Re-implementation of Bidirectional Attention Flow For Machine Comprehension

Suchandra Chakraborty
Researcher

Abstract

In this paper, I reimplement a multi-stage hierarchical network, Bi-Directional Attention Flow (BiDAF), for the task of machine comprehension (MC), specifically question answering (QA) task. The network represents the context at multiple levels of granularity. Unlike traditional attention mechanisms that summarize the context into a fixed-sized vector, BiDAF employs a bi-directional attention flow mechanism to achieve a query-aware context representation without early summarization. This approach mitigates information loss. The model achieves the highest Exact Match (EM) score of 52.5 and the highest F1 score of 65.7 on the Stanford Question Answering Dataset (SQuAD) validation set. On the test set it achieves an EM of 44.6 and an F1 score of 59.8.

1 Introduction

Machine comprehension (MC) and question answering (QA) have made substantial progress recently mainly due to the development of neural attention mechanisms. MC involves training systems to read and understand text to accurately answer questions, a key challenge in creating advanced AI. These systems with neural attention mechanisms examine context paragraphs to pinpoint and retrieve information relevant to the questions (Weston et al., 2015).

I reimplement the Bi-Directional Attention Flow (BiDAF) (Seo et al., 2018) network, which improves upon earlier attention models through several key innovations:

1) The attention mechanism in BiDAF avoids early summarization into a fixed-size vector. Instead, it computes attention continuously at each time step, preserving detailed information and preventing information loss. 2) Attention is iteratively computed through time as in (Bahdanau et al., 2016), each step of attention calculation depends only on the current query and context, avoiding reliance on previous attention results. This reduces error accumulation. 3) BiDAF uses query-to-context and context-to-query attention mechanisms, enhancing the richness of the information used for comprehension. BiDAF performs well on the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016). I have provided a comprehensive performance table demonstrating BiDAF's performance compared to traditional MC models.

2 Problem Statement

The machine comprehension task involves answering questions based on a given context passage. Given a *question*, the primary goal here is to accurately predict the *start* and *end* indices of the answer span within the *context*.

3 Related Work

Previous works on end-to-end machine comprehension have largely utilized attention mechanisms in three distinct ways:

Dynamic Attention Mechanisms: Inspired by (Bahdanau et al., 2016), this approach updates attention weights dynamically, considering the query, context, and previous attention states. (Hermann et al., 2015) demonstrated that this dynamic model outperforms the use of a single fixed query vector for attending to context words, specifically on the CNN and DailyMail datasets.

Static Attention Mechanisms: This method computes attention weights once and feeds them into an output layer for the final prediction (Kadlec et al., 2016). The Attention-over-Attention model by (Cui et al., 2017), utilizes a 2D similarity matrix between query and context words to compute a weighted average of query-to-context attention.

Multi-hop Attention Mechanisms: These are variants of Memory Networks (Weston et al., 2015) and involve repeatedly computing an attention vector between the query and the context through multiple layers. This method, often referred to as multi-

hop attention (Sordoni et al., 2016), (Dhingra et al., 2017)

4 Model Description

The model [1] consists of the following seven layers

4.1 Word Embedding Layer

In the word embedding layer all the tokenized words from context and query are mapped to a high-dimensional vector space. These mapped vectors contain semantic and syntactic information about the words they represent. I used pretrained 100 dimensional GloVe embeddings (Pennington et al., 2014) to get the vector representation of the words. The output of this layer is two matrices, $\mathbf{X} \in \mathbb{R}^{d1 \times T}$ for the context and $\mathbf{Q} \in \mathbb{R}^{d1 \times J}$ for the queries, where $d1$ is the GloVe embedding dimension, T and J are the context and query lengths respectively.

4.2 Character Embedding Layer

Similar to the word embedding layer, the character embedding layer also assigns each word to a high-dimensional vector space. Convolutional Neural Networks (CNNs) (Kim, 2014) are used to derive character-level embeddings for each word in context and query. This layer produces two matrices, one $d2 \times T$ matrix for the context and one $d2 \times J$ matrix for the query. The dimensions correspond to the number of words (T for context, J for query). The height $d2$ of these matrices is determined by the number of convolutional filters used in the CNN.

