KnowGQA: Using Knowledge Attention to Integrate Extra Graph Knowledge in Question Answering(Unfinished Version)

Stanford CS224N {Default} Project

Jiarui Feng
Washington University in St.Louis
feng.jiarui@wustl.edu

Abstract

In the project, we proposed model KnowGQA to present a general framework to integrate extra knowledge into current question answering system using graph convolutional neural network and knowledge attention. We collect extra knowledge from ConceptNet to build sub-graph for each context and apply knowledge attention to combine extra knowledge into QANet which is well-performed and computation-saving model in question answering problem. Finally, we test our model in on the Stanford Question Answering Dataset(SQuAD) 2.0. Results show that our framework significantly improve the performance of QANet in terms of EM and F1scores.

1 Introduction

Machine reading comprehension equip computer with the ability to read and understand given textual context and answer questions corresponding to context. Recently, a sheer number of neural architectures achieved super-human performance in question answering problems. Typically, The models can be divided into two types. The first is based on Pre-trained Contextual Word Embedding(PCE) techniques like ELMo[1] and Bert[2]. PCE based model become really popular in recent years due to its power of generalizing to many different Nature Language Processing(NLP) tasks and high performance in all the tasks. Currently, almost all the models that achieve state of art results in question answering tasks are PCE based. We can regard PCE as integrate extra knowledge into the model, which entail model with better ability to understand both contexts and questions. The second type of models are not based on PCE but pay more attention to the structure and design of model to enable each part of model to be specific in one part of task. There are also many well-known models in question answering task that don't depend on PCE like BiDAF[3] and QANet[4]. Although the performance of non-PCE based model is not good compare to the counterpart, it is still worth to investigate it, which allow researcher to find better architectures and study what the model really learned in such tasks.

Recently, the concept of knowledge graph become popular because it characterizes the relationship among different entity that we have seen in normal world. Such relationship provides a different way of modeling the world and could in theoretically improve the ability of question answering system because this is also a kind of extra knowledge integration like PCE. Several methods have been proposed to combine knowledge graph information into question answering

system like KABLSTM[5]. However, it only use the knowledge graph information as pre-trained embedding in the model.

Graph Convolutional Neural Network(GCN)[6] and many refined model have been proposed in recent year to capture and represent information from graph data. Many studies have utilize GCN to modeling the information in the knowledge graph. In this work, we proposed KnowGQA to integrate extra knowledge graph information into the question answering system using GCN and knowledge attention. We retrieve the knowledge graph data for each word entity and its related edges from ConceptNet[7] and build knowledge sub-graph for each context and generate knowledge representation using GCN. Such representation is then be integrated into question answering system using knowledge attention. The knowledge attention mechanism can be used in any question answering system. Here, we combine knowledge attention with QANet train the combined model in SQuAD 2.0 dataset. Results show that the extra knowledge extensively improve the performance of QANet in SQuAD 2.0, which prove the ability of our framework.

2 Related Work

In the field of non-PCE model, Before QANet, the majority of model are based on RNN architecture and attention mechanism. In BiDAF[3], researchers used multi-layers Bi-direction LSTM and innovative context to question(C2Q) and question to context attention(Q2C) mechanism. C2Q and Q2C is good for capturing question-aware representation of context, which enable model to combine both information from context and question in order to answer the questions. Model achieve state-of-art result in SQuAD1.0 when it published. However, one big drawback of BiDAF is that it depends on RNN architecture, which is hard to parallelize. Thus the training and inference time of BiDAF is slow.

In 2017, Transformer[8] is proposed, which achieved super good performance in Machine translation task and also show great potential to generate to other tasks. One major advance is that Transformer using self-attention to modeling sequence data, which does not depend on RNN architecture and thus easy to be parallelized. QANet[4] draw on the idea of encoder of Transformer and generalize it to question answering task. QANet replace all the RNN architecture in the model with modified-transformer encoder, which also integrate depthwise convolution layer to modeling local information. QANet achieved state-of-art result in SQuAD1.0. Meanwhile, the training time of QANet was 4.3 times faster than BiDAF and the Inference time of QANet was 7.0 times faster than BiDAF.

