Methods for Multi-hop Question Generation

A Project Report

submitted by

SHUBHANGI GHOSH

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.
May 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Methods for Multi-hop Question Generation**, submitted by **Shubhangi Ghosh**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by her under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Mitesh Khapra
Research Guide
Assistant Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Prof. Umesh S.
Research Co-Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: IIT Madras, Chennai

Date: 5th May 2019

ACKNOWLEDGEMENTS

I would like to take this opportunity express my sincere gratitude to my guide **Prof. Mitesh M. Khapra** for his constant encouragement and guidance. I would like to express my sincere gratitude to him for instilling in me a sense of discipline, dedication and systematic approach to work, which he has personally inspired by example. His active and intuitive style of pedagogy has motivated me to pursue Deep Learning as a field. I, thus, hope that the intuition that I have developed over time while working on the project is reflected in this report.

I would like to express my sincere thanks to my friends, Sasank G., Alfred Festus Davidson, Briti Ghosh, Ameet S. Deshpande and Monisha J., for being emotional pillars of support through all the stressful times, and keeping me motivated and cheerful through the process. The intellectual and philosophical discussions with them have helped me learn and grow leaps and bounds, and have played a crucial role in shaping this project.

I would like to thank Anjali Kashyap and Dhanush Bhatt for providing me Amazon AWS credits for running my experiments.

ABSTRACT

KEYWORDS: Encoder-Decoder models, Multi-hop Question Generation, Coverage Mechanism, Graph Convolutional Networks, Hierarchical Encoder

Existing Question-Generation systems often lack the ability to link multiple concepts in relation to a concept, in this case the expected answer. Conventional Encoder-Decoder modules, which often form a basis for question-generation frameworks, process information sequentially, and hence fail to understand common links between disjoint, yet related sequences. Methods to incorporate dependencies between common or related entities across sequences have been studied and implemented through the course of this project have been an endeavour in modelling the current shortcomings.

The two notable techniques that have been ideated and incorporated are: (1) Sentence-level coverage mechanism (2) Graph Convolutional Networks. Incorporating sentence-level coverage mechanism is a novel idea to ensure that the encoder-decoder model has sufficiently attended to all the sentences in the supporting passage while generating the question. The results of this intuition have been largely favourable. The results upon incorporating Graph Convolutional Networks have been unconvincing, however. The shortcomings in training of Graph Convolutional Networks, owing to the structural complexities, have been analysed and the current performance has been adequately justified.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Recent research in natural language sequence generation problems have been greatly dominated by sequence-to-sequence (seq-to-seq) models (Sutskever *et al.* [2014]). A sequence of recursive units at the Encoder end encodes the input sequence. The encoded version is fed to the decoder, which generates the output sequence step-by-step. The elegance in the End-to-End trainability and model flexibility (in terms of there being minimum constraints on input sequence length and output sequence length) of sequence-to-sequence models initially led to their use for a plethora of downstream tasks, for example, Machine Translation, Question Generation, Speech Recognition and Video Captioning. The task that we focus on in this project is that of Question Generation. Enhancements have been made to the Vanilla seq-to-seq model which have led to landmark performance boosts, which have enabled sequence-to-sequence models to achieve competitive and even improved performance over conventional NLP pipelines on a much wider range of tasks than traditional methods.

One of the milestone additions to seq-to-seq models is the attention mechanism (Bahdanau *et al.* [2014]). To overcome the bottleneck of seq-to-seq models in terms of having to express a large input sequence in terms of a single feature vector, this approach proposes to soft-search the input sequence, at every decoder time-step, for relevant tokens to generate the current output token. An additional improvement over the attention mechanism is the coverage mechanism (See *et al.* [2017]). When the output probability distribution at a decoder time-step doesn't clearly indicate any vocabulary word, an *unknown* token is predicted. The coverage mechanism fills in the *unknown* values by replacing them with the highest attended-to input token at the corresponding decoder time-step. The mentioned mechanisms have enabled the model to achieve improved performance with a significantly smaller vocabulary size (160,000 words to 30,000 words) and have thus reduced the model complexity greatly.

Recent observations have shown that seq-to-seq models have the ability to learn to recognize syntactic patterns. General semantic information is introduced into seq-to-seq models through input word embeddings (Pennington *et al.* [2014]). Words which share similar context have closer word embeddings in the embedding space. Through this project, we attempt to introduce input-specific semantic information into seq-to-seq models and study the results.

An ideal problem statement to evaluate our endeavours has been through the task of Multi-hop Question Generation. In the context of the tasks Question Generation and Question Answering, most existing datasets often have the limitation of testing the ability of reasoning within a single paragraph or document, or *single-hop reasoning*. A good way of testing the model's ability to capture input semantics is through *multi-hop reasoning tasks*, where the system has to reason with information taken from more than one document to arrive at the answer. The HotpotQA dataset is used for evaluation of Multi-hop Question Generation.

Two primary approaches were tried to improve the performance of existing models on the given task.

1. Sentence-level coverage mechanism and loss is introduced to enforce the model to attend to all input sentences adequately. We expect that on attending to all input sentences, the model should be able to find semantic links between common entities across sentences better.

2. Graph Convolutional Networks for explicitly providing the model with input

semantic information in the form of entity-linked graphs.

The results of the above approaches are discussed in the report, and their success or failure is accordingly evaluated.

CHAPTER 2

BACKGROUND WORK

Question generation (QG) aims to create natural-sounding questions from a given support paragraph. Question Generation finds significant downstream applications: in the field of education, chatbots, and creating synthetic Question Generation/Question Answering datasets for NLP research.

Classical Question-Generation methods focused on the objective of transforming declarative sentences in the support passage to interrogative ones through rule-based approaches. (Heilman and Smith [2010]) These methods typically involved sophisticated Natural Language Processing (NLP) pipelines for text-parsing and feature-extraction and the model's performance depends critically on the constructed rules. Most importantly, these methods fall short on the fluency-front at generating natural-sounding questions. Natural-sounding questions are mostly *abstractive* in nature and *abstraction* cannot be trivially incorporated through rule-based approaches. This is because rule-based models typically exploit the syntactic role of words rather than their semantic roles.

Inspired by the breakthrough results of neural Machine Translation (Sutskever *et al.* [2014], Bahdanau *et al.* [2014]) , modern approaches to Question Generation (QG) typically frame QG as a sequence-to-sequence learning problem (Du *et al.* [2017]). This approach is a step forward in abstractive question generation as it better models the semantic role of words.

