SQuAD 2.0: Improving Performance with Optimization and Feature Engineering

Stanford CS224N Default Project - IID SQuAD

Robert Schmidt
rschm@stanford.edu

Brian Powell
bpowell4@stanford.edu

Abstract

Our goal was to build a QA system that performs well on the SQuAD 2.0 dataset [1] and improves on a baseline inspired by the Bi-Directional Attention Flow (BiDAF) model [2]; however, we were not able to leverage pretrained models to accomplish this task. We hypothesized that providing our model a more nuanced view of the data through hand-engineered features would refine its ability to generalize to both answerable and non-answerable questions. To this end, we took inspiration from the DrQA document reader [3] and augmented the BiDAF architecture to incorporate character-level word embeddings and tagged input features. Our hypothesis proved correct, as we achieved top-five test set performance (F1=$\mathbf{68.753}$, EM=$\mathbf{65.714}$) through a detailed hyperparameter search and by ensembling our best models.

1 Introduction

Question answering is a widely studied NLP task with clear human benefits, as we often seek efficient and accurate answers to complex, vague, and possibly unanswerable questions. Some QA systems, such as Chen et al.'s DrQA [3], focus on the open-domain setting, in which the model endeavors to answer a query given a large knowledge base such as Wikipedia. Our task is simplified: given a question and an associated Wikipedia paragraph, our system must either find the span of text that best answers the question, or else conclude that the provided passage does not contain the correct answer. The second edition of the Stanford Question Answering Dataset (SQuAD) is a standard benchmark in this setting, as it includes both answerable and unanswerable questions.

While many of the top performers on SQuAD 2.0 [4] are based on BERT [5] and its descendants, we have eschewed pretrained Transformer models in favor of constructing a model that can extract as much information as possible from the context and question input. In addition to building character-level embeddings to bring the provided Bi-Directional Attention Flow (BiDAF)-inspired baseline in line with the official [2] model, we also incorporate incorporate part-of-speech (POS), named entity recognition (NER), and exact match features for both the context and question words in the vein of the DrQA document reader. In particular, we hypothesize that the benefits of DrQA's feature engineering will be able to generalize even to a different underlying network architecture, and propose that feature engineering can be highly effective in the absence of pretraining.

2 Related Work

BiDAF. Seo et al [2] introduce the Bi-Directional Attention Flow mechanism. By combining Context-to-Query (C2Q) and Query-to-Context (Q2C) attention, the model is able to identify which context words are most relevant to the question, as well as which question words are most relevant to the context. Our exact match features have a similar goal; they indicate if a context (resp. question) word can be found in the question (resp. context). The BiDAF network begins with Character, Word, and Contextual Embedding Layers, which retrieve and refine the pretrained GLoVe embeddings of both the context and question words. These hidden states are then passed through the Attention Flow and Modeling Layers, the latter of which is a two-layer bidirectional LSTM. The Output Layer

produces probability distributions for the start and end token indices of the span. Our baseline is a simplified version of the BiDAF model, omitting the Character Embedding Layer.

DrQA. The Document Reader in Chen et al.'s DrQA system [3] provides a framework for searching relevant documents for the span that best answers the given question. By incorporating hand-engineered features for the context paragraphs, the authors were able to achieve top performance (as of 2017) on many notable QA tasks, including SQuAD 1.0. Instead of focusing solely on the context tokens, we also generate POS, NER, and exact match features for the question words.

Ensembling. The top entries on the SQuAD 2.0 leaderboard [4] all exploit ensembling, the practice of amalgamating several model outputs to improve performance. Huang et al.'s FusionNet [6], once a state-of-the art model on SQuAD, used majority vote as their ensembling technique. This simple method chooses the most common span among model outputs. Instead of breaking ties at random, we opted to favor the model with the higher F1 score.

