Building a QA system (IID SQuAD track)

Stanford CS224N Default Project

Yi Wen
Department of Computer Science
Stanford University
wyi@stanford.edu

Ruoyan Chen
Department of Electrical Engineering
Stanford University
ruoyan85@stanford.edu

Abstract

For this project, we are dealing with building a Question Answering System that expected to perform well on SQuAD. We conducted several experiments about improving on embedding, modification of attention and application of transformer. Even though coattention didn't perform as well as we anticipated, we obtained pretty satisfying results on BiDAF with character-level embedding and Question Answering Network models. In this report, we are going to introduce our approaches and experiments in depth and give some kind of visualizations to help evaluate our results.

1 Key Information

- Mentor: None
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

The task for this project is to build a question answering system that performs well on SQuAD. In this task, the input is a context paragraph with a related question and the output is the answer to that question. The goal for out models is to answer that question correctly. The following is an example:

- Input
  - **Question:** Why was Tesla returned to Gospic?
  - **Context paragraph:** On 24 March 1879, Tesla was returned to Gospic under police guard for not having a residence permit. On 17 April 1879, Milutin Tesla died at the age of 60 after contracting an unspecified illness (although some sources say that he died of a stroke). During that year, Tesla taught a large class of students in his old school, Higher Real Gymnasium, in Gospic.
- Output
  - **Answer:** not having a residence permit

The task of question answering has gained significant popularity over the past few years, and the use of neural attention mechanism successfully enables the system to focus on a targeted area within a context paragraph. Typical attention mechanisms in previous works are usually uni-directional, wherein the query attends on the context paragraph. In [1], the author introduces the Bi-Directional Attention Flow (BiDAF) network, which is a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity. BiDAF includes character-level, word-level, and contextual embeddings, and uses bi-directional attention flow to obtain a query-aware context representation. The given baseline model in this project is based on this original BiDAF implementation but without the character level embedding.

Our experimental approaches include reproducing the baseline training results, improving default baseline model with character-level embedding, working on Dynamic Coattention Networks as well as Question Answering Network (QANet), and finally comparing performances of these models with that of the baseline model.

3 Related Work

- **BiDAF with character-level embedding [1]:** From the original BiDAF model features mentioned in the above Introduction section, specifically its character-level embeddings allow us to condition on the internal structure of words, which also known as morphology, and better handle out-of-vocabulary words. These are advantages compared to lookup-based word embeddings.

  BiDAF offers three major improvements to the previously attention paradigms. First, the attention is computed for every time step, and the attended vector at each time step, along with the representations from previous layers, is allowed to flow through to the subsequent modeling layer. This technique reduces the information loss caused by early summarization. Second, the author uses a memory-less attention mechanism. While iteratively computing attention through time, the attention at each time step is a function of only the query and the context paragraph at the current time step and does not directly depend on the attention at the previous time step. This simplification amortizes the labor between the attention layer and the modeling layer as well as allows the attention at each tine step to be unaffected from incorrect attendances at previous time steps. Third, using attention mechanisms in both directions, query-to-context and context-to-query provides complimentary information to each other.

- **Coattention:** In [2], several deep learning models have been proposed for question answering, but due to their single-pass nature, they have no way to recover from local maxima corresponding to incorrect answers.

  To address this problem, the author introduces the Dynamic Coattention Network (DCN), which consists of a coattentive encoder that captures the interactions between the question and the document, as well as a dynamic pointing decoder that alternates between estimating the start and end of the answer span. DCN first fuses co-dependent representations of the question and the document in order to focus on relevant parts of both. Then a dynamic pointing decoder iterates over potential answer spans, enabling the model to recover from initial local maxima corresponding to incorrect answers.

- **QANet[3]:** QANet adapts ideas from the Transformer[4] to question answering. Previous end-to-end machine question answering models are primarily based on recurrent neural networks (RNNs) with attention. Despite their success, these models are often slow for both training and inference due to the sequential nature of RNNs. The QANet architecture does not require recurrent networks: Its encoder consists exclusively of convolution and self-attention, where convolution models local interactions and self-attention models global interactions.

  The main component of the QANet model is called an Encoder Block. The Encoder Block draws inspiration from the Transformer: they consist of positional encoding, residual connections, layer normalization, self-attention, and feed-forward sublayers. The difference is that the Transformer uses stacked convolutional sublayers, which use depthwise-separable convolution to capture local dependencies in the input sequence.