In Sequence-to-sequence models, the encoder iterates over input sequences using recursive units, such as Recurrent Neural Network (RNN) or Long Short

Table 2.1: Examples of transformative and abstractive questions

Passage:	*Oxygen is used in cellular respiration and released by photosynthesis, which uses the energy of sunlight to produce oxygen from water.*
Abstractive question:	*What life process produces oxygen in the presence of light?*
Transformative question:	*Which uses the energy of sunlight to produce oxygen from water?*

term Memory (LSTM) cells (Hochreiter and Schmidhuber [1997]). The decoder generates the output sequence step-by-step.

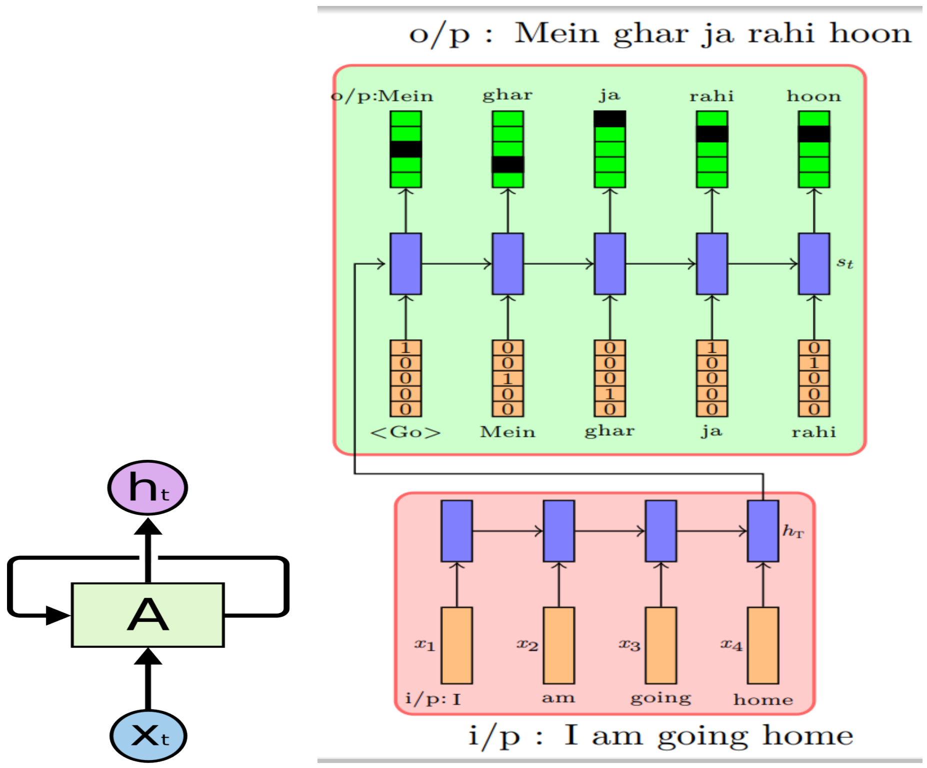

Figure 2.1: Left: An RNN cell, Right: A sequence-to-sequence framework
Image Source: Prof. Mitesh Khapra's slides

RNN Cell – Recursive unit of a Sequence-to-Sequence model

$s_i = \sigma(Ux_i + Ws_{i-1} + b)$

$y_i = \sigma(Vs_i + c)$

Sequence-to-Sequence model

Encoder

$h_i = RNN(h_{i-1}, x_i)$

Decoder

$s_0 = h_T$ (T is the length of the input)

$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$

$p(y_t|y_i^{t-1}, x) = softmax(Vs_t + b)$

Typically, the recursive unit used in a Sequence-to-sequence model is an LSTM cell, which is a more advanced and gated version of the RNN cell. In our method, we have used the Bidirectional LSTM cell, which runs two LSTM sequences – one forward and one backward to incorporate bidirectional context.

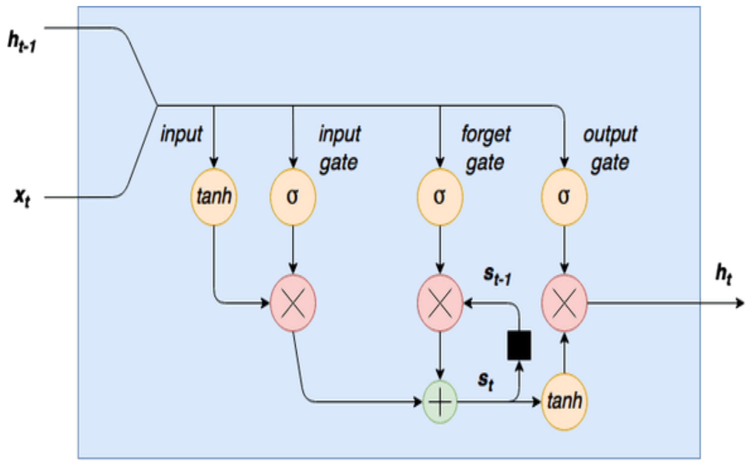

Figure 2.2: An LSTM cell
Image Source: Google Images

The semantics of words is initially captured through pre-trained word embeddings (Pennington *et al.* [2014]). Words which appear in similar context are spatially closer to one-another in the embedding-space. These embeddings are further trained in the context specific to the given dataset, owing to sequential iteration over multiple input sequences.

2.3 ATTENTION MECHANISM

The primary shortcoming of the described sequence-to-sequence model is that, at each decoder time-step, a representation of the entire input sequence is presented to the decoder cell as input. However, focusing on a certain portion of the input sequence while generating the output token at a particular time-step would definitely be more optimal.

For example, in an instance of Machine Translation,
To translate from a language that follows a Subject-Verb-Object order, say English, to a language which follows a Subject-Object-Verb order, say Hindi,

Table 2.2: Attention mechanism for Translation

Main	aam	khaata hoon
I	eat	a mango.

In the second decoder time-step, while generating the output token 'aam', we need to pay *attention* to the fourth word in the encoder sequence, 'mango'. Similarly, for every output token in a sequence-generation task, there is usually a particular input token that needs to be attended to more. Thus, attending selectively to different parts of the input sequence at different decoder time-steps is more beneficial than feeding a representation of the entire input sequence at all decoder time-steps.