3 Approach

Of particular importance to the BiDAF architecture is its **Word Embedding Layer**, which creates a refined embedding for each word in the input by projecting each GloVe embedding via a learned linear transformation. In particular, we highlight the projection operation of this layer. Letting $\boldsymbol{W}_{\text{proj}} \in \mathbb{R}^{H \times D}$ be a learnable matrix of parameters with hidden size H and letting $\boldsymbol{v}_i \in \mathbb{R}^D$ represent the i^{th} input word embedding, the projection operation is given by:

$$\boldsymbol{h}_i^{\text{word}} = \boldsymbol{W}_{\text{proj}} \boldsymbol{v}_i \in \mathbb{R}^H$$

In the baseline model, these refined word embeddings $\boldsymbol{h}_i^{\text{word}}$ are then fed into the Highway Network and supplied to the later modeling layers. However, we propose adding as much information as possible into the Highway Network's final input vector in order to free up training resources from automatically developing features that would otherwise be learned through pretraining. Hence, we move the Highway operation into its own **Highway Layer**, and augment the word embeddings with character-level word embeddings and hand-engineered tagged features.

3.1 Character-level CNN

First, taking cues from the original BiDAF model, we employ GloVe character embeddings to construct "character-level word embeddings" via a 1D Convolutional Neural Network (CNN) [7] consisting of a `CONV + ReLU + MaxPool` operation. More specifically, we slide multiple convolutional windows of a provided kernel size over the characters of each word to condense each of the 64-dimensional character embeddings into one d-dimensional embedding for the entire word. We adopt the same parameters as in the original BiDAF model, using a kernel size of 5 and $d = H$ (from the **Word Embedding** layer), yielding final character-level word embeddings $\boldsymbol{h}_i^{\text{char}} \in \mathbb{R}^H$. We constructed this layer in `PyTorch` by adapting the BiDAF authors' `TensorFlow` implementation [2].

3.2 Feature Engineering

In their DrQA paper, Chen et al. [3] showed that feature engineering can greatly improve performance on question-answering tasks. In a similar vein, we add the following inputs for each context paragraph:

1. **Exact Match**: Three binary features $\alpha_1, \alpha_2, \alpha_3$, where (1) $\alpha_{1,i} = 1$ if context word i matches a question word exactly; (2) $\alpha_{2,i} = 1$ if context word i matches a question word in their uncased forms; (3) $\alpha_{3,i} = 1$ if context word i matches a question word in their lemma forms. Note that the lemma form of a word is its "base form"; for example, $\{am, are, is\}$ are all mapped to be [8]. This increases the likelihood that a context word finds a match in the question.
2. **Token Features**: Part-of-speech (POS) and named entity recognition (NER) tags for each context word, which are converted to indices using dictionaries. These are optionally one-hot encoded, so the vectors are $\boldsymbol{v}_i^{\text{NER}} \in \mathbb{R}^{K_{\text{NER}}}$ and $\boldsymbol{v}_i^{\text{POS}} \in \mathbb{R}^{K_{\text{POS}}}$, where K_{NER} is either 1 (appending the raw index) or 21 (one-hot encoding with 21 categories), and similarly $K_{\text{POS}} \in \{1, 52\}$.

While DrQA's Document Reader only incorporated metadata for the context paragraphs, we also generate exact match and token features for each of the question tokens. Thus, each context and question word has additional input features of the form:

$$(1)\ \boldsymbol{v}_i^{\text{NER}} \in \mathbb{R}^{K_{\text{NER}}} \quad (2)\ \boldsymbol{v}_i^{\text{POS}} \in \mathbb{R}^{K_{\text{POS}}} \quad (3)\ \alpha_{1,i} \in \{0,1\} \quad (4)\ \alpha_{2,i} \in \{0,1\} \quad (5)\ \alpha_{3,i} \in \{0,1\}$$

3.3 Final Modeling Considerations

If all features are present for a word, we construct vector $\boldsymbol{x}_i$ as the final input to the **Highway Layer**:

