Exploring Question Answering on SQuAD
Based on Stanford's Default Project for CS224N/2020

Michael Kamfonas

July 6, 2021

Abstract

In 2020, Stanford created a project for their CS224N graduate NLP course, described in the handout [3]. The document and the associated github repository have been made publicly available. We attempted to follow a few of their non-PCE recommendations for enhancement included in section 5 of the handout, and we discuss our experiences here. The acronym PCE stands for Pre-Trained Contextual Embeddings, and refers to enhancements that provide contextual pretrained features such as ELMo, or frameworks that use finetuning to generate contextual embeddings like BERT and ALBERT.

The specific enhancements we attempted are:

- Change the RNN from LSTM to GRU
- Extend the model to use character along with word embeddings, per the original BiDAF
- Introduce a self-attention block similar to the BiDEF Attention block before the final output layer.

1 Introduction

1.1 Useful References

- The CS 224N Default Final Project Handout
- The original Default Project Github Repository referenced in the handout.
- A publicly accessible repository (squadNon-PCE) contains a parameter-driven python program that can invoke any combination of three enhancements discussed. The same repository contains branches with a copy of the original baseline BiDAF code provided by Stanford and several intermediate milestones.
- Tensorboard results of the runs attempted using this link.
- The SQuAD page and Leader-board.

1.2 Background

This is a "learning" project meant for experimentation and hands-on experience development with NLP models and tasks that go beyond toy examples. We follow the Stanford 224N Default Project from 2020, a question answering challenge, using the SQuAD data set and BiDAF-based starting code. It approaches a realistic level of complexity, while the envisioned models can still be trained on a workstation or high end laptop equipped with a decent GPU such as the Nvidia RTX 3080.

The Stanford project comes with a handout [3] which gives a detailed explanation of the starting model and suggests numerous potential enhancements along with citations of relevant papers and resources. This document is an extension of the handout, focused on a few of the non-PCE recommended enhancements. A follow-up project may address PCE enhancements in the future.

The way the default project has been setup by Stanford is to reserve a subset of the publicly available SQuAD training dataset for testing, without disclosing the true answers. The participants develop and train their solutions independently, and run their final tests on the provided test dataset producing a predictions file with answers to the test questions. These predictions are submitted and scored by an auto-grader based on the hidden true answers. This auto-scoring capability is not available to us, at least not readily. Consequently we can either base our results on the validation (dev-set) which is performed after every 50,000 steps of training and skip independent testing, or we could use the official SQuAD training data, split out a test set, and calculate our own scores. For now we opt to follow the former approach and use the results of our validation runs as our test results.

1.3 Literature Summary

The following sections briefly summarize relevant literature for the enhancements considered.