The attention mechanism (Bahdanau *et al.* [2014]) is implemented as:

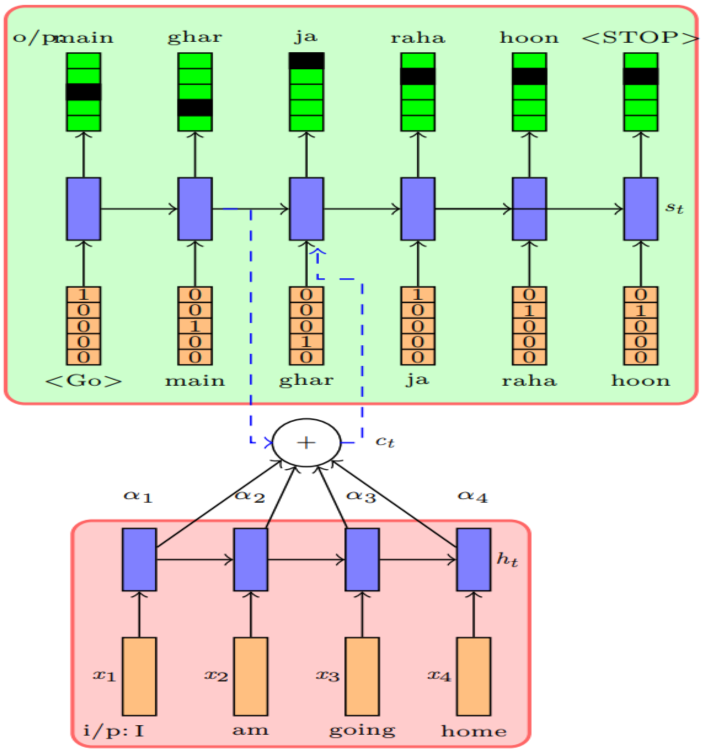

Figure 2.3: Attention Mechanism
Image Source: Prof. Mitesh Khapra's slides

$$\alpha = softmax(V_{attn}^{T} tanh(U_{attn}\mathbf{s_{t-1}} + W_{attn}\mathbf{h_j})) \tag{2.3.1}$$

where, α - vector of attention weights, **h** - vector of encoder states, **s** - vector of decoder states.

The attention weights are calculated at each decoder time-step. The encoder states are weighted by the attention weights calculated at the present time-step and fed to the decoder cell.

2.4 METRICS

The metrics that have been used for evaluating the performance of the models are as follows:

1. **BLEU - Bilingual Evaluation Understudy score** (Papineni *et al.* [2002])
   This score was first used to evaluate the quality of translated texts, hence the term 'bilingual'. However, it is a universal metric for text generation which compares the quality of the generated text with a given reference text. The BLEU-n score reports a precision values for the matching n-gram counts between the candidate and the reference text, adjusted for the number of words being different in the above-mentioned documents.
2. **ROUGE-L** (Lin [2004])
   ROUGE-L score reflects Longest Common Subsequence-based statistics. The Longest Common Subsequence is between the candidate text and the reference text.

2.5 BASELINE MODEL FOR QUESTION GENERATION

The seminal work that forms the baseline for neural Question Generation is Du *et al.* [2017] . This is the first work that uses the end-to-end sequence-to-sequence models for this task rather than a pipeline of classical NLP techniques. Three levels of hierarchies at the encoder stage is presented: word-level, sentence-level and paragraph-level. Bidirectional LSTMs are used as the recursive units of the Encoder and the Decoder. Pre-trained GloVe embeddings ((Pennington *et al.* [2014]) for words have been used. Attention mechanism is used at word-level. The highest BLEU-4 score of **12.28** is obtained when sentence-level encoding is used. Supplementing with paragraph-level encoding diminishes the BLEU-4 score slightly.

Table 2.3: Baseline model scores

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE$_L$
Vanilla seq-to-seq without attention	31.34	13.79	7.36	4.26	29.75
Baseline with sentence-level encoding	**43.09**	**25.96**	**17.50**	**12.28**	**39.75**
Baseline with paragraph-level encoding	42.54	25.33	16.98	11.86	39.37

2.6 DATASET

The specific type of Question-Generation that has been targeted in this project is Multi-hop Question Generation. The task of Multi-hop Question Generation aims at generating questions which require finding a common concept and reasoning over multiple supporting documents. This task is quite apt on the HotpotQA dataset (Yang *et al.* [2018]), which presents a diverse set of questions not constrained to any pre-existing knowledge base. Although the intended use of the dataset is for Question-Answering tasks, we have structured the inputs appropriately and used the same for the task of Question Generation.

The questions presented by the HotpotQA dataset can be primarily classified into two types:

1. Bridge Questions
   These questions are framed connecting the descriptors of a common entity across two distinct supporting documents.
2. Comparison Questions
   These questions are framed comparing instances based on a common entity across two distinct supporting documents.

Each datapoint in the dataset has four components: 1. Answer 2. Question 3. Context Documents 4. Selected sentences from all the context documents which form the principal supporting facts for the Question Answering/ (in our case) Question Generation task.

In training our model, we have used supporting facts as the context passage for running and analyzing most of the experiments.

We have tried certain approaches to improve upon the baseline scores for Question Generation in our model. The following section reviews the literature and details the intuition and implementation of these cornerstone methods.

2.7 ADDITIONAL APPROACHES FOR IMPROVEMENT

2.7.1 Gated Self-Attention Zhao *et al.* [2018]

Long text has posed challenges for sequence to sequence neural models in question generation. In reality, however, it often requires the whole paragraph as context in order to generate high quality questions.

Specifically to the task of Multi-hop question generation, this approach is useful because gated self-attention is a step forward in embedding intra-passage dependency. Thus, now the encoder representation of entities which are common across supporting documents will carry contextual information from all the supporting documents.

Gated self-attention is implemented in two steps:

1. **Self-representation**

   $\mathbf{a_t^s} = softmax(\mathbf{U^T W_s u_t})$

$\mathbf{s_t} = \mathbf{U}\mathbf{a_t^s}$

where,
$\mathbf{U}$ - n x d matrix of all the word embeddings, n - number of words, d - dimensionality of embeddings
$\mathbf{W^s}$ - trainable weight matrix
$\mathbf{s_t}$ - weighted sum of all words encoded representation in passage based on their corresponding matching strength to current word at t.
$\mathbf{s} = \mathbf{s_t}_{t=1}^{M}$ - final self matching representation

2. **Gating**

   $\mathbf{f_t} = tanh(\mathbf{W^f}[\mathbf{u_t}, \mathbf{s_t}])$
   $\mathbf{g_t} = tanh(\mathbf{W^g}[\mathbf{u_t}, \mathbf{s_t}])$
   $\hat{u}_t = \mathbf{g_t} \bigodot \mathbf{f_t} + (1 - \mathbf{g_t}) \bigodot \mathbf{u_t}$

   where,

   The self matching representation $\mathbf{s_t}$ is combined with original passage-answer representation $\mathbf{u_t}$ as the new self matching enhanced representation $\mathbf{f_t}$. A learnable gate vector $\mathbf{g_t}$, chooses the information between the original passage-answer representation and the new self matching enhanced representation to form the final encoded passage-answer representation $\mathbf{\hat{u}_t}$.

