# Embedding and Attending: Two Hearts that Beat as One

Stanford CS224N Default Project, IID SQuAD

**Manuka Stratta**
Department of Computer Science
Stanford University
mstratta@stanford.edu

**Raymond Yao**
Department of Computer Science
Stanford University
ryao28@stanford.edu

## Abstract

Neural attention mechanisms have proven to be effective at leveraging relevant tokens of the input data to more accurately predict output words [1]. In this paper, we implement three different attention mechanisms (co-attention from Dynamic Coattention Networks [2], key-query-value self attention, and R-Net self-attention [3]) in the domain of the Question-Answering (QA) paradigm. Our goal was to produce a model that is highly performant compared to the baseline BiDAF model on the Stanford Questioning Answering Dataset (SQuAD 2.0) [4]. We combine these attention mechanisms with character-level embeddings to provide more local contextual information, and finally enhancing these embeddings by including additional input features (part-of-speech and lemmatized forms of words). Augmenting the baseline with these techniques produces a significant improvement compared to the baseline and results in an F1 score of 65.27 and EM score of 61.77 (an increase of 5.6% and 5.5%, respectively).

## 1 Introduction

The Question-Answering task has seen widespread interest among the NLP community in recent years due to its far-reaching applications (search engines, database entry retrieval, etc.), especially with the release of the SQuAD/SQuAD 2.0 data sets [4]. While QA (and more generally, reading comprehension as well) appears straightforward and simple for humans, it has proven to be a rather difficult task for machines to accurately process/understand a context passage & query, and output the span of the correct answer (if the answer exists at all). Numerous state of the art techniques have emerged, some utilizing transformer architectures, neural attention mechanisms, LSTMs, reformers, and countless other techniques, each with their unique benefits and drawbacks.

In this paper, we explore current performant methods in the QA space. We were particularly intrigued with the concept of neural attention mechanisms, which have seen tremendous growth in recent years in the ability to "look back" on words of the input sequence [1]. Namely, we experimented with different types of attention: co-attention [2], self-attention [5], and R-Net self-attention [3]. We were also motivated by feature engineering in the embedding layer, integrating character-level embeddings to condition on the morphology of tokens [6] and investigating the effectiveness of additional input features (the part-of-speech tag and lemmatized forms of words) [7]. Additionally, we were curious as to exploring the most ideal hyperparameters that produced the highest EM/F1 scores for our model by modifying different values of the learning rate (i.e. learning rate annealing) and dropout probabilities.

Our model performed best with the configuration of character-level embeddings and RNet self-attention, producing competitive EM/F1 scores of 61.771/65.268, respectively, on the dev set leaderboard. In our analysis of the results, we also qualitatively evaluate key characteristics of the errors of our outputs, noting cases where our model excelled in particular linguistic conditions as well as situations where our model performed poorly. Additionally, we reflect on the benefits/trade-offs of feature engineering, as well as detailing limitations and avenues for future courses of work.

## 2 Related Work

This paper lies in the space of question answering (QA), and more generally, machine comprehension. This field has seen great leaps of progress to not only increased amounts of data (e.g. with the 100k examples in SQuAD), but particularly thanks to neural attention mechanisms. Attention is a means by which a model can utilize specific aspects/words of the input sentence (by "attending" to input words), which also resolves the issue of semantic loss in longer sentences being compressed into a fixed-length vector during the encoding process. Numerous types of attention exist, such as self-attention, two-way attention, global attention, local attention, hierarchical attention, and so forth. Our baseline model uses the Bidirectional Attention Flow (BiDAF) implementation [8], which utilizes a bidirectional context-query attention mechanism and character-level/word-level embeddings. Exploration in attention mechanisms have been extensive.

In the field of coattention, Xiong et. al. [2] proposed the concept of Dynamic Coattention Networks (DCN) composed of a coattentive encoder that represents the co-dependent relationships between the query and context, as well as a dynamic decoder that modulates between estimating the beginning and end of the answer span. The authors are interested in evaluating how utilizing a specific variety of attention, namely coattention, can yield improved results by computing an alignment matrix on every pair of context and query words. Coattention is unique in that the words in the query inform on the words in the context words, and crucially, vice versa as well. They propose the concept of Dynamic Coattention Networks (DCN) composed of a coattentive encoder that represents the co-dependent relationships between the query and context, as well as a dynamic decoder that modulates between estimating the beginning and end of the answer span.

Another type of self-attention is leveraged in R-net [3], which poses the idea of coupling a Context-to-Question attention layer (to obtain the question-aware passage representation), with a self-matching attention mechanism (to refine the representation by matching the passage against itself). We were interested in implementing the self-matching layer from R-Net.

Finally, we took inspiration from the paper by Chen et. al [7], which pioneered the concept of leveraging manual token features (i.e. part-of-speech, named entity recognition tags, and term frequency) to aid learning by the machine comprehension system.