4.3 Highway Network

The highway network (Srivastava et al., 2015) is a type of feed-forward neural network architecture designed to simplify the training of deep networks. It introduces gating mechanisms that facilitate the learning of certain paths through the network. It serves as a shortcut for information to bypass some layers. The matrices from the word and character embedding layers are vertically concatenated to create two matrices: one for the context and another for the query. These two matrices are then passed through a 2-layer highway network. This results in two matrices, $\mathbf{X} \in \mathbb{R}^{d \times T}$ denotes the matrix for the context and $\mathbf{Q} \in \mathbb{R}^{d \times J}$ represents the matrix for the query, where d is $d1 + d2$, T is the number of words in the context paragraph and J is the number of words in the query.

[1] Code Link

4.4 Contextual Embedding Layer

In this layer, Long Short-Term Memory Networks (LSTMs) (Hochreiter and Schmidhuber, 1997) are used to capture temporal dependencies among words, using embeddings from previous layers. LSTMs are applied in both forward and backward directions to model these interactions comprehensively. The output from the highway network serves as input to this layer. The results from the forward and backward LSTMs are combined: $\mathbf{H} \in \mathbb{R}^{2d \times T}$ for context word vectors $\mathbf{X}$ and $\mathbf{U} \in \mathbb{R}^{2d \times J}$ for query word vectors $\mathbf{Q}$. Each column vector in $\mathbf{H}$ and $\mathbf{U}$ is 2d-dimensional, reflecting the combination of the outputs from both LSTM directions, each originally d-dimensional.

4.5 Attention Flow Layer

The inputs to this layer are the contextual vector representations of the context $\mathbf{H}$ and the query $\mathbf{U}$. It outputs query-aware vector representations of the context words $\mathbf{G}$, along with the contextual embeddings from the previous layer. The main goal of this layer is to fuse together information from context $\mathbf{H}$ and query $\mathbf{U}$. The first step in the attention layer is to calculate the share similarity matrix $\mathbf{S} \in \mathbb{R}^{T \times J}$ where T is number of words in context and J is number of words in the query. The element $\mathbf{S}_{tj}$ represents the similarity between the t-th word in the context and the j-th word in the query. The similarity matrix is computed using the following equation:

$$\mathbf{S}_{tj} = \alpha(\mathbf{H}_{:t}, \mathbf{U}_{:j}) \in \mathbb{R}$$

In this equation, α is a trainable function that measures the similarity between two input vectors. Here, $\mathbf{H}_{:t}$ refers to the t-th column vector of $\mathbf{H}$, and $\mathbf{U}_{:j}$ refers to the j-th column vector of $\mathbf{U}$. $\alpha(h, u)$ is defined as:

$$\alpha(h, u) = w_{(\mathbf{S})}^{\top}[h; u; h \circ u]$$

where $w_{(S)} \in \mathbb{R}^{6d}$ is a trainable weight vector, $\circ$ denotes element-wise multiplication, $[;]$ represents vector concatenation across rows, and implicit multiplication is matrix multiplication. The similarity matrix $\mathbf{S}$ is then used to derive the attentions and attended vectors in both directions.

