-speech embeddings had the most beneficial effect out of the three additional token features. The dev EM and F1 scores of the POS model were nearly as high as the scores of the model using the full embeddings. Adding only named entity recognition tags actually decreased performance as compared to the model with only word and character embeddings; adding only normalized term frequency resulted in limited improvement. These results indicate that the model could be simplified and sped up by removing NER tag and TF embeddings, while maintaining similar performance.

Table 1: BiDAF Model EM and F1 Scores

Model	EM Score	F1 Score
Baseline BiDAF	56.14	59.89
Word+Character Embeddings	58.91	62.06
Word+Character+POS Embeddings	59.74	62.80
Word+Character+NER Embeddings	58.34	61.78
Word+Character+TF Embeddings	59.13	62.38
Word+Character+POS+NER+TF (All) Embeddings	59.50	62.82
All Embeddings, Hyperparameters Tuned (Best)	60.07	63.19

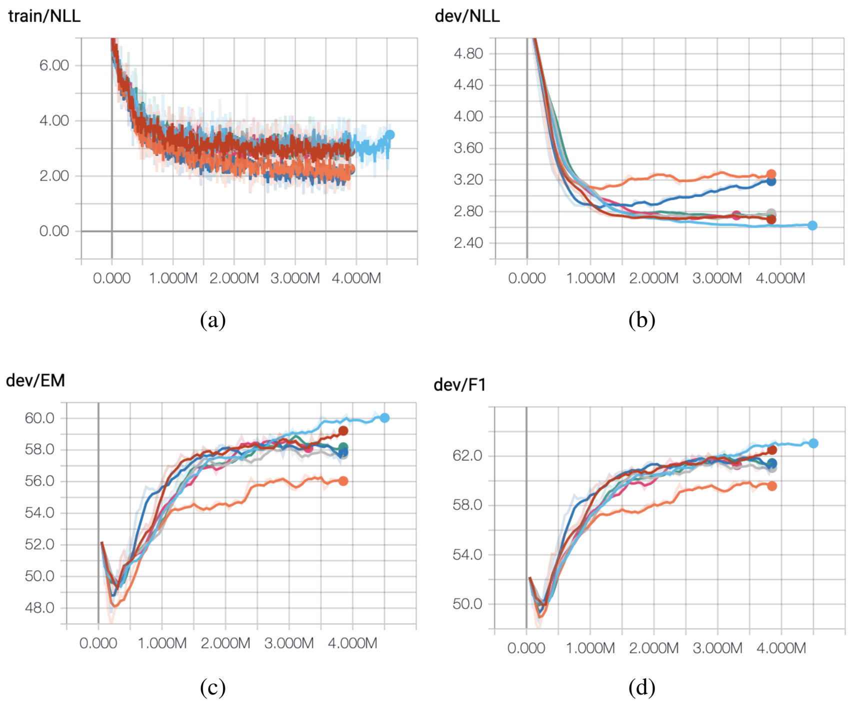


Figure 2: Final BiDAF model results. (a) Training loss, (b) Dev loss, (c) Dev EM, (d) Dev F1 Orange: Baseline, Cyan: All Embed Tuned, Red: All Embed, Blue: Word/Char Embed, Magenta: Word/Char/POS Embed, Gray: Word/Char/NER Embed, Green: Word/Char/TF Embed.

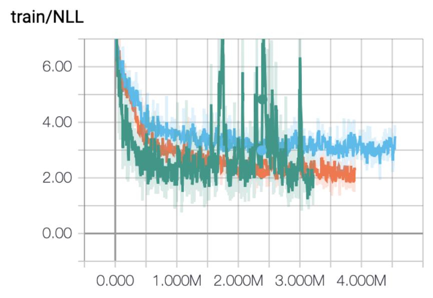
4.4.2 QANet Model

Table 2 and Figure 3 provide a summary of our single QANet model results. After 25 epochs of training, the QANet model achieved a dev EM score of 57.00 and F1 score of 60.59. These scores are marginally better than the baseline results, but worse than all other BiDAF models.