For example,

Table 2.4: Gated self-attention

Passage:	Scott Derrickson: Scott Derrickson is an American director, screenwriter and producer. Ed Wood: Edward Davis Wood Jr. was an American filmmaker, actor, writer, producer, and director.
Question:	Were Scott Derrickson and Ed Wood of the same nationality?

Gated self-attention enables the encoder representations to link the word *American* to its two distinct contexts, *Scott Derrickson* and *Ed Wood*, thus enables the model to generate the right question.

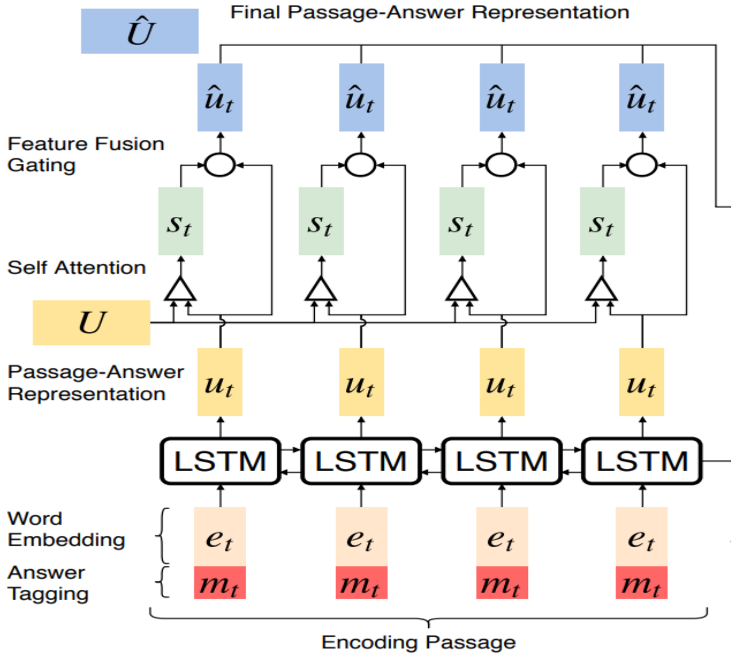

Figure 2.4: Encoder structure for Gated Self-attention

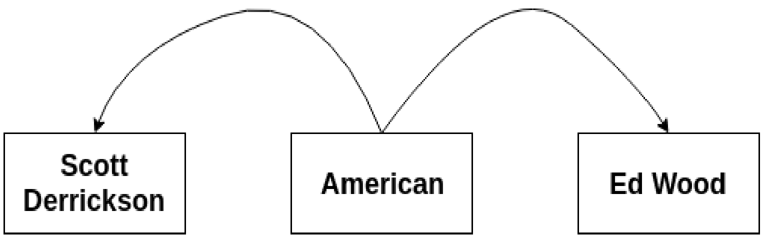

Figure 2.5: Gated Self-attention example
Image Source: Zhao *et al.* [2018]

2.7.2 Co-attention Seo *et al.* [2016]

This method is used to compute answer-aware representations of passage words. These answer-aware representations are passed on to the Encoder LSTMs and attended over in the conventional fashion.

The answer-aware representations of passage words are freshly computed at every time-step to prevent *early summarization*.

The computation of representations do not depend on the previous decoder state and hence, are *memoryless*. The attention weights computed by the Encoder LSTMs over these answer-aware representations of passage words depend on the previous decoder state. This simplification leads to the division of labor between the answer-passage co-attention layer and the Encoder attention layer.

$$\alpha(\mathbf{h}, \mathbf{u}) = \mathbf{w}_{(\mathbf{s})}^{\mathbf{T}}[\mathbf{h}; \mathbf{u}; \mathbf{h} \circ \mathbf{u}]$$

where,

$\mathbf{h}$ - passage word representation

$\mathbf{u}$ - encoder answer-phrase state representation

$\mathbf{w}_{(\mathbf{s})}$ - trainable weight matrix

$\alpha(\mathbf{h}, \mathbf{u})$ - answer-aware passage word representation

2.7.3 Word-level Coverage Mechanism See *et al.* [2017]

Repetition of output tokens is commonly observed in sequence-to-sequence models. We adapt the *coverage model* to solve this problem. We maintain a coverage vector c^t, which is the sum of attention distributions over all previous decoder time-steps.

$$c^t = \Sigma_{t'=0}^{t-1} a^{t'}$$

Intuitively, c^t is a (unnormalized) distribution over the source document words that represents the degree of coverage that those words have received from the attention mechanism so far. The coverage vector is used as extra input to the attention mechanism. Thus, the coverage term is added to the attention mechanism described in Section 2.3, Equation 2.3.1.

$$\alpha = softmax(V_{attn}^{T} tanh(U_{attn}\mathbf{s_{t-1}} + W_{attn}\mathbf{h_j} + C_{attn}\mathbf{c^t}) \tag{2.7.1}$$

A coverage loss is additionally defined to penalize attending to the same input tokens repeatedly.

$$covloss_t = \Sigma_i min(a_i^t, c_i^t) \tag{2.7.2}$$

The coverage loss is bounded; $covloss_t \leq \Sigma_i a_i^t = 1$. The loss intuitively minimises the attention over maximally covered locations.

2.7.4 Pointer-Generator Copy Mechanism See *et al.* [2017]

This mechanism is an elegant way of handling Out-of-Vocabulary (OOV) tokens generated at the decoder end. When an OOV token is generated by the decoder, it is replaced by the input token with the highest attention weight assigned at that decoder time-step.

This enables reducing the vocabulary size at both source and target end. We have used a vocabulary size of 30K at both source and target end, while conventionally vocabulary sizes of around 150K at source end and 60K at target end are used in the absence of Copy mechanism Nallapati *et al.* [2016].