## 3 Approach

At a high-level, we focused on implementing models that concern primarily 1) the embedding layer (character-level embeddings, input features) and 2) attention techniques (coattention, key/query/value (KQV) self-attention, and RNet self-attention). Additionally, we ran hyperparameter tuning experiments (learning rate, dropout probabilities).

**Baseline**: We utilize the provided baseline that implements the BiDAF model specified by Seo et al. [8] (without character embeddings). Based on our initial training, the baseline model produced the following metrics (maximized across all values except minimized for NLL): $AvNA \approx 66.91, EM \approx 56.24, F1 \approx 59.69, NLL \approx 3.013$. For the implementation of our model, we seek to improve across these metrics on the SQuAD 2.0 dataset.

**Character-Level Embeddings**: As proposed by Seo et al. [8], our model implements character-level embeddings, in addition to word embeddings, to obtain a more nuanced context representation of the input sequence. This additional level of granularity provided by the character embeddings serves to add robustness on out-of-vocabulary terms and enables us to condition on the morphology of words.

We follow a similar approach to the BiDAF paper by using character-level convolutional neural networks (CNN) [6]. We pass our pre-processed context words $c_1, ..., c_N$ and query words $q_1, ..., q_M$ into a CNN that produces our character-level embeddings $c_{emb,1}, ..., c_{emb,N}$ and $q_{emb,1}, ..., q_{emb,M}$ for each word. We proceed to max-pool across the width of each embedding to find the maximal vector values for each word. Then, we concatenate the resulting character embedding with our word embeddings, and pass the result into a Highway Network for future processing.

**Input Features**: In taking inspiration from DrQA [7], we integrated the input features of part-of-speech (POS) and the lemmatized form of the context/query tokens into our model. Our reasoning was that including the part-of-speech feature provides useful information for our model to generalize patterns such as the order of parts-of-speech in identifying the answer. Similarly, utilizing lemmas may aid the model in understanding when word forms differ in the context and query. For example, with the context passage of "Godzilla ate a plane..." and the query of "What did Godzilla eat?", the system may be able to see that the lemmatized form of "ate" (namely "eat") is a direct match for a token in the query ("eat"), which may yield a more "directed search" for the eventual answer.

Heeding guidance from our TA, for each word in the context/query, we tokenize on the POS and lemmas of each token, in addition to obtaining the unmodified token text. We create maps from the POS/lemma to their respective indices, as well as POS/lemma embedding matrices that we eventually pass to our embedding layer. Finally, we concatenate POS/lemma embeddings with the word & character-level embeddings. While there is the added benefit of more embeddings for our model to process and extract patterns from, this significantly increased our setup running time by a factor of two.

**Co-attention**: We sought to implement a coattention mechanism akin to the one described by Xiong et al. [2] and the project handout.

Let $c_1, ..., c_N$ denote the context hidden states and let $q_1, ..., q_M$ denote the question hidden states, both in $\mathbb{R}^l$. We first apply a linear layer with a tanh nonlinearity to produce question hidden states $q'_1, ..., q'_M$. After we append sentinel vectors to each hidden state, we compute an affinity matrix $L$ to describe the similarity between the pairs of context and hidden states. Next, we take the masked softmax of $L$ row-wise to produce $\alpha$, then take the weighted sum with each question hidden state to obtain the Context-to-Question attention output $A$. We also take the masked softmax of $L$ column-wise to produce $\beta$, then take the weighted sum with each context hidden state to obtain the Question-to-Context attention output $B$. Crucially, we take the weighted sum of the C2Q attention distributions with the Q2C attention outputs to produce our second-level attention outputs $S$:

$$S = \alpha @ B,$$

where @ represents the batched matrix multiplication operation. Finally, we concatenate the second level attention output with our C2Q attention outputs and feed the result through a bidirectional LSTM layer, which is part of the modeling layer.

**KQV Self-Attention**: This is the second type of attention that we implemented, the "vanilla" key/query/value self-attention from lecture. We decided to apply the KQV self-attention layer right after the embedding layer becuase in lecture we saw that self-attention can be useful towards the beginning of the model. We could have experimented by placing the layer elsewhere, however. We made use of existing Assignment 5 code [9] to build the KQV self-attention block, while adapting to our model configurations, number of attention heads, and removing additional layers to focus on just the linear normalization and residual connection steps. The primary equation for self-attention is the following:

$$\text{att} = \text{softmax}(\frac{QK}{f})V$$

where $f$ is a normalization factor dependent on the size of $K$. Following what we learned in Assignment 5, we preceded this step by a normalization layer and completed the KQV self-attention layer with a residual connection as follows:

$$\text{att} = x + L(x),$$

where $x$ is the input to the KQV self-attention layer and $L$ is a LayerNorm.

