Question Answering by Reasoning across common Knowledge bases

Prasanna Patil
Department of
Computer Science
and Automation

Kapil Pathak
Department of
Computer Science
and Automation

Vikram Bhatt
Department of
Computational
and Data Sciences

Abstract

Recent state of the art works in reading comprehension focused on question answering on individual documents. (Cao et al., 2018) proposed a method, which integrates information across documents by posing it as inference problem on graph. Graph Convolution Networks are applied to train the network, to capture the relations between mention of entities. Entity-GCN is compact and scalable because every node can perform differential message passing in parallel fashion, which also makes it feasible to apply frameworks similar to Pregel(Malewicz et al., 2010) or Spark Giraph. We propose graph attention network based model on ARC dataset.

.

1 Introduction

Question Answering is a problem in which a question is posed to an AI system and is expected to formalize an appropriate answer to the question. Recently, deep learning methods are being applied to task and they have shown promising results.

Most of the deep learning methods use some kind of supervised learning where a training dataset and a test dataset is available. Question Answering (QA) is a diverse task where depending upon dataset, the complexity of method being applied changes.

Different kind of datasets for QA have been developed over the past. Some datasets, such as SQuAD (Rajpurkar et al., 2016), include a passage and questions based on the text given in passage. Another dataset is Natural Questions which has a wikipedia page as a passage and a question for each page. The model is required to output a long answer and a short answer for a question.

Here, we are focusing on ARC dataset (Clark et al., 2018). The ARC dataset has multiple choice question where each question has 4 choices. It also comes with a supplementary knowledge base which can be used to answer the questions. More details on dataset in later section. The dataset is compiled with specific focus on questions such that simple algorithms have difficult time to answer the questions. The questions are designed such that multiple level of inference from the knowledge base is required and hence captures machine comprehension task implicitly.

2 Related work

Recently, many deep learning methods have been proposed to address the problem of open domain question answering. The ARC, being relatively new dataset, has been researched slightly less than other datasets which have been available and address similar challenge.

2.1 RNN based approaches

Since, RNNs have been promising in capturing the semantics of a sequence, in the past deep learning methods have been focused on using RNNs to answer such questions. Recent methods include DrQA (Chen et al., 2017) which tries to address open domain QA by reading Wikipedia pages related to the query and using LSTMs to encode question as well as passages extracted from Wikipedia pages. The model then outputs beginning and ending of the answer from one of the candidate passages extracted in the beginning. Hence, it performs extractive question answering. Other approaches such as BiDAF (Seo et al., 2016) and FastQA (Weissenborn et al., 2017) use varients such as BiLSTM and attention to perform question answering.

2.2 Graph based approaches

Recently, a graph convolutional network was proposed which was found useful in question answer-

ing where reasoning across documents is required. One such approach is Entity-GCN(Kratzwald and Feuerriegel, 2018) which extracts named entities from question as well as supporting document and forms a graph between them depending upon whether entities are in same document or different documents. Then, it performs multi step reasoning with the help of a GCN. GCNs have also been applied to task of text classification such as Natural Language Inference (Yao et al., 2018).

3 Problem Definition

We are tackling problem of question answering across common knowledge base. We have a dataset $D = \{(Q_1, C_1, A_1), (Q_2, C_2, A_2), ..., (Q_n, C_n, A_n)\}$ where we have n training instances and each instance is defined by a tuple (Q_i, C_i, A_i) where Q_i is question, C_i are choices and A_i is the correct answer.

Further we have a common knowledge base (KB) which may contain sentences through which we can reason answer for a particular question. However, these sentences are irrelevant and the right set of sentences for a given question is unknown priori.

Knowledge base can be very huge and may contain millions of sentences, many of which are irrelevant for any question in the dataset.

4 Approach

We attempt to solve the question answering problem using graph attention network (Velikovi et al., 2018) and relational graph convolution network (Schlichtkrull et al., 2017). Below we define functional components of our network. Figure to depicts a compact view of our network.

4.1 Extracting entities

Before we perform any preprocessing on dataset, we extract entities from question and choices of each training instance using parts-of-speech tagging. We extract pos tag of each word in sentence (sentence can be a question or a choice). We keep only those words which are Nouns, Proper Nouns, Verbs and Adverbs.

4.2 Finding relevant sentences from KB

The knowledge base of ARC dataset consists of around 14 M sentences and all relevant statements have been scattered across the knowledge base. Hence we need to have a sophisticated method to extract relevant sentences from the knowledge base and filter out other irrelevant sentences.

We used unigrams based method to extract relevant sentences. We first calculate unigram counts for words, which are extracted in above step, in knowledge base and assign a negative log to the normalized unigrams probabilities to calculate weight of each unigram. Hence $W_u = -log(P_u)$ where P_u is unigram probability and W_u is weight of unigram.

Once we have extracted weights as above, we find relevant sentences by using a weighted containment similarity measure as below:

$$sim(q, s) = \frac{\Sigma_{u \in q \, and \, u \in s} W_u}{\Sigma_{u \in q} W_u}$$

We keep top N relevant sentences by ranking sentences of knowledge base on descending order of their containment similarity with question. N is hyperparameter in our network. We call these sentences as supporting sentences of a question.

The core idea behind this method is that this method assigns higher weight to the sentence consists of more rare words matched with the given question. We select top 100 sentences obtained after ordering these sentences according their weights. Here the number of top sentences to be selected is a hyperparameter.

4.3 Generating Graph

Once we have extracted supporting sentences for each question as described above,we generate graph for each question by generating a dependency parsing of each sentence in the set of supporting sentences. Hence, we have a separate graph for each question in the dataset.

Once we have dependency parse of each sentence, we prune the graph as follows:

- We remove dependencies which seem irrelevant for our the purpose of question answering.
- We remove stop words by directly connecting two nodes u and v if they are connected by

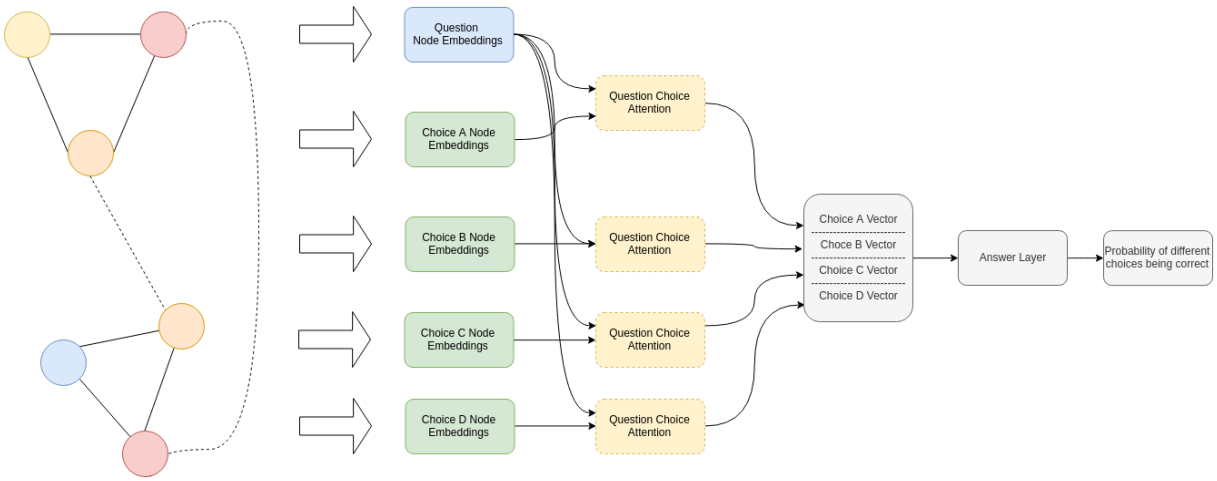

Figure 1: Compact view of relational graph network. (i) Dotted lines represent match based edges while (ii) solid line represents sentence based edges. The Question Choice layer is shared between all choices.

some other node k such that k is a node for stop word.

After pruning the graph, we generate the complete final graph for a given question as below:

1. We have two kinds of edges in the graph: (i) sentence based and (ii) match based.

2. Sentence based edges connect two nodes if they are part of same sentence. Hence these edges in graph are the same ones which are generated by the dependency parse of a sentence.

3. Match based edges connect two nodes across sentences if the entities of the node are same word. Here, we use a direct match heuristic.

4. To further reduce size of the graph, we consider only those nodes which are within a fixed depth of anchor nodes. Anchor nodes are the nodes with word which are part of question or one of the choices.

Two types of edges extracted above represent two relations in our graph on which we use a relational graph convolution network.

4.4 Relational Graph Network

Once we have extracted graph for each question in the dataset we use graph convolution network. There are many components in our network. We describe each one of them in brief.

Initial node embedding:- Initial embedding are pretrained ELMo embedding. Since each node in the graph comes from a supporing sentence, we pass supporing sentence through ELMo to extract nod embedding. We use small ELMo model which outputs 3 vectors, each of dimension 256. We simply concatenate all these 3 embedding and form a 768 dimension embedding for each node.

Question Embedding:- We extract question embedding by first extracting question word embedding for words in question using ELMo and then using Key Value attention as described in (Vaswani et al., 2017). However, we do not use multiheaded attention, instead we restrict it to single head. Moreover, we don't have 3 different projection weights for Query, Key and Value. We use a single weight to project all 3 into same dimension. We also scale this attention output by $\sqrt{dim}$. We apply dropout right before applying attention and after applying linear projection. After Key Value attention we simply do maxpool over each dimension of question word embeddings to get question embedding. The process applied above gives additional regularization effect with fewer trainable parameters.

Question Aware Node Embeddings:- Instead of using node embedding extracted from ELMo as it is we use question aware node embedding for each node. These embedding are generated as below:

$$X_v = ELMo_v * P_q v + Q_e * (1 - P_g)$$

where $ELMo_v$ = ELMo embedding of each node extracted from ELMo and Q_e represents question

embedding. $P_q v$ is a scalar value between 0 and 1. It represents probability of keeping ELMo embedding vs probability of keeping question embedding. The value of $P_q v = f_\sigma(Q_E^T \cdot W \cdot X_v)$ where $W \in R^{dxd}$ where d = dimension of initial embedding (768 in our case). f_σ is sigmoid function.

Relational Graph Convolution Network:- Once we have processed node embedding as described above, we perform several passes of relational graph convolution over the graph of each question. The relational graph convolution is performed as below:

$$X_{ur}^{l+1} = f(X_u^l + \Sigma_{v \in N_r(u)} X_v^l)$$

Here X_{ur}^{l+1} represents embedding of a node u for relation r at layer $l+1$. The function f is any non linear function, we have used a leakyrelu function with parameter value 0.2.

Then we extract node embedding for each layer and apply layer norm to get the embedding of layer $l+1$.

$$X_u^{l+1} = LayerNorm(MaxPool(X_{ur}^{l+1}))$$

Answer Classification:- Once we have node embeddings for final layer of GCN, we perform answer classification as below:

- We extract the node embedding of question nodes from final layer of GCN.
- We extract the node embedding of choice nodes from final layer of GCN.
- Again we apply Key Value based attention to get question aware embedding of choices by setting query Q = question node embedding and K = V = choice node embedding.
- These embedding are then projected to a scalar with the help of a learnable parameter $V \in R^{dx1}$ where d is dimension size of graph layers. Softmax is applied to this scalar and vectors obtained in previous step are combined using these softmax probabilities.
- The parameter vector V is same for all choices as label of a choice doesn't give any information about the choice being correct answer.

Once we have obtained a vector for each choice as described above, it is passed through a softmax linear layer which calculates probability of choice being correct answer. The final loss of network is calculated as cross entropy loss between true labels and predicted labels where true label is the correct choice.

Regularization for Graph Networks At the first site, we found that the model is overfitting on ARC dataset. Hence, to achieve regularization over training dataset, we use a technique described in (Zhang et al., 2019). This trick uses a decoder on top of GCN layers and tries to reconstruct initial embedding of nodes. In our case, decoder is also a muli layer GCN which takes the hidden dimension embedding of each node and tries to reconstruct initial ELMo embedding of each node at final layer. This adds additional loss term corresponding to reconstruction error. Hence, we jointly optimize of

$$\mathcal{L} = \mathcal{L}_{xent} + \alpha \mathcal{L}_{rec}$$

where $\mathcal{L}_{xent}$ is cross entropy loss for correct answer classification and $\mathcal{L}_{rec} = ||X - \hat{X}||_2^2$ where $\hat{X}$ is reconstructed node embedding by decoder GCN. α is a hyper parameter of our network. This gives additional regularization effect with 0.6 percent additional accuracy on validation dataset.

5 Dataset and Metrics

5.1 ARC Dataset

As mentioned in the beginning we are planning to use ARC (Clark et al., 2018) dataset. The ARC dataset has following characteristics:

- Questions with 4 choices.
- A knowledge base consisting 14M lines of text data. The authors hypothesize that knowledge base covers at least 95% of the data.
- A split between Easy and Challenge questions.

The ARC dataset is split as Easy and Challenge. The Easy dataset consists of 2251 training instances, 570 validation instances and 2376 instances for test. The challenge dataset consists of 1119 training, 299 validation and 1172 test instances. We are focusing on Easy dataset as of

now.

The ARC dataset because it contains natural and grade-school questions and it is largest of such datasets. It cover broad variety of topics and comes with a supporting knowledge base (the use of KB for challenge is optional). The models that have been developed for SQuAD and SNLI aren't able to outperform significantly compared to a random baseline. It also covers broad type of questions and reasoning required to answer questions correctly.

5.2 OpenBookQA Dataset

Apart from ARC, Allen Institute also released another dataset named OpenBookQA. It is a new kind of question-answering dataset modeled after open book exams for assessing human understanding of a subject. It consists of 5,957 multiple-choice elementary-level science questions (4,957 train, 500 dev, 500 test), which probe the understanding of a small "book" of 1,326 core science facts and the application of these facts to novel situations. For training, the dataset includes a mapping from each question to the core science fact it was designed to probe. Answering OpenBookQA questions requires additional broad common knowledge, not contained in the book. The questions, by design, are answered incorrectly by both a retrieval-based algorithm and a word co-occurrence algorithm

Metrics:- We use accuracy to measure and compare performance of our classifier.

6 Experimental Setup

Our experimental setup is as below:

- Initial embedding are ELMo embedding of dimension 768.
- Dropout rate is same throught the network and all experiments and it is set to 0.4
- Leakyrelu value is set to 0.2.
- Number of relations is fixed to 2 (sentence based edges, match based edges). We do not consider other experiments where we combine both relation into single relation or introduce other complex types of edges.
- We use StanfordCoreNLP (Manning et al., 2014) for dependency parsing during preprocessing.
- We fix the number of supporting sentences to 100 for all our experiments.
- We also use L_2 regularization over all parameters of the network. The coefficient of L_2 loss is fixed to 0.001.
- We use Exponential Learning rate scheduler and apply it at every 2nd epoch.
- We have fixed batch size to 8 for all of our experiments.

We carry out experiments over both OpenBook and ARC dataset. Note that, to carry out our experiments, we only train network for 5 epochs.

7 Experiment Results

7.1 Initial study

At the very beginning we had a very simple graph attention network model. We had components for generating question aware embedding, question embedding and relational graph convolution network layers, however these layer were designed in very simple manner.

Question embedding were generated by combining question word embedding and using dot product attention to calculate there respective weights. Question aware node embedding were generated by simple linear project of question embedding concatenated with ELMo node embedding to hidden layer dimension size. The outputs of relational graph convolution networks were combined in a similar manner. This caused very high number of parameters in our network.

We trained on ARC dataset with depth set to 2 and embedding dimension set to 128, 256 and 512 on top of this network. Our findings are summarised in table 1. Note that the reported accuracy are best over all epochs.

This caused us to look for more rigorous methods regularization and reduce number of parameters as much as possible. We also tried adding L_2 regularization to above network but we couldn't improve it much further. Finally, we decided the current approach and carried out our

Table 1: Accuracy for different hidden dimensions on simple graph network for ARC dataset

Hidden dim	Training accuracy	Validation Accuracy
128	94.5	33.5
256	96.5	35
512	99.4	39.2

Table 2: Accuracy for different hidden dimensions on simple graph network for OpenBookQA dataset

Hidden Dim	Training accuracy	Validation Accuracy	Loss
128	50.8	45.4	1.16179
256	49	46.6	1.17006
512	48.4	47.4	1.20841

experiments on it.

We also observed that gradients were exploding in such simple network. Hence, we used gradient clipping to normalize gradients to have unit norm. We have kept it as it is in final model.

These results also motivated us to look at different values of depth parameter and increase the dataset size by combining multiple datasets into one. Hence, we also started working on OpenBook dataset in parallel.

7.1.1 Effect of Regularization

We further experimented with models to see whether there is any effect of the decoder reconstruction regularizer that we have applied compared to no regularizer. We trained two models with hidden dimension size 128 and depth set to 5 with 5 layers of GCN.

Regularized network had 2 decoder layers to reconstruct the initial ELMo embedding of nodes and Non-regularized network had no layers for decoder or even reconstruction loss.

7.2 Hidden dimension

We experimented with OpenBook dataset and tried three different hidden dimensions: 128, 256, 512. Our results from the experiments conclude that we should keep dimension to 256 to allow faster learning as well as increase complexity of model. Results are shown in Table 2 and Figure 2.

Our results are shown in the graph above. Higher hidden embedding had lower learning rate, this could be reasoned as number of parameters

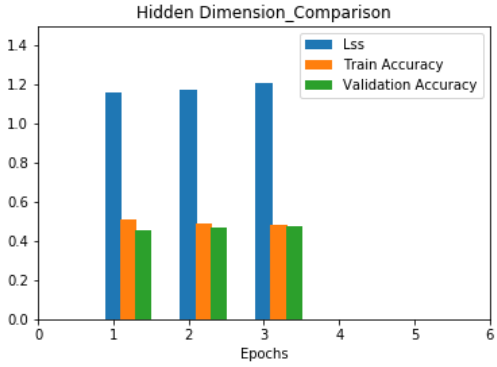

Figure 2: Comparison of different hidden dimension for Graph Attention Network with depth 5.

Table 3: Accuracy for different depths on simple graph network for OpenBookQA dataset

Hidden dim	Training accuracy	Validation Accuracy	Loss
2	50.4	46	1.17374
5	50.8	45.4	1.16179
7	44.2	45.8	1.24562

are more which requires more training time for parameters to adjust.

Our results show that convergence is slower when learning dimensions are lower. This could be because model isn't complex enough to learn patterns quickly.

7.3 Varying depth of the graph

We experimented with various values of depth when generating graph from supporting sentences. Specifically, we tried 3 different values: 2, 5 and 7. Upon manual inspection we found out that sometimes, not enough context was being captured when we limited nodes to be at most 2 hops far from anchor nodes. However, we still went ahead and carried out our experiments.

We found that almost all the networks performed similar when depth was set to 2 or 5 or 7. Their validation accuracy were almost same after every epoch. The results are shown in Table 3 and Figure 3.

We decided to stay with depth of 5 because the number of nodes in depth 7 and 5 were almost similar, meaning that most of the important nodes were only depth 5 far from anchor nodes. This is something we didn't observe in depth 2. Since

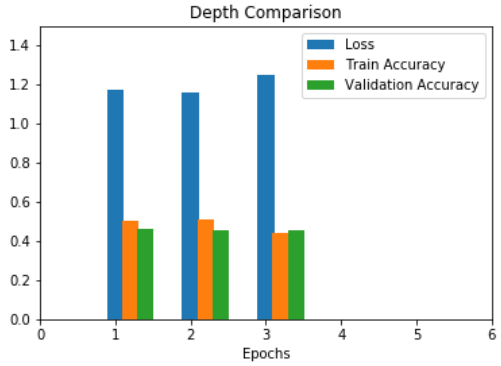

Figure 3: Comparison of different depths for Graph Attention Network with hidden dimension 256.

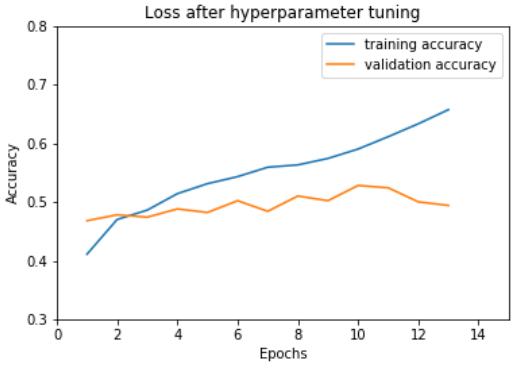

Figure 5: Training and validation loss accuracy for final model

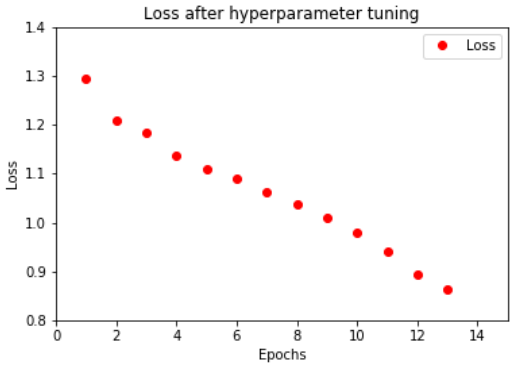

Figure 4: Final model loss after hyper-parameter training.

with depth set to 2, sometimes important context was lost, we decided against it.

7.4 Fine tuning

We weren't able to do much fine tuning due to limited time and lack of available resources. However, we found following steps to be useful. Final accuracy of our model on OpenBook dataset is 52.8%.

- Reduce regularization hyperparemeter α at successive epochs.
- Apply learning rate scheduler at every 2-3 epochs.

The results during fine tuning are shown in Figure 4 and Figure 5. Figure 3 shows loss as number of epochs proceed while Figure 4 shows training and validation accuracy.

We observed that, during training, once validation set accuracy peaked, it would start getting low, along with training accuracy, suggesting that the network is now dominated by reconstruction loss rather than answer loss. We also found that this effect happens earlier if hidden size is larger.

Moreover, from our experience with simpler version of network, we had learned that keeping lower learning rate helped network learn faster but to give network a warm start we start with high learning rate and then reduce learning rate using exponential learning rate scheduler.

8 Future Work

- Try combining OpenBook and ARC dataset to improve results further.
- Use a different method of regularization in which cosine similarity between vectors should match with cosine similarity of initial embedding and decoder generated embedding.
- Use BERT instead of ELMo and finetune over BERT during training.
- Instead of a single head, we can use multi-head attention

References

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2018. Question answering by reasoning across documents with graph convolutional networks. *CoRR*, abs/1808.09920.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. *CoRR*, abs/1704.00051.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind

Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

Bernhard Kratzwald and Stefan Feuerriegel. 2018. Adaptive document retrieval for deep question answering. *CoRR*, abs/1808.06528.

Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

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

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2017. Modeling Relational Data with Graph Convolutional Networks. *arXiv e-prints*, page arXiv:1703.06103.

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

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

Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Dirk Weissenborn, Georg Wiese, and Laura Seiffe. 2017. Fastqa: A simple and efficient neural architecture for question answering. *CoRR*, abs/1703.04816.

Liang Yao, Chengsheng Mao, and Yuan Luo. 2018. Graph convolutional networks for text classification. *CoRR*, abs/1809.05679.

Shengzhong Zhang, Ziang Zhou, Zengfeng Huang, and Zhongyu Wei. 2019. Few-shot classification on graphs with structural regularized GCNs.