3 Method

3.1. Task Description

Given a context $\boldsymbol{C}$ with sequence of words $\boldsymbol{c_1}, \boldsymbol{c_2}, \dots, \boldsymbol{c_N} \in \mathbb{R}^D$ and a question $\boldsymbol{Q}$ with sequence of words $\boldsymbol{q_1}, \boldsymbol{q_2}, \dots, \boldsymbol{q_M} \in \mathbb{R}^D$, where D is embedding size, model what to find an answer, where the answer is guaranteed to be a continuous span in the context. Let the start and end point of answer be $\boldsymbol{p}$ and $\boldsymbol{q}$, we want to find a model that maximize the $\mathrm{p}(\boldsymbol{p}|\boldsymbol{C}, \boldsymbol{Q})$ and $\mathrm{p}(\boldsymbol{q}|\boldsymbol{C}, \boldsymbol{Q})$ for each $\boldsymbol{C}$ and $\boldsymbol{Q}$.

3.2. Proposed Model

Figure 1. Architecture of KnowGQA

The framework of KnowGQA is largely based on QANet with minimal modification, show in the Figure 1. We will illustrate of our model in a bottom up fashion.

3.3. Embedding Blocks

In embedding blocks, we combine initial word embedding and char embedding to get our final word embedding. Let the dimension of initial word embedding be D_w. Thus, the initial word embedding of context $\boldsymbol{C_w} \in \mathbb{R}^{N \times D_w}$ and initial word embedding of question $\boldsymbol{Q_w} \in \mathbb{R}^{M \times D_w}$. Let the dimension of char embedding be D_c, the number of chars in each word be C_h. Thus, the char embedding of context $\boldsymbol{C_c} \in \mathbb{R}^{N \times C_h \times D_c}$ and the char embedding of question $\boldsymbol{Q_c} \in \mathbb{R}^{M \times C_h \times D_c}$.

Let the dimension of hidden size in the model be *H*. The processing of initial word embedding is simple, we use a linear projection $\boldsymbol{W_{proj}} \in \mathbb{R}^{D_w \times H/2}$ to project the word vector to the dimension of *H/2:*

$$\widehat{\boldsymbol{C}}_{\boldsymbol{w}} = \boldsymbol{C_w W_{proj}} \quad \in \mathbb{R}^{N \times H/2}$$

$$\widehat{\boldsymbol{Q}}_{\boldsymbol{w}} = \boldsymbol{Q_w W_{proj}} \quad \in \mathbb{R}^{M \times H/2}$$

For the char embedding, we use char-CNN to do our processing. To be specific, a Convolution layer with ReLU activation function is applied with kernel size be 3 and output channel be *H/2:*

$$\widetilde{\boldsymbol{C}}_{\boldsymbol{c}} = \mathrm{CNN}(\boldsymbol{C}_{\boldsymbol{c}}) \ \in \mathbb{R}^{N\times H/2\times D_c}$$

$$\widetilde{\boldsymbol{Q}}_{\boldsymbol{c}} = \mathrm{CNN}(\boldsymbol{Q}_{\boldsymbol{c}}) \ \in \mathbb{R}^{M\times H/2\times D_c}$$

Then, a Max pooling layer with stride be D_c is used to Extract final feature for each wrods:

$$\widehat{\boldsymbol{C}}_{\boldsymbol{c}} = \mathrm{MaxPooling}\big(\widetilde{\boldsymbol{C}}_{\boldsymbol{c}}\big) \ \in \mathbb{R}^{N\times H/2}$$

$$\widehat{\boldsymbol{Q}}_{\boldsymbol{c}} = \mathrm{MaxPooling}\big(\widetilde{\boldsymbol{Q}}_{\boldsymbol{c}}\big) \ \in \mathbb{R}^{M\times H/2}$$

Finally, we concatenate both word embedding and char embedding to get our final word embedding:

$$\boldsymbol{h}_{\boldsymbol{C}} = \mathrm{Concat}\big(\widehat{\boldsymbol{C}}_{\boldsymbol{w}}, \widehat{\boldsymbol{C}}_{\boldsymbol{c}}\big) \ \in \mathbb{R}^{N\times H}$$

$$\boldsymbol{h}_{\boldsymbol{Q}} = \mathrm{Concat}\big(\widehat{\boldsymbol{Q}}_{\boldsymbol{w}}, \widehat{\boldsymbol{Q}}_{\boldsymbol{c}}\big) \ \in \mathbb{R}^{M\times H}$$

3.4. Highway Blocks

Then, we input the word representation of $\boldsymbol{h}_{\boldsymbol{C}}$ and $\boldsymbol{h}_{\boldsymbol{Q}}$ into highway blocks with two highway network layers. The computation in each highway layer is following:

$$\boldsymbol{g} = \sigma\big(\boldsymbol{W}_{\boldsymbol{g}}\boldsymbol{h} + \boldsymbol{b}_{\boldsymbol{g}}\big) \ \in \mathbb{R}^{(N\ or\ M)\times H}$$

$$\boldsymbol{t} = ReLU(\boldsymbol{W}_{\boldsymbol{t}}\boldsymbol{h} + \boldsymbol{b}_{\boldsymbol{t}}) \ \in \mathbb{R}^{(N\ or\ M)\times H}$$

$$\widetilde{\boldsymbol{h}} = \boldsymbol{g} \odot \boldsymbol{t} + (1 - \boldsymbol{g})\boldsymbol{h} \in \mathbb{R}^{(N\ or\ M)\times H}$$

Where $\boldsymbol{W}_{\boldsymbol{g}}, \boldsymbol{W}_{\boldsymbol{t}} \in \mathbb{R}^{H\times H}$ and $\boldsymbol{b}_{\boldsymbol{g}}, \boldsymbol{b}_{\boldsymbol{t}} \in \mathbb{R}^{H}$ are learnable parameters. $\boldsymbol{h}$ is either $\boldsymbol{h}_{\boldsymbol{C}}$ or $\boldsymbol{h}_{\boldsymbol{Q}}$. σ is sigmoid function, $\odot$ is pointwise matrix production. Highway is used to preliminarily extract contextual representation of contexts and questions.

3.5. Stacked Embedding Encoder Blocks

To further extract information from embedding, stacked embedding encoder block is applied. Typically, a block contains several encoder layers. The structure of encoder layer is similar to the encoder layer in Transformer[8] with additional depthwise separable convolutional layers. Such convolutional layer is memory saving and can capture local information from the surrounding of each words. The kernel size of conv-layer is 7 and the number of conv-layer in each encoder layer is 4. In each encoder layer, input $\boldsymbol{h}$ will first pass through # convolutional layers, where # is the number of convolutional layers. Then pass through multi-head self-attention layer and feed-forward network layer. Residual connection and layer normalization layer are added after each layer. The hidden size of input and output are all *H* and the number of encoder layer in encoder blocks is 1.

3.6. Residual GCN Blocks

For each context, we build sub-graph based on the edges retrieved from ConceptNet. The node for each sub-graph are the words in the context and additional words that have edges start from word in the contexts. Let $\boldsymbol{X} \in \mathbb{R}^{a\times D_w}$ be the node matrix, where a is the number of nodes in the sub graph and D_w be the embedding size. Let $\boldsymbol{A} \in \mathbb{R}^{a\times a}$ be the adjacency matrix that indicate the edges in the sub graph, where $\boldsymbol{A}_{\boldsymbol{ij}} = 1$ indicate there is an edge between *i* and *j*, otherwise $\boldsymbol{A}_{\boldsymbol{ij}} =$

0. Let $\boldsymbol{D} \in \mathbb{R}^{a \times a}$ be degree matrix used for normalization. The sub-graph information will first be processed by Residual GCN blocks to get knowledge representation in the context.

The Residual GCN Blocks contains one diffpool[9] layer and *N* GCN layers. Let The $\boldsymbol{W_k} \in \mathbb{R}^{H \times H}$ and $\boldsymbol{b_k} \in \mathbb{R}^{H}$ be learnable parameters in the GCN layers, *H* is the hidden size, the computation in GCN layer is:

$$\widetilde{\boldsymbol{X}} = GCN(\boldsymbol{A}, \boldsymbol{X}) = ReLU\left(\boldsymbol{D^{-1}}(\boldsymbol{A} + \boldsymbol{I})\boldsymbol{X}\boldsymbol{W_k} + \boldsymbol{b_k}\right) \in \mathbb{R}^{a \times H}$$

Where $\boldsymbol{I}$ is identity matrix.

Before the GCN layers, we introduce diffpool layer in order to first clustering nodes into different groups, which give us compacted graph representation. The diffpool layers contain two GCN layers:

$$\boldsymbol{Z} = GCN_{embed}(\boldsymbol{A}, \boldsymbol{X}) \in \mathbb{R}^{a \times H}$$

$$\boldsymbol{S} = GCN_{pool}(\boldsymbol{A}, \boldsymbol{X}) \in \mathbb{R}^{a \times H}$$

In diffpool layer, we learn a current graph representation $\boldsymbol{Z}$ and an embedding matrix $\boldsymbol{S}$. Here, the learnable matrix $\boldsymbol{W_k}$ in GCN_{embed} and GCN_{pool} is $\in \mathbb{R}^{D_w \times H}$. The S is used to get compacted embedding of graph. The embedded graph representation and adjacency matrix is computed by:

$$\tilde{\boldsymbol{S}} = softmax(\boldsymbol{S}) \in \mathbb{R}^{a \times H}$$

$$\widetilde{\boldsymbol{X}} = \tilde{\boldsymbol{S}}^{\boldsymbol{T}}\boldsymbol{Z} \in \mathbb{R}^{H \times H}$$

$$\widetilde{\boldsymbol{A}} = \tilde{\boldsymbol{S}}^{\boldsymbol{T}}\boldsymbol{A}\tilde{\boldsymbol{S}} \in \mathbb{R}^{H \times H}$$

Where $\widetilde{\boldsymbol{X}}$ and $\widetilde{\boldsymbol{A}}$ are the output of diffpool layer. Thus, from the diffpool layer, we can get compacted embedding of sub-graph. $\widetilde{\boldsymbol{X}}$ and $\widetilde{\boldsymbol{A}}$ will be the input of following GCN layers. Between each GCN layers, we also add residual connection and layer normalization. The final knowledge representation is generated by:

$$\boldsymbol{k} = \sum_{i=1}^{H} \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} \in \mathbb{R}^{H \times 1}$$

Where $\widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$ is *i*-th row vector in $\widetilde{\boldsymbol{X}}$.

3.7. Knowledge Attention Blocks

Next, we proposed Knowledge Attention Blocks to integrate knowledge learned in Residual GCN Blocks into context and question. Let the output of Stacked Embedding Encoder Blocks be $\boldsymbol{h_C}$ and $\boldsymbol{h_Q}$**.** The knowledge attention score is computed by**:**

$$\boldsymbol{c_{attn}} = softmax(\boldsymbol{h_C}\boldsymbol{k}) \in \mathbb{R}^{N \times 1}$$

$$\boldsymbol{q_{attn}} = softmax(\boldsymbol{Q_C}\boldsymbol{k}) \in \mathbb{R}^{M \times 1}$$

The softmax function is performed on each row. $\boldsymbol{c_{attn}}$ indicate the relevance or importance of each words with knowledge. Next, we use knowledge attention score to compute knowledge aware context representation and question representation:

$$\boldsymbol{c_k} = \boldsymbol{h_C}^{\boldsymbol{T}} \boldsymbol{c_{attn}} \quad \in \mathbb{R}^{H \times 1}$$

$$\boldsymbol{q_k} = \boldsymbol{h_Q}^{\boldsymbol{T}} \boldsymbol{c_{attn}} \quad \in \mathbb{R}^{H \times 1}$$

The we tile the $\boldsymbol{c_k}$ and $\boldsymbol{q_k}$ N and M times respectively to get $\boldsymbol{C_k}$ and $\boldsymbol{Q_k}$. Next, we learn a gate function to merge the knowledge aware representation with original representation. Let $\boldsymbol{W_g} \in \mathbb{R}^{4*H \times 1}$ and $\boldsymbol{b_g} \in \mathbb{R}^1$ be learnable parameters:

$$\boldsymbol{g}(\boldsymbol{h_C}, \boldsymbol{C_k}) = \sigma(Concat(\boldsymbol{h_C}, \boldsymbol{C_k}, \boldsymbol{h_C} \odot \boldsymbol{C_k}, \boldsymbol{h_C} - \boldsymbol{C_k})\boldsymbol{W_g} + \boldsymbol{b_g}) \quad \in \mathbb{R}^{N \times 1}$$

$$\boldsymbol{g}(\boldsymbol{h_Q}, \boldsymbol{Q_k}) = \sigma(Concat(\boldsymbol{h_Q}, \boldsymbol{Q_k}, \boldsymbol{h_Q} \odot \boldsymbol{Q_k}, \boldsymbol{h_C} - \boldsymbol{Q_k})\boldsymbol{W_g} + \boldsymbol{b_g}) \quad \in \mathbb{R}^{N \times 1}$$

Finally, we merge two representations get finally representation by:

$$\widetilde{\boldsymbol{h_C}} = \boldsymbol{g}(\boldsymbol{h_C}, \boldsymbol{C_k})\boldsymbol{h_C} + \big(1 - \boldsymbol{g}(\boldsymbol{h_C}, \boldsymbol{C_k})\big)\boldsymbol{C_k} \quad \in \mathbb{R}^{N \times H}$$

$$\widetilde{\boldsymbol{h_Q}} = \boldsymbol{g}(\boldsymbol{h_Q}, \boldsymbol{Q_k})\boldsymbol{h_Q} + \Big(1 - \boldsymbol{g}\big(\boldsymbol{h_Q}, \boldsymbol{Q_k}\big)\Big)\boldsymbol{Q_k} \quad \in \mathbb{R}^{N \times H}$$

3.8. Context2Question and Question2Context Attention Blocks

After we get knowledge aware context and question representation, we combine the context and question using with Context2Question and Question2Context Attention Blocks. The mechanism of attention is the same as attention block in BiDAF. Let the $\widetilde{\boldsymbol{h_{C_i}}}$ be the representation of i-th word in context and $\widetilde{\boldsymbol{h_{Q_j}}}$ be the representation of j-th word in question. We compute similarity matrix $\boldsymbol{S} \in \mathbb{R}^{N \times M}$, which contains a similarity score $\boldsymbol{S_{ij}}$ for each pair $(\widetilde{\boldsymbol{h_{C_i}}}, \widetilde{\boldsymbol{h_{Q_j}}})$ of context and question representation:

$$\boldsymbol{S_{ij}} = Concat(\widetilde{\boldsymbol{h_{C_i}}}, \widetilde{\boldsymbol{h_{Q_j}}}, \widetilde{\boldsymbol{h_{C_i}}} \odot \widetilde{\boldsymbol{h_{Q_j}}})\boldsymbol{W_{sim}}$$

Where $\boldsymbol{W_{sim}} \in \mathbb{R}^{4*H \times 1}$. $\boldsymbol{S}$ contain information about how each context word related to each question word. Next, we can compute Context2Question attention and Question2Context attention. Context2Question attention is computed by:

$$\tilde{\boldsymbol{S}} = softmax(\boldsymbol{S}) \quad \in \mathbb{R}^{N \times M}$$

$$\boldsymbol{a} = \tilde{\boldsymbol{S}}\widetilde{\boldsymbol{h_Q}} \quad \in \mathbb{R}^{N \times H}$$

Here, softmax is taken in each row. Next, the Question2Context attention is computed by:

$$\overline{\boldsymbol{S}} = softmax(\boldsymbol{S}) \quad \in \mathbb{R}^{N \times M}$$

$$\overline{\overline{\boldsymbol{S}}} = \tilde{\boldsymbol{S}}\overline{\boldsymbol{S}}^{\boldsymbol{T}} \in \mathbb{R}^{N \times N}$$

$$\boldsymbol{b} = \overline{\overline{\boldsymbol{S}}}\widetilde{\boldsymbol{h_C}} \quad \in \mathbb{R}^{N \times H}$$

Lastly, the question aware context representation is computed by:

$$\boldsymbol{g_C} = Concat(\widetilde{\boldsymbol{h_C}}, \boldsymbol{a}, \widetilde{\boldsymbol{h_C}} \odot \boldsymbol{a}, \widetilde{\boldsymbol{h_C}} \odot \boldsymbol{b}) \quad \in \mathbb{R}^{N \times 4*H}$$

$\boldsymbol{g_C}$ can be view as combine both context, question, and extra knowledge.

3.9. Stacked Modeling Encoder Blocks

Next, we use Stacked Modeling Encoder Blocks to further process our context representation $\boldsymbol{g_C}$. We first project $\boldsymbol{g_C}$ back to hidden size H. Then input $\boldsymbol{g_C}$ to Stacked Modeling Encoder Blocks. The structure of modeling encoder blocks is the same as embedding encoder blocks. But in modeling encoder blocks, the number of encoder layer is 7, The kernel size of conv-layer is 5 and the number of conv-layers in each encoder layer is 2. We stack 3 modeling encoder blocks and share the weights in each block. Finally, we can get representation $\boldsymbol{m_0}, \boldsymbol{m_1}, \boldsymbol{m_2} \in \mathbb{R}^{N \times H}$.

3.10. Answer Extraction Blocks

Finally, we extract answer prediction from our representation $\boldsymbol{m_0}, \boldsymbol{m_1}, \boldsymbol{m_2}$. Let be learnable parameters $\boldsymbol{W_{start}}, \boldsymbol{W_{end}} \in \mathbb{R}^{2*H \times 1}$, the distribution of start position and end position is computed by:

$$P_{start}(\boldsymbol{C}) = softmax(Concat(\boldsymbol{m_0}, \boldsymbol{m_1})\boldsymbol{W_{start}}) \in \mathbb{R}^{N \times 1}$$

$$P_{end}(\boldsymbol{C}) = softmax(Concat(\boldsymbol{m_0}, \boldsymbol{m_2})\boldsymbol{W_{start}}) \in \mathbb{R}^{N \times 1}$$

4 Experiments

4.1.Dataset

We evaluate our model in Stanford Question Answering Dataset(SQuAD)2.0[10]. SQuAD2.0 is a popular machine comprehension dataset consisting of more than 100000 (context, question, answer) triples. The context is a paragraph from Wikipedia articles, questions are collected by crowd workers. If question has answer, the answer must be a span in the context, or "No answer" is answer cannot be found in context. Here, we use a slightly modified version of dataset get from CS224N default final project, where the training dataset is identical, but the dev dataset is only a half of original dev dataset. Finally, our dataset contains 129941 training samples and 5951 dev samples.

4.2. Evaluation Method

We use two metrics to evaluate our model, which are Exact Match(EM) and F1 scores. EM measures whether the answer span matches exactly with the ground truth answer. F1 scores which measure the weighted average of precision and recall, where precision is calculated as the number of correct words divided by the length of the predicted answer, and recall is calculated as the number of correct words divided by the length of the ground truth answer. For evaluation, the predicted answer is measured against 3 human answers for each question, and the highest score among the three is recorded.

4.3. Training Details

Data Preprocessing. We use spaCy to preprocess data. The maximum context length is set to 400 during training and the contexts that have length longer than 400 will be ignored. The maximum length of answer is set to 15. We use the pretrained 300-D GLoVe[11] word embedding. All the out-of-vocabulary words are replaced with special token <OOV>, initialized with random value. The char embedding is also initialized with random value and fine-tune during training.

Model Setting. Regularization is used in the model. To be more specific, we apply L2 weight decay on all the trainable parameters with $\lambda = 3 \times 10^{-7}$. We also apply dropout

on the word, char embedding and each layer with dropout rates set as 0.1. Meanwhile, similarly as QANet, we adopt layer dropout[12]to each layers in embedding encoder blocks and modeling encoder blocks with probability $p_l = 1 - \frac{l}{L}(1 - p_L)$, where l is current layer and L is the last layer in the block, $p_L = 0.9$. We experiment two model setting, KnowGQA-small has hidden size of 96 for both Residual GCN block, embedding encoder block and modeling encoder block and heads of 1 in two encoder blockw. The hidden size and the number of heads for KnowGQA-large is 128 and 8, respectively. Batch size is 16. Loss function is the sum of the negative log-likelihood (cross-entropy) loss for the start and end distribution. The optimizer is set to Adadelta.

Prediction. During prediction, we discretize the soft predictions of the model to get start and end indices. We choose the answer span (i, j) that maximizes $p_{start}(i) * p_{end}(j)$ subject to $i \leq j$ and $j - i + 1 \leq L_{max}$, where L_{max} is the maximum length of answer span. To equip our model with the ability to make no-answer predictions, we prepend <NULL> token to the beginning of each context. During prediction, if $p_{start}(0) * p_{end}(0)$ is greater than any other span, then model will give no-answer, which is introduced in[13].

Evaluation. During evaluation, exponentially weighted moving average of model parameters is applied with decay rate 0.999.

4.4. Results

We compared our model with BiDAF and standard QANet model, where in QANet model, we experiment two different setting, QANet-small have hidden size 96 and number of headers 1, which is the same with our model. In QANet-large, we set the hidden size to 128 and number of headers is 8, which is default in QANet paper. The training result is show in Figure 2.

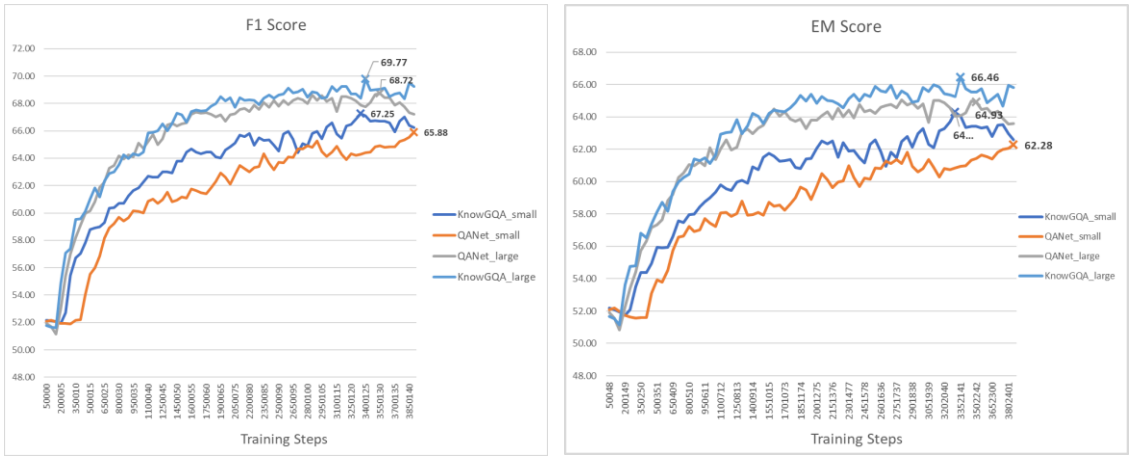

Figure 2 F1 score and EM score of each model during training

Table 1 EM score and F1 score in dev dataset

Model	F1	EM	AvNA
BiDAF-nochar	61.52	57.89	68.56
BiDAF-char	64.55	60.91	71.08
QANet-small	65.64	62.21	71.7
QANet-large	68.57	64.95	74.72
KnowGQA-small	67.25	64.29	72.86
KnowGQA-large	**69.77**	**66.46**	**75.11**

Table 1 shows the EM and F1 result for each model in modified dev dataset. Notice that BiDAF-nochar is the baseline model in CS224 project starter code. The only difference between BiDAF-char and BiDAF-nochar is that the former combine char embedding, which is the default BiDAF in original paper. Compare KnowGQA-small and QANet-small, we can see that the F1 and EM score increased by 1.61 and 2.08 respectively after we integrate extra knowledge into model. Meanwhile, the F1 score and EM score of KnowGQA-large increased by 1.2 and 1.51 respectively compare to QANet-large. The result demonstrates the effect of adding extra knowledge using our proposed knowledge attention mechanism. We notice that the improvement of performance due to knowledge attention is relatively small compare KnowGQA-small and KnowGQA-large. We think it is because large model size gives model more power to learn some latent relationship inside the context.

An interesting thing is that KnowGQA always converge faster and achieve best score earlier than QANet. This may because the extra knowledge give model an outline to learn the latent relationship and representation, like a teacher teach student how to learn. The effect of knowledge attention is some kind like pre-trained word embedding.

However, a wired thing is that model will have better performance if we freeze the randomly initialized char embedding than making it trainable during training. We are not sure why this is the case now and more experiments are needed to analysis this.

4.5. Error Analysis

Wait for future analysis.

5 Conclusion and Future Work

Our model demonstrates a general way to integrate extra knowledge using GCN and knowledge attention, which achieve good performance in experiments. However, due to time limit, we have not yet tested more experiments. We will further work on it in the future. The future work will incl

ude: 1. Evaluate on whole dev dataset and test dataset. 2. Improve GCN architecture. 3. Combine graph word embedding. 4. Test knowledge attention in other model structure. 5. Detailed error analysis.

6 Acknowledgment

Thank for Standford University and teacher group of CS224N for providing such wonderful course and online-free material. It really helps me learn both fundamental and state-of-art techniques in NLP field, which also inspired me the interest to explore more in this field. Meanwhile, thanks for the starter code in this default project.

[1] M. Peters *et al.*, “Deep contextualized word representations,” Feb. 2018.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018.

[3] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, “Bidirectional Attention Flow for Machine Comprehension,” Nov. 2016, Accessed: Aug. 22, 2020. [Online]. Available: https://arxiv.org/abs/1611.01603.

[4] A. W. Yu *et al.*, “QaNet: Combining local convolution with global self-attention for reading comprehension,” 2018.

[5] Y. Shen *et al.*, “Knowledge-aware attentive neural network for ranking question answer pairs,” in *41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2018*, Jun. 2018, pp. 901–904, doi: 10.1145/3209978.3210081.

[6] T. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” Sep. 2016.

[7] R. Speer, J. Chin, and C. Havasi, *ConceptNet 5.5: An Open Multilingual Graph of General Knowledge*. 2019.

[8] A. Vaswani *et al.*, “Attention Is All You Need,” Jun. 2017.

[9] R. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, *Hierarchical Graph Representation Learning withDifferentiable Pooling*. 2018.

[10] P. Rajpurkar, R. Jia, and P. Liang, *Know What You Don’t Know: Unanswerable Questions for SQuAD*. 2018.

[11] J. Pennington, R. Socher, and C. Manning, *Glove: Global Vectors for Word Representation*, vol. 14. 2014.

[12] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, *Deep Networks with Stochastic Depth*, vol. 9908. 2016.

[13] O. Levy, M. Seo, E. Choi, and L. Zettlemoyer, “Zero-Shot Relation Extraction via Reading Comprehension,” Jun. 2017.