Comparing Model Size and Attention Layer Design Impact on Question-Answer Tasks

Stanford CS224N {Default} Project

Roshini Ravi
Department of Computer Science
Stanford University
roshini@stanford.edu

Sean Konz
Department of Computer Science
Stanford University
swkonz@stanford.edu

Abstract

In this project, we explore the use of various Neural Language Models applied to Question Answer tasks from the SQuAD dataset. We're specifically interested in exploring the transition from RNN-based models to transformer-based models. RNN Neural Language Models were dominant in language tasks for many years, but the introduction of the transformer demonstrated that the fall-backs of RNN models could be overcome by using architectures that optimize for larger, more parallelizable models. In this work, we compare the impacts of expanding model size with the impact of changing attention layer implementations using a Bi-Directional Attention Flow baseline model. We find that model size has a significantly greater impact on model performance on the SQuAD dataset, but larger models fail to improve performance on unanswerable question-answer examples.

1 Key Information to include

- Mentor: None
- External Collaborators (if you have any): None
- Sharing project: No

2 Introduction

Question Answering (QA) tasks are one of the fundamental activities for modern neural language models. Over the last five years, there have been various datasets curated and released to help researchers develop and evaluate the performance of different language modelling approaches for QA tasks. SQuAD 1.0 [1] was one of the first successful datasets published that provided a benchmark and leaderboard for comparing model architectures and training approaches. Within two years of SQuAD being released, models using various Recurrent Neural Network approaches had demonstrated near-human performance on the initial SQuAD dataset, but it was unclear whether these models would perform well on adversarial examples that intentionally distract from the question presented to the model. Jia and Liang augmented the SQuAD dataset with adversarial examples that intentionally distract model-generated answers without changing the groundtruth answer, or confusing humans [2]. They found that models trained on the SQuAD dataset perform poorly on adversarial examples at test-time, achieving an F1 score of 36% on an adversarial test set, versus an initial 75% F1 score on SQuAD. This raised the question of how much true language understanding these models were gaining, and led to the development of SQuAD 2.0 [3].

SQuAD 2.0 augmented the initial dataset by adding 50,000 unanswerable QA examples to the initial 100,000 QA examples included in SQuAD 1.0. This augmentation of the initial dataset forces models to consider how to perform well on a more diverse corpus of QA examples in order to attain a performant score. Since the introduction of SQuAD 2.0, leading models have begun to deviate from the traditional Recurrent Neural Network (RNN) models that had exceeded human performance on SQuAD 1.0 in exchange for transformer-based models that prioritize attention layers rather than recurrent, sequential layers. These transformer approaches are currently the leading models on the SQuAD 2.0 leaderboard and have begun to replace older RNN models in nearly all language tasks. The defining feature of transformer models is the replacement of recurrence and convolution layers with attention layers, achieving comparable performance on various language tasks, including on the SQuAD dataset [4] [5]. In this work, we focus on understanding the impact of this transition towards exclusively transformer-based models by implementing slight alterations on a baseline model architecture based on the BiDirectional Attention Flow (BiDAF) model used as the baseline model in the SQuAD 2.0 paper [3] as well as the impacts of various attention layers on the same BiDAF model. We find that model size adjustments yield a significantly greater impact on validation set performance as opposed to variations in attention layer design. Additionally, we conduct an evaluation of model performance on answerable and unanswerable QA examples, finding that that models with more complex attention layers are more capable of recognizing questions that are not answered by the given context as opposed to models with larger embedding layers.

3 Related Work

The ability to utilize context about a situation in answering a question is a fundamental element of language understanding. This task comes naturally for humans, we're able to extract important information from a given contextualizing paragraph, and apply that context to a question prompt. An important aspect of this activity is our ability to apply judgement on the contextualizing paragraph to extract what we deem to be important details to retain. This extraction exercise is akin to language model attention layers that now dominate most neural language model architectures.

