RNet Re-Implementation for SQuAD

Stanford CS224N Default Project

Danny Tse
Department of Computer Science
Stanford University
dannytse@stanford.edu

Abstract

We reimplement the R-Net model described in *R-Net: Machine Reading Comprehension with Self-Matching Networks* [1] for the Stanford Question Answering Dataset (SQuAD). Furthermore, we experiment with different types of encoding layers, particularly one encoding characters with a Gated Recurrent Unit (GRU), and one encoding characters with a Convolutional Neural Network (CNN). We analyze the performance of these edited baseline models, and look toward completing the reimplementation and possibly improving on it.

1 Approach

We wish to develop a model that performs relatively well on the Stanford Question Answering Dataset (SQuAD). To do this, we reimplement the complex neural model described in the paper *R-Net: Machine Reading Comprehension with Self-Matching Networks* [1], including its contributions such as the gated attention-based recurrent network and self-matching mechanism. Furthermore, we wish to determine how different ways of processing character embeddings interact with the rest of the model architecture, and how the self-attention mechanisms described in the paper compare to the given baseline RNN model for question-answering. We suspect that, since the self-attention mechanism allows hidden states to consider previous hidden states, this model can record long-distance dependencies, and as a result have more complete answers to questions. The project is a direct reimplementation of the paper, even training on a similar dataset (SQuAD 2.0), so reimplementing the model, tuning hyper-parameters, and making few adjustments should suffice. Variants of self-attention have proven to be vastly successful in modern natural language processing models (ex. usage of multi-headed self-attention in a Transformer Encoder-Decoder block introduced in Vaswani et al. (2017)'s *Attention Is All You Need* [2]), thus in theory, this model should outperform our Bidirectional Attention Flow baseline.

We describe the model architecture below. First, we process the question and passage through a bi-directional recurrent network. Then, we match each question and passage with gated attention-based recurrent networks, followed by self-matching attention. This provides us with a question-aware representation for the passage. Then, we refine the passage representation with self-matching attention and feed it into the output layer to return the boundary of the answer span. The model layers are:

1. **Question and Passage Encoder** Passing in a question $Q = \{w_t^Q\}_{t=1}^m$ and a passage $P = \{w_t^P\}_{t=1}^n$, we begin by converting the question and passage into their corresponding word embeddings ($\{e_t^Q\}_{t=1}^m$ and $\{e_t^P\}_{t=1}^n$ respectively). We obtain the character-level embeddings by taking the final hidden states of a bi-directional recurrent neural network (RNN) applied to the embeddings of characters in the token. Then, we apply a Gated Recurrent Unit (GRU) to obtain representations for words in the question and the passage. That is,

$$u_t^Q = \text{GRU}_Q(u_{t-1}^Q, [e_t^Q, c_t^Q])$$
$$u_t^P = \text{GRU}_P(u_{t-1}^P, [e_t^P, c_t^P])$$

2. **Gated Attention-Based Recurrent Networks** Now, given question and passage representations u_t^Q and u_t^P from the Question and Passage Encoder, we use a gated attention-based recurrent network to match question information with passage representation. Wang & Jiang (2016) [3] propose match-LSTM:

   $$v_t^P = \text{GRU}(v_{t-1}^P, [u_t^P, c_t])$$

   where c_t is given by the attention-pooling vector of the entire question u^Q:

   $$\begin{gathered} s_j^t = v^T \tanh(W_u^Q u_j^Q + W_u^P u_t^P + W_v^P v_{t-1}^P) \\ a_i^t = \exp(s_i^t) / \textstyle\sum_{j=1}^m \exp(s_j^t) \\ c_t = \textstyle\sum_{i=1}^m a_i^t u_i^Q \end{gathered}$$

   Then, we add another gate to the input to the input $[u_t^P, c_t]$:

   $$\begin{gathered} g_t = \text{sigmoid}(W_g[u_t^P, c_t]) \\ [u_t^P, c_t]^* = g_t \odot [u_t^P, c_t], \end{gathered}$$

   which is establishes a relationship between the current passage word and its attention-pooling vector of the question. $[u_t^P, c_t]^*$ is utilized in subsequent calculations in place of $[u_t^P, c_t]$, and is called the "gated attention-based recurrent network".

3. **Self-Matching Attention** Now, we are given the question-aware passage representation from previous layers, or $\{v_t^P\}_{t=1}^n$. An issue with such a representation is that it lacks extensive knowledge of context. To circumvent this issue, the paper proposes using attention to match $\{v_t^P\}_{t=1}^n$ against itself, obtaining h_t^P which encodes evidence from words in the passage and their matching question information:

   $$h_t^P = \text{GRU}(h_{t-1}^P, [v_t^P, c_t])$$

   where c_t is given by the attention-pooling vector of the whole passage v^P:

   $$\begin{gathered} s_j^t = v^T \tanh(W_v^P u_j^P + W_u^{\bar{P}} u_t^P) \\ a_i^t = \exp(s_i^t) / \textstyle\sum_{j=1}^n \exp(s_j^t) \\ c_t = \textstyle\sum_{i=1}^n a_i^t v_i^P \end{gathered}$$

   Once again, an additional gate as stated above is applied to $[v_t^P, c_t]$.