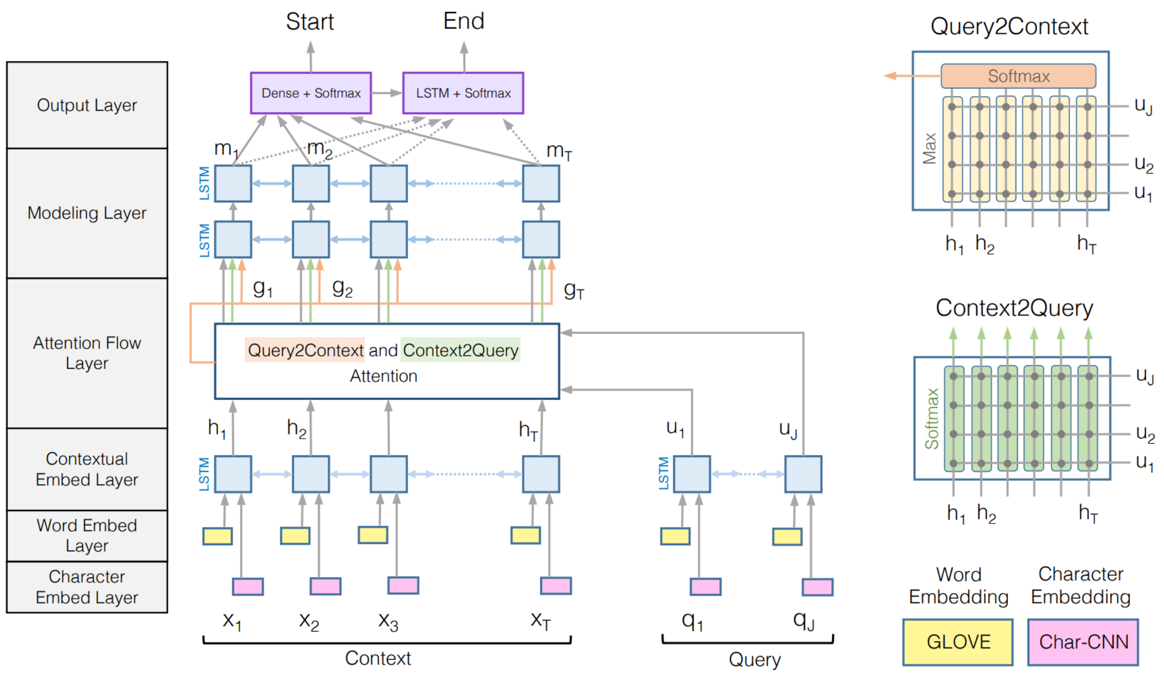

Figure 1: Model architecture from the original BiDAF paper [6]

1.3.1 Replacing LSTM RNNs with GRU

The BiDAF article originally published by Seo et al [6] uses LSTM RNNs. The GRU came as a later development, simplifying the LSTM. In an empirical evaluation [1], Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio compared the performance and training time of LSTM, GRU and simple sinh-RNN, on music and speech sequences. They demonstrated that the gated RNNs (LSTM and GRU) perform better than simple RNNs and that the performance of the GRU is very close to an LSTM, while the training time of the simpler GRU architecture can be significantly (up to 40%) shorter.

An informal popularity comparison, by searching the ArXiv library, finds 3678 hits on LSTM and 563 on GRU, a 0.15 ratio, which seems to have remained about the same regardless of the historical horizon. For example, restricting the search to the last 12 months or the first half of 2021 yields about the same ratio.

Swapping the LSTM with GRU is fairly straightforward as they have an almost identical interface.

1.3.2 Enabling Character Embeddings

Although character embeddings have been excluded from the baseline code provided for this project, the original BiDAF model [6] includes character embeddings.

Figure 1 is the BiDAF model diagram borrowed as-is from the original article. The first layer, starting from the bottom, is the Character Embedding Layer that converts character embeddings into word embedding vectors. The method used to derive these vector embeddings follows Yoon Kim's "Convolutional Neural Networks for Sentence Classification" article published in 2014 [5], whereby character embeddings for each word are passed through a one-dimensional CNN whose output channel size matches the desired word embedding size. Max-pooling over the character embedding dimension results in a fixed-size embedding vector passed to the Word Embedding Layer.

At the Word Embedding Layer character embedding vectors get concatenated with the pretrained word embeddings to form a double-wide embedding consisting of both word and character-derived information. This flows into a Highway layer, not explicitly shown in figure 1 but better delineated in figure 2, where it is labeled as Pre-Process layer. This is done both for the context passage and the query. The output of the Word embedding layer is then fed into the Contextual Embedding Layer and the rest of the BiDAF layers with no change.

1.3.3 Adding Self-Attention to BiDAF

In a progress report, the Microsoft-Asia Natural Language Computing [4] proposes R-Net, a four layer architecture that adopts both character and word embedding input. The layers of R-Net are:

1. Question and Passage Encoder similar to the

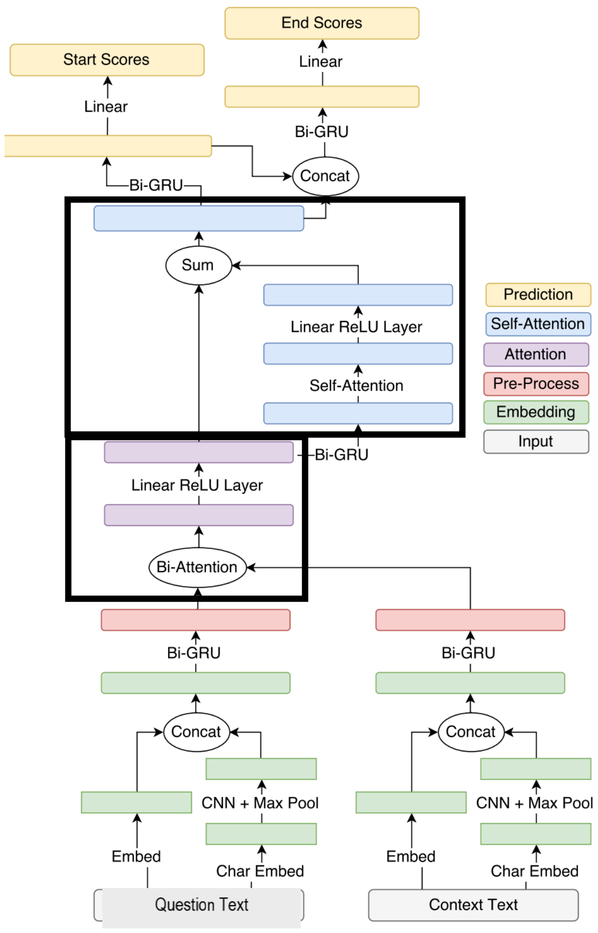

Figure 2: Model architecture by Clark and Gardner. Image is borrowed from their article [2] and annotated

BiDAF model

2. Gated Attention Recurrent Network which blends question information into the context and corresponds to the BiDAF attention and modeling layers, using GRUs instead of LSTMs .

3. A Self-Matching Attention is intended to improve context awareness for the question-enriched passage representation.

4. Output Layer is similar to the start-end pointer prediction in BiDAF.

Clark and Gardner in their article about multi-paragraph question answering [2] also propose a residual self-attention layer as shown in figure 2. The lower outlined block includes the bidirectional attention layer followed by a linear layer with ReLU activation. The output is fed to the upper block, where it is forked into two branches, one becoming the input to a self attention sequence and the other added to the self-attention output producing the "residual" self-attention. The self-attention sequence consists of the following sub-steps:

1. Bidirectional GRU receiving the output of the modeling layer ReLU.

2. Self-Attention of question-aware passage to itself

3. Linear layer with ReLU activation whose output is the output of the self-attention block.

4. Residual self-attention calculated by summing the previous step's output with the Bi-Attention output, i.e. this block's input.

1.4 Paper Structure

The rest of the document is organized into three sections: Methods, Results and Discussion. Each section is divided into subsections corresponding to the baseline and each enhancement.

2 Methods

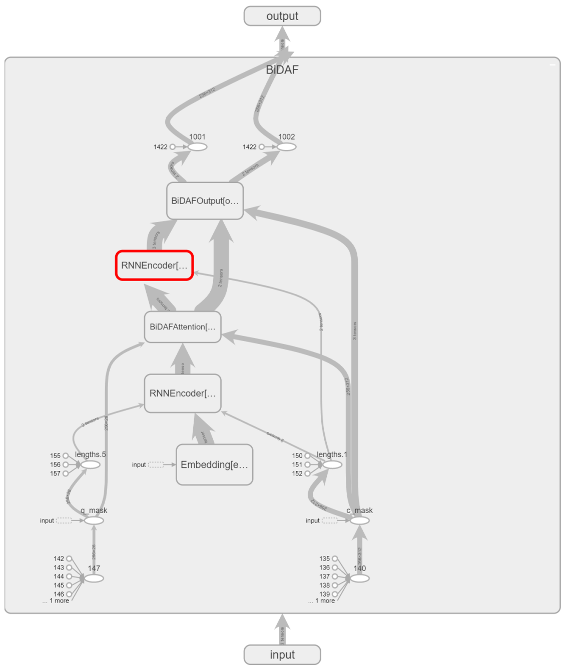

Figure 3: Top level topology of the base model. Notice that the type of RNN is not visible unless one looks into the encoder layer detail

The model architecture can be shaped by the permutation of three options producing eight distinct configurations.

- The first option introduces the ability to choose the type of RNN. Figure 3 shows the top level flow, as extracted from the Tensorboard graph.

  The RNNEncoder class, which contains the RNN, is used more than once. The first use serves as a true RNN Encoder. It is also used after each attention layer to consolidate exposed residual temporal dependencies, leveraging the inherent structural similarity with the encoder. This follows the practice used in the starter code provided for the project.

- Adding character embeddings is limited to replacing the Embedding class with the EmbeddingCE class. Figure 4 shows this replacement. In all other respects, the model is identical to the baseline.