2.7.5 Hierarchical Encoder with Sentence-level and word-level attention Nallapati *et al.* [2016]

In the context of multi-hop question generation, in addition to paying attention to key words at every decoder time-step, it is also important to identify key sentences which detail the properties in relation with the bridging or comparison entity in the HotpotQA dataset.

The difference between the approach detailed in this paper and the Baseline sentence-level hierarchical model is that this model incorporates both sentence-level attention and word-level attention in conjugation, whereas the baseline model incorporates only word-level attention.

In this approach, the word-level attention weights are re-weighted by the sentence-level attention weights.

$$\alpha(j) = \frac{\alpha_w(j)\alpha_s(s(j))}{\Sigma_{k=1}^{N_d}\alpha_w(k)\alpha_s(s(k))} \tag{2.7.3}$$

where,

$s(j)$ denotes the sentence to which the word j belongs.

α_w denotes initial word-level attention weights.

α_s denotes sentence-level attention weights.

α denoted weighted word-level attention weights.

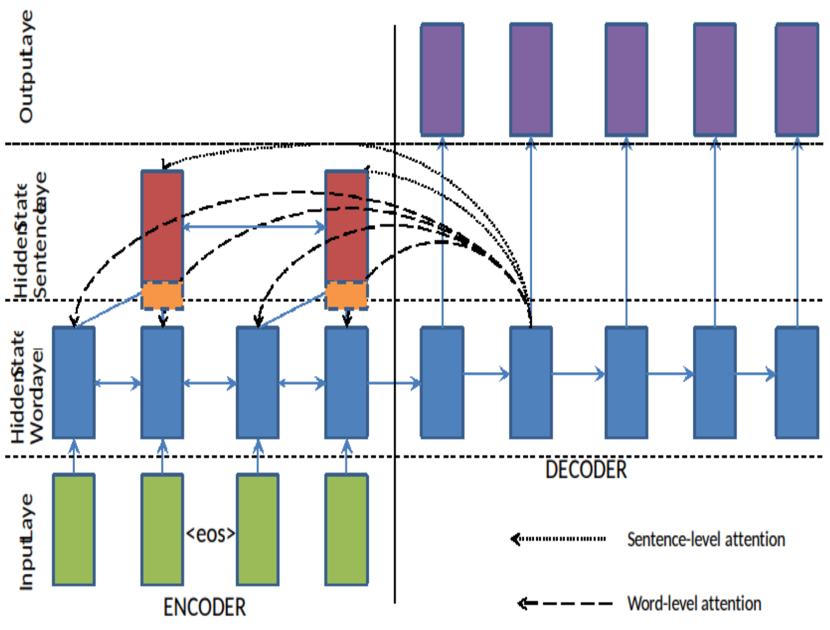

Figure 2.6: Hierarchical Encoder with both word-level and sentence-level attention
Image source: Nallapati *et al.* [2016]

The Basic sequence-to-sequence model supplemented by word-level attention + word-level coverage + co-attention + gated self-attention + copy mechanism shall be referred to as *Primary model* in future references.

CHAPTER 3

RELATED WORK

Graph Convolutional Networks (GCNs) have been used to incorporate non-sequential relations between context passage words/entities. The intuition behind using GCNs for Multihop Question Generation is that the properties that describe the entities and therefore expected to form the question are often non-linearly linked to the entity. The objective of using GCNs is to make these non-linear connections explicit to obtain more useful input entity representations.

This section aims to convey of the general overview about the functioning of GCNs.

3.3.1 Graph Convolutional Networks Kipf and Welling [2016]

GCNs devise a way to generalize neural networks to work on arbitrarily structured graphs. GCNs have primarily been used so far in node-classification tasks in complex graphs: for example, classifying Technology and Fashion pages on social media platforms, such as Facebook, classifying documents into different areas of research on citation networks.

A GCN takes as input:

1. Input feature matrix X: N x D, N nodes and D-dimensional node embeddings
2. A matrix representation of the graph - typically the adjacency matrix, A

For our task of Question Generation, we have used two kinds of adjacency matrices.

1. Dependency parses
2. Capitalized entity linking

Thus, for our task, the adjacency matrix changes for each input point. This makes our task significantly more complex than page-classification in social media networks or document-classification in citation networks, since for those tasks the adjacency matrix remains fixed across datapoints.

Every GCN layer can be written as a non-linear function:

$$H^{(l+1)} = f(H^{(l)}, A) \tag{3.3.1}$$

where,
$H(l)$ - layer-wise outputs of the GCN
$H(0) = X$

The non-linear function $f(.)$ is typically modelled as:

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}) \tag{3.3.2}$$

where,
$W^{(l)}$ is a trainable weight matrix. Each row and column of the adjacency matrix, A is typically normalized by the node degree, to ensure that GCNs don't change the scale of the input feature vectors. Also, multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself. We can "fix" this by enforcing self-loops in the graph: we simply add the identity matrix to A.

Thus, the final expression for the non-linearity function looks like:

$$f(H^{(l)}, A) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{3.3.3}$$

where,

$\hat{A} = A + I$

$\hat{D}$ = diagonal degree matrix of $\hat{A}$

Intuitively GCNs can be visualised as a generalization of our known and loved Convolutional Networks.

A fully-connected neural network requires all input nodes to be connected to all output nodes.

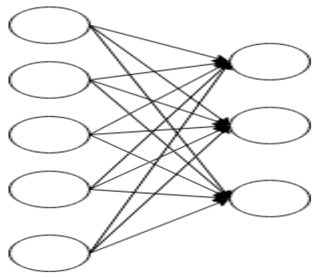

Figure 3.1: Fully connected Network

For data where there is a notion of spatial proximity and spatially distant connections do not yield much information, typically image datasets, Convolutional Neural Networks (CNNs) are used. The spatially distant connections are eliminated and weights are shared among spatially local filters.

Figure 2.3 depicts one layer of a Convolutional Neural Network. The first layer of the CNN identifies primary features (say edges), the second layer identifies secondary features which are an aggregation of the primary features (say shapes)

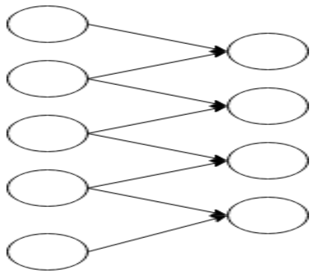
Figure 3.2: Convolutional Neural Network layer

and so on.