For several years, language models relied heavily on RNN models utilizing Long-Short Term Memory (LSTM) unit architectures to process language sequences for most language tasks. These models work by using numerous LSTM cells processing sequences of data and remembering a variable percentage of each element it views in the sequences [6]. Though intuitive, the sequential nature of these models makes parallelization during training challenging, which becomes problematic when expanding model sizes or when processing examples with long sequences. In contrast, Transformers solely rely on the use of attention layers, which eliminate the need for considering sequences and allow for modelling of dependencies without regard for their distance in the input or output sequences [7]. Seo et al formally introduced the Transformer architecture, which utilized an encoding and decoding layer derived from attention mechanisms that had been applied as an additional layer in many sequential models previously [4]. By eliminating the sequential elements of the model architecture and thus increasing the parts of the model that can be run in parallel, the model size for transformer architectures can potentially be larger than traditional RNN models. We focus on the attention and size aspects of transformers in our work in order to augment a BiDirectional Attention Flow (BiDAF) model, presented in [3], to improve its performance on the QA tasks using the SQuAD dataset.

The SQuAD 2.0 dataset was released by Rajpurkar et al in 2018, and replaced the SQuAD 1.1 dataset as a standard QA task training and evaluation dataset [3]. SQuAD 2.0 consists of 100,000 traditional QA examples extracted from Wikipedia articles, and 50,000 QA examples that do not contain answers, but have been curated to match the format and style of the answerable questions. Along with SQuAD, the authors utilized a version of BiDAF to evaluate the dataset and to provide a baseline reference for other models to compare against. We use this model as a starting point in this work.

The BiDAF architecture is a multi-stage, hierarchical model that utilizes context at various levels of granularity as well as attention flow between encoder and decoder layers to obtain a query-aware context representation on QA tasks [8]. BiDAF computes attention scores in order to generate a similarity matrix for question-context inputs. In its initial evaluation on SQuAD, Rajpurkar et al. find that BiDAF achieved an average exact match score (EM) of 59.2 and an F1 score of 62.1, significantly lower than those scores achieved by the same model on SQuAD 1.1 (EM of 68.0, F1 of 77.3), but still providing a competitive baseline for the dataset. We expand upon the BiDAF model and our alterations to the architecture in part 4.

4 Approach

As previously mentioned, we use the BiDAF model as a reference architecture to evaluate the impact on improving model performance on the SQuAD 2.0 dataset using model size changes versus changing attention layer formats. We expand upon the baseline BiDAF model by implementing a character embedding layer, co-attention layer [9], and a self-attention layer [10].

4.1 Bi-Directional

Our baseline model follows a variant of the BiDAF architecture proposed in [11]. This model uses the same word embedding layer approach, but omits the character level embeddings, which we include as a means of testing the impact of model size on performance on SQuAD. A description of our baseline model architecture follows.

4.1.1 Embedding Layer

The embedding layer takes the word indices of the context and question to produce embeddings $\boldsymbol{c}_1, ..., \boldsymbol{c}_N \in \mathbb{R}^D$ and $\boldsymbol{q}_1, ..., \boldsymbol{q}_N \in \mathbb{R}^D$ respectively. Each embedding is projected to have dimensionality H to produce hidden vectors $\boldsymbol{h}$ and then transformed twice by a highway network.

For a hidden vector, $\boldsymbol{h}_i$, a one-layer highway network will compute

$$\boldsymbol{g} = \sigma(\boldsymbol{W}_g \boldsymbol{h}_i + \boldsymbol{b}_g) \in \mathbb{R}^H$$
$$\boldsymbol{t} = \text{ReLU}(\boldsymbol{W}_t \boldsymbol{h}_i + \boldsymbol{b}_t) \in \mathbb{R}^H$$
$$\boldsymbol{h}'_i = \boldsymbol{g} \odot \boldsymbol{t} + (1 - \boldsymbol{g}) \odot \boldsymbol{h}_i \in \mathbb{R}^H$$

where $\boldsymbol{W}_g, \boldsymbol{W}_t \in \mathbb{R}^H$ and $\boldsymbol{b}_g, \boldsymbol{b}_t \in \mathbb{R}^H$ are learnable parameters.