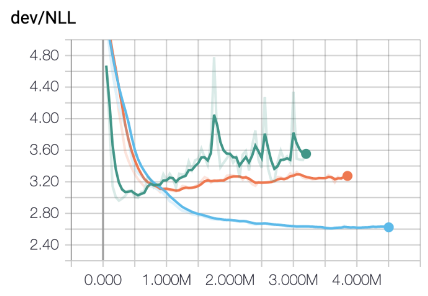
The training and dev plots in Figure 3 suggest that further tuning the model's hyperparameters could result in far superior results. The QANet model decreased loss faster than the baseline and best BiDAF models during both training and dev. However the loss then began to increase, often erratically. Decaying the learning rate after approximately 500,000 iterations would likely improve this behavior and result in higher scores. Increasing the batch size could also lead to improvement in this area. A small batch size of 8 was chosen due to computing constraints; a larger batch size such as 64 would lead to less variability between batches and more consistent training.

Table 2: QANet Model EM and F1 Scores

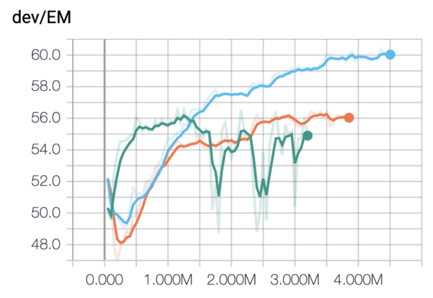
Model	EM Score	F1 Score
All Embeddings with QANet	57.00	60.59



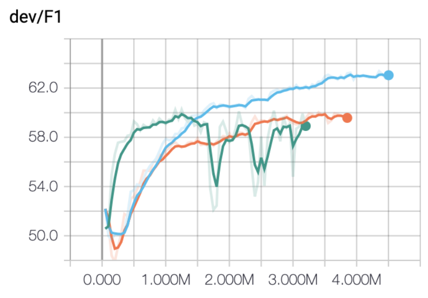
(a)



(b)



(c)



(d)

Figure 3: Final QANet model results. (a) Training loss, (b) Dev loss, (c) Dev EM, (d) Dev F1 Orange: Baseline BiDAF, Cyan: All Embed Tuned BiDAF (best), Green: All Embed QANet

5 Analysis

5.1 Ablation Studies

Based on an ablative analysis of the BiDAF model experiments, it is possible to gather some insight in to which changes to the original baseline had a significant positive effect on the result. The final tuned model (drop prob = 0.3, hidden size = 128) using all embeddings achieved an F1 score of 63.19. The same model using untuned hyperparameters (drop prob = 0.2, hidden size = 100) achieved a score of 62.82. The difference of 0.37 (0.6%) indicates that tuning the hyperparameters provided only a minor advantage. We can then compare the model with all (word+character+POS+NER+TF) embeddings to the three models with word, character, and one of POS/NER/TF embeddings. The

POS model achieved an F1 score of 62.80, while the NER achieved 61.78 and TF achieved 62.38. From these results, we see evidence that the POS embeddings provided a significant positive effect while the other two provided very little, since performance drops off heavily when removing the POS embeddings, but remains nearly constant when removing both NER and TF.

Finally, we can analyze the effect of removing character embeddings from the model by comparing the results of the character embeddings experiment with the original baseline model experiment. The model with character embeddings achieved an F1 score of 62.06 while the baseline achieved 59.89. Of all the model components analyzed so far, this difference of 2.17 (3.5%) represents the greatest effect. It is likely that, with the addition of character embedding information, the model is better able to handle instances of unknown words appearing in either the question or context by using character-level information to gain insight into these unknown words.

5.2 Characteristic Examples

The example outputs viewable in TensorBoard provide some evidence for general strengths and weakness of the trained models. For this analysis, the best performing BiDAF model (enhanced-embeddings, fine-tuned) and the enhanced-embedding QANet model are considered. Although it is difficult to generalize about model performance, we note several interesting behaviors below.