4 Approach

The baseline model is based on Bidirectional Attention Flow (BiDAF) [1] but includes only word-level embedding. Our approach focused on training the baseline as well as improving embedding, modifying attention and adding adapting transformer.

A. **Baseline (provided): BiDAF [1] without character-level embedding with coattention[2]-like attention layer** is a hierarchical multi-stage process and consists of four layers:

(1) **Embedding Layer** maps each word to a vector space using a pre-trained word embedding model.

(2) **Contextual Embedding Layer** utilizes contextual cues from surrounding words via a Recurrent Neural Network to refine the embedding of the words.

(3) **Attention Flow Layer** couples the query and context vectors and produces a set of query-aware feature vectors for each word in the context.

(4) **Modeling and Output Layer** employs a Recurrent Neural Network to scan the context and provides an answer to the query.

B. **Improvement on embedding: Character-Level Word Embedding** conditions on the internal structure of words, and better handles out-of-vocabulary words. We added "Character_Embedding" layer which maps each word to a vector space using character-level CNNs; we renamed the given "Embedding" layer to "Word_Embedding" layer; and did some tweaks in both "Word_Embedding" and "HighwayEncoder" layers such as getting rid of the highway network originally applied purely on the word embedding, for the purpose of applying highway on the concatenation of character embedding and word embedding.So that inputs into highway now become character embedding and word embedding, and then after the highway, the output is in the shape of (batch_size, seq_len, 2*hidden_size). Besides, we did projections in both "Char_Embedding" and "Word_Embedding" layers so that their outputs are in the same shape of (batch_size, seq_len, hidden_size). We also created a new Convolutional Neural Network (CNN) class used by the character-level embedding following the model details described in the handout from an archived cs224n assignment [5]. For the CNN class, we used 1-dimensional convolutions, with kernel size of five and number of filters tuned from arguments.

C. **Modification of attention:** : The provided baseline already uses the coattention-like implementation. We try several variants based on this.

- In [2], the question hidden states are projected via tanh nonlinearity and sentinel vectors are added to both the context and question states. These are not done in baseline.
- The similarity matrix is calculated as BiDAF does. The attention weights and attention outputs are derived as coattention does. In [2], the matrix is replaced by affinity scores.
- Attention weights and attention outputs are calculated in the same method.
- The attention layer finally feeds a sequence feature through a bidirectional LSTM. Denote context hidden states as c, question hidden states as q, C2Q attention outputs as a, Q2C attention outputs b, and second-level attention outputs as s. [2] feeds $[s; a]$ and the baseline feeds $[c; a; c * a; c * s]$ where $[\cdot; \cdot]$ is the concatenation along the hidden feature dimension for the sequence and $*$ is element-wise multiplication.

D. **Alternative RNN** can be GRU in place of RNN.

E. **Application of transformer[4]:** The idea of transformer is adapted and we call the new thing *Encoder Block* [3].

(a) A positional encoding is added to the input at the beginning of each encoder block consisting of sin and cos functions at varying wavelengths.

$$\begin{cases} Position[pos, 2i] & = \sin(pos/M^{2i/hidden\_size}) \\ Position[pos, 2i+1] & = \cos(pos/M^{2i/hidden\_size}) \end{cases}$$

where M is hardcoded as $1e4$, pos is the word position and i is the embed index.

(b) After that, there is a stack of depthwise separable convolutions, each of which is composed of two convolution layers:

- Depth-wise convolution: The filter and image are broken into different channels and each filter channel is applied only at one input channel.
- Point-wise convolution: The layer behaves as usual along depth and the kernel size is 1×1.

(c) In the following self-attention layer, we adopt the multi-head attention mechanism [4] by taking the input as query, key and value.

(d) The final is the feed-forward-layer made up of two fully-connected layers and a relu activation layer between them.

Consider *each* of (a) the convolution layers in the stack, (b) the self-attention-layer, and (c) the feed-forward layer as a functional operation f with input x, the output is $f(layernorm(x)) + x$ to establish residual connections.

Our best model still consists of four layers, the same as the baseline [A], but with some modifications [from B, C, D]:

(1) **Embedding Layer [A+B]** The combination of word- and char-level embedding uses the pre-trained word embedding model to map the input into vector space.

(2) **Contextual Embedding Layer [A+E]**
The Encoder Block utilizes contextual cues from surrounding words to refine the embedding of the words

(3) **Attention Flow Layer [A]**
The coattention-like implementation produces a set of query-aware feature vectors for each word in the context.