- *Refined word and character vector*: $\boldsymbol{h}_i = [\boldsymbol{h}_i^{\text{word}};\ \boldsymbol{h}_i^{\text{char}}] \in \mathbb{R}^{2H}$
- *Token vector*: $\boldsymbol{t}_i = [\boldsymbol{v}_i^{\text{NER}};\ \boldsymbol{v}_i^{\text{POS}}] \in \mathbb{R}^{K_{\text{NER}}+K_{\text{POS}}}$
- *Exact match vector*: $\boldsymbol{e}_i = [\alpha_{1,i};\ \alpha_{2,i};\ \alpha_{3,i}] \in \{0,1\}^3$
- *Final vector*: $\boldsymbol{x}_i = [\boldsymbol{h}_i;\ \boldsymbol{t}_i,\ \boldsymbol{e}_i] \in \mathbb{R}^{2H+K_{\text{POS}}+K_{\text{NER}}+3}$

At this point we have constructed a highly modular system: any number of components to the vector $\boldsymbol{x}_i$ can be swapped in or out. Furthermore, we consider that we can construct features at the context level only (as was done in DrQA [3]) or also append tagged features to the question input. To this end, we built a highly customizable modeling script with the following options:

- The input types (word embeddings, character embeddings, tagged tokens, and exact match features) can be used in any combination separately for the context input and question input.
- The POS or NER tokens can either be appended directly or one-hot encoded and appended to the final feature vector $\boldsymbol{f}_i = [\boldsymbol{t}_i;\ \boldsymbol{e}_i]$.
- In the feature vector construction step, we can project the feature vector $\boldsymbol{f}_i$ as in the **Word Embedding** layer, or simply provide the concatenated feature vector to $\boldsymbol{x}_i$.
- After concatenating to create $\boldsymbol{x}_i$, we can optionally project this vector before providing it to the **Highway Layer** as a means of refinement and regularization.

The embedding procedure is summarized in Figure 1: the final vector from this stage (for both context and question) is passed into the Highway Layer and fed into the remainder of the BiDAF architecture.

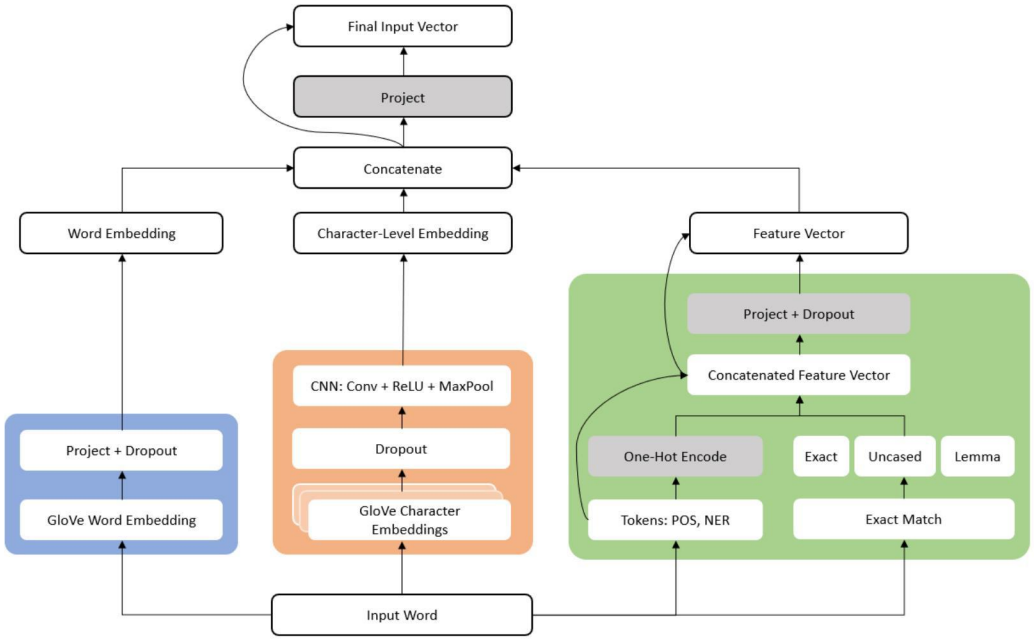

Figure 1: Encoding module for our modified BiDAF architecture (best viewed in color). Greyed-out steps with skip connections indicate an optional element.