Results from the enhanced-embedding, fine-tuned BiDAF model show many chosen answers contain context phrases that are close to correct, but too lengthy. For example, the correct answer of the question 'What is the most important type of Norman art preserved in churches?' is 'mosaics', but the model chose 'sculptured fonts, capitals, and more importantly mosaics'. Similarly, the correct answer of the question 'The time required to output an answer on a deterministic Turing machine is expressed as what?' is 'state transitions', but the model chose 'M on input x is the total number of state transitions'. Continuing the trend, the correct answers of two other questions are 'lack of net force' and 'problem instance', but the model chose 'a lack of net force' and 'a problem instance', respectively. Strategies to address these types of errors were not implemented in this project, but future work could be done in this area.

The enhanced-embedding QANet model demonstrated an ability to answer with the correct type of item or idea, but often struggled with choosing which of the items from the context to choose from. For example, the correct answer to the question 'Which directive mentioned was created in 1994?' from the context '...the 1994 Works Council Directive, which required workforce consultation in businesses, and the 1996 Parental Leave Directive...' is 'Works Council Directive', but the model instead answered with 'Parental Leave Directive'. This could be due to the sub-optimal training of the QANet architecture resulting in a weak ability of the attention mechanism to identify from where in the context the answer is likely to come.

6 Conclusion

As demonstrated by this project, there are multiple paths to improving the performance of a baseline BiDAF model competing in the SQuAD 2.0 challenge. Supplementing word-level embeddings with character-level embeddings leads to EM and F1 scores both improved by approximately 2 points. EM and F1 scores can be further increased by approximately 1 point each by fine-tuning hyperparameters and adding three additional features to the embeddings: part-of-speech, named entity recognition tags, and term frequency. Adding only the part-of-speech feature seems to be nearly as effective as adding all three. Further tuning the model's hyperparameters and adding different, more descriptive token feature embeddings could result in even better performance.

Our work with the QANet model also achieved results better than the baseline, but there are many opportunities to further improve its performance. Many more experiments could be run using different hyperparameters. Specifically, decaying the learning rate at later time steps and increasing the batch size would likely improve the sometimes eratic training behavior. Also, increasing the model's hidden size could allow for a more expressive and therefore more accurate model.

Overall, it is recognized that the increase from baseline performance achieved in this project was relatively small. However, given more time and computing power, we are confident that the simple improvements described above could achieve far better results.

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603, 2016.* https://arxiv.org/pdf/1611.01603.pdf

[2] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822, 2018* https://arxiv.org/abs/1806.03822

[3] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated Self-Matching Networks for Reading Comprehension and Question Answering. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, 2017, pages 189–198* http://aclweb.org/anthology/P17-1018

[4] Shuohang Wang and Jing Jiang. Machine Comprehension Using Match-LSTM and Answer Pointer. *arXiv preprint arXiv:1608.07905, 2016* https://arxiv.org/abs/1608.07905

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805, 2018.* https://arxiv.org/pdf/1810.04805.pdf

[6] The initial Bert Github repository. huggingface/pytorch-pretrained-BERT

[7] Danqi Chen, Adam Fisch, Jason Weston, Antoine Bordes. Reading Wikipedia to Answer Open-Domain Questions. *arXiv:1704.00051v2 [cs.CL] 28 Apr 2017* https://arxiv.org/pdf/1704.00051.pdf

[8] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi,and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading. comprehension. *arXiv preprint arXiv:1804.09541, 2018.*

[9] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs.LG] 30 Jan 2017* https://arxiv.org/abs/1412.6980

[10] John Duchi, Elad Hazan, Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research 12, 2011, pages 2121-2159* http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[11] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez,Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, pages 5998–6008, 2017.

[13] Attention is all you need in pytorch repository: jadore801120/attention-is-all-you-need-pytorch

[14] Another Attention is all you need in pytorch repository: SamLynnEvans/Transformer