For data consisting of arbitrary graphs, where there is no notion of spatial proximity, yet there is a notion of neighbours and possible-communities, GCNs are used.

Figure 2.4 depicts one partial layer of a Graph Convolutional Network. Only two sets of neighbours have been shown to ease the visual complexity of the figure.

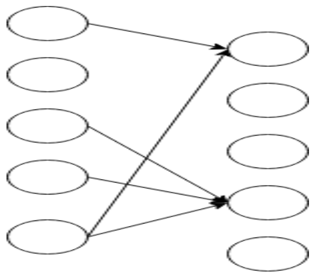
Figure 3.3: Graph Convolutional Network layer

Graph Convolutional Networks are a generalization over CNNs in the following ways:

1. There is no spatial ordering of nodes.
2. Each node may have a different number of neighbours (thus, weight sharing is not possible)

GCNs also perform layer-wise feature-abstraction. The proof of this is through fundamentals of spectral graph theory as detailed here Defferrard *et al.* [2016].

GCNs can identify community structures in a graph even with untrained weights by exploiting the loops in a given graph.

CHAPTER 4

CURRENT APPROACH

Through this project, I have come up certain ideas to supplement the primary model with principal objective of improving performance on the task of *Multi-hop Question Generation*. The intuition behind these ideas and their implementation have been detailed in the current section. The performance and analysis of the same has been presented in latter sections.

Three primary approaches have been tried:

1. Sentence-level coverage mechanism
2. GCNs for supplementing input embeddings, trained with the adjacency matrices:
   (a) Dependency parses
   (b) Linking capitalized entities
3. GCNs for supplementing primary model with sentence-level coverage mechanism

4.5 Sentence-level coverage mechanism

The task of Multi-hop Question Generation involves linking properties of common entities across multiple support documents. To be able to link properties of an entity across multiple sentences, all the sentences should be sufficiently attended to. Thus, to ensure that attention is adequately distributed across all sentences and no sentence is attended to repeatedly, sentence-level coverage mechanism is incorporated.

Hierarchical encoder with word-level and sentence-level attention is implemented as detailed in Section 2.7.5. Coverage mechanism, as detailed in Section 2.7.3, is implemented for both word-level attention and sentence-level attention.

The sentence-level coverage loss and the word-level coverage loss are both added to the standard cross-entropy loss, with a weighting factor of 1.0 (as the cross-entropy loss and the coverage losses are observed to be of a similar order of magnitude).

Sentence-level attention and coverage

$$c_s^t = \Sigma_{t'=0}^{t-1} \alpha_s^{t'}$$

$$\alpha_\mathbf{s} = softmax({V_{attn}^s}^T tanh(U_{attn}^s \mathbf{s_{t-1}} + W_{attn}^s \mathbf{h_j} + C_{attn}^s \mathbf{c_s^t}) \tag{4.5.1}$$

$$covloss_t^s = \Sigma_i min(\alpha_{si}^t, c_{si}^t) \tag{4.5.2}$$

where,

c_s - coverage vector for sentence

α_s - attention weights for sentence

i - encoder steps, t - decoder steps

h - encoder state, **s** -decoder state

Word-level attention and coverage

$$c_w^t = \Sigma_{t'=0}^{t-1} \alpha^{t'}$$

$$\alpha_{\mathbf{w}} = softmax({V_{attn}^{w}}^{T} tanh(U_{attn}^{w} \mathbf{s_{t-1}} + W_{attn}^{w} \mathbf{h_j} + C_{attn}^{w} \mathbf{c_w^t}) \quad (4.5.3)$$

$$\alpha(j) = \frac{\alpha_w(j)\alpha_s(s(j))}{\Sigma_{k=1}^{N_d} \alpha_w(k)\alpha_s(s(k))} \quad (4.5.4)$$

$$covloss_t^w = \Sigma_i min(\alpha_i^t, c_{wi}^t) \quad (4.5.5)$$

where,

c_s - coverage vector for sentence

α_w - attention weights for words

α - sentence-attention weighted attention weights for words

i - encoder steps, t - decoder steps

h - encoder state, **s** -decoder state

4.6 Graph Convolutional Networks

GCNs were largely used with the intention to model non-linear sequential relationships between the words in a given document. These non-linear dependencies are incorporated as a part of the input feature representation.

Two kinds of adjacency matrices were used as input to the GCN (in disjoint experiments):

1. **Dependency Parses**
   The dependency parses were generated by the Spacy Natural Language Processing toolkit. This approach enhances connections between entities and the properties that describe it.

2. **Capitalized Entity Linking**
   This approach enhances connections between common entities across different supporting documents.

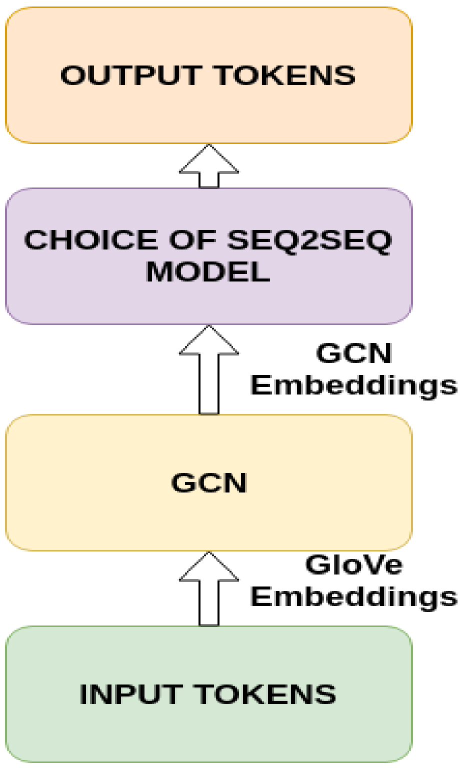

Figure 4.1: Model with GCN flow diagram

CHAPTER 5

Experimental Results and Analysis

Maximum number of epochs run for all experiments = 20

Learning Rate = 0.0004

Optimizer = Adam

Batch-size = 64

5.6 Sentence-level coverage

Weighting factor for Sentence-level coverage loss = 1.0

Table 5.1: Sentence-level Coverage

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE$_L$
Primary model	36.7	26.1	19.5	15.0	34.2
Primary model + sentence-level coverage	**37.5**	**26.8**	**20.2**	**15.7**	**34.6**

Sentence-level coverage improves the **BLEU-4** score by **0.7 points** over our primary model. The intuition behind sentence-level coverage, thus, seems largely successful.