4 Experiments

4.1 Data

We were provided with ⟨question, context, answer⟩ triples split into three sets: `train` (129,941 examples), `dev` (6078), and `test` (5915). This data came from SQuAD 2.0's training and development sets [1], plus some additional hand-labeled examples for testing. We adapted `preprocess.py` from the DrQA repository [3] with the `spaCy` [9] tokenizer to generate POS and NER tags as well as the lemma forms for each of the context and question words. The metadata output allowed us to construct the additional features mentioned in section 3.2.

4.2 Evaluation Method

To evaluate our models, we submitted to the course SQuAD leaderboard and compared both F1 and EM performance against our peers. The AvNA score was a useful barometer in determining the differences between models. We also examined performance qualitatively, as described in section 5, including an ablation study and confusion matrix analysis.

4.3 Experimental details

4.3.1 Initial Experimentation

Baseline. Our experiments began with the provided baseline BiDAF model; in particular, we wanted to explore this model as much as possible before introducing new components. We iterated over the following hyperparameter combinations, taking cues from the hyperparameter investigation performed by Britz et al. (2017) [10] for fruitful search directions:

- RNN module type: {LSTM, GRU}
- Number of layers in BiDAF **Modeling Layer** [2]: {2, 4}
- Dropout: {0.1, 0.2, 0.3}

For each run, we employed the AdaDelta [11] optimizer with learning rate 0.5, used hidden size 100, a batch size of 64, and ran for 30 epochs with manual early stopping when overfitting began to significantly impact performance. While the GRU model did reduce the training time by about two hours on our `Standard NC6_Promo` VM (the original baseline took eleven hours), it was not able to replicate the performance of the default model. In fact, the initial configuration, an LSTM model with 2 modeling layers and 0.2 dropout, proved to be quite optimized and exceeded the performance of any other hyperparameter configuration at this stage; this baseline achieved a dev set performance of 61.36 F1 and 58.19 EM, with 68.14 AvNA. Given the challenges in hyperparameter optimization during our initial experiments, we turned our efforts towards developing the components of our embedding framework to boost performance.

Char-CNN. After implementing the character-level CNN, we first ran an experiment with the default BiDAF hyperparameters on the concatenated word and character-level vector. This yielded immediate improvements on every dev score, achieving 62.82 F1, 59.32 EM, and 69.69 AvNA. Given these positive results, we were curious to see if the model would further improve by increasing the hidden size from 100 to 150 (i.e. the full concatenated vector increased in size from 200 to 300) while keeping other hyperparameters fixed. Unfortunately, this decreased performance on the dev set. Given our lack of progress in optimizing the earlier baseline over numerous configurations, we decided to move to implementing the engineered features to improve performance. In all future models, we continued to build upon the default parameter Baseline + Char-CNN model created at this step.

4.3.2 Feature Engineering Exploration

Context-Only Features. Mirroring the DrQA procedure, we began by constructing features for the context only. We kept the baseline hidden size of 100 and first appended the raw indices for the tokens in addition to the exact match features, yielding a final context vector size of 205 and question vector size of 200. However, the difference in size meant that we could no longer employ the same

Highway Layer for the context and question. We posited that creating separate context and question layers would make the question weights much less powerful (from training on fewer words). This sparked the creation of the projection layer at the concatenation step from Figure 1; by projecting the context vector back to dimension 200, we would be able to leverage the same Highway weights for both the context and question. We also theorized that this would allow the model to re-weight the varying inputs and learn importance and interactions for the different features. Running such a model with the default BiDAF hyperparameters was a resounding success; we increased each dev score by 3 to 4 points from the Baseline + Char-CNN model, yielding scores of 66.10 F1, 62.88 EM, and 72.37 AvNA (see entry (7) in Table 1).