**RNet Self-Attention**: To experiment with a third and final type of attention, we implemented the self-attention layer described in the RNet paper [3] (which they call Self-Matching Attention). This layer is an application of additive attention (coupled with a gated layer which resembled our BiDAF attention layer) whose goal is to refine the question-aware passage representation by matching the passage against itself. We took inspiration from an existing RNet implementation which we found online [10]. Our approach consists in keeping the BiDAF attention layer and then passing the output into the RNet self-attention layer to further enhance the question-aware passage representation. In our self-attention layer, we implement the following steps (where $v^P$ is the question-aware passage representation obtained by previous attention layer, $W^P$ and $W^{P\prime}$ are learnable linear layers, and $v^T$

is a learnable vector parameter):

$$s_j^t = \mathrm{v}^{\mathrm{T}}\tanh(W_v^P v_j^P + W_v^{\tilde{P}} v_t^P)$$
$$a_i^t = \exp(s_i^t)/\Sigma_{j=1}^n \exp(s_j^t)$$
$$c_t = \Sigma_{i=1}^n a_i^t v_i^P$$

We finally pass the concatenation of $v^P$ with scores $c$ into a "gate", which is defined as:

$$g = \mathrm{sigmoid}(W_g[v^P, c])$$
$$g = g * [v^P, c],$$

which gives us our final output of the self-attention layer $g$.

**Hyperparameter tuning**: Our approach consists of picking the model with character-level embeddings (and BiDAF attention only) and varying the hyperparameters of learning rate and dropout rate. We decided to focus on just 1 model (instead of trying with a few of our models) for straight-forward comparison, so as to clearly identify the impact of varying a hyperparameter. We run a dozen experiements increasing/decreasing the learning rate and dropout rate separately. We also try annealing the learning rate, as to manually decrease it once the training loss stabilizes.

# 4 Experiments

## 4.1 Data

We use the SQuAD 2.0 dataset which contains ≈150k total questions, with $\approx \frac{1}{2}$ of the questions unanswerable. The dataset consists of (context, question, answer) triplets, to which we will separate into training and dev sets. We made use of the start code which helped us preprocess the data. We use this dataset to build a model that performs the question answering task—answering a question by determining the start and end position of the answer (a span of text) from the context paragraph provided.

## 4.2 Evaluation method

We use the automatic evaluation metric that is standard for machine comprehension / question-answering tasks: F1 score and Exact Match (EM). Combining F1 and EM provides a reasonable metric to evaluate the performance of our model. We also consider the metrics of NLL (negative log-likelihood loss) and AvNA (Answer vs. No Answer).
To understand the evaluation of one of our specific models, we compare our scores to the baseline's, but also to the scores obtained by our other models. Indeed, to evaluate the effectiveness of the techniques we implement, we integrate aspects of an ablation study to present different scores with/without a particular technique technique. This provides great insights into which of the techniques are most successful.

## 4.3 Experimental details

We ran all of our models on NC6Promo and NC6sv3 Azure machines, resulting in training times of $\approx$ 15+ hours and $\approx$ 6 hours, respectively. Each completed experiment ran for thirty epochs. For our self-attention models, our batch size was 32, as our VMs were consistently running out of memory, and 64 for the rest of the models that we tested. As we had conducted a series of hyperparameter tuning experiments, our learning rates differed model to model (refer to Section 4.4: Results for more details), but we ended up using the default values of a learning rate of 0.5, dropout probability of 0.2, kernel size of 5 (for character embeddings), hidden size of 100, character limit of 16, paragraph limit of 400, question limit of 50, and character dimensions of 64.

## 4.4 Results

After implementing each of our models following our approach detailed above, we evaluated their performance on the validation set. We summarize our quantitative results in the following table,

where each row presents a particular model's metrics. The color-coding indicates in blue models that applied techniques to the embedding later, and in green models that applied attention techniques.

| Model | F1 | EM | NLL | AvNA |
|---|---|---|---|---|
| Baseline | 59.69 | 56.23 | 3.13 | 66.71 |
| Character-Level Embeddings | 63.19 | 59.81 | 2.93 | 69.79 |
| Character-Level Embeddings + POS | 63.44 | 60.14 | 2.82 | 69.84 |
| Character-Level Embeddings + Lemma | 62.13 | 58.78 | 2.81 | 68.98 |
| Character-Level Embeddings + POS + Lemma | 61.86 | 58.73 | 2.85 | 68.91 |
| Character-Level Embeddings + KQV Self-Attention for context | 61.49 | 58.09 | 3.20 | 68.32 |
| Character-Level Embeddings + KQV Self-Attention for context & query | 63.46 | 60.11 | 2.95 | 69.72 |
| **Character-Level Embeddings + Self-Matching Attention (RNet)** | **65.27** | **61.77** | **2.77** | **71.79** |
| Character-Level Embeddings + Coattention | 61.39 | 57.57 | 3.10 | 68.46 |
| Coattention (without char-level embeddings) | 59.23 | 55.44 | 3.20 | 66.48 |