(4) **Modeling and Output Layer [A+E]**
A stack of Encoder Blocks with fully-connected layers are used to scan the context and provide an answer to the query.

5 Experiments

5.1 Dataset

The SQuAD dataset [6] contains many (context, question, answer) triples. Each context is an excerpt from Wikipedia. The question is the question to be answered based on the context. The answer is a span from the context. The official SQuAD 2.0 dataset has three splits: train, dev and test. We will use the train split and the dev split provided by the course to train, tune and evaluate the models respectively. Here, the dev split is a subset of the official dev split. The details for the splits here:

- train (129,941 examples): exactly the official SQuAD 2.0 train set
- dev (6,078 examples): from the official dev set
- test (5,915 examples): from the official dev set, plus hand-labeled examples

5.2 Evaluation method

Performance is measured via two metrics: Exact Match (EM) score and F1 score.

- **Exact Match** is a binary measure (i.e. True/False) of whether the system output matches the ground truth answer exactly.
- **F1** is the harmonic mean of precision and recall, which reflects how much the answer hits the ground truth and misses the ground truth respectively.

$$F1 = \frac{1}{precision^{-1} + recall^{-1}} = \frac{2tp}{2tp + fp + fn}$$

In SQuAD, every question has three answers provided – each answer from a different crowd worker. When evaluating, we take the maximum F1 and EM scores across the three human-provided answers for that question.

5.3 Experimental details and Results

Baseline We reproduced the experiment of training the baseline model, with default hyperparameters: hidden_size = 100,batch_size = 64, lr = 0.5, drop_prob = 0.2, ema_decay = 0.999, epoch=3000000. We obtained similar results as expected. Over 3 million iterations, the dev AvNA reaches about 67, the dev F1 reaches about 60, the dev EM score reaches around 57, and the dev NLL reaches 3.2.

BiDAF with char-level embedding In the original BiDAF paper [1], the author uses 100 1-dimensional filters for CNN char embedding. In our experiments, we scaled the number of filters up and used 200 and 300 1-dimensional filters respectively as the character embedding dimension, and kept other hyperparameters unchanged. The results are shown in Table 1.

Table 1: The results of comparing char-level embedding

model	dev		test	
	F1	EM	F1	EM
Baseline	60.31	57.21	-	-
char embed (200)	63.24	59.69	-	-
char embed (300)	63.72	60.39	64.32	60.34

Over 3 million iterations of training the model with 200-dimension char-level embedding, the dev AvNA reaches about 69 and the dev NLL reaches about 3.1; for the 300-dimension, the dev AvNA reaches about 70 and the dev NLL reaches about 3.1.

Alternative RNN is replaceing all the LSTMs with GRUs in the baseline. The corresponding results are in Table 2.

Table 2: The results of comparing RNNs

model	dev	
	F1	EM
LSTM	60.31	57.21
GRU	60.83	57.57

Ablation study of coattention keeps all implementation and hyperparameters the same except for the Attention Flow Layer, which involves the following elements:

- `pre`: some preprocess operations at the beginning
  - `No`: No related operations are use
  - `Yes`: The question hidden states are projected via tanh nonlinearity and sentinel vectors are added to both the context and question state.
- `score`: the matrix reflecting the context-question relation used later to apply softmax operation
  - `sim`: It is the same in BiDAF.
  - `aff`: It uses the affinity scores.
- `lstm`: the sequence passed as input to the final bidirectional LSTM
  - `x2`: It feeds $[s; a]$.
  - `x4`: It feeds $[c; a; c * a; c * s]$.

We study some different versions of the attention as shown in Table 3. Based on the experiments, the implementation in the provided baseline is already very good.

Table 3: The results of ablation study of coattention. ($*$ is the Baseline.)

	model			dev	
	pre	score	lstm	F1	EM
	No	aff	x2	53.64	51.65
	Yes	aff	x2	53.67	51.62
	Yes	aff	x4	59.44	56.02
$*$	No	sim	x4	60.31	57.21
	Yes	sim	x4	59.93	57.30

Trial of QANet is run for only once and turns out to be our best model. It uses the hyperparameters in the [3], the following are different from the baseline:

- `hidden_size = 128` which should be divisible by `num_head` later
- `batch_size = 16` due to limited CUDA memory
- `lr = 0.5` optimized cross entropy loss instead of negative log likelihood loss
- `drop_prob = 0.1`

Embedding Layer uses pre-trained embedding model. The modification with respect to the baseline applies the idea of Encoder Block to Contextual Embedding Layer and Modeling and Output Layer.