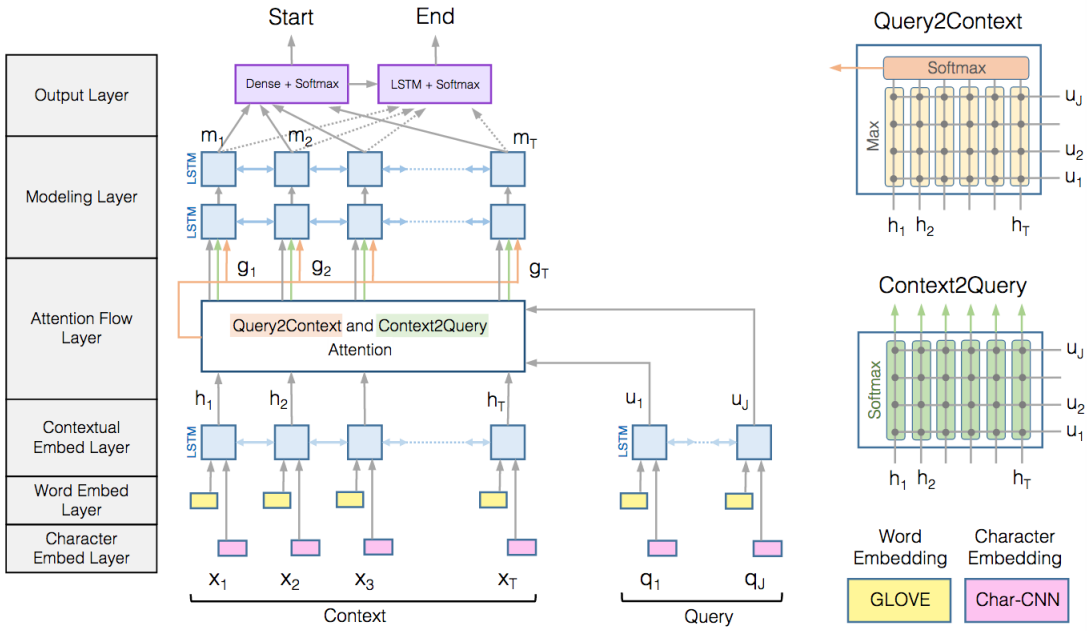

Figure 1: Architecture of BiDAF

4.5.1 Context-to-Query attention (C2Q)

The goal of this step is to find which query words are most relevant to each context words. Let $\mathbf{a}_t \in \mathbb{R}^J$ denote the attention weights assigned to the query words by the t-th context word. The attention weight is computed as $\mathbf{a}_t = \text{softmax}(\mathbf{S}_{t:}) \in \mathbb{R}^J$, where $\mathbf{S}$ is a similarity matrix. Each attended query vector is then calculated as $\tilde{\mathbf{U}}_{:t} = \sum_j a_{tj}\mathbf{U}_{:j}$. $\tilde{\mathbf{U}}$ forms a $2d \times T$ matrix containing the attended query vectors for the entire context.

4.5.2 Query-to-Context (Q2C) Attention

The goal of this step is to find which context word is most similar or relevant to either one of the query words. These words are most importanFt for answering the given query. Let $\mathbf{b} \in \mathbb{R}^T$ denote the attention weights on context words. It is computed by taking the maximum across row of $\mathbf{S}$ and then applying softmax on the resulting column vector. Then $\mathbf{b}$ is used to calculate the weighted sum $\tilde{\mathbf{h}}$ of most important words in context with respect to the query. Equation for $\tilde{\mathbf{h}}$ is $\tilde{\mathbf{h}} = \sum_t b_t \mathbf{H}_{:t} \in \mathbb{R}^{2d}$. $\tilde{\mathbf{h}}$ is copied T times across column to create $\tilde{\mathbf{H}} \in \mathbb{R}^{2d \times T}$.

Finally, the contextual embeddings and the attention vectors are combined together to form matrix $\mathbf{G}$. $\mathbf{G}$ is defined as $\mathbf{G}_{:\mathbf{t}} = \beta(\mathbf{H}_{:\mathbf{t}}, \tilde{\mathbf{U}}_{:\mathbf{t}}, \tilde{\mathbf{H}}_{:\mathbf{t}}) \in \mathbb{R}^{d_G}$. where $\mathbf{G}_{:\mathbf{t}}$ is the t-th column vector (corresponding to t-th context word), β is a trainable vector function and d_G is the output dimension of the β function. $\beta(h, \tilde{u}, \tilde{h})$ is defined as:

$$\beta(h, \tilde{u}, \tilde{h}) = [h; \tilde{u}; h \circ \tilde{u}; h \circ \tilde{h}] \in \mathbb{R}^{8d \times T}$$

, $d_G = 8d \times T$

4.6 Modeling Layer

The modeling layer focuses on the interactions among context words conditioned on the query. The input to the modeling layer is the matrix $\mathbf{G}$, which contains query-aware representations of context words. A bi-directional LSTM with two layers, where each direction has an output size of d is used in this layer. As a result, we get a matrix $\mathbf{M} \in \mathbb{R}^{2d \times T}$. This matrix $\mathbf{M}$ is then used in the output layer to predict the answer span. Each column of $\mathbf{M}$ encapsulates contextual information about a word in relation to the entire context and the query.

4.7 Output Layer

The output layer involves predicting the start and end indices of a sub-phrase within the context that answers the query. Initially the model computes the probability distribution for the start index p^1 across the entire context paragraph using a softmax function applied to a linear transformation of concatenated $\mathbf{G}$ and $\mathbf{M}$ matrices. It is represented as $p^1 = \text{softmax}(w_{p^1}^\top [\mathbf{G}; \mathbf{M}])$, where w_{p^1} is a trainable weight vector of dimension $10d$.