Table 1: Model performance results on the validation set

Here are the F1 and EM scores we obtained on the test leaderboard for our best model:

| **Character-Level Embeddings + Self-Attention (RNet)** | F1 | EM | NLL | AvNA |
|---|---|---|---|---|
| Baseline (Dev) | 59.69 | 56.23 | 3.13 | 66.71 |
| Dev Set | **65.27** | **61.77** | 2.77 | 71.79 |
| Test Set | **63.99** | **60.14** | | |

Table 2: Test leaderboard results

We also ran a series of hyperparameter tuning experiments, where we varied learning rate and dropout rate, summarized in following table:

| Model | LR | Dropout | F1 | EM | NLL | AvNA |
|---|---|---|---|---|---|---|
| Baseline | 0.5 | 0.2 | 59.69 | 56.23 | 3.13 | 66.71 |
| 0-Character-Level Embeddings | 0.5 | 0.2 | 63.19 | 59.81 | 2.93 | 69.79 |
| 1-Character-Level Embeddings | 0.9 | 0.3 | 60.74 | 57.67 | 2.74 | 67.23 |
| 2-Character-Level Embeddings | 0.7 | 0.3 | 61.34 | 58.14 | 2.77 | 67.40 |
| 3-Character-Level Embeddings (stopped half-way through) | 0.05 | 0.2 | 52.19 | 52.19 | 5.83 | 52.14 |
| 4-Character-Level Embeddings | 0.3 | 0.2 | 62.28 | 58.90 | 2.99 | 69.15 |
| 5a-Character-Level Embeddings: LR annealing | 0.5, then 0.3 (at 1.5mil iterat$^n$) | 0.2 | 61.62 | 58.14 | 3.30 | 68.56 |
| 5b-Character-Level Embeddings: LR annealing | 0.5, then 0.3 (at 3 mil it) | 0.2 | 62.74 | 59.33 | 3.16 | 69.70 |
| 6- Character-Level Embeddings | 0.2 | 0.2 | 60.84 | 57.07 | 3.31 | 68.19 |

Table 3: Hyperparameter tuning results

**Results analysis for character-level embeddings and input features**: Our results reveal that character-level embeddings produce a drastic improvement on EM/F1 scores, producing an approximate 5.7% increase in F1 and 6.4% increase in EM scores from the baseline. However, we were rather disappointed with results for input features; while POS produced a slight increase in F1/EM to 63.44/60.14, the integration of lemma features reduced scores to 62.13/58.78, and the combined POS + lemmas produced a rather unimpressive 61.86/58.73 (see Appendix for more details). We think that this may be an indication that we are "over-engineering" our model, and that it might be best for the model to extract its own patterns in an "end-to-end" manner.

**Results analysis for attention models**: Our table shows results for three types of attention: coattention, key/query/value (KQV) self-attention, and RNet self-attention. Each of these attention techniques had significantly different results. Coattention performed worse than we expected, with lower results than with BiDAF attention (see Appendix for more details). However, the other two types of attention performed quite well. To better visualize the differences, displayed below is a graph plotting RNet self attention in pink/gray , KQV self attention (applied to context + query) in blue, and KQV self attention (applied to context only) in orange :

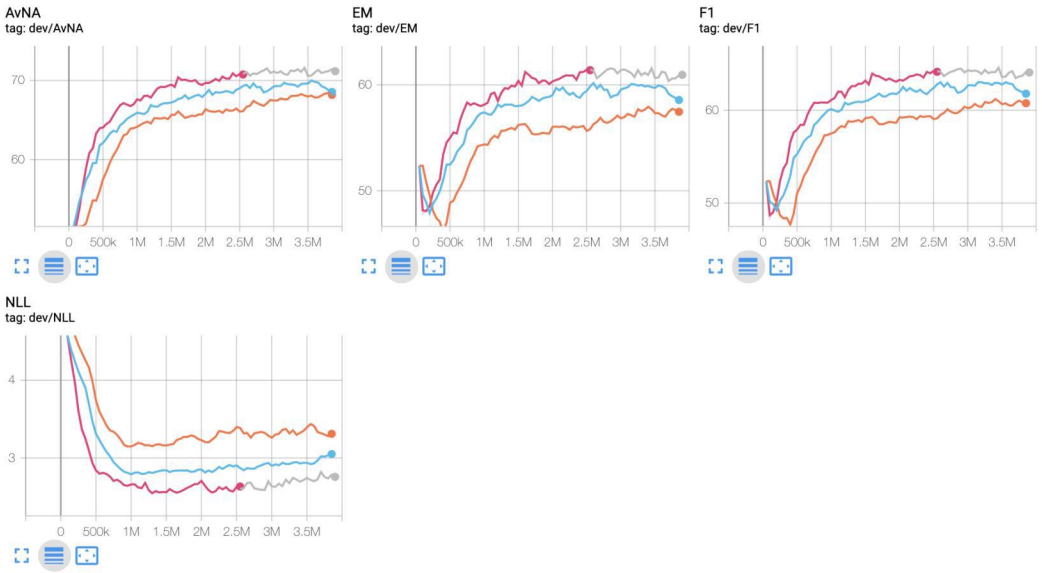


Figure 1: Graph of R-Net self attention vs. KQV self attention (context and query) vs. KQV self attention (context only)