- Adding the self-attention layer causes more substantial change to the topology as shown in figure 5.

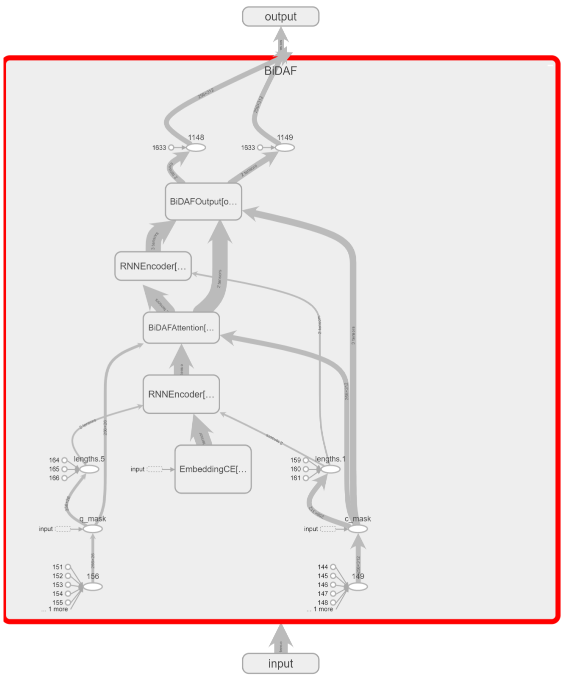

Figure 4: Topology of the basic model with added character embeddings. Notice that the original Embedding class is now replaced by EmbeddingCE.

The published Tensorboard results include graphs for each model topology where more detail is available by drilling into each sub-module.

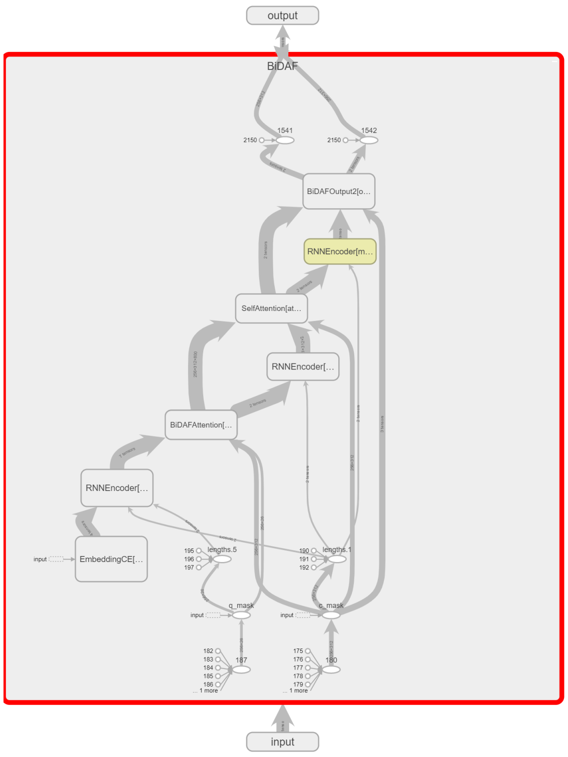

Figure 5: Topology of the model with both character embeddings and self-attention.

2.1 Character Embeddings

The baseline model uses word embeddings only. We extended the model by adding character embeddings guided by the original BiDEF architecture [6]. We take advantage of the preprocessing already included in the starter code. An added command-line argument *–char_embed* controls the inclusion or exclusion of character embeddings from the model. Figure 6 is borrowed from an assignment handout of CS229N and it depicts the detailed steps of converting characters to character embeddings and from those to a word level embeddings, using a convolutional and a highway network.