4.2 Encoding Layer

The Encoding layer takes the output of the embedding layer and applies a bidirectional LSTM to generate temporal dependencies between timesteps of the embedding layer output. The output are the hidden states at each position. For a given timestep i, the full output is the vector concatenation of the forward and backward hidden states, $\boldsymbol{h'}_i \in \mathbb{R}^{2H}$.

4.3 Attention Layer

The fundamental element of the BiDAF baseline model, and the layer that we focus our work on is the bidirectional attention flow layer which enabless attention to flow from the context to the question, as well as from the question to the context.

First we computer the the similarity matrix $\boldsymbol{S} \in \mathbb{R}^{NxM}$ where for a given $(\boldsymbol{c}_i, \boldsymbol{q}_j)$

$$\boldsymbol{S}_{i,j} = \boldsymbol{w}_{sim}^T[\boldsymbol{c}_i; \boldsymbol{q}_j; \boldsymbol{c}_i \odot \boldsymbol{q}_j] \in \mathbb{R}$$

We determine Context-to-Question (C2Q) attention as follows

$$\hat{\boldsymbol{S}}_{i,:} = \text{softmax}(\boldsymbol{S}_{i,:}) \in \mathbb{R}^M$$
$$\boldsymbol{a}_i = \sum_{j=1}^{M} \hat{\boldsymbol{S}}_{i,j} \boldsymbol{q}_j \in \mathbb{R}^{2H}$$

and Question-to-Context (Q2C) attention as follows,

$$\hat{\hat{\boldsymbol{S}}}_{:,j} = \text{softmax}(\hat{\boldsymbol{S}}_{:,j}) \in \mathbb{R}^N$$
$$\boldsymbol{S'} = \hat{\boldsymbol{S}} \hat{\hat{\boldsymbol{S}}}^\top \in \mathbb{R}^{N \times N}$$
$$\boldsymbol{b}_i = \sum_{j=1}^{N} \boldsymbol{S'}_{i,j} \boldsymbol{c}_j \in \mathbb{R}^{2H}$$

The final output of the bidirectional attention flow layer is

$$\boldsymbol{g}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \odot \boldsymbol{a}_i; \boldsymbol{c}_i \odot \boldsymbol{b}_i] \in \mathbb{R}^{8H}$$

4.4 Modelling Layers

A two layer bidirectional LSTM is applied to the inputs $\boldsymbol{g}_i \in \mathbb{R}^{8H}$ to integrate temporal information between context representations conditioned on the question. The full output of this layer is the vector concatenation of the forward and backward states, $\boldsymbol{m}_i \in \mathbb{R}^{2H}$.

4.5 Output Layers

The output layer generates a vector of probabilities corresponding to the start and end positions in the context. It applies a bidirectional LSTM to the outputs from the modelling layer $\boldsymbol{m}_i \in \mathbb{R}^{2H}$ to give $\boldsymbol{m'}_i \in \mathbb{R}^{2H}$ by concatenating the forward and backward states. Then,

$$\boldsymbol{p}_{\text{start}} = \text{softmax}(\boldsymbol{W}_{\text{start}}[\boldsymbol{G}; \boldsymbol{M}])$$
$$\boldsymbol{p}_{\text{end}} = \text{softmax}(\boldsymbol{W}_{\text{end}}[\boldsymbol{G}; \boldsymbol{M'}])$$

where $\boldsymbol{W}_{\text{start}}$ and $\boldsymbol{W}_{\text{end}}$ are learnable parameters, $\boldsymbol{G} \in \mathbb{R}^{8H \times N}$ is the matrix with columns $\boldsymbol{g}_i, ..., \boldsymbol{g}_N$ from the bidirectional attention flow layer and $\boldsymbol{M}, \boldsymbol{M'} \in \mathbb{R}^{2H \times N}$ be matrices with columns $\boldsymbol{m}_i, ..., \boldsymbol{m}_N$ and $\boldsymbol{m'}_i, ..., \boldsymbol{m'}_N$.

Notice that we are again using the output of the attention layer here in the final span prediction. The attention layer is a core element of the BiDAF model, and we intend to explore this dependency further in our experiments.

4.6 Character Embeddings

Our baseline model uses only pre-trained lookup word embeddings, as compared to the original model presented in [11] which also utilizes character embeddings. Character level work embeddings are used in order to expand the capacity for the model to condition at a character level within words, rather than solely conditioning directly on word embeddings [11]. They allow us to condition on the internal structure of words and better handle out-of-vocabulary words. We implement character embeddings and use the embedding layer size to tune our model size.

4.7 Co-Attention

The first alternative attention layer we experiment with is the Co-Attention layer proposed in [9]. Similar to the simple bi-directional attention implemented in [11], co-attention utilizes a second-level attention representation to further extract context from the presented question and context paragraph. First, a linear layer with tanh nonlinearity is applied to the question hidden states $\boldsymbol{q_1}, ..., \boldsymbol{q_M} \in \mathbb{R}^l$ to obtain $\boldsymbol{q'_1}, ..., \boldsymbol{q'_M} \in \mathbb{R}^l$ projected hidden states.

$$\boldsymbol{q'_j} = \tanh(\boldsymbol{W}\boldsymbol{q_j} + \boldsymbol{b}) \in \mathbb{R}^l$$
$$\forall j \in 1, ..., M$$

Sentinel vectors $c_\emptyset, q_\emptyset \in \mathbb{R}^l$ which are trainable parameters of the model and facilitate not attending the provided hidden states are incorporated to the question and context states to give $\boldsymbol{q'_1}, ..., \boldsymbol{q'_M}, \boldsymbol{q_\emptyset} \in \mathbb{R}^l$ and $\boldsymbol{c_1}, ..., \boldsymbol{c_N}, \boldsymbol{c_\emptyset} \in \mathbb{R}^l$. The affinity matrix with scores $\boldsymbol{L}_{ij}$ for every pair $(\boldsymbol{c}_i, \boldsymbol{q'}_j)$

$$\boldsymbol{L}_{ij} = \boldsymbol{c}_i^\top \boldsymbol{q'}_j \in \mathbb{R}$$

The affinity matrix is then used to determine attention outputs for both Context-to-Question (C2Q) Attention and Question-to-Context(Q2C) Attention. C2Q attention outputs are calculated as

$$\alpha_i = \text{softmax}(\boldsymbol{L}_{i,:}) \in \mathbb{R}^{M+1}$$
$$\boldsymbol{a_i} = \sum_{j=1}^{M+1} \alpha_j^i \boldsymbol{q'_j} \in \mathbb{R}^l$$

and for Q2C attention,

$$\beta_j = \text{softmax}(\boldsymbol{L}_{:,j}) \in \mathbb{R}^{N+1}$$
$$\boldsymbol{b_j} = \sum_{i=1}^{N+1} \beta_j^i \boldsymbol{c_i} \in \mathbb{R}^l$$

Second-level attention outputs are calculated by using C2Q attention distributions to take weighted sums of the Q2C attention outputs.

$$\boldsymbol{s_i} = \sum_{j=1}^{M+1} \alpha_j^i \boldsymbol{b_j} \in \mathbb{R}^l$$
$$\forall i \in 1, ..., N$$

The concatenated sequence of $\boldsymbol{s}_i$, $\boldsymbol{a}_i$ are run through a bidirectional LSTM and the results $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_N\}$ are the coattention encoding to be fed into the modelling layer or self-attention layer.

4.8 Self-Attention

A self-attention layer was incorporated so that the inputs were able to interact with each other, i.e. every hidden state is able to attend to all the hidden states including itself. Self-attention by matching question-aware passage representations against themselves collects evidence from the whole passage for words in passage and encodes the relevant evidence of the passage word of interest and its respective question [10].

We calculate the attention scores of each word representation $\boldsymbol{g}_i$ with all the other word representations $\boldsymbol{g}_j$ to determine the final weighted sum representation of every word.

First we compute a matrix $\dot{\boldsymbol{S}} \in \mathbb{R}^{N \times N}$ where for a given context words i, j in a passage,

$$\dot{\boldsymbol{S}}_{i,j} = v^\top \tanh(\boldsymbol{W}_m^P \boldsymbol{g_j} + \boldsymbol{W}_m^{\tilde{P}} \boldsymbol{g_i}) \in \mathbb{R}$$

Next, we take the softmax of every column of $\dot{\boldsymbol{S}}$ to give $\ddot{\boldsymbol{S}}$ and use it to take weighted sums of the attention outputs from the bidirectional attention flow layer $\boldsymbol{g_i}$

$$\ddot{\boldsymbol{S}}_{i,:} = \text{softmax}(\dot{S_{i:}}) \in \mathbb{R}^N$$
$$\boldsymbol{a}_i = \sum_{j=1}^{N} \ddot{\boldsymbol{S}}_j \boldsymbol{g}_j \in \mathbb{R}^{8H}$$

where $v^\top$, W_m^P, and $W_m^{\tilde{P}}$ are trainable weights.

We also include a gate function to reduce the contribution of information with less significance.

$$g = (\boldsymbol{W}_g[\boldsymbol{a}; \boldsymbol{g}])$$
$$a^* = g \odot [\boldsymbol{a}; \boldsymbol{g}]$$

5 Experiments

5.1 Data

The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset that contains around 150k (context, question, answer) triples where the contexts are excerpts of Wikipedia articles, the question is the the query to be answered given the context, and the answer is either the subsequence of text from the context that appropriately responds to the query or indicates the question is unanswerable. The goal of the model is to determine the answer associated with the context-query pair in the SQuAD database.

5.2 Evaluation method

Both Exact Match (EM) scoring, and F1 scoring is used to evaluate the model. Exact match is a strict evaluation of whether the system output matches the exact expected output for each example. F1 score is a less strict evaluation metric that uses a mean of precision and recall that matches each word on the expected output result (i.e. if some words match, the resulting F1 score will be higher than if no words match).

5.3 Experimental details

Our experimentation was limited to tuning the hyperparameters we chose to train our models with. We wrote a bash script to cycle through dropout probabilites (ranging from 10% - 40% in 10% increments), hidden layer size (ranging from 70 - 130 in increments of 10), learning rate (ranging from 0.1 to 0.8 in increments of 0.1), and learning rate decay regularization (ranging from 0.9 to 0.99 in increments of 0.01). We randomly sample 10 sets of values for these parameters from the specified ranges, and train each model for 3 epochs with the specified parameters. We then select the models with the lowest loss at the end of that 3 epoch training period and train the model with the selected hyperparameters for the full training duration. Our baseline model uses all the default hyperparameters provided in the starter code. Each of our experimental models use the following hyperparameters:

Char Embed + Basic Attention: dropout: 30%, hidden: 110, lr: 0.6, lr decay: 0.93
Char Embed + Co-Attention: dropout: 10%, hidden: 100, lr: 0.5, lr decay: 0.98
Char Embed + Self-Attention: dropout: 20%, hidden: 50, lr: 0.7, lr decay: 0.99

5.4 Results

We present the results for our experimental models on the dev set below:

Dev Set EM / F1 Scores		
Model type	Exact Match	F1
BiDAF (Baseline)	57.23	60.97
Char Embed + Basic Attention	61.37	64.57
Char Embed + Co-Attention	61.82	65.27
Char Embed + Self-Attention	52.08	53.14

From our models, it's clear that the addition of character embeddings in our model yielded the most impactful accuracy increase on our validation set. By adding sizable (applying a 1D convolution with 256 output channels) character embedding layers to the baseline model, we increased our EM score by 4.14%, in contrast to our addition of Co-Attention, which only yielded a performance increase of 4.59%.

Since we were working on the default final project for the IID track, we chose to submit our highest performing model to the Test Set leaderboard. We submitted our Character Embedding + Co-Attention model to the leaderboard since that model had the highest EM and F1 scores on our validation set and saw a test set performance for the model of 61.82.% exact match score, and an F1 score of 64.39%. As expected, our performance on the test set is slightly worse than that of our performance on the validation set since we use the validation set to track our performance throughout training.

The most striking reaction to these results is generally the poor performance of our model implementing self-attention. The self-attention model performs significantly worse than the baseline model, despite implementing character embeddings at the same layer size used in both the character embeddings model and the character embeddings + co-attention model. Successful execution of our implementation on the virtual machine necessitated a 50% decrease in the size of the hidden layer and it is likely that this modification impacted the accuracy of our model, though not necessarily to the extent of the decrease seen. We attribute this result to a potential bug in our self-attention implementation but its plausible that this result could also be because of a numerical overflow when computing our gradients for the self-attention layer. We chose to include the results in order to point out the impacts of a faulty attention layer in these models. The model still performs reasonably well (better than a random guess), but it becomes clear from these results that a faulty attention implementation can yield dramatically worse results for this model architecture.

The poor results from our self-attention model, provides an interesting data point to consider when analyzing the impacts of different architecture elements on model accuracy performance. The addition of character embeddings seems to yield a significant performance increase, but the elimination of a proper attention layer yields a dramatic drop in model performance that insinuates that attention may hold a greater importance on model performance than was initially supported by the properly functioning models. This becomes an important consideration when evaluating which layer improvements may yield the greatest performance depending on the hardware platform used for running a model.

6 Analysis

We reviewed 30 sample outputs from the dev set evaluation for each of our models in order to better understand what kind of examples performed better for each model. Of the 30 examples, we randomly chose 15 that were correctly predicted, and 15 that were incorrect. From our analysis, we found that our character embedding model seemed to perform better on context examples that were answerable, whereas the co-attention model seemed to perform better on examples that were unanswerable. Additionally, the self-attention model, seemed to only perform reasonably well on example inputs that were answerable, suggesting that the faulty attention layer could be distracting from recognizing that a context example did not contain an answer. These points require significant more analysis in order to reach a more nuanced explanation, but from a general perspective these observations were made about our models.

In general, we find that our models are more successful on examples that contain shorter context paragraphs and longer, more descriptive questions. This intuitively makes sense since a shorter context paragraph would provide fewer variations in where the answer might lie, and a more specific question paragraph offers more opportunities for the model to trigger off of a specific word.

7 Conclusion

Overall, we present three models building off of a baseline BiDirectional Attention Flow architecture and demonstrate that models increasing their effective learnable parameter size appear to have the greatest impact on increasing the performance of the model on the SQuAD 2.0 dataset. Our highest performing model utilized both character embeddings with a layer size of 256 in addition to implementing a co-attention layer to replace the original bi-directional attention layer. We struggled to refine our implementation of self-attention and thus the performance of our self-attention model was disappointing. Our models in this work do not fully represent the various neural language models that can be applied to this task, and a reasonable next-step would be to expand on our exploration of attention in our model by implementing a transformer-based model for SQuAD 2.0 and compare the learnable parameter count for the transformer to that of the BiDAF model to further estimate the impact of model size on performance on SQuAD.

References

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

[2] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *CoRR*, abs/1707.07328, 2017.

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

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[5] Chris Chute. Attention is all squad needs. https://github.com/chrischute/squad-transformer.

[6] Sepp Hochrieter and Jurgen Schmidhuber. Long short-term memory. 1997.

[7] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. *CoRR*, abs/1702.00887, 2017.

[8] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. *CoRR*, abs/1706.04115, 2017.

[9] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[10] Microsoft Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. 2017.

[11] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.