Examining the two KQV experiments, we notice that KQV self-attention performed much better when applied to both context and query (blue) than just context (orange), with a difference of nearly 2 percentage points for the F1 score. This highlights that attending to just one of the inputs is not enough and that it possibly even worsens results as perhaps the embeddings of context/query are no longer very comparable or scaled the same way if only one is passed through the KQV self-attention layer.

Finally, we can compare the results from KQV self-attention (blue, both applied to context + question) to the self-attention from the Rnet paper (pink/gray). We saw a large improvement with the RNet self-attention, obtaining an F1 score of 65.27. The F1 score for RNet self-attention is almost 2 percentage points greater than the score for KQV self-attention. KQV may have not performed as well as expected, mainly because when transformers use this technique, they have dozens of layers, whereas we only have 1. We believe the strong performance of RNet self-attention is due to the attention layer's increased level of nuance in obtaining question-aware representation for the passage, which could have helped the model gain a better contextual understanding of the passage which led to better results.

**Results analysis for hyperparameter tuning**: The results we obtained in our dozen of hyperparameter tuning experiments were rather disappointing. We ran these experiments on the model with just character-level embeddings for easy comparison. We tried independently increasing or decreasing the

learning rate and the dropout rate, but none of our results improved upon the default parameters for the character-level embeddings model. This appears to suggest that the default parameters provided in the starter code (learning rate of 0.5, dropout of 0.2) were very ideal hyperparameters for our task and dataset, and that varying them did not lead to immediate improvements. It was interesting to perform the experiments of learning rate annealing, where we started the run at LR of 0.5 but then decreased the LR when the training loss started to stabilize. With more time, we would have experimented with different types of annealing (cosine, linear, etc).

## 5 Analysis

In this section, we do an initial quantitative analysis but then spend time qualitatively evaluating our best model (which incorporates character-level embeddings and RNet self-attention).

It is valuable to break down the different kinds of questions. Examining the validation evaluation set, we obtain the following percentages of questions by type ("how", "what", etc):

"How": 9.7%, "What": 57.7%, "Why": 1.5%, "Which": 4.5%, "Who": 10.6%, "When": 9.1%,
"Where": 4.4%, other: 6.7%,

We then calculated the F1 scores of our model on the dev set, broken down by question type. We obtain the following F1 scores, in decreasing order:

"Which": 78,
"When": 72.6,
"Who": 65.5,
"What": 64.7,
"Where": 64.3,
"How": 62.5,
other: 62.5,
"Why": 61.1,

The break down is quite interesting. We can notice that our model performs the best on "which" questions, obtaining the highest F1 score of 78. This makes sense, as in a way, "Which" questions often contain a lot of information that gives the answer away. For example, in the question "Which character is known for being small and having blue skin?" there's a lot of information about the characteristics of that character (small, blue skin) that the model could easily find in the passage and use to correctly predict the answer.
Our models also performs extremely well on the "When" questions, obtaining a very high F1 score of 72.6. This suggests that our model successfully captures temporal dimensions. This is also perhaps because many of the "When" questions have the answer follow the pattern "in..." (like "in 1999", or "in May"), a pattern that could be easily learned by the model.
On the other hand, our models performs poorly on the "Why" questions, obtaining a low F1 of 61.1. This actually isn't that surprising, since "Why" questions often have a more complex answer that requires a much deeper level of understanding of the passage and question. With more data available, it would be interesting to increase the number of "Why" questions in our training set so our model can hopefully better grasp a true semantic understanding of the passage and question.

To further qualitatively understand our system (e.g. when it succeeds and when it fails), we will look at a few interesting examples of correct/incorrect predictions.