Character input for each minibatch enters the model as a tensor of character indexes to the character embedding vector lookup table shaped as $Y \in \mathcal{R}^{b \times m_s \times m_w}$ where b is the batch size, m_s the maximum number of words in a sentence of the batch, and m_w the maximum number of characters in a word. Padding symbols have already been substituted in non-character and non-word positions. These character indexes are converted to their embedding vector by a dictionary lookup, which takes place using an "Embedding" NN-module, thus adding one more dimension to

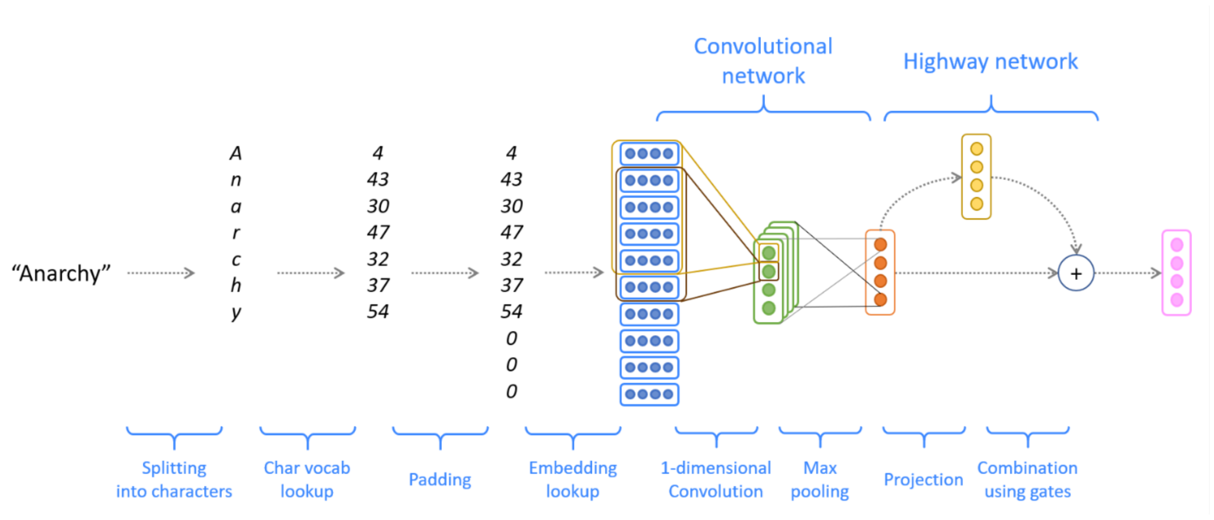

Figure 6: Character embedding steps in converting word "Anarchy" to a word embedding. Image copied from Stanford 224N - assignment 5 handout

the tensor: $Y_{\text{emb}} \in \mathcal{R}^{b \times m_s \times m_w \times e_{\text{ch}}}$ where e_{ch} is the character embedding vector length.

In practice, we introduce two reshaping operations which produce an equivalent more efficient representation:

1. Since the rest of the character embedding layer aggregates character vectors to word-level embeddings, we can temporarily string all words of all sentences in the batch into a single dimension of size bm_s. After the character embeddings get aggregated into word embeddings, then we can reshape back to the two original $b \times m_s$ before passing them on to the word embedding layer.

2. Since we intend to apply 1D-convolutions along the characters of each word but not across the embedding features, we treat the embeddings as input channels. Input channels act like color channels in a colored picture representation; they take part in the convolution summation, but the kernel window does not slide along them (here is a simple illustration.) Following the implementation conventions of LSTMs and GRUs, we permute the last two dimensions, so that the channel dimension precedes the convolution dimensions.

As a result, the tensor shape becomes:

$$Y_{\text{emb}} \in \mathcal{R}^{(bm_s) \times e_{\text{ch}} \times m_w} \tag{1}$$

The result of equation 1 is fed into a convolution network (LSTM or GRU) which is defined with e_{ch} input channels , H output channels and a 1D-convolution of size $k = 5$. Its output is shaped as:

$$Y_{\text{conv}} \in \mathcal{R}^{(bm_s) \times H \times m_w - k + 1} \tag{2}$$

The last dimension is derived from the number of convolutions of size k that fit in m_w. After applying ReLU activation and max-pooling on the last dimension we get the aggregated character embeddings that get concatenated with their corresponding word embeddings, producing the input to the highway layer.

$$[X_{\text{word\_proj}}; Y_{\text{conv\_out}}] \in \mathcal{R}^{(bm_s) \times 2H} \tag{3}$$

The representation $X_{\text{word\_proj}}$ captures the word embedding processing steps whereby the pre-trained word embeddings of size e_w are projected to the hidden dimension of size H. The concatenation causes the last dimension to be twice the hidden size.

2.2 Attention

For this enhancement, we will first review the BiDAF attention implementation and then describe how it was modified to form the self-attention layer.

2.2.1 Baseline Model Implementation

The input to the bidirectional attention layer consists of two tensors containing pairs of passage and question encodings. Both tensors have the batch as their first dimension and the encoding as the last. More specifically the two tensors are:

Context Passage Encoding: A batch of passages $C \in \mathcal{R}^{B,N,E}$ where B is the batch size, N is the size of the longest passage in the batch, and E is the encoding dimension size for each word token at the time it is fed to the attention layer. These encodings are formed by fusing sub-word and character level embeddings along with contextual information derived by the encoder. the resulting representation of the context passage c takes the form:

$$c = c_1 \dots c_N \in \mathcal{R}^E$$

Notice that the handout uses $2H$ and so our $E = 2H$. H is the globally defined hidden layer size which has to be doubled because of the bidirectional RNN in the encoding layer.

Question Encoding: A batch of questions, in the order corresponding to that of the associated passages above, $Q \in \mathcal{R}^{B,M,E}$ where M is the size of the longest question in the batch.

$$q = q_1 \dots q_M \in \mathcal{R}^E$$

A batch consists of pairs of context passages and the associated question. For each such passage-question pair, we construct a similarity matrix $S(c, q) \in \mathcal{R}^{N,M}$ [7] denoting the strength of the relationship between each word token of the context passage with each word token of the question. for the i-th passage token and j-th question token, the element $S_{i,j} \in \mathcal{R}$ is:

$$\begin{aligned} S_{i,j} &= \alpha(c_i, q_j) \\ &= w_{(S)}^T [c_i; q_j; c_i \circ q_j] \\ &= [w_c; w_q; w_{qc}]_{(S)}^T [c_i; q_j; c_i \circ q_j] \\ &= w_c^T c_i + w_q^T q_j + w_{qc}^T (c_i \circ q_j) \end{aligned}$$

This shows that the trainable weight vector $w_{(S)} \in R^{3E}$ can be decomposed into three components, the first carrying weights for the passage token encoding, the second for the question token encoding and the last for the product of the two encodings. This allows for a more memory efficient implementation where instead of concatenating all encodings and multiplying by $3E$ dimensional weights, we can sum three smaller products of E-dimensional weights.

The same similarity matrix can be used for calculating both query-to-passage attention by taking a softmax row-wise, and passage-to-query attention by taking a softmax along the columns as elaborated in the handout.

Since the fixed size lengths N and M represent the longest respective sequences of passages and questions in the batch, we use a mask to force a very low weight for the unused slots.

2.2.2 Self Attention

Adding a layer of self-attention before the output layer is based on the ideas of [2] and [9]. No implementation code was posted, as far as we know. Using the articles for guidance we implemented a residual self attention block that receives the output of the ReLU activation of the second linear modeling layer and applies the following three sequential layers:

- A bidirectional RNN which can be a GRU or an LSTM depending on the command-line option
- A self-attention layer. similar to BiDEF Attention, which evaluates the strength of the relationship between each element of the input sequence to every other of its elements To avoid each element's relationship to itself to dominate, [2] proposes that the diagonal elements of the similarity matrix be overridden by a value representing negative infinity. We chose to leave the self-attending elements to their computed value, and rely on the scaling factor $\sqrt{E}$ to dampen the weight fluctuations [8]. E is the embedding dimension size.
- A linear layer with ReLU activation delivers the self-attention block output.

Since we want a residual output from this block, we add the block input to the output of the last Relu.

3 Results

3.1 Tensorboard

Since this is an on-going project implementation details are continuously refined, more effective hyperparameters are discovered and more evaluation runs are planned and executed, a static report of results - at least at this point - is unproductive. We are continuing to build on the tensorboard logging provided with the baseline code in a few ways:

- We replaced library tensorboardx with torch.util.tensorboard
- Added code to display the graph of the model
- Added all arguments as hyperparameters recorded at the beginning of the run
- Added a record of final metrics recorded after the end of the last epoch of each run.

Results are published here so that they can be viewed via Tensorboard on the web.

3.2 20-Epoch Random Runs

This first series of 60 training/evaluation runs used randomized assignment of select hyperparameter and architecture choices. The intention is to understand which architectural choices and hyperparameters have significant influence on the F1 metric achieved. Here is a list of these select choices and their value-ranges:

Number of Epochs: 20 for all runs

Batch Size: All runs used a batch size of 32

RNN Type: Choice between GRU or LSTM

RNN Number of Layers: Choice of 1 or 2 for each of the the three RNN encoder instances used,i.e. encoding, BiDAF modeling and Self-Attention modeling.

Character Embeddings: True or False

Self-Attention: True or False

Learning Rate: Choice of values 0.5, 0.75, 1.0 or 1.2

L2 Weight Decay: Choice of values 0.0, 0.0001, 0.0005, 0.001

Drop Probability: choice of values 0.0, 0.1,0.2,0.3

The runs are published and can be viewed via Tensorboard as described earlier. A listing also appears in the appendix of this document. An observant reader may notice that a few of the entries, have drop probability, learning rate or L2 weight decay outside the range described above. This is because the ranges were initially broad, and after 20 or so runs, based on the results observed, we removed extremely bad performing runs and narrowed the ranges. Although most extreme cases were removed, a few marginal low performing runs with parameters outside the final ranges remained.

We used step-wise regression to identify what options have the most significant impact on the F1 score. The analysis is in a separate document which can be accessed here.

3.3 60 Epoch GRU Random Runs

Looking at the top 5 from the previous phase, we see that the F1 trends upward implying potentially better performance with longer training. We run a 60 epoch phase next, and focus on a narrower set of choices, GRU-only, with no Self-Attention.

Results will be updated shortly.

4 Discussion

4.1 Issues and To-Do-List

Unify to One Program: Currently the different versions reside in three different branches of a gitub repo. Desirable to consolidate into one program and let options control the variations.

Self-Attention: The Self-Attention model underperforms. Need to review the architecture, consider alternatives that show improvement or validate this result.

Fix Warning Message: Deal with the non-contiguous memory message (see this post). Here is the message:

UserWarning: RNN module weights are not part of single contiguous chunk of memory. This means they need to be compacted at every call, possibly greatly increasing memory usage. To compact weights again call flatten_parameters().

References

[1] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[2] Christopher Clark and Matt Gardner. *Simple and Effective Multi-Paragraph Reading Comprehension.* 2017. arXiv: 1710.10723 [cs.CL].

[3] Stanford CS224N. *CS 224N Default Final Project: Question Answering on SQuAD 2.0.* 2020. URL: http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf.

[4] Natural Language Computing Group. “R-NET: Machine Reading Comprehension with Self-matching Networks”. In: 2017. URL: https://www.microsoft.com/en-us/research/publication/mcr/.

[5] Yoon Kim. *Convolutional Neural Networks for Sentence Classification.* 2014. arXiv: 1408.5882 [cs.CL].

[6] Minjoon Seo et al. “Bidirectional Attention Flow for Machine Comprehension”. In: *ArXiv* abs/1611.01603 (2017).

[7] Minjoon Seo et al. *Bidirectional Attention Flow for Machine Comprehension.* 2018. arXiv: 1611.01603 [cs.CL].

[8] Ashish Vaswani et al. *Attention Is All You Need.* 2017. arXiv: 1706.03762 [cs.CL].

[9] Wenhui Wang et al. “Gated Self-Matching Networks for Reading Comprehension and Question Answering”. In: *ACL.* 2017.