4. **Output Layer** Finally, pointer networks are used to predict the start and end position of the answer. Attention-pooling over the question representation is used to generate the initial hidden vector for the pointer network. Given $\{h_t^P\}_{t=1}^n$ from the previous layers, we utilize the attention mechanism to select start and end positions p^1 and p^2 from the passage:

   $$\begin{gathered} s_j^t = v^T \tanh(W_h^P h_j^P + W_h^a h_{t-1}^a) \\ a_i^t = \exp(s_i^t) / \textstyle\sum_{j=1}^n \exp(s_j^t) \\ p_t = \text{argmax}(a_1^t, \ldots, a_n^t) \end{gathered}$$

   where

   $$\begin{gathered} c_t = \textstyle\sum_{i=1}^n a_i^t h_i^P \\ h_t^a = \text{GRU}(h_{t-1}^a, c_t) \end{gathered}$$

   Then, using h_a^{t-1} as the initial hidden state and r^Q as the initial state,

   $$\begin{gathered} s_j = v^T \tanh(W_u^Q u_j^Q + W_v^Q V_r^Q) \\ a_i = \exp(s_i) / \textstyle\sum_{j=1}^n \exp(s_j) \\ r^Q = \textstyle\sum_{i=1}^m a_i u_i^Q \end{gathered}$$

Though the paper describes exactly what layers to use, and what model architecture components to incorporate, we may experiment with other adjustments that have proven to work. One example of such an adjustment is described in Seo et al. (2017)'s *Bi-Directional Attention Flow for Machine Comprehension* [4], where a Convolutional Neural Network (CNN) Layer is utilized to process character embeddings. In this modification, we simply obtain the character-level embedding of each word using a CNN, instead of a GRU. Characters are embedded into vectors whose size is the input-channel size of the CNN, which are then fed as 1-dimensional inputs. Then, we apply a max-pool on the CNN's outputs over the entire width to obtain a fixed-size vector for each word.

The baseline we use is a Bi-Directional Attention Flow (BiDAF) model on the word embeddings. This model is described in the Default Final Project Handout (SQuAD Track), and consists of a Word Embedding Layer, an Encoder Layer, an Attention Layer, a Modeling Layer, and an Output Layer.

2 Experiments

As data, we use the Stanford Question Answering Dataset 2.0 (SQuAD 2.0), which has the splits train (129,941 examples), dev (6078 examples), and split (5915 examples). In this dataset, passages are selected from the English Wikipedia (usually 100-150 words), questions are crowd-sourced, and each answer is either contained within the passage, or does not exist at all (not answerable). Also, each answerable SQuAD question has three answers, each from a different crowd worker.

To evaluate model performance, we use two well-defined, numerical, automatic evaluation metrics: Exact Match (EM) and F1 score. The Exact Match (EM) captures the percentage of the prediction that completely matches with one of the ground truth answers. For example, if the ground truth was "Panic of 1901" and the model responded with "1901", then the ground truth score would be 0. On the other hand, the F1 score is less strict - it is the harmonic mean of precision and recall. An example of an F1 score is with "1901" (model answer) and "Panic of 1901" (ground truth). Since the model answer is completely contained in the ground truth, it would have 100% precision, though only 33.3% recall, as it only included one out of the three words in the ground truth answer. Thus we have an F1 score of $2 \times 100 \times 33.3/(100 + 33.3) \approx 50.0\%$.

In experimentation thus far, we have run three versions of the model: (1) the baseline, (2) baseline, but replacing the first word embedding layer with a word + character embedding layer, which is processed through a CNN as described above, and (3) baseline, but replacing the first word embedding layer with a word + character embedding layer, passed through the Gated Recurrent Unit described in the model description above. We used default baseline hyper-parameters for all three test runs, and they ran at different times, with iteration (3) taking almost four times as long as the baseline. Furthermore, we observed that model (3) performed slightly worse than the baseline, and model (2) performed slightly better than the baseline. We observed the following results:

F1 and EM Scores		
Model	**EM Score**	**F1 Score**
Baseline	57.889	60.965
Baseline with CNN	**57.973**	**61.381**
Baseline with GRU	52.731	55.390

We noticed that the baseline + CNN model outperformed the standard baseline. We suspect that it may be because we simply have more information – given character embeddings, the LSTM can process subword relationships that normal embeddings may not encode. Furthermore, though the model involving the GRU performs slightly worse than the baseline, this can be attributed to the fact that the GRU is slightly more compatible with the other parts of the described model. After all, this work-in-progress uses the first bit of the described model, and fills the rest in with the baseline. Another potential reason may simply be suboptimal hyperparameters – in training, we observed that the scores were still increasing when we reached 30 epochs, so we can potentially increase the learning rate to observe an improvement.

3 Future work

In the future, we will completely reimplement the model described in the paper and in section 1. That is, we will replicate the gated attention-based recurrent network layer, the self-matching attention layer, and the output layer. Furthermore, we will consider the different types of encoding layers. As we saw above that the CNN layer performed (with the rest of the layers being the baseline) better than the GRU, we will experiment with a CNN-based encoding layer on our model, as well as adjust the GRU's hyperparameters, particularly the learning rate.

References

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

[2] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N Gomez Lukasz Kaiser Ashish Vaswani, Noam Shazeer and Illia Polosukhin. Attention is all you need.

[3] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. In *arXiv preprint arXiv:1608.07905*, 2016.

[4] Ali Farhadi Hananneh Hajishirzi Minjoon Seo, Aniruddha Kembhavi. Bi-directional attention flow for machine comprehension. In *arXiv preprint arXiv:1611.01603*.