**Correct predictions**:

- **Question:** How many people were at a Harvard sponsored regatta in 1875?
- **Context:** Harvard's 2,400 professors, lecturers, and instructors instruct 7,200 undergraduates and 14,000 graduate students. The school color is crimson, which is also the name of the Harvard sports teams and the daily newspaper, The Harvard Crimson. The color was unofficially adopted (in preference to magenta) by an 1875 vote of the student body, although the association with some form of red can be traced back to 1858, when Charles William Eliot, a young graduate student who would later become Harvard's 21st and longest-serving president (1869–1909), bought red bandanas for his crew so they could more easily be distinguished by spectators at a regatta.
- **Answer:** N/A
- **Prediction:** N/A

This correct prediction is interesting because it demonstrates that the model isn't just trying to find

words from the question in the passage and select a substring that is near it. Indeed, although the word "regatta" is present in the passage, the model understands the semantic difference between the type of answer the question is expecting ("how many", which is a quantity), and the type of answer that the passage seems to suggest ("who", as in the crew and spectators who were at the regatta). The understanding of the mismatch between these two types enables the model to correctly predict N/A.

**Incorrect predictions**:

- **Question:** What can the exhaust steam not fully do when the exhaust event is insufficiently long?
- **Context:** The simplest valve gears give events of fixed length during the engine cycle and often make the engine rotate in only one direction. Most however have a reversing mechanism which additionally can provide means for saving steam as speed and momentum are gained by gradually "shortening the cutoff" or rather, shortening the admission event; this in turn proportionately lengthens the expansion period. However, as one and the same valve usually controls both steam flows, a short cutoff at admission adversely affects the exhaust and compression periods which should ideally always be kept fairly constant; if the exhaust event is too brief, the totality of the exhaust steam cannot evacuate the cylinder, choking it and giving excessive compression ("kick back").[citation needed]
- **Answer:** evacuate the cylinder
- **Prediction:** N/A

This is a very interesting incorrect prediction. While the question contains "insufficiently long", the answer in the passage is preceded by "too brief". It seems like the model did not realize that "insufficiently long" and "too brief" actually mean the same thing (just flipping antonyms around), leading to the incorrect prediction of N/A. If the model could better encode the similarity of phrases that contain different words that together build up the same meaning, we believe this would lead to better results in these types of scenarios.

- **Question:** Who was Kaidu's grandfather?
- **Context:** Instability troubled the early years of Kublai Khan's reign. Ogedei's grandson Kaidu refused to submit to Kublai and threatened the western frontier of Kublai's domain. The hostile but weakened Song dynasty remained an obstacle in the south. Kublai secured the northeast border in 1259 by installing the hostage prince Wonjong as the ruler of Korea, making it a Mongol tributary state. Kublai was also threatened by domestic unrest. Li Tan, the son-in-law of a powerful official, instigated a revolt against Mongol rule in 1262. After successfully suppressing the revolt, Kublai curbed the influence of the Han Chinese advisers in his court. He feared that his dependence on Chinese officials left him vulnerable to future revolts and defections to the Song.
- **Answer:** Ogedei
- **Prediction:** Kublai

This is a relatively simple example that the model failed to succeed on. The answer lies in the phrase "Ogedai's grandson Kaidu": any human can immediately tell that Ogedai is the grandfather of Kaidu. But the model did not pick up on the semantics of "'s" and did not understand how to relate "grandfather" from the question to "grandson" from the passage. If the model's embeddings for "grandson" and "grandfather" had greater similarity, the model would be more likely to process the question-aware passage representation and predict the correct answer for this type of scenario.

- **Question:** Abilene was a prime investor in what project?
- **Context:** Internet2 is a not-for-profit United States computer networking consortium led by members from the research and education communities, industry, and government. The Internet2 community, in partnership with Qwest, built the first Internet2 Network, called Abilene, in 1998 and was a prime investor in the National LambdaRail (NLR) project. In 2006, Internet2 announced a partnership with Level 3 Communications to launch a brand new nationwide network, boosting its capacity from 10 Gbit/s to 100 Gbit/s. In October, 2007, Internet2 officially retired Abilene and now refers to its new, higher capacity network as the Internet2 Network.
- **Answer:** N/A
- **Prediction:** National LambdaRail

This is a complex example, which could probably actually fool some humans. The subtlety lies in grammatical structures and subject/verb linking. The model seems to focus on local, smaller chunks of the sentence and loses the larger picture. Zooming out, one can see that the subject of "was a prime investor" is "the internet2 community", and not "Abilene" as in the question. The model was also probably thrown off by the length of the sentence, broken up into chunks separated by commas.

## 6 Conclusion

In this paper, we evaluated the results of experimenting with embeddings, with regards to character-level embeddings and POS/lemma input features, as well as exploring unique neural attention mechanisms (i.e. co-attention, key/query/value self-attention, and R-net self-attention). Our highest-performing model ended up being the configuration of character-level embeddings with R-net self-attention, producing F1/EM scores of 63.99/60.14 on the leaderboard test set (and scores of 65.27/61.77 on the leaderboard dev set). We also conducted hyperparameter tuning experiments, and found that the best-yielding results were the default parameters provided in the starter code (LR = 0.5, dropout = 0.2).