Appendix

save_dir	lr	l2_wd	drop_prob	final/NLL	final/F1	final/EM	final/AvNA
GRU111-CEF-SAF-01	0.5	0.001	0.2	2.76	61.2	58	67.5
GRU111-CEF-SAF-02	0.75	0	0.4	3.21	55.3	51.7	63.5
GRU111-CEF-SAF-03	0.5	0.005	0.4	3.8	49.1	46.1	57.6
GRU111-CEF-SAF-04	0.5	0.0005	0.2	2.71	63.8	60.6	69.9
GRU111-CET-SAF-01	1.2	0.0001	0	2.62	66.2	62.9	72.2
GRU112-CET-SAF-01	0.5	0.0005	0.2	2.45	68	64.8	73.3
GRU112-CET-SAT-01	1.2	0.0005	0	2.48	65.7	62.4	71.2
GRU121-CEF-SAF-01	1	0.0005	0.2	2.72	62	58.9	67.9
GRU121-CET-SAF-01	0.75	0.005	0.2	4.15	50.6	49.8	56.1
GRU121-CET-SAT-01	0.5	0.0001	0.1	2.89	67.2	63.6	73.7
GRU122-CEF-SAT-01	1	0	0	3.15	56.8	53.7	64.4
GRU122-CET-SAF-01	1.2	0.001	0	2.62	62.9	59.9	68.3
GRU122-CET-SAT-01	1	0	0.2	3.1	57.6	53.9	65.4
GRU211-CEF-SAF-01	0.75	0.001	0	2.8	60.9	57.9	67
GRU211-CEF-SAF-02	0.75	0.001	0.2	2.86	59.7	56.5	65.7
GRU211-CET-SAF-01	0.75	0	0.2	3.02	57.7	54.6	64.5
GRU211-CET-SAF-02	1.2	0.0001	0	2.79	62.7	59.5	68.9
GRU211-CET-SAT-01	0.5	0.0001	0.2	2.63	67.2	63.8	73.3
GRU211-CET-SAT-02	0.75	0.0001	0.2	2.6	66.3	62.8	72.1
GRU211-CET-SAT-03	1	0	0.3	3.59	51.6	48.7	61.5
GRU212-CET-SAT-01	1	0.001	0.3	2.73	60.3	57.5	65.5
GRU221-CEF-SAF-01	1	0.0001	0.4	3.26	54	51.1	62.6
GRU221-CEF-SAF-02	1.2	0.001	0	2.76	60.5	57.9	66.1
GRU221-CEF-SAT-01	1	0.0005	0.2	2.77	59.6	56.7	65.2
GRU221-CET-SAF-01	0.75	0.0001	0.1	2.51	66.8	63.6	72.4
GRU222-CET-SAF-01	0.5	0	0.4	2.92	58.6	55.2	66.3
GRU222-CET-SAF-02	1.2	0.0001	0.2	2.92	59.4	56	66
GRU222-CET-SAT-01	0.5	0	0.3	2.78	62.1	58.8	68.7
GRU222-CET-SAT-02	0.75	0	0.3	3.07	57.5	54.4	65.2
LSTM111-CEF-SAT-01	0.5	0.001	0.2	2.8	59.5	56.7	65.8
LSTM111-CET-SAF-01	1.2	0.0005	0.3	2.54	64.8	61.8	70.1
LSTM111-CET-SAF-02	0.5	0.001	0.1	2.59	63.7	60.7	69.4
LSTM112-CEF-SAF-01	0.75	0.001	0.2	2.82	59.8	56.8	65.6
LSTM112-CEF-SAT-01	1	0.0001	0.2	2.76	61.6	58.6	67.5
LSTM112-CEF-SAT-02	0.5	0.001	0	2.84	61.4	58.4	67.2
LSTM112-CEF-SAT-03	0.5	0.0005	0.3	2.82	60.3	57.2	66.8
LSTM112-CET-SAT-01	0.5	0.001	0.3	2.65	61.8	58.9	67.4
LSTM121-CEF-SAT-01	0.75	0	0.4	2.91	57.9	54.4	64.7
LSTM121-CET-SAT-01	1.2	0.0005	0.1	2.53	64.4	61.5	69.7
LSTM122-CEF-SAF-01	1	0.001	0.2	2.79	59.4	56.4	64.8
LSTM122-CEF-SAT-01	0.5	0.0001	0.1	2.87	64.2	60.8	70.4
LSTM211-CEF-SAT-01	1	0	0	2.77	64.2	60.9	71.3
LSTM211-CEF-SAT-02	1.2	0.0005	0	2.65	62.5	59.8	68.1
LSTM211-CET-SAF-01	1.2	0.0001	0.2	2.65	62.5	59.5	68.4
LSTM211-CET-SAT-01	0.5	0.001	0.2	2.68	60.4	57.7	65.6
LSTM212-CEF-SAF-01	1.2	0	0.2	3.04	56.9	53.8	64
LSTM212-CET-SAF-01	0.75	0	0.3	2.93	59.7	56.5	66.8
LSTM222-CEF-SAF-01	0.75	0	0.2	2.8	60.6	57.4	67.4
LSTM222-CEF-SAT-01	1.2	0.0005	0	2.74	61.2	58.5	66.6