A visual analysis of the attention-plots reveals that upon incorporating sentence-level coverage mechanism, the model attends over both the support sentences

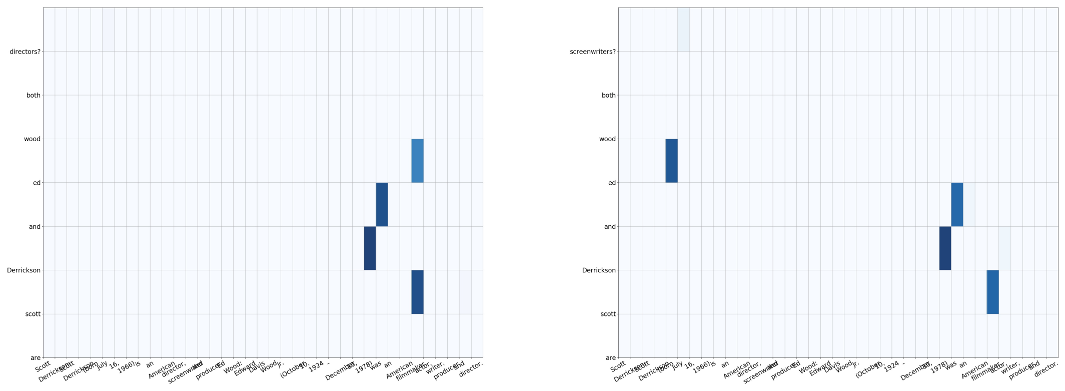

Figure 5.1: Left: Word-level coverage Right: Sentence-level coverage

more proportionately. This enables the model to take a step forward in identifying the common entities shared by *Scott Derrickson* and *Ed Wood*, as it now attends sufficiently to supporting facts that discuss both these entities.

5.7 Graph Convolutional Networks

Experiments have been run with two different kinds of adjacency matrices as input to GCNs.

1. Depencdency Parse created using Spacy, an existing NLP toolkit.
2. Capitalized entity linking across the passage.

GCNs have further been run in conjugation with Sentence-level coverage mechanism.

To our disappointment, the performance of GCNs has been greatly dismal. The poor performance of GCNs, as implemented, on the current task of Question-Generation can be attributed to three primary intuitions.

1. The tasks that GCNs have been greatly successful on, typically classfication tasks, have one adjacency matrix across all input datapoints. For example, in

Table 5.2: Sentence-level Coverage

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE$_L$
GCN with Spacy dependency parse	24.2	13.1	7.7	4.8	25.6
GCN with Entity graph	26.8	16.0	10.6	7.3	27.3
GCN with Spacy dependency parse + Sentence-level coverage	24.5	12.6	7.0	4.1	23.4
GCN with Entity graph + Sentence-level coverage	30.7	18.4	11.9	8.0	27.8

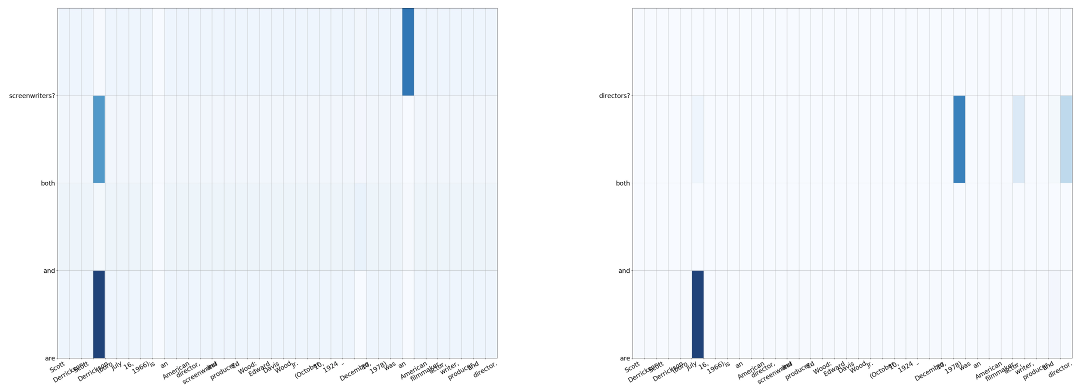

Figure 5.2: GCN: Left: Spacy dependency parse Right: Entity graph

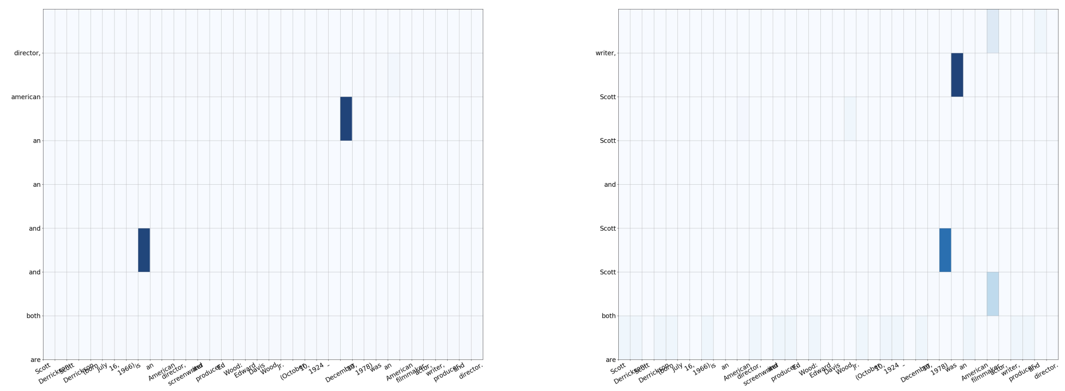

Figure 5.3: GCN + Sentence-level coverage: Left: Spacy dependency parse Right: Entity graph

a document classification task over a citation network, the adjacency matrix that describes the citation network is fixed across all input datapoints to be classified.

In the task of Question Generation, however, the adjacency matrix changes with every input passage, thus rendering the each element of the GCN weight-matrix to be very sparsely trained.

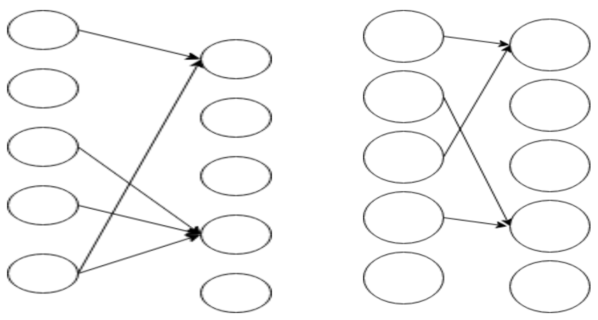

Figure 5.4: GCN weights that are being trained at two distinct datapoints

GCNs are a generalization over CNNs in the sense that each node may have a different number of neighbours. For our task, the neighbours of each numbered-node has is changing across input datapoint. Thus, each element in the weight-matrix, is not trained while iterating over every input.