We were rather pleased with our high EM/F1 scores, producing increases of nearly 6% from the baseline for both EM/F1. Moreover, we were satisfied with the sheer number of models we implemented (6 models + a dozen hypertuning experiments), exploring various attention techniques and achieving good performance on the SQuAD 2.0 data set. Our implementation of different types of attention cemented our understanding that "Attention is *pretty much* all you need" (a reference to [1]), although results with attention aren't as performant with fewer layers and model complexity. We also learned that different embedding techniques may produce varied results; character-level embeddings resulted in immense improvements to our model, but POS/lemmatization of words, for the most part, did not.

Additionally, we were able to reflect over the tradeoffs between feature engineering (as we had done with the POS/lemmatization input features) and end-to-end learning. Based on our results, we concluded that feature engineering may enable the machine to target specific features and extract patterns with human intervention, resulting in improved pattern recognition. However, it may be dangerous to "over-engineer"; features that one may believe to be optimal may end up not being particularly useful for the model, and in turn may end up hurting results. Additionally, human-chosen input features may result in correlated/redundant features, which may be damaging as the system may erroneously pick up on correlations between features that are unimportant details. End-to-end learning allows for the natural extraction of features, but oftentimes such features may be rather trivial and insignificant.

One limitation of our study was that we did not get the opportunity to tune hyperparameters for our most performant model (due to time constraints); instead, we solely tuned our character-level embedding model. It would be interesting to explore as to whether our results hold true (that the default learning rate/dropout probability parameters are most ideal) when we implement different attention mechanisms/input features. Another limitation is that we run KQV self-attention only after the embedding layer; however, results may have panned out differently if we were to run vanilla self-attention in other locations (such as after the encoding layer).

Future courses of action include what is listed above, as well as experimenting with a greater variety of input features (such as term frequency and named entity recognition).

## References


[1] Ashish Vaswani. Attention is all you need. In *NeurIPS*, 2017.

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. In *ICLR*, 2017.

[3] Microsoft Research Asia Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. In *ACL*, 2017.

[4] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[5] 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. In *ICLR*, 2018.

[6] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[7] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *ACL*, 2017.

[8] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirz. Bidirectional attention flow for machine comprehension. In *ICLR*, 2017.

[9] CS224N Assignment 5 code. `http://web.stanford.edu/class/cs224n/assignments/a5.pdf`.

[10] Existing R-net code. `https://github.com/tailerr/R-NET-pytorch`.

# A Appendix

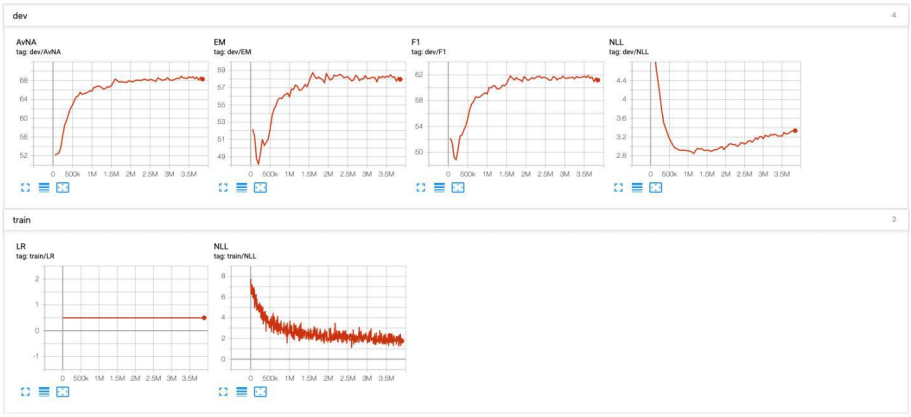


Figure 2: Graph of input features with POS and lemmatized words

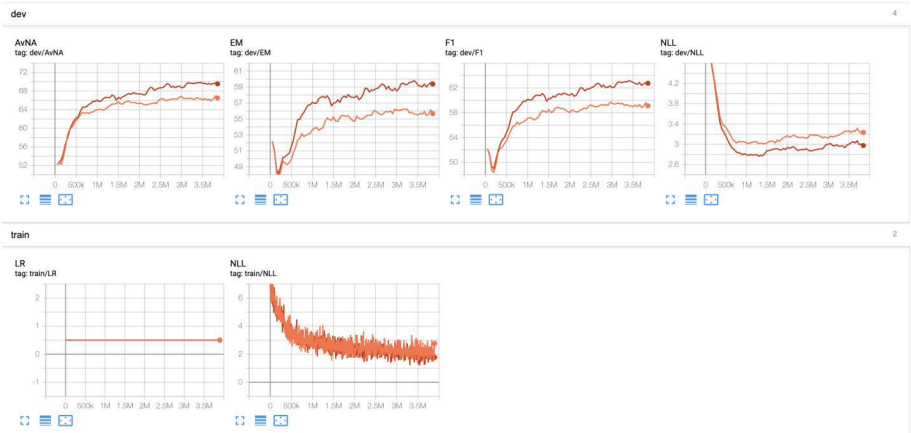


Figure 3: Graph of baseline (orange) vs. character-level embeddings (red)