Adding Question Features. Given the success achieved by appending the engineered features to the context, we supposed that we may be able to further boost model performance by also appending features to the question input. While token features have the same interpretation for each input type, exact match in the question implies that one of the question words is found in the context; we hoped that having exact match in both the context and question would allow the model to better map said matches. While this did not yield as large a boost as adding the context features, it did improve model performance across each score by about one point. While we did not know it yet, this run (with all features and projection performed at the concatenation step for both question and context) actually ended up being our best-performing model (see Table 1). In fact, we observed that the model had two "best" iterations: one with a top F1 score (1 - F1), and another with a top EM score (1 - EM). The best-EM version is in some ways the better model as it improves the EM score more than the best-F1 version improves the F1 score.

Model Expansion and DrQA. At this stage of experimentation, we returned to the analysis done by Chen et al. (2017) [3] to gather insight on directions for our hyperparameter search. In particular, we noted that, as opposed to our modified BiDAF model, DrQA was larger (128 hidden units and 3 modeling layers), had a smaller batch size (32), used a larger dropout (0.4), and employed the AdaMax optimization algorithm [12]. We thus ran the following configurations (with rank reported from Table 1):

1. DrQA hyperparameters, using our model's projection layer $\rightarrow$ (not top-7)
2. DrQA hyperparameters, but not using the projection layer $\rightarrow$ (not top-7)
3. Increasing the size of our model to match DrQA, but keeping all other BiDAF parameters fixed (and using projection) $\rightarrow$ (2)
4. Running our model with BiDAF parameters and no projection $\rightarrow$ (5)

While some of these configurations were successful, none of them surpassed the original run we performed with all features and our projection layer. Notably, neither run that took the DrQA parameters wholesale ended up making our list of top-7 models. This may imply that our BiDAF model is sufficiently different from the DrQA architecture for their hyperparameters to be non-optimal for our design. By far the most successful modification was increasing the size of the model, although it did not surpass the top run (though it was extremely close). It is likely that with more hyperparameter adjustments, the larger model could surpass our first run - this may serve as a direction for future study.

One-Hot Encoding Features. Thus far, we had been appending the token features without one-hot encoding to our input vector and had seen improvements to performance. However, one-hot encoding the tokens before appending would arguably be more sensible, as there is no inherent meaning in the order of the tokens. Furthermore, given that the largest token is of size 52, the decreased magnitude when shifting to one-hot encoding may improve model performance. We also supposed that, rather than our current form of projection after the concatenation of all features, it may be better to project on the features directly before concatenation (as is done in the **Word Embedding** and **Char-CNN** layers). To this end, we created a feature projection step that would take the full feature vector (size 76 when using tokens and exact match) and project it to dimension 50 with dropout. We did not experiment with using both forms of projection at once as we thought that it would entail too much regularization and tuning.

Final Model Runs. After developing the one-hot encoding and feature projection scripts, we ran the following configurations on with one-hot encoded features (with rank reported from Table 1, and "BiDAF default" denoting the default hyperparameter set):

1. BiDAF default + feature projection layer $\rightarrow$ (not top-7)
2. BiDAF default + projection after concatenation $\rightarrow$ (6)
3. BiDAF default + no projection $\rightarrow$ (3)
4. Dropout 0.4 + no projection $\rightarrow$ (4)

Model	F1	EM	AvNA
(1 - F1) All Features + Concat Projection (Best F1)	**67.00**	63.55	**73.11**
(1 - EM) All Features + Concat Projection (Best EM)	66.98	**63.72**	73.08
(2) All Features + Concat Projection (LARGE)	66.67	63.43	73.06
(3) All Features + No Projection + Token One-Hot	66.61	63.47	72.22
(4) All Features + No Projection + Token One-Hot (Dropout 0.4)	66.39	63.25	72.27
(5) All Features + No Projection	66.26	63.13	72.26
(6) All Features + Concat Projection + Token One-Hot	66.17	62.91	71.92
(7) Context-Only Features + Concat Projection	66.10	62.88	72.37

Table 1: Best dev set results in order of F1 with minimum cutoff 66. For more details on the various hyperparameter configurations, see Table 7 in the Appendix.

Contrary to our initial expectations, we saw that the one-hot encoded token runs did not surpass our best-performing model, regardless of whether we used the new feature projection layer or the concatenation projection layer. In fact, the one-hot model runs without any projection tended to perform the best. It may be that the default set of hyperparameters was better tuned to the vectors without one-hot encoding, and that further experimentation would be required to optimize for this new configuration.