To showcase the sparsity in training GCNs in a more pronounced manner, The maximum number of words in each context passage is 100. The size of the adjacency matrix for every input is, thus, 100 x 100. There $2^{100x100}$ possible adjacency matrices across all inputs, whereas the HotpotQA dataset provides only 86,400 training points, which is orders of magnitude lower than the number of adjacency matrices. A majority of the weights in the weight matrix may not get trained at all!

Imagine how challenging it would be to train a CNN where the size or structure of the filters changed with each input point.

2. Another issue with the complexity of trainability of GCNs is that, if the weights aren't trained well, in the process of obtaining the node's final embedding, you could be enforcing more information about the node's neighbours than optimal for the task. This might be responsible for diluting the amount of *information* present in the node's original embedding. This is termed as *Node Information Morphing* Vijayan *et al.* [2018].

   For example,

   In a dependency parse of the passage sentence in Table 5.3, the words, '*The* and '*dog*' are related. When we use GCN output embeddings to generate a question based on *dog*, we have lost some information about *dog* as compared to its original GloVe embedding. This is likely to reduce the quality of the question generated.

Table 5.3: Node Information Morphing

Passage: Given Answer:	*The dog jumps over the fence.* *dog*
Expected question to be generated:	*What jumps over the fence?*

3. On analysing the questions generated, we observe that some words appear in loop on using GCN embeddings for question generation. This phenomenon is not observed when GCN embeddings are not used. The loops present in the adjacency matrix of input graph to the GCN seem to appear in the generated text. The loops in the input graph appear to be likely to manifest as loops in the decoder states.

5.7.1 More observations

Using entity graphs as the input graphs for GCN gives better results than using dependency parses. Two possible reasons for this are:

1. Entity graphs link capitalized entities across sentences, while the links given by dependency parses are local to each sentence. Since the task is of multi-hop question generation, having information of entities across different sentences in the supporting passage aids the model.
2. The links in Dependency parses are denser than the links in entity graphs. Hence, the problem of *Node Information Morphing* is likely more pronounced when dependency parses are used as input graphs.

On a side note, on the note on the SQuAD dataset (Rajpurkar *et al.* [2016]), GCNs with edge-wise gating mechanism seems to get closer to competitive performance than on the HotpotQA dataset. On closer examination of the datasets, we attribute this to the reasoning that context passage sentences in SQuAD have more patterns while the sentences in HotpotQA are more diverse. Thus, since the input graph diversity in SQuAD is much lower, it is likely that the GCN weights are trained more consistently and meaningfully.

CHAPTER 6

Future work and GCNs for NLP

Some future work directions are described as follows:

1. Edge-wise Gating for GCNs (Marcheggiani and Titov [2017])
   Edges are assigned scalar gates to upscale or downscale their contribution to the output. Although Edge-wise gating may help alleviate Node Information Morphing, the usefulness of this technique can be commented about only after trial. The poor trainability of GCNs in our use-case may adversely impact the expected improvements from this method.

2. Node-tagging
   When dependency parses are used as GCN input graph, nodes can be supplemented by typed dependency relations (De Marneffe and Manning [2008]) they are derived from.

3. Input graph information may instead be used to set attention priors (Shankar *et al.* [2018]) rather than manipulate word embeddings. This approach is more flexible as attention weights vary with decoder time-steps. This approach allows to retain all input information.

4. Training GCN weights jointly for Question Generation and Question Answering tasks
   Among the given inputs for Question Generation and Question Answering tasks, i.e. the context passage, the question and the answer, the context passage plays an equivalent role in both the tasks while the question and answer play inverse roles. Since the GCN is typically used for embedding the input tokens of the context passage, training the GCN weights jointly for both tasks may enhance training outcome.

REFERENCES

Bahdanau, D., **K. Cho**, and **Y. Bengio** (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

De Marneffe, M.-C. and **C. D. Manning** (2008). Stanford typed dependencies manual. Technical report, Technical report, Stanford University.

Defferrard, M., **X. Bresson**, and **P. Vandergheynst**, Convolutional neural networks on graphs with fast localized spectral filtering. *In Advances in neural information processing systems*. 2016.

Du, X., **J. Shao**, and **C. Cardie** (2017). Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106*.

Heilman, M. and **N. A. Smith**, Good question! statistical ranking for question generation. *In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010.

Hochreiter, S. and **J. Schmidhuber** (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Kipf, T. N. and **M. Welling** (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.

Marcheggiani, D. and **I. Titov** (2017). Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*.

Nallapati, R., **B. Zhou**, **C. Gulcehre**, **B. Xiang**, *et al.* (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Papineni, K., **S. Roukos**, **T. Ward**, and **W.-J. Zhu**, Bleu: a method for automatic evaluation of machine translation. *In Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002.

Pennington, J., **R. Socher**, and **C. Manning**, Glove: Global vectors for word representation. *In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

Rajpurkar, P., **J. Zhang**, **K. Lopyrev**, and **P. Liang** (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

See, A., **P. J. Liu**, and **C. D. Manning** (2017). Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Seo, M., **A. Kembhavi**, **A. Farhadi**, and **H. Hajishirzi** (2016). Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Shankar, S., **S. Garg**, and **S. Sarawagi**, Surprisingly easy hard-attention for sequence to sequence learning. *In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018.

Sutskever, I., **O. Vinyals**, and **Q. V. Le**, Sequence to sequence learning with neural networks. *In Advances in neural information processing systems*. 2014.

Vijayan, P., **Y. Chandak**, **M. M. Khapra**, and **B. Ravindran** (2018). Hopf: Higher order propagation framework for deep collective classification. *arXiv preprint arXiv:1805.12421*.

Yang, Z., **P. Qi**, **S. Zhang**, **Y. Bengio**, **W. W. Cohen**, **R. Salakhutdinov**, and **C. D. Manning** (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Zhao, Y., **X. Ni**, **Y. Ding**, and **Q. Ke**, Paragraph-level neural question generation with maxout pointer and gated self-attention networks. *In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018.