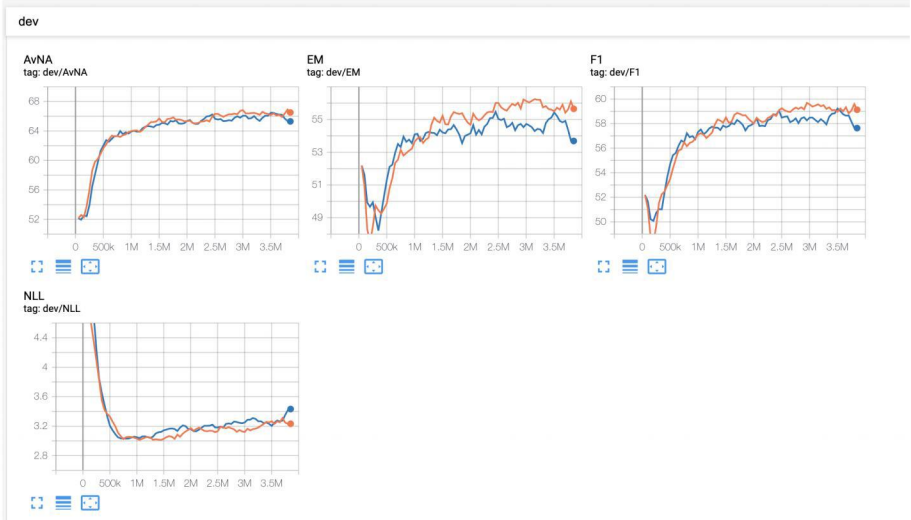


Figure 4: Graph of baseline (orange) vs. coattention (blue)

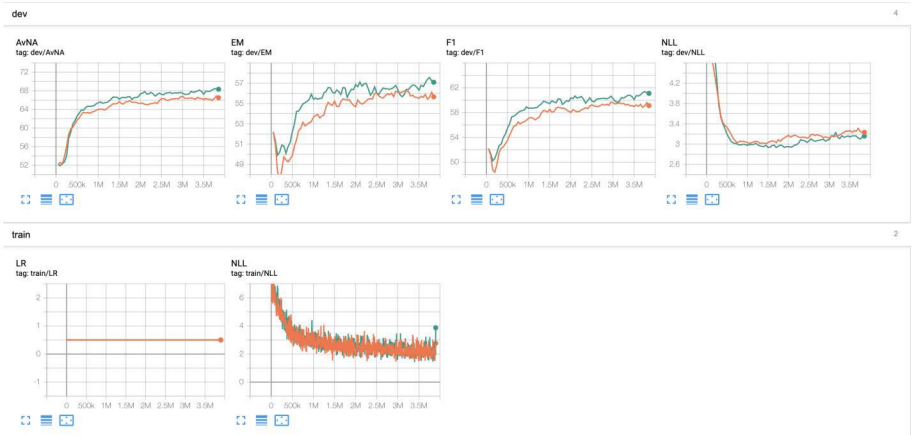


Figure 5: Graph of baseline (orange) vs. character-level embeddings + coattention (green)

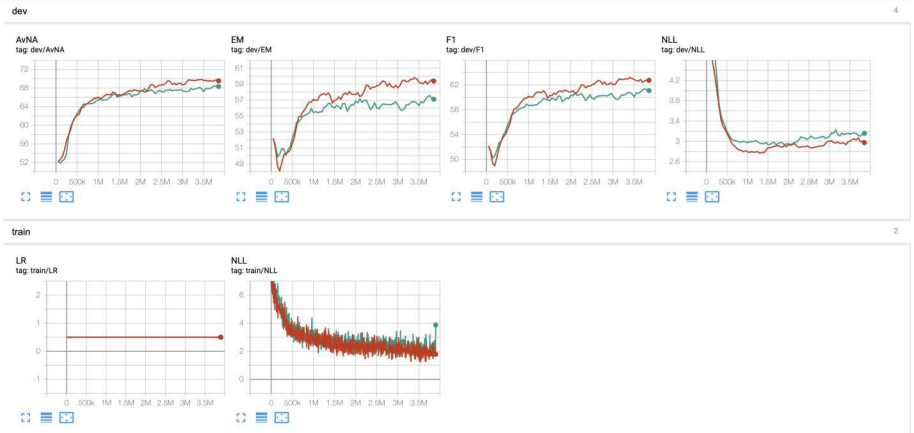


Figure 6: Graph of character-level embeddings (red) vs. character-level embeddings + coattention (green)