4.3.3 Ensembling

Lastly, we combined selections of our best performing models using majority vote: given a set of models $M = \{m_1, m_2, \dots\}$ with output spans $S = \{s_{m_1}, s_{m_2}, \dots\}$, our ensemble chooses the most common span, i.e. the mode of S. If S has multiple modes $S^* = \{s_1^*, s_2^*, \dots\}$, the ensemble outputs the span in S^* from the model with the highest F1 score.

We created several ensembles by combining subsets of the top 7 models (by both F1 and EM) as described in Table 1, which all had F1 scores above 66. This cutoff ensured that the models included were diverse and high-performing. Since the top two iterations were checkpoints from the same model, we experimented with leaving one out to decrease the correlation between model outputs and thus hopefully improve generalization. We evaluated the ensembles by submitting to the validation leaderboard and obtaining EM and F1 scores; however, we were wary of trying too many combinations as we would likely be overfitting to the dev set. As a result, we tried to motivate our choices of subsets by examining the relative advantages of each model.

Our very first ensemble included our best four models at the time: (1 - EM), (2), (6), and (7). Adding (4), the model with one-hot encoding and higher dropout, helped diversify the selection of spans and produced our highest scoring ensemble on the validation set, with approximately 2 point improvements on the F1 and EM scores of our best single model. Our second best ensemble combined our top 7 models, opting for the version of (1) with the best EM. As we suspected, including both iterations of (1) hurt performance slightly. The best 5 ensembles are reported in Table 2.

(1 - F1)	(1 - EM)	(2)	(3)	(4)	(5)	(6)	(7)	F1	EM
	✓	✓		✓	✓		✓	**68.808**	**65.787**
	✓	✓	✓	✓	✓	✓	✓	68.755	65.703
✓	✓	✓	✓	✓	✓	✓	✓	68.577	65.670
✓		✓	✓	✓	✓	✓	✓	68.543	65.586
	✓	✓			✓		✓	68.541	65.552

Table 2: Best dev set results for ensembles in order of F1.

4.4 Results

Concluding our experiments, we made four submissions to the test leaderboard as shown in Table 3: one of the best single models (1 - EM), and three promising ensemble models given the performance boost seen in the previous section. In particular, we were interested to see which version of model (1) would yield the best final test performance.

Model	F1	EM
Top-7 Ensemble with (1 - F1)	**68.753**	**65.714**
Top-7 Ensemble with (1 - EM)	68.630	65.630
Best Dev Set Ensemble	68.333	65.275
Best Single Model (1 - EM)	66.192	62.959

Table 3: Final test set model submissions.

Our best-performing model on the test set is the Top-7 Ensemble model with (1 - F1) representing model (1) - at the time of writing, this model is ranked fourth overall on the final test leaderboard. Notably, we see that the Best Dev Set Ensemble is not the best-performing test model, lending some credence to our idea that iterating to achieve the best dev set ensemble could potentially lead to overfitting. This effect is extremely marginal, however, given that the two Top-7 Ensemble models did not drastically improve upon the Best Dev Set Ensemble scores. Overall, we are very pleased with how we balanced the bias and variance of these models, as comparing their scores reveals that the dev set results were highly predictive of their final test scores. Our iterative model-building procedure would suggest that each part of the architecture was significant to improving the final performance, and that ensembling the final model results allowed us to leverage the insights gained by the different hyperparameter configurations. However, we thought it important to further examine the components of our best models to see how much each feature contributed to the final performance.

5 Analysis

5.1 Ablation Study

In order to examine the individual effects of our modifications, we conducted an ablation analysis on the additional token (NER & POS) and exact match features for the context and question words. Here, we focus on the ablation impact to the (1 - F1) model given its status as the best individual model in our best-performing Top-7 Ensemble (as judged by test set scores). Table 4 shows that removing the 3 binary exact match inputs engendered a substantial drop in performance across all metrics. Excluding the token features led to a smaller but significant decrease in scores.

Features	F1	EM	AvNA
Full	67.00	63.55	73.11
No exact match	60.41 (-6.59)	57.44 (-6.11)	66.49 (-6.62)
No NER or POS	63.15 (-3.85)	60.21 (-3.34)	69.32 (-3.79)

Table 4: Feature ablation analysis. The "full" model is our best performing model by F1 score.

These results replicate Chen et al.'s findings [3] that exact match features are particularly crucial to performance improvements. Since our BiDAF-inspired architecture differs considerably from the

DrQA Document Reader, our results suggest that the efficacy of hand-engineered features extends beyond DrQA's model. The inclusion of metadata appears to positively influence a neural network's ability to perform reading comprehension tasks.

From the `Tensorboard` plots in Figure 2, we see that the models with exact match features (in dark red and pink) have AvNA curves that are a vertical shift up from the iterations without them (in light blue and orange). Moreover, when exact match inputs are included, the EM and F1 scores increase almost monotonically. When models exclude them, initial strong performance from predicting no-answer is followed by a downward trajectory such that they are soon surpassed by their exact match counterparts.

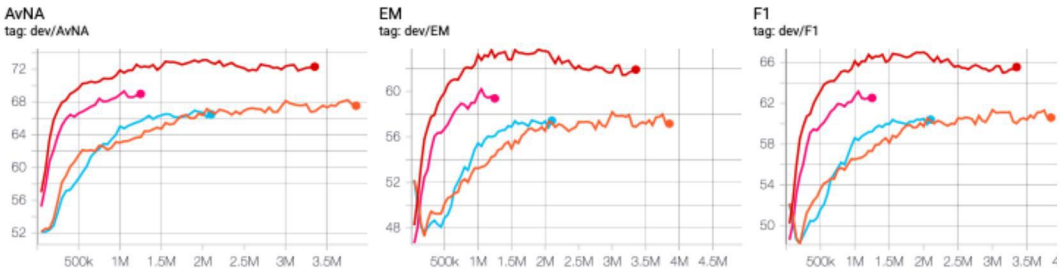

Figure 2: `Tensorboard` plots for best model (dark red), best model minus token features (pink), best model minus exact match (light blue), baseline model (orange).

5.2 AvNA and Exact Match

As suggested by the improved AvNA score, we posit that the exact match features are particularly important in determining whether or not the model believes the answer can be found in the context paragraph. Comparing the confusion matrices in Tables 5 and 6, we see that the inclusion of exact match features increases the True Negative rate by 9% and decreases the False Positive rate by 8%, while performance on questions which can be answered remains fairly stable. This shift suggests that these 3 binary inputs help the neural network identify adversarial questions, which are likely to have fewer words in common with their associated context paragraph.

Predicted/Actual	Answer	No Answer
Answer	36%	16%
No Answer	11%	37%

Table 5: Best single model by F1 score.

Predicted/Actual	Answer	No Answer
Answer	38%	24%
No Answer	10%	28%

Table 6: Best single model w/o exact match.

6 Conclusion

In this project, we significantly improved baseline performance on the SQuAD 2.0 question answering task through optimization and feature engineering. Instead of overhauling the original network architecture, we focused on extracting as much information as possible from the input data. We first constructed character-level word embeddings via a 1D Convolutional Neural Network, and then added token and exact match features for both the context and question words. We also conducted thorough hyperparameter searches and experimented with one-hot encoding, projection, and drop-out layers. Ensembling our best models by majority vote achieved validation set F1 and EM scores over 7 points higher than the baseline, with comparable performance on the test data.

We argue that it would be difficult for a model to learn meaningful metadata representations solely from the training text without pretraining. Explicitly providing token and exact match features as input ensures that the model leverages this useful information and frees up training resources for the model to better fit the dataset. Given the performance gains we realized in the paper, incorporating additional features, such as term frequency and aligned question embeddings [3], may lead to further improvements. Overall, our findings suggest that feature engineering is an effective approach to improve model performance in the absence of pretraining.