- **Contextual Embedding Layer** is an Encoder Block with the following hyprtparameters: in the convolution stack, `conv_num=4` convolutions with `kernel_size=7`; in self-attention, `num_head=8`.
- **Modeling and Output Layer** is step by step as following:
  - A fully-connected layer model the attention outputs by mapping the hidden dimension from `hidden_size * 4` to `hidden_size = 128`.
  - There is a stack of Encoder Blocks with: `num_block=7`; in the convolution stack, `conv_num=2` convolutions with `kernel_size=5`; in self-attention, `num_head=8`. Passing through the Encoder Block Stack sequentially thrice, the outputs are denoted as M_0, M_1, M_2.
  - The final prediction logits of the starting and ending position are modeled as $W_{start}[M_0; M_1]$ and $W_{end}[M_0; M_2]$ where W_{start}, W_{end} are trainable parameters.

The results are compared with the baseline Table 4.

Table 4: The results of comparing our best model compare with the baseline

model	dev		test	
	F1	EM	F1	EM
Baseline	60.31	57.21	-	-
w/ char embed	63.72	60.39	64.32	60.34
Best model	68.41	64.95	66.43	62.45

6 Analysis

- The re-implementation of BiDAF with character-level embedding performs relatively better than the baseline does. This is expected because the addition of character-level embedding increases model's learning capability from the perspective of morphology. Specifically, by scaling up the dimension of character embedding, we kept getting slightly higher results but we think this might gradually reach to a ceiling because on the one hand, the model will be too heavy to train and on the other hand, the amount of benefits getting from larger embedding dimension will reach to a bottleneck.

  After experimenting with character-level embedding, we have learnt several lessons. For examples, for polysynthetic language which characterized by complex words consisting of several morphemes, it's better to split a word into sub-characters for the semantic analysis purpose. Further, using character-level embedding avoids the out-of-vocabulary problem, makes it possible to encode words that were not in the training set, and gives manageable vocabulary sizes as well as mitigates the data sparsity.
- For the attention flow layer, the provided baseline implementation already involves a second-level attention computation. Compared with the implementation in the original paper of coattention [2], there are some differences. With the variable-control ablation study, we see that the baseline's choice of the context-question scores and the sequence concatenation passing into Bi-LSTM already reaches the level of the best performance. Also, the post-processing of question state projection and sentinel vectors does not make big difference.

The main contribution to the performance difference is what to pass to the final LSTM to fuse information. Compared with feeding $[s; a]$ in [2], we feed $[c; a; c * a; c * s]$. This seems reasonable since the concatenation with the double hidden size is likely to carry more information. One more thing we can see is that this choice pay much more attention to the context hidden state c input into the attention layer.

- When replacing LSTM with GRU, the performance is at the same level. The main difference is that GRU converges at almost double the speed of LSTM. In this project, the dataset size is kind of small and thus GRU, which is less complex because of less number of gates, may be more helpful than RNN.
- At the trial of QANet, the experiment time is very limited. We can only show that the introduction of Encoder Block inspired by Transformer to the architecture can improve the performance obviously. The model are much larger and takes much longer to learn. One interesting finding is that the performances on dev and test splits vary quite a little. It seems that it might be overfitting to some part of the data. There is expected to be some improvement after tuning based on this version of the architecture.

  The advance made by this model is similar to the benefit of a transformer over an LSTM. Using convolutional neural networks instead of recurrent neural networks, the Encoder Block processes sentences as a whole rather than word by word, which helps relieve the suffer from long dependency issues. Meanwhile, the multi-head attention and positional encoder both provide information about the relationship between different words.

7 Conclusion

Although at the beginning of this project we only planed to implement BiDAF with character-level embedding and Dynamic Coattention Network, during the experiments and per TA's suggestion, we finally decided to focus on a new architecture called Question Answering Network given the unsatisfying results of our work on coattention model. Fortunately, both BiDAF (with character-level embedding) and QANet (inspired by transformer) achieved fairly satisfying results compared to the baseline performance.

References

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

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering, 2018.

[3] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 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. *arXiv preprint arXiv:1706.03762*, 2017.

[5] Character-based convolutional encoder for nmt, archived cs224n assignment #5, Winter-2019.

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