To predict the end index of the answer phrase, matrix $\mathbf{M}$ is processed through another bi-LSTM

layer, resulting in $\mathbf{M^2} \in \mathbb{R}^{2d \times T}$. Subsequently, the model computes the probability distribution p^2 for the end index using a similar softmax function applied to a linear transformation of concatenated $\mathbf{G}$ and $\mathbf{M^2}$: $p^2 = \text{softmax}(w_{p^2}^\top \mathbf{G}; \mathbf{M^2}])$, where w_{p^2} is another trainable weight vector.

5 Dataset

The Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) serves as the benchmark dataset for evaluating the performance of the BiDAF model. Based on a large collection of Wikipedia articles, SQuAD comprises over 100k human-annotated questions and their corresponding answers. The answers are always a span within the context. For my experiment, I split 80k random samples from train dataset into train dataset and dev dataset with a 80/20 ratio, utilizing 10k random samples from dev dataset as the test set.

6 Evaluation Metric

Two metrics are used to evaluate the model on the SQuAD dataset: 1) Exact Match (EM): This metric measures the percentage of predictions that match the ground truth exactly character by character. A higher EM score implies that the model can predict answers that are an exact match to the correct answer. 2) F1 Score: This is the harmonic mean of precision and recall, specifically at the character level. Precision measures the accuracy of the model's predictions, while recall measures the ability to capture all relevant answers. The F1 score provides a balance between precision and recall, giving a single value that reflects the model's overall performance.

7 Parameter settings

The model's hidden state size h is 100.

8 Training

The training loss is defined as the sum of negative log probabilities (NLLLoss) computed for the predicted answer $start$ and end indices relative to their ground truth values. The goal is to minimize this training loss.

9 Inference

Based on input contexts and questions, the model predicts $start$ and end indices (where $start \leq end$) for answer spans. Each example undergoes a process where the indices for $start$ and end positions are identified using the argmax function to pinpoint the highest probabilities.

10 Model Details and Hyper parameter settings

The model architecture for this task, shown in 1 involves tokenizing each context paragraph and question using NLTK before feeding them into the model. The word embedding dimension d set for the model is 100. Character-level word embeddings are generated using a CNN with 100 filters. The kernel width is 5 and the character embedding dimension is set to 8. The model contains approximately 2.6 million parameters. The BiDAF model was trained on various combinations of learning rate (0.01, 0.001, 0.0001, 0.00001, etc) and batch size (10, 25, 32, 50, 100, etc), and the training performance was observed. Finally, training is done using the AdamW optimizer (Loshchilov and Hutter, 2019) with a batch size of 32 and a learning rate of 0.001, over 40 epochs. A dropout rate of 0.2 (Srivastava et al., 2014) is applied to both the CNN and LSTM layers to prevent overfitting.

11 Results and Comparison

Table 1: Results on SQuAD Dataset

Model (Single Model)	EM	F1
Logistic Regression Baseline	40.4	51.0
Dynamic Chunk Reader	62.5	71.0
Fine-Grained Gating	62.5	73.3
Match-LSTM	64.7	73.7
Multi-Perspective Matching	65.5	75.1
Dynamic Co-attention Networks	66.2	75.9
R-Net	68.4	77.5
BiDAF (on validation set)	**52.5**	**65.7**
BiDAF (on test set)	**44.6**	**59.8**

Results of the model and and competing approaches shown in Table 1. BiDAF achieves a highest EM score of 52.5 and an F1 score of 65.7 on validation data and EM score of 44.6 and an F1 score of 59.8 on test data.

12 Conclusion

In this paper, I reimplement BiDAF, a model that uses a multi-stage hierarchical process to represent the context at different levels of granularity. It employs a bidirectional attention flow mechanism to achieve a query-aware context representation without early summarization. The simplified

reimplementation almost achieved the performance reported in the original paper, falling short by only 10-15%. Further hyperparameter tuning and debugging are necessary to fully realize the model's capabilities.

13 Acknowledgement

I would like to express my sincere gratitude to my team members for their invaluable support and help throughout this project.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural machine translation by jointly learning to align and translate.

Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. 2017. Attention-over-attention neural networks for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2017. Gated-attention readers for text comprehension.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. 2016. Text understanding with the attention sum reader network.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Bidirectional attention flow for machine comprehension.

Alessandro Sordoni, Philip Bachman, Adam Trischler, and Yoshua Bengio. 2016. Iterative alternating neural attention for machine reading.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks.