Text summarization using abstract meaning representation (AMR)

A thesis submitted in fulfillment of the requirements

for the degree of Master of Technology

by

Amit Nagarkoti

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

July 2017

Certificate

It is certified that the work contained in this thesis entitled *"Text summarization using abstract meaning representation (AMR)"* by *Amit Nagarkoti* has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

Dr. Harish Karnick

July 2017

Professor,

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

Abstract

Name of the student: **Amit Nagarkoti** Roll No: **15111009**

Degree for which submitted: **M.Tech.** Department: **Computer Science and Engineering**

Thesis title: **Text summarization using abstract meaning representation (AMR)**

Thesis supervisor: **Dr. Harish Karnick**

Month and year of thesis submission: **10 July 2017**

Text Summarization is an important and widely studied problem in NLP. The ability to represent and store information concisely gives us the ability to process much more information in a limited time. Also the ever growing number of documents makes it difficult to retrieve the most important facts. Summarizing a natural language text requires both semantic and syntactic understanding of the language. We want summaries to not only have the important keywords from the original document but also retain the grammatical structure. Text summarization is generally categorized as extractive and abstractive, extractive summarization is finding the most relevant sentences or words from the document whereas abstractive summarization is a much harder problem where we need to rephrase the text and generate new words with similar meanings. We study both problems using the Abstract Meaning Representation or AMR which is a semantic representation of the text as graph. An AMR tries to capture the meaning of a sentence with as much abstraction as possible. We study methods where AMRs are used for extractive summarization as well as abstractive summarization. For abstractive summarization we use Seq2Seq learning models which are based on the encoder-decoder architecture. We study a few variations of these architectures for generating summaries directly from linearized AMRs as well as raw text. We also explore data augmentation or extension techniques to have a more informed

neural network by using both AMRs and POS tags to get richer decoder states. We use attention based models and other techniques from Machine Translation suggested in literature to deal with large vocabularies and handling(OOV or out of vocabulary words). Further we experimented with a mixed system where in phase one we use extractive summarization to get the most relevant text and than generate abstractive summaries from it. For the extractive phase we use dependency parse trees and Latent Semantic Analysis or LSA.

Acknowledgements

I would like to extend my sincerest gratitude to my thesis supervisor, Prof. Harish Karnick, for his unparalleled guidance and support. I was new to this field, I can't be much more thankful for all the wild ideas and new things i got to learn and explore under his guidance. I am grateful for his patience and all those weekly sessions of discussion.

I would also thank my Family for believing in me and my Friends for their support. These 2 years would not have been same without you all.

Last but not the least I would like to thank Prof. Gaurav Sharma and Prof. Gaurav Pandey for letting me use their computational resources and making it possible to complete the work on time.

Contents

Certificate i

Abstract ii

Acknowledgements iv

Contents v

List of Figures vii

List of Tables viii

1 Introduction 1
- 1.1 Background and problem definition . . . 2
- 1.2 Organization of the thesis . . . 4

2 Background Literature Review 5
- 2.1 Neural Language models and word Embeddings . . . 5
  - 2.1.1 Word2Vec[Mikolov et al., 2013b] . . . 7
  - 2.1.2 Glove or Global word vectors . . . 8
- 2.2 AMR : Abstract Meaning Representation . . . 8
  - 2.2.1 JAMR Parser [Flanigan et al., 2014] . . . 12
- 2.3 Sequence to Sequence Learning : s2s . . . 13
  - 2.3.1 Attention . . . 14
- 2.4 Extractive summarization using Latent Semantic Analysis (LSA) . . . 15

3 Methods and Models 18
- 3.1 Approximate AMR based summarization . . . 18
  - 3.1.1 Graph based summarization . . . 18
    - 3.1.1.1 Alignments for word mapping . . . 23
  - 3.1.2 AMR Linearization . . . 24
- 3.2 Data Augmentation . . . 25

3.2.1 Using POS tags . . . 27
3.2.2 AMR . . . 28
3.3 Handling large vocabularies . . . 29
3.3.1 Byte Pair Encoding or BPE . . . 29
3.3.2 Pointer-Generator . . . 31
3.3.3 Coverage . . . 32
3.4 Extractive + Abstractive . . . 33
3.4.1 Using Dependency Parsing . . . 33
3.4.2 Latent Semantic Analysis . . . 34

4 Results and Analysis **35**
4.1 Rouge or Recall-Oriented Understudy for Gisting Evaluation . . . 35
4.1.1 Rouge-N . . . 35
4.1.2 Rouge-L . . . 36
4.2 Beam Search . . . 37
4.3 Datasets . . . 38
4.4 Results . . . 40
4.4.1 S2S results on CNN/Dailymail . . . 40
4.4.2 Models using AMRs . . . 43
4.4.2.1 AMR Augmentation . . . 43
4.4.2.2 AMR to AMR s2s . . . 44
4.4.2.3 Graph based summarization . . . 46
4.4.3 Extractive + Abstractive . . . 46
4.5 Some Generated Summaries . . . 47
4.6 Attention Visualization . . . 48

5 Conclusion and Future work **52**
5.1 Conclusion . . . 52
5.2 Future work . . . 53

Bibliography **54**

List of Figures

2.1 A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature [LeCun et al., 2015] 6
2.2 Overview Neural Language Model 6
2.3 Skip-gram, recreated from [Mikolov et al., 2013b] 8
2.4 AMR PENMAN 9
2.5 AMR Dictionary screenshot from *https://www.isi.edu/ ulf/amr/lib/amr-dict.html* . 10
2.6 Overview of Sequence to Sequence Learning Model 13
2.7 S2S in action 14
2.8 Attention in S2S 14
2.9 LSA for summarization recreated from [Steinberger and Jezek, 2004] 16

3.1 Raw Document Graph (modified from [Liu et al., 2015]) 19
3.2 Article Document Graph 19
3.3 AMR DFS Traversal 25
3.4 Bounding at d-3 will give **-TOP-**(*want-01* **ARG0**(*boy*)**ARG0 ARG1**(*believe-01* **ARG0**(*girl*)**ARG0 ARG1**(*RET*)**ARG1**)**ARG1**)**-TOP-** 25
3.5 Multiple inputs combined to get a single input sequence 26
3.6 Using two LSTMs to encode separate input sequences 26
3.7 s2s model generating multiple output distribution 27
3.8 hierarchical models learn the document vector using level-wise learning 28
3.9 Hierarchical Attention Network 29
3.10 Pointer Generator Model source: [See et al., 2017] 32
3.11 Dependency Parse Example 34

4.1 Number of sentences in article/summary 39
4.2 Number of words in article/summary 39
4.3 Number of words in article/summary for AMR Data 40
4.4 AMR PCA sentence embeds 45
4.5 Attention heatmap 1 49
4.6 Attention heatmap 2 50
4.7 Attention Probability for decoding 51

List of Tables

3.1 Node features as defined in [Liu et al., 2015] . . . 20
3.2 Node word alignment . . . 24

4.1 CNN/Dailymail split . . . 38
4.2 CNN/Dailymail average stats for Training sets . . . 39
4.3 hyper-parameters for s2s model . . . 41
4.4 metric short-names mapping . . . 41
4.5 Model Description . . . 42
4.6 *cov* indicates model is using coverage loss, $no - cov$ indicated model is not using coverage loss, *pos-wt-loss with $\lambda = 0.25$ model in eq 3.10, pos-full combines loss for word generation and POS tag generation with equal weight of 1 model in eq 3.9, all models use pointer copying mechanism by default except the bpe model, text-100 indicates glove vectors of size 100 were used to initialize the embeddings and text-200 indicates glove vectors of size 200 being used to initialize the embeddings . . . 43
4.7 Model Description . . . 44
4.8 AMRs as Augmented Data . . . 44
4.9 1: cnn no pointer gen 2: cnn with pointer gen 3: cnn with cov and pointer gen . . . 44
4.10 1: giga headline generation dataset with pgen-cov . . . 46
4.11 ref-gold refers when we compare the generated summaries to the original gold summaries, *ref generated summaries were generated using word alignment using nodes in the summary graph . . . 46
4.12 Model Description . . . 47
4.13 Comparing the mixed approach for summarization, Dependency based method used $L = 7$ as context window value . . . 47

Dedicated to the Universe

Chapter 1

Introduction

This thesis is on automatic summarization. Given the ever increasing rate at which information is being generated multiple methods have to be used to accurately extract those documents or pieces of text that are important for a particular purpose.

Accurate document summaries is one of many techniques that can help humans sift through large quantities of information in the form of text documents to retain only those that are useful. Conceptually, summaries can be of different types. For example they can be: key or topic words, headlines, abstracts or precis. Normally, a summary is a grammatical, human readable piece of text that is much smaller than the original text. So, topic or keyword tagging of text is not considered summarization. Summaries can be a few words (e.g. headlines), a fixed upper limit on the number of words (e.g. abstracts) or a fraction of the original document length (e.g. precis - typical a third of the original).

Broadly, three kinds of summarization algorithms exist. We have extractive algorithms that select some sentences from the original document to produce a summary. Abstractive summarization converts the original text into an intermediate representation that represents the meaning of the text; compresses or shortens the representation by editing the representation in some form while retaining the meaning and then regenerates or synthesizes text from the edited representation. Finally, we can have hybrid methods that typically first do a permissive form of extraction and then use abstraction on the extracted text.

Algorithms for summarization vary depending on the length of the summary text. For example, headline generation is most often extractive. Longer summaries like abstracts or

precis can be extractive or abstractive or hybrid. In general, abstractive summarization is much harder to automate than extractive summarization.

1.1 Background and problem definition

Traditional methods that use the concept of centrality[Gong and Liu, 2001] and graph based algorithms[Erkan and Radev, 2004] work quite well for extractive summarization but fail in case of abstractive summarization as it requires text understanding beyond the assumption of a document as a *BOW*(Bag of Words) or a syntax tree. To rephrase a text requires semantic knowledge of the content. To this date abstractive summarization systems are far from human level performance opposite to what is being observed with images where *CNN*(Convolutional Neural Networks)[Krizhevsky et al., 2012] have been shown to capture abstract features of images reasonably well. We are yet to find a suitable framework that captures both the syntax and semantics of text (not including pragmatics). *AMR* or *Abstract Meaning Representation*[Banarescu et al., 2012] tries to address the problem of language semantics by using a graph based approach to represent a sentence as a Directed Acyclic Graph.

Text summarization can be defined as follows: given a sequence of words $D_i = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ which represents a document of size N our goal is to find a sequence of words $S_i = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\}$ of size M where $M << N$ such that the new sequence S_i captures the meaning of the original document D_i and is human readable. We generally limit the size of the summary by bounding the length of the summary.

Ideally, we would like summaries to be abstractive. However, as mentioned earlier abstractive summarization is hard since it involves multiple steps some of which are poorly understood and therefore do not have algorithms that can perform at a high level. Taking our cue from recent machine translation approaches that start with sentence aligned corpora in the two languages and then train recurrent neural network models that function as sequence transformers from one language to another. Such models do not have an intermediate representation and completely avoid the hard problem of meaning representation. Counter intuitive though this may seem such translation methods are able to achieve or exceed state-of-art translation.

In this thesis summarization is modeled as a sequence to sequence learning[Sutskever et al., 2014] task. Sequence learning is widely used in machine translation where we are given a parallel aligned corpus in different languages and the goal is to learn an

efficient translation from one language to another. Recurrent neural network based architectures are used for the translation problem and we use them to model sequence learning for summarization. At a very high level we have an encoder which is a Long Short Term Memory(LSTM)[Hochreiter and Schmidhuber, 1997] or Gated Recurrent Unit(GRU)[Bahdanau et al., 2014] unit and there is a decoder(LSTM or GRU) which uses the output from encoder to generate the final summaries. The models will be discussed in detail in later chapters.

The main goals we tried to address here were how can we learn a good encoding of the input so that our decoder can generate good summaries. This is where it becomes necessary to have richer representations for the input text. Also, we wish to explore whether we can use the intermediate encoder states to have a more informed decoder? The attention mechanism first proposed by Bahadanu et. al. [Bahdanau et al., 2014] discussed later attempts to overcome this problem by using encoder states when decoding.

We quickly discovered that machine translation approaches are not directly applicable since we cannot align the text and its summary. So, directly learning to generate a summary sequence of words from the document sequence of words does not seem to work at least with currently available recurrent neural network models. One way this problem can be mitigated is by providing extra information in the form of richer representations of the text in conjuction with the text for learning the sequence model. While this does imply that we have to use extra representations of the document it still falls considerably short of using a fully intermediate representation based abstractive summarization.

We tried data augmentation techniques such as providing extra information to our model in terms of POS tags or AMR sequence. Another major problem with neural NLP models is they tend to learn very poorly if vocabulary size is large, which also results in increased computation cost. We used a few techniques such as word categorization and the recently introduced idea of pointer generator[See et al., 2017] models or copying mechanism[Gu et al., 2016] to handle larger vocabularies. We also experimented by having an encoded vocabulary by using the *BPE* or Byte Pair Encoding[Sennrich et al., 2015] which drastically reduces the vocabulary size and has been quite effective in machine translation.

Also a few proposed graph based methods[Liu et al., 2015] are discussed. Summarization can easily be put in the category of NLP-Hard problems. We believe a complete solution will require a more complete representation and modeling of language itself along with better models to exploit them.

1.2 Organization of the thesis

This chapter concluded after giving background information on summarization and a statement and motivation for our approach to text summerization. The problems stated above will be discussed in more detail in following chapters. The rest of this thesis is organized as follows: Chapter 2 discusses Neural Language Model, Word Embeddings (word2vec, glove), AMR, JAMR Parser, Sequence to Sequence Learning, Extractive summarization using LSA. Chapter 3 describes the models and methods used in this thesis. It starts with Extractive Summarization using AMRs and describing AMR linearization. Data augmentation techniques using POS tags and AMR sequences. Augmentation models, hierarchical neural networks for document embedding. BPE, pointer-generator model, coverage mechanism[See et al., 2017], Dependency and LSA based methods for two phase summarization. Chapter 4 Starts by describing the Rouge metric, followed by CNN/Dailymail dataset analysis, Beam search method, hyper-parameters for our models, results analysis, attention heatmap and qualitative analysis. Chapter 5 concludes and contains pointers for further work.

Chapter 2

Background Literature Review

This chapter covers in brief the basic fundamentals of the most important topics along with techniques and tools which have been developed to tackle the problem of language understanding and text summarization. These tools will be used to develop models described in chapters 3 and 4.

2.1 Neural Language models and word Embeddings

The most natural task in *NLP* is to calculate the probability $P(W)$ of a sentence or a sequence of words $< w_1, \ldots, w_n >$. Often we have the related task of calculating probability of next upcoming word $P(w_n|w_1, \ldots, w_{n-1})$ given the current sequence of words $< w_1, \ldots, w_n - 1 >$. A model which calculates the sentence probability $P(W)$ or next word probability $P(w_n|w_1, \ldots, w_{n-1})$ is called the language model. The goal of a language model is to assign a higher probability to valid, grammatically correct sequence of text $T_x = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ compared to random text sequences $T_y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$. The next word probability is given by

$$P(\boldsymbol{w}_{N+1}|\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}) = \arg\max_V P(w|\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}) \tag{2.1}$$

where $D_i =< \boldsymbol{w}_1, \ldots, \boldsymbol{w}_N >$ is the current given sequence of words of length N, V is the set of all words in the dictionary and $\boldsymbol{w}_{N+1}$ is the next word we want the model to predict.

In non neural network based models and even in many neural network architectures a word is generally represented as a one hot vector of size $|V|$, the size of vocabulary. A word with index j in the dictionary of all words will be represented as $[0, \ldots, 1, \ldots, 0]$ with a 1 at

index j and 0 everywhere. The problem is that we have no correlation between other words in the dictionary. Also with large vocabularies of $50k$ and $100k$ it becomes quite resource heavy. To overcome this problem idea of distributional semantics or word embeddings was introduced where a word is represented as a fixed size vector of dimension D in a R^D space, instead of the normal one-hot representation. The distributional representations are such that words with similar meanings are expected to be closer to each other than the words with opposite meanings.

Classical models used Markov Approximation to model the above probability. In 2003 Bengio et. al.[Bengio et al., 2003] suggested the idea of using neural networks to model the probability of word prediction. They suggested using Recurrent Neural Nets (*RNNs*) for modeling word dependency on earlier words in the sequence. A *RNN* is a class of artificial neural network where connections between units form a directed cycle. They have an internal state s_t which makes them useful for learning from long sequence of text. They help us exploit the fact that a word in a sentence is related to the previous words in the sentence. Diagrammatically

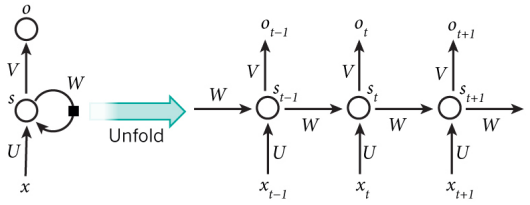


Figure 2.1: A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature [LeCun et al., 2015]

Using a *RNN* a language model can be represented as by the figure below

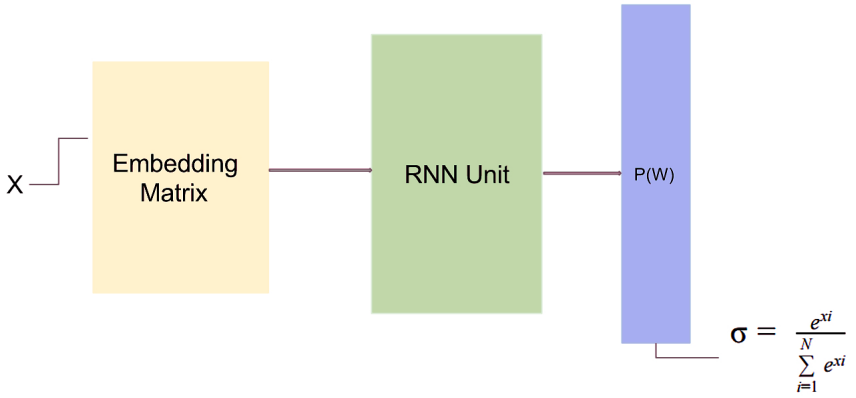


Figure 2.2: Overview Neural Language Model

In the fig 2.6 embedding matrix is a $|V|$XD matrix with rows equal to number of words in the vocab $|V|$ and D is the size of the word vector. The input X is a sequence of words with word row ids, which are used to fetch the embedding vector from the embedding matrix. Thus the inputs to the RNN is the word vector for the current word in the sequence. At each step the RNN state s can be used to derive probability of the next word using a softmax layer and some projection matrix W. This softmax probability for next word is given by $P(w)$ and is used to calculate the network loss generally cross-entropy loss.

2.1.1 Word2Vec[Mikolov et al., 2013b]

Mikolov et. al. in 2013 gave a method to efficiently learn word vectors using distributional semantics - that is using the context in which the word occurs. They used the word context to learn a fixed size vector representation for words such that words with similar meanings occur closer together in the vector space than words with different meanings. They compared two methods to learn a word embedding *Skip-gram*[Mikolov et al., 2013a] where the target word is used to predict the context words and *CBOW Continuous Bag Of Words*[Mikolov et al., 2013a], which is the converse of *Skip-gram*, where context words are used to predict the target word. Skip-gram tries to maximize the following log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, c\neq 0}\log p(w_{t+j}|w_t) \tag{2.2}$$

where T is the length of the corpus, c is the size of the context window for skip-gram, $p(w_{t+j}|w_t)$ is the probabilty of word w_{t+j} occuring in the context of word w_t. So basically the model here tries to optimize the observation of words around any particular word over the entire corpus.

The probability of context word given the current word $p(w_{t+j}|w_t)$ is defined as

$$p(w_O|w_I) = \frac{\exp v'^{\perp}_{w_O} v_{w_I}}{\sum_{w=1}^{W}\exp v'^{\perp}_{w_w} v_{w_I}} \tag{2.3}$$

Where v'_w and v_w are the input and output representation vector for word w. The model assumes two representations for each word which are than combined to give a final word vector. To deal with large vocabularies and to make the process efficient they used *Hierarchical Softmax*[Morin and Bengio, 2005].

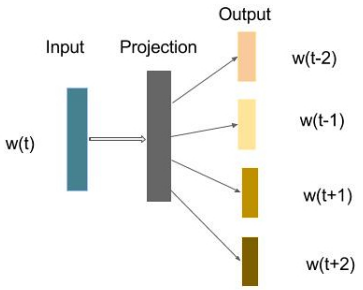


FIGURE 2.3: Skip-gram, recreated from [MIKOLOV ET AL., 2013b]

In the figure above the word w_t at current time-step t is used to predict the context words $\{w(t-2), w(t-1), w(t+1), w(t+2)\}$. The projection is done using state of the RNN cell as explained earlier.

2.1.2 Glove or Global word vectors

Similar to word2vec Glove[PENNINGTON ET AL., 2014] gives word vectors and uses distributional word semantics to compute such vectors but unlike word2vec which focuses just on the local context glove tries to incorporate global information by using word co-occurrence counts. Word-word co-occurrence count between a word w_i and w_j is the number of times w_i and w_j appears together in the corpus in some context. The matrix capturing the co-occurrence relations between all words in the vocab is called the co-occurence matrix. The objective is changed to minimize the following cost-function $\hat{J}$:

$$\hat{J} = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{i,j})(u_i^T u_j - \log P_{i,j})^2 \tag{2.4}$$

where $P_{i,j}$ is the probability that the word j appears in context of i obtained using the word co-occurence matrix. Glove performs much better than word2vec on many tasks and a corpus of trained vectors of dimensions 100 and 200 and 300 is available which we will be using to initialize our word vectors.

2.2 AMR : Abstract Meaning Representation

The idea of AMR dates back to 1998 and was introduced by Langkilde and Knight[LANGKILDE AND KNIGHT, 1998]. AMR was designed to capture the semantic content of a sentence

where a simple data structure tries to represent most of the concepts in a sentence. AMR is a *Directed Graph* and uses PENMAN notation[Schneider et al., 2015] to represent the actual structure. For example the sentence "The dog is eating a bone." is represented as

(e / eat-01 :**ARG0** (d / dog) :**ARG1** (b / bone))

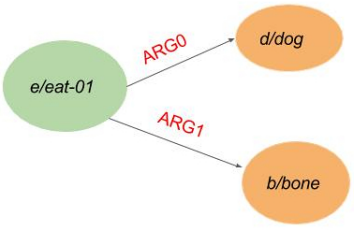


Figure 2.4: AMR PENMAN

The edges represent relations where the edge labels $ARG0$ and $ARG1$ have special meanings depending on the word sense. An AMR basically captures "who is doing what to whom" in a sentence. So these labels $ARG0$, $ARG1$ have fixed roles depending the on the current word-sense. For example if we check the PropBank frame for $eat-01$ we find the following description

- eat-01: is used in sense for to consume or consuming
- Arg0: consumer, eater - *the dog* in our example
- Arg1: meal - *the bone* in our example

These roles may vary depending on the core-concept or the node. Nodes are concepts and are marked with a variable, for example e, **;** d, **;** b above. The top or head concept is typically the main verb or focus of the sentence. Core concepts or frames in AMR are based on the PropBank Lexicon[Bonial et al., 2010]. For example run may have the following senses based on how it is used

- run-01 :: operate, proceed, operate or proceed
- run-02 :: walk quickly, run/jog
- run-03 :: cost

There are five core arguments ARG0 to ARG4 which themselves have meaning based on the word sense. For example: for the sense `run-01`

- ARG0 :: operator

- ARG1 :: machine, operation, procedure
- ARG2 :: employer
- ARG3 :: coworker
- ARG4 :: instrumental

As an example for `run-01` consider the following sentence "65 percent or more of automobiles run by Gustavo and his companion Hector are on alcohol". In this example

- ARG0 : Gustavo
- ARG1 : 65 percent or more of automobiles
- ARG3 : Hector
- ARG4 : on alcohol

Here the alcohol is the instrument or tool to drive the automobile. Note we have not used *ARG2* here which is quite common in natural language as we may not require all the relations in a single sentence.

for `run-02`

- ARG0 :: runner
- ARG1 :: cource, race, distance
- ARG2 :: opponent

AMR also defines non-core arguments to express things like :time, :manner, :part, :location, :frequency. These are described in the AMR dictionary. There are many design guidelines

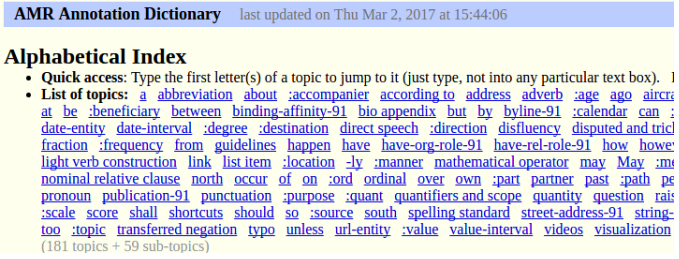


FIGURE 2.5: AMR Dictionary screenshot from *https://www.isi.edu/ ulf/amr/lib/amr-dict.html*

on how to actually get an AMR for a sentence, which we do not discuss here completely.

Typically an AMR removes tense related information, uses sentence level co-referencing, adds wiki-fication etc.

A few example AMRs highlighting how rules from amr-guidelines are used for amr-creation

- All the below words/phrases gets mapped to the same AMR removing the tense information

  ```
  (e / eat-01)
  ```

  - eating
  - eats
  - ate
  - will eat

- Removing articles and plurality

  ```
  (c / cat)
  ```

  - A cat
  - The cat
  - cats
  - the cats

- Co-referencing, example "The man saved himself"

  ```
  (s / save-01
      :ARG0 (m / man)
      :ARG1 m)
  ```

- Non-core role, example "The yummy food "

  ```
  (f / food
      :mod (y / yummy))
  ```

  here *:mod* indicated the attribute *i.e yummy* is the attribute of the *food*.

- Named-entities, example "Sachin Tendulkar"

  ```
  (p / person
      :name (n / name
          :op1 "Sachin"
          :op2 "Tendulkar"))
  ```

2.2.1 JAMR Parser [Flanigan et al., 2014]

The JAMR parser, is one of several implementations, that can be used to obtain AMR graphs for a given sentence. The parser is quite efficient in identifying concepts in a graph. At a high level JAMR works in two phases. First, it identifies the main concepts in a sentence. A JAMR parser views a sentence as a series of tokens. Based on some built in rules multiple concepts may be predicted for a given sequence of tokens resulting in many different concept assignments. The problem is than reduced to choosing the highest weighted concept assignment for the sequence of tokens. The goal is given word span $< w_{b0..b1}, w_{b1..b2}, \cdots, w_{bn-1..bn} >$ of tokens in a sentence, where $w_{bi..bj}$ is the tokens between index i and j in the sentence we have to find the best matching concept $C_i \cup \phi$ or node for that span. The objective is to maximize the score

$$score(b, c; \theta) = \sum_{i=1}^{k} \theta^T f(w_{b_{i-1}:b_i}, b_{i-1}, b_i, c_i) \tag{2.5}$$

where θ is a learnable model parameter, $\mathbf{f}$ is a feature vector of span which is based on some predefined rules to extract values for the span. The output is the mapping of word spans to concepts such that the above score is maximized *i.e* the output is a table of mappings from $w_{bi..bj}$ to c_k. This table is the alignment table which will be used in later chapters where we discuss on how AMR can be used for summarization task. More details can be found in the original JAMR paper by Flanigan et. al.

In the second phase JAMR parser tries to find the edge labels or relations between the selected nodes/concepts in step 1. First we create a fully connected dense graph by adding all possible relations between the nodes obtained in step 1. The second phase of relation identification than finds a *MSCG* (Maximally Connected Spanning Sub-graph) from a fully connected dense graph such that the selected sub-graph has the highest score given in *eq.* 2.6. The algorithm maximizes the score over the edge set E_g of the sub-graph given by

$$score(E_G; \psi) = \sum_{e \in E_G} \psi^T g(e) \tag{2.6}$$

where ψ is the learnable model parameter, $\mathbf{g}$ gives the feature vector for the edge e_i based on predefined rules. The set of edges e_i maximizing above score forms the part of the JAMR output graph. A very similar idea will be discussed in chapter 3 where we use AMR to do extractive summarization directly from AMR graph.

2.3 Sequence to Sequence Learning : s2s

Sequence learning is a common method in NLP which is widely used in machine translation. In sequence learning, generally, we are given two sequences called the source and the target which may be based on the same or different vocabularies. The task is than to learn a model which given a source sequence generates the matching target sequence - for example a translation system from English to French. Here we use recurrent network based sequence to sequence learning models. The high level idea is summarized by the figure below There

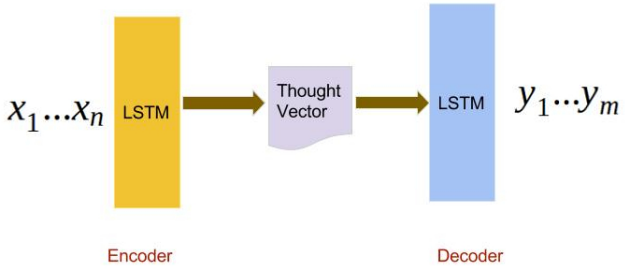


FIGURE 2.6: Overview of Sequence to Sequence Learning Model

are two main components of a s2s model. An encoder which takes as input the source sequence and generates a fixed dimensional representation for the source sequence called the thought vector. It is expected to encapsulate the semantics of the source sequence. The thought vector is also used to initialize the initial state of the decoder. The decoder generates the output target sequence one word or char at a time, while also updating its current state. The encoder can be modelled using an RNN (LSTM or GRU) or a CNN (Convolutional Neural Network). The decoder is modelled using an RNN. During training the decoder learns from the input target sequence and tries to minimize the negative log likelihood of the target word

$$loss_D = \sum_{t=1}^{T} -\log(P(w_t)) \tag{2.7}$$

here w_t is the target word at step t.

Multiple S2S architectures have been suggested in the literature where different encoders and decoders have been used. Variants include CNN based encoders or multi-layered RNN encoders. The decoder can itself be multi-layered. S2S models perform fairly well in practice but the idea of using a single thought vector as the complete representation of the source sequence does affect the performance of the decoder as it limits the information available to the decoder. To address this problem the idea of attention [BAHDANAU ET AL., 2014] was suggested.

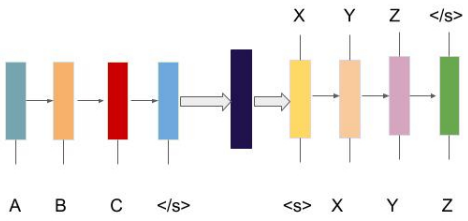


FIGURE 2.7: S2S in action

2.3.1 Attention

Attention in s2s attempts to make a more informed decision when decoding the next word. Now the decoder has access to all the past encoder states. The decoder can decide how much weight must be given to a particular encoder state. A soft attention model performs the following computation to learn the probability distribution over the vocabulary.

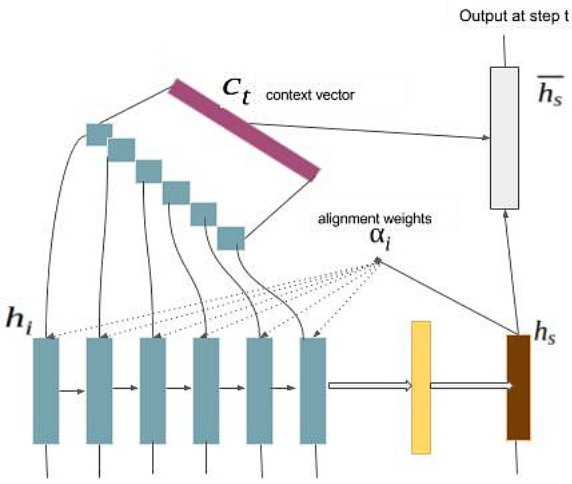


FIGURE 2.8: Attention in S2S

The attention model described in figure above uses the encoder states h_i obtained from a given sequence $< x_1, \ldots, x_n >$. h_s is the current decoder state at say the current decode step t. In eq 2.8 a score e_i is obtained for each encoder state using the encoder state h_i and the current decoder state h_s, v, W_{h_e}, W_{h_d} are model learnable parameters. A softmax weight α is then obtained using the above scores as given in eq 2.9. A context vector c_t is obtained combining encoder states according with the softmax weights as in eq 2.10. To use this information for decoding we concatenate the context vector c_t with current decoder state h_s to obtain a new richer state $\bar{h_s}$ as in eq 2.11. This new state can now be used with the projection matrix W_w to generate next word probability as given in eq 2.12.

$$e_i^t = v^T \tanh(W_{h_e} h_i + W_{h_d} h_s + b_{attn}) \tag{2.8}$$

$$\alpha^t = softmax(e^t) \tag{2.9}$$

$$c_t = \sum_{i=1}^{encsteps} \alpha_i^t h_i^e \tag{2.10}$$

$$\bar{h_s} = [c_t, h_s] \tag{2.11}$$

$$P(w) = softmax(W_w \bar{h_s} + b_w) \tag{2.12}$$

where $P(w)$ is the probability of next word generation. As can be seen the output word is derived using the richer vector $\bar{h_s}$.

2.4 Extractive summarization using Latent Semantic Analysis (LSA)

Extractive summazrization aims at finding the k most relevant sentences from a document containing n sentences. The idea in extractive summarization methods is to get those sentences which cover the major topic(s) of the document assuming a document contains multiple topics but not all may be equally relevant. Latent semantic analysis (LSA) is a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of sentences/documents and the terms they contain by producing a set of concepts related to the sentences/documents and terms. Given a document with n sentences first a term frequency vector $T_i = [t_{1i}, t_{2i}, \cdots, t_{mi}]$ is created for each sentence where t_{ji} refers to the frequency of word j in sentence i. A weighted term frequency vector is defined as $A_i = [a_{1i}, a_{2i}, \cdots, a_{mi}]$ where $a_{ji} = L_{ji} \cdot G_{ji}$ and L_{ji} is the local weighting and G_{ji} is the global weighting for term j in sentence i ex. $tf - idf$. In LSA first a weighted sentence matrix is created using the weighted term frequency vectors. For n sentences we have $\mathbf{A} = [A_1 A_2 \cdots A_n]$ which gives an $m \times n$ matrix. Being a sparse matrix if the vocabulary is large as all words may not occur in each sentence, it is safe to assume that a low rank approximation of matrix $\mathbf{A}$ exists. The rank k approximation should have the smallest error wrt Forbenius norm. SVD or Singular Value Decomposition is used to obtain the low rank approximation k. Then SVD of $\mathbf{A}$ is defines such that

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \tag{2.13}$$

where $\mathbf{U}$ is a $m \times n$ orthogonal matrix whose columns are called the left singular vectors, Σ is a $n \times n$ diagonal matrix with non-negative values, $\mathbf{V}$ is $n \times n$ matrix whose columns are called the right singular vectors. Now U is such that it captures the correlation between the term vectors givens by AA^T and V captures correlation between the sentence vectors given by A^TA. If $rank(A) = r$ then Σ satisfies $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$. So SVD derives a transformation from m dimensional space of the term vectors to the r dimensional singular vector space. Each column of i of A is projected to column vector $\psi_i = [v_{i1}v_{i2}\cdots v_{ir}]^T$ of $\mathbf{V}^T$ and each row vector j of $\mathbf{A}$ is projected to row vector $\phi_j = [u_{j1}, u_{j2}, \cdots, u_{jr}]$ of $\mathbf{U}$. v_{ix} and u_{jy} are called the index with xth and yth singular vectors.

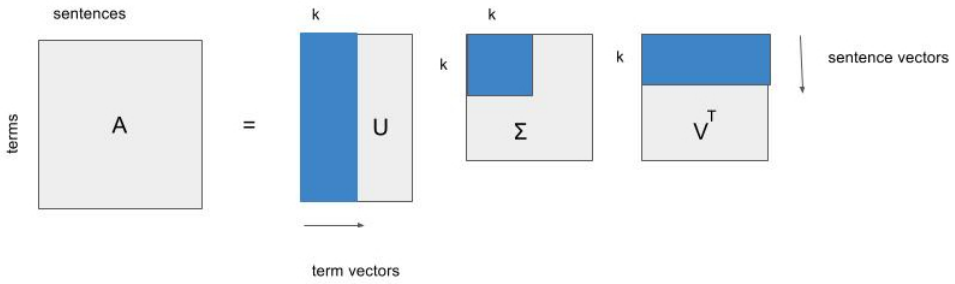


FIGURE 2.9: LSA for summarization recreated from [STEINBERGER AND JEZEK, 2004]

GIven the analogy above the reduced sentence vectors from V^T can be used to calculate a score for each sentence given by

$$S_k = \sqrt{\sum_{i=1}^{n} v_{k,i}^2 \cdot \sigma_i^2} \tag{2.14}$$

n is a hyperparameter of the model. The sentences with the highest scores are then selected as part of the summary.

As an example consider the excerpt from *CNN* on the recent *Google EU* lawsuit

" *Regulators were worried that Google used its might to take away the market share of various price-comparison websites. Why? Two reasons. First, Google's dominance of Internet shopping drove out competitors. This made it tough for existing companies and new companies to get a foothold in the market. The second, more important, reason was that customers would only see companies who paid Google to display their wares at the top of the page when they did a search. The upshot was that customers would have less choice when they went online to shop. They could easily end up paying more for a smaller range of products. Arguably the EU investigation focused on the less controversial anti-competitive*

practices of Google, staying away from the much more profound issues of privacy and data ownership. These are more troubling because Google can make small changes to its search function–as in the earlier FTC case– to create a semblance of fair competition, but changing the way it deals with data goes to the heart of its business model, which is to collect and sell data about its users. ”

The article basically talks about How *Google* is using its dominance to its own advantage when showing Ads and how it effects the consumers choice to get best result and find the best products online.

LSA on above text gives the following ordering for the $top - 3$ sentences

1. The second, more important, reason was that customers would only see companies who paid Google to display their wares at the top of the page when they did a search.
2. Arguably the EU investigation focused on the less controversial anti-competitive practices of Google, staying away from the much more profound issues of privacy and data ownership.
3. These are more troubling because Google can make small changes to its search function–as in the earlier FTC case– to create a semblance of fair competition, but changing the way it deals with data goes to the heart of its business model, which is to collect and sell data about its users.

We did find that first two sentences do cover the gist, with the third one going into more details.

Chapter 3

Methods and Models

In this chapter we will discuss the methods and suggested approaches and possible changes to old models obtain new models for text summarization.

3.1 Approximate AMR based summarization

3.1.1 Graph based summarization

This method is modified from [Liu et al., 2015]. In this method given AMRs for each sentence in the document, first a document graph is built using the following pre-processing steps

1. Collapsing :: collapse concepts for named-entities and date-entities merging all nodes to a single node
2. Merging :: merge common concepts across all sentences in the document, edges labels are removed and parallel edges are merged
3. Connectivity :: add a dummy root and connect it to all the individual sentence roots/concepts so that the graph is connected
4. Expansion :: add new edges between nodes within a sentence

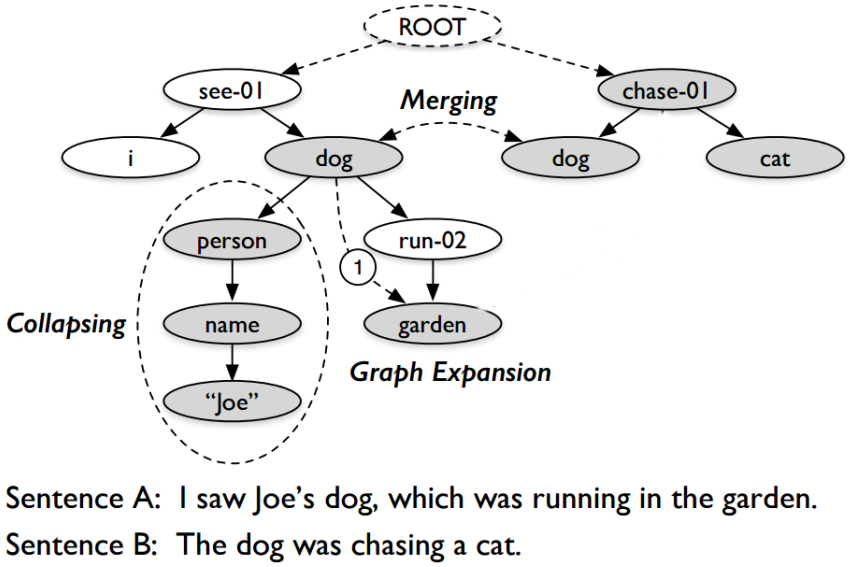


FIGURE 3.1: Raw Document Graph (modified from [LIU ET AL., 2015])

Both the article and the provided human summaries are transformed to their respective document graphs. The article graph after all the pre-processing steps will be modified to a graph as shown in *Fig.* 3.2

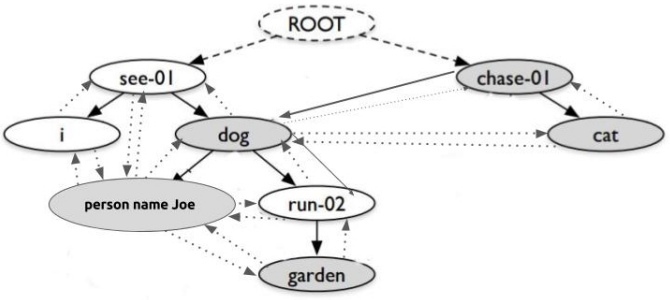


FIGURE 3.2: Article Document Graph

Now given the document graph the goal is to find the closest subgraph similar to the summary graph. Structured prediction is used to get the closest subgraph. The chosen subgraph maximizes the following score

$$score(G_{sub}) = \sum_{v_i} \theta^T f(v_i) \tag{3.1}$$

where θ is learnable model parameter and $f(v)$ is feature vector for nodes defined in table 3.1. We only maximize score with respect to the nodes.

TABLE 3.1: Node features as defined in [LIU ET AL., 2015]

Concept	Identity feature for concept label
Freq	Concept freq in the input sentence set; one binary feature defined for each frequency threshold t = 0/1/2/5/10
Depth	Average and smallest depth of node to the root of the sentence graph; binarized using 5 depth thresholds
Position	Average and foremost position of sentences containing the concept; binarized using 5 position thresholds
Span	Average and longest word span of concept; binarized using 5 length thresholds; word spans obtained from JAMR
Entity	Two binary features indicating whether the concept is a named entity/date entity or not

Obtaining a subgraph from document graph with N vertices is reduced to an optimization problem with the following constraints. Constraints were defined inorder to get a valid connected sub-graph. The selected variables which forms part of the sub-graph *i.e* v_i for the node i and e_{ij} for the edge between node i and node j are binary variables and will have value 1 and unselected node and edge variables v_k and e_jk will have value 0. If edge e_{ij} is selected than both vertices v_i and v_j must be selected which gives the following sanity constraints for all node and edge pairs in the graph.

$$v_i - e_{ij} \geq 0 \quad v_j - e_{ij} \geq 0 \quad , \forall i, j \leq N \tag{3.2}$$

which basically says that edge e_{ij} is included only if both its end points v_i and v_j are. Connectivity is enforced using flow constraints introduced in [MARTINS AND SMITH, 2009]. If the root(node 0) is supposed to be sending one unit of flow($f_{i,j}$ is the flow from node i to node j) to each included node i then,

$$\sum_i f_{0,i} - v_i = 0 \tag{3.3}$$

which says all the flow originates from the root node.

Each included node consumes one unit of flow which translates to the constraint

$$\sum_i f_{i,j} - \sum_k f_{j,k} - v_j = 0, \forall j \tag{3.4}$$

i.e preservation of flow at a node assuming the selected node consumes a unit flow and unselected node passes everything. Flow is sent only over selected edge, again a sanity

constraint

$$N\dot{e}_{ij} - f_{ij} \geq 0, \forall i, j \tag{3.5}$$

The size of subgraph is constrained by limiting number of edges selected by the model

$$\sum_{i,j} e_{i,j} \leq L \quad L\, is\, a\, constant \tag{3.6}$$

Also the subgraph is forced to have single incoming edge per selected node

$$\sum e_{i,j} = 1, \forall j \leq N \tag{3.7}$$

Ramp loss [Gimpel and Smith, 2012a] is used to perform updates using AdaGrad [Duchi et al., 2011] algorithm. Summaries generated are extractive as the most common words aligned to the selected nodes forms the summarized text. Ramp loss is normally used when the outputs and inputs might not be in the same hypothesis space. The below form of structured ramp loss is used for calculating loss

$$-max_G(score(G) - cost(G, G^*)) + max_G(score(G) + cost(G, G^*)) \tag{3.8}$$

where G is the selected sub-graph, G^* is the given summary graph, $score(G)$ is as defined in eq 3.1, $cost(G, G^*)$ penalizes nodes in the set $G \cup G^*/(G \cap G^*)$, we used a penalty of -1.

All the model parameters given by θ are initially initialized to zero and are updated in subsequent iterations. At the time of finding the best sub-graph these values are assumed as constants so the only variables are all integer variables v_i's, e_ij's and f_ij's. Once we get the best set of nodes for particular value of θ, we update it accordingly. The updated θ is used in the next iteration. At a high level the model performs the following steps

```
theta = {} # the model parameters, initially empty, missing parameters are
    taken as zeros
corpus # the training corpus of document and summary amrs
eta = 1.0 # learning rate for adagrad
for cur_doc in corpus:
  create doc graph using method discussed earlier
  assign labels to nodes in both summary and document graph
  add node features as defined in table 3.1, also updating model parameter
    dictionary
  create ILP constraints and solve the optimization problem eq 3.1
```

```
9  calculate ramp loss as in eq 3.8
10 calculate gradients for the model parameters in theta using loss above
11 update theta using adagrad
```

LISTING 3.1: Sub-graph finding algorithm overview

Once we have the final model parameters we can use them as constants for eq 3.1 and just solve the ILP to get the decoded nodes for summary generation.

As an example consider the summarization task for the sentence " *the sri lankan government on wednesday announce the closure of government school with immediate effect as military campaign against tamil separatist escalate in the north of the country*". The amr obtained using the JAMR parser is

```
(a / announce-01
   :ARG0 (g3 / government-organization
         :ARG0-of (g4 / govern-01
               :ARG1 (l / lankan)))
   :ARG1 (c3 / closure
         :poss (s2 / school
               :mod (g / government-organization
                     :ARG0-of (g2 / govern-01
                           :ARG1 (c / country)))))
   :ARG1-of (e2 / effect-03
         :ARG3 (c2 / campaign-01
               :ARG1 (m / military)
               :prep-against (e / escalate-01
                     :ARG0 (s / separatist)
                     :ARG1 (w / wednesday)
                     :location (n / north)
                     :mod (t / tamil)))
         :time (i / immediate)))
```

The document graph has 16 nodes excluding the root. The summary sentence "*sri lanka close school as war escalate*" has the following amr

```
(e / escalate-01
  :ARG1 (w / war)
```

```
:prep-as-of (s / school
      :ARG1-of (c2 / country
            :name (n / name
                  :op1 "sri"
                  :op2 "lanka"))
      :mod (c / close)))
```

summary and document has 3 nodes in common $escalate-01$, *school*, *country* in common. A few model parameter values learned from the single example

- cpt school 1.235702 – concept parameter in table 3.1 for node *school*
- dep avg 5 0.666667 – depth average parameter in table 3.1
- dep fmst 5 0.666667 – depth smallest parameter in table 3.1
  etc

These parameter values are used as such in decoding phase to select the sub-graph.

3.1.1.1 Alignments for word mapping

To generate summary text *alignment* from amr graph to original sentence is used. Each node is marked with the most common word span it aligns to which is used to generate summaries in decoding step. An example is given below, a random sentence taken from CNN dataset

marseille prosecutor says so far no videos were used in the crash investigation despite media reports

The amr parse for this sentence along with alignments obtained using JAMR is

```
(s / say-01 |  0
  :ARG0 (p / prosecute-01 | 0.0
        :ARG1 - | 0.0.0)
  :ARG1 (u / use-01 | 0.1
        :ARG1 (v / video | 0.1.0)
        :ARG2 (i / investigate-01 | 0.1.1
              :ARG1 (r / report-01 | 0.1.1.0
```

```
            :ARG0 (m / media | 0.1.1.0.0)
            :ARG1 (m2 / marseille | 0.1.1.0.1))
      :ARG1-of (c / crash-01 | 0.1.1.1
            :ARG0 (f / far | 0.1.1.1.0)))))
```

The alignments with original sentence and amr nodes is represented in table below

TABLE 3.2: Node word alignment

Amr-node	**Alignment**	**Word**
say-01	2-3—0	says
prosecute-01	1-2—0.0	prosecutor
-	5-6—0.0.0	no
use-01	8-9—0.1	used
video	6-7—0.1.0	videos
investigate-01	12-13—0.1.1	investigation
report-01	15-16—0.1.1.0	reports
media	14-15—0.1.1.0.0	media
marseille	0-1—0.1.1.0.1	marseille
crash-01	11-12—0.1.1.1	crash
far	4-5—0.1.1.1.0	far

We can also see that many nodes have no alignments to the original text.

3.1.2 AMR Linearization

In order to use amr graphs for neural s2s models we need a linearized representation of amr graph. For amr linearization we used amr bracketing suggested by [PENG ET AL., 2017] The amr structure is captured by doing a DFS traversal on the amr graph starting from root. For example the graph below will be linearized as **-TOP-**(*eat-01* **ARG1**(*bone*)**ARG1 ARG0**(*dog*)**ARG0**)**-TOP-**

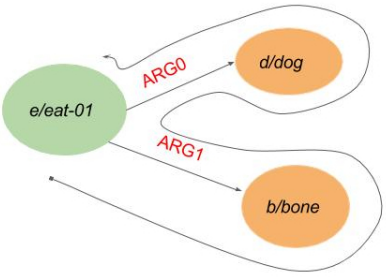


FIGURE 3.3: AMR DFS Traversal

Further as the linearized amr strings can be very long. We used depth based traversal to limit the size of linearized amr. In depth based linearization we stop traversal once a certain concept depth from the root is reached. We used depth 5 in our experiments, a depth 3 was able to cover entire graph for small sentences of length 10 and depth 5 was able to cover sentences with average length $15 - 20$. Also as AMRs place the main verbs/concepts closer to the root, we expect a minimal semantics loss, thereby keeping the linearized amr sequence from becoming too long.

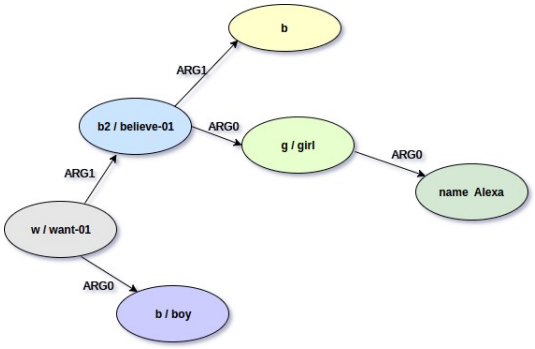


FIGURE 3.4: Bounding at d-3 will give **-TOP-**(*want-01* **ARG0**(*boy*)**ARG0 ARG1**(*believe-01* **ARG0**(*girl*)**ARG0 ARG1**(*RET*)**ARG1**)**ARG1**)**-TOP-**

3.2 Data Augmentation

Data augmentation is the technique where we provide a learning algorithm some extra information besides our input and output text. This data is a supplementary information to the network and enhances its ability to make more informed decisions. In our experiments we used *POS* tags and *AMR* graphs for data augmentation. The figures below shows two possible network architectures with data augmentation

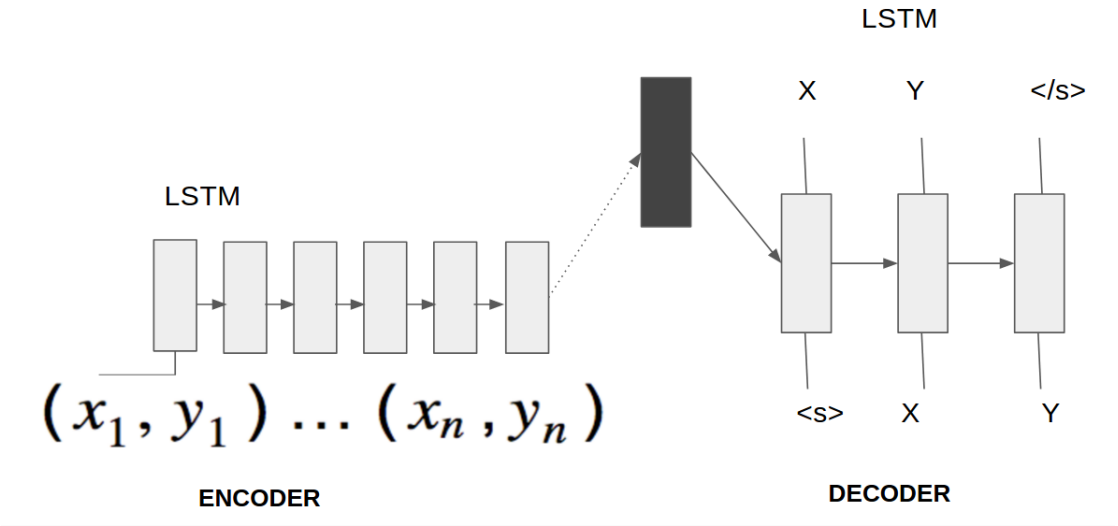


FIGURE 3.5: Multiple inputs combined to get a single input sequence

In the fig 3.5 x_i's are the original text input and y_i's are the augmented or extended input ex POS tags.

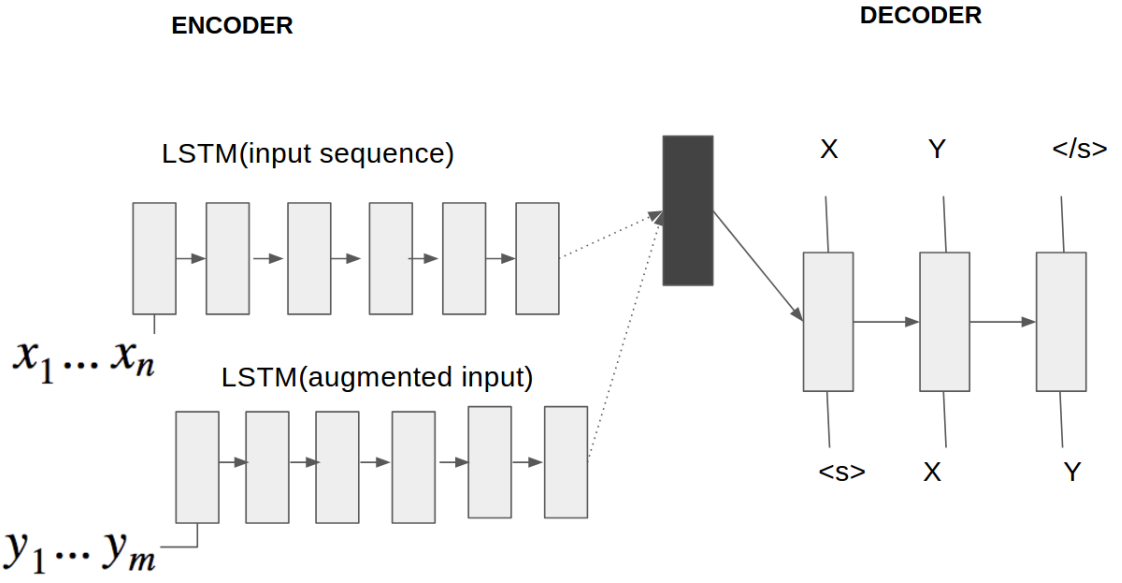


FIGURE 3.6: Using two LSTMs to encode separate input sequences

In the fig 3.6 x_i's are the original text input and y_i's are the augmented or extended input ex POS tags or amr sequences. The inputs have the flexibility that they might not be of same length contrary to fig 3.5.

3.2.1 Using POS tags

We provide two input sequences $T_i = x_1 \cdots x_n$, the original text sequence and $P_i = y_i \cdots y_n$, the POS sequence both going through a different LSTM. The outputs from both networks is used to derive a new initial state for the decoder. We used basically two variations with POS tags

- First where POS tags is used only in encoder part to get initial decoder state basically the architectures in *Fig* 3.6.
- In the second variation our decoder learns to jointly generate separate distributions over both output *POS* sequence and output text sequence.

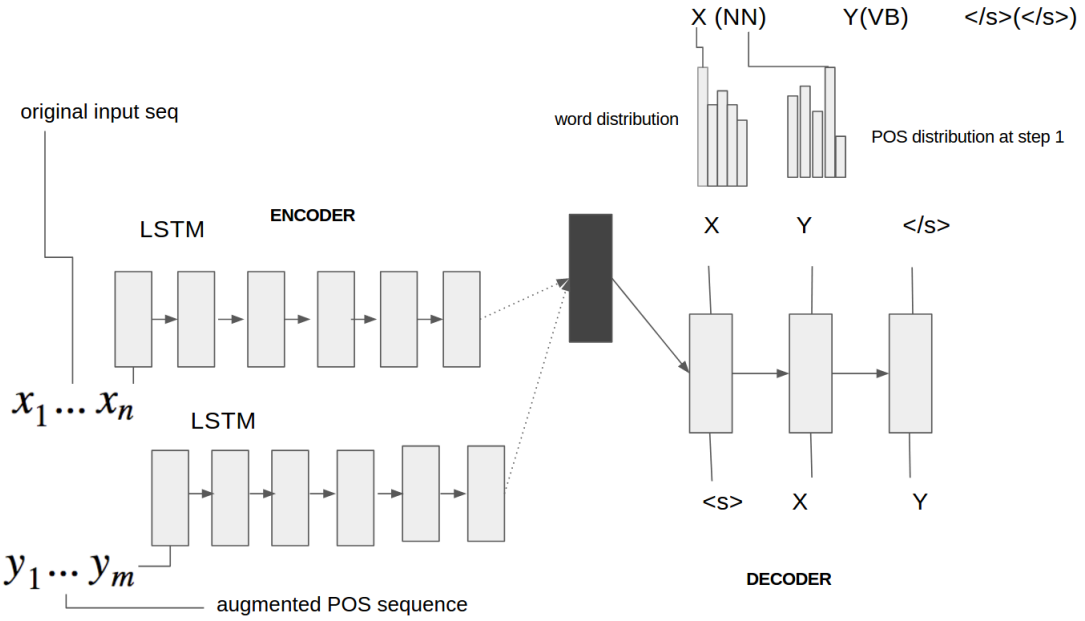


FIGURE 3.7: s2s model generating multiple output distribution

For a decoder sequence of length T the *loss* of the model is given by

$$loss = -\frac{1}{T}\sum_{i=1}^{T}(log(P_{vocab}(w_i)) + log(P_{pos}(p_i))) \tag{3.9}$$

where P_{vocab} is the distribution over text vocab and P_{pos} is the distribution over *POS* vocab. Here we are minimizing the cross-entropy loss over both the text sequence and the augmented POS sequence. During training we will also feed the POS sequence for the decoder/target sequence for loss calculation.

We can also define a different variation with a weighted loss weighted by the parameter λ_{pos} given by

$$loss = -\frac{1}{T}\sum_{i=1}^{T}(\lambda_{pos} * log(P_{vocab}(w_i)) + (1 - \lambda_{pos}) * log(P_{pos}(p_i))) \tag{3.10}$$

The intuition behind the joint optimization is to force the network to generate sequences which are also grammatically relevant. This setting does improves on *Rouge-2* values discussed in next chapter.

3.2.2 AMR

We also experimented with using AMRs as augmented input to our network. The amrs were used only in encoder phase to learn the augmented thought vector. One problem with AMR linearized sequence is that the sentence length becomes quite large and it's inefficient to run LSTMs on very long sequences. Normally we strip very long sentences to a predefined $max - len$ when using recurrent networks but stripping the linear amr sequences will effect the amr structure and meaning of the represented text. So to solve this problem we are using a hierarchical encoder for linear amr sequences.

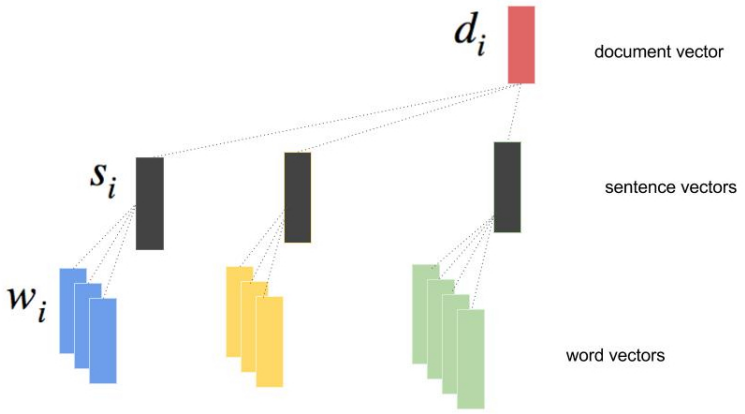


FIGURE 3.8: hierarchical models learn the document vector using level-wise learning

The model above combines word representations to learn sentence representations which are combined to learn document representation. As all words may not contribute equally to the sentence meaning, similarly not all sentences might not have equal contribution to the final document meaning, an attention based model is suitable for the hierarchical network.

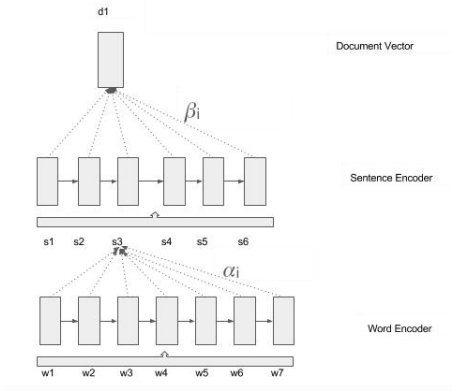


FIGURE 3.9: Hierarchical Attention Network

We used the hierarchical attention network defined in [YANG ET AL., 2016]. The word states are LSTM states for word-encoder, sentence vectors are derived using attention on word states. These sentence vectors are than inputs to the sentence encoder. The states from sentence encoder are used to learn the final document vector again using attention over sentence states. Basically we are trying to combine the most relevant sentences to learn the final document embedding. Finally using an architecture similar to *Fig* 3.7 these document embedding or vector is concatenated with LSTM state for the text sequence to act as the initial decoder state. Due to time complexity of JAMR parser we used a small fraction of dataset to do learning on this model, which surprisingly still achieves comparable results discussed later.

3.3 Handling large vocabularies

Neural networks are still notorious when learning on a large vocabularies. Also large vocabularies adds to the computation complexity of the model as at decode time we need to calculate $softmax$ over entire vocab. Also we do not want too many *UNK* tokens in the decoder. *UNK* is a special token assigned to out of vocabulary words. We experimented with two approaches to address this issue.

3.3.1 Byte Pair Encoding or BPE

BPE is a compression technique where the most frequent pair of consecutive bytes is replaced by a byte not in the document. A table of mappings is generated which is used for the actual encoding. For *ex* consider the sequence of text "abaabcbcaadefaa", in first iteration the most common pair that is "aa" is replaced by say "X" to give

"abXbcbcXdefX", in next iteration the next most common pair "bc" is replaced by say "Y" to give "abXYYXdefX". which is the final sequence with mapping table $X = aa$ and $Y = bc$.

BPE has been adapted for NMT [Sennrich et al., 2015] where we view the text as a sequence of characters and each word ending is marked by a special character $'@'$ *for ex.* lower will be represented as "l o w e r @". The initial vocabulary is all the characters in the corpus. BPE works by combining the most frequent pair of characters in a word to a single sub-word in each iteration thereby adding the combined sub-word to the new vocabulary. The final vocab size is the $numOfIterations + numOfChars$. One drawback of this approach in case of text summarization is that document length increases thereby making it harder for sequence learning. BPE merge operations learned from dictionary 'low', 'lowest', 'newer', 'wider' using 4 merge operations.

r @ ---------→ r@
l o ---------→ lo
lo w ---------→ low
e r@ ---------→ er@

```
i = 0
num_merges : C # number of merges to perform
word_dict : dictionary of word frequency # spaced ending with @
merge : ordered dictionary of learned merge operations
while i < num_merges:
  pairs = {}
  for word in word_dict:
    char_list = [char in word]
    for all consecutive character pairs in char_list:
      pairs[cur_char_pair] += word_dict[word]
  best_pair = max(pairs)
  word_dict_new = {}
  for word in word_dict:
    if word has best_pair:
      updated_word = concat best_pair in word_dict
      word_dict_new[updated_word] = word_dict[word]
  word_dict = word_dict_new
  update merge to include best_pair
```

Listing 3.2: BPE Algorithm

3.3.2 Pointer-Generator

In this section we will discuss the recently introduced Pointer-Generator [See et al., 2017] model which was built on the idea of copy-mechanism [Gu et al., 2016] using attention. The core of the idea is to handle out of vocabulary(OOV) words using word copying. The model is trained to learn a probability distribution not only for generating a new word from its limited vocab but also to copy a word directly from the source text. The model learns a generation probability called p_{gen}, given by

$$p_{gen} = \sum (w_{\bar{h}_s}^T * \bar{h}_t \; + \; w_x^T * x_t \; + \; b_{pg}) \tag{3.11}$$

$w_{\bar{h}_s}$, w_x, b_{pg} are model parameters, $\bar{h}_s$ is the updated decoder state given in eq 2.11, x_t is the embedding vector for current input word. p_{gen} is used to make the decision on when to generate a new word or when to copy a source word. A extended vocab is defined as words in the vocab + OOV words in the current input sequence. The new probability over the extended vocab is defined as

$$P_{ptr-gen}(w) = p_{gen} * P_{vocab}(w) \; + \; (1 - p_{gen}) * \sum_{w_i = w} \alpha_i^t \tag{3.12}$$

here w belongs to the extended vocab, a_i^t is attention weight(eq 2.9) for word w_i at time-step t. The attention mechanism can be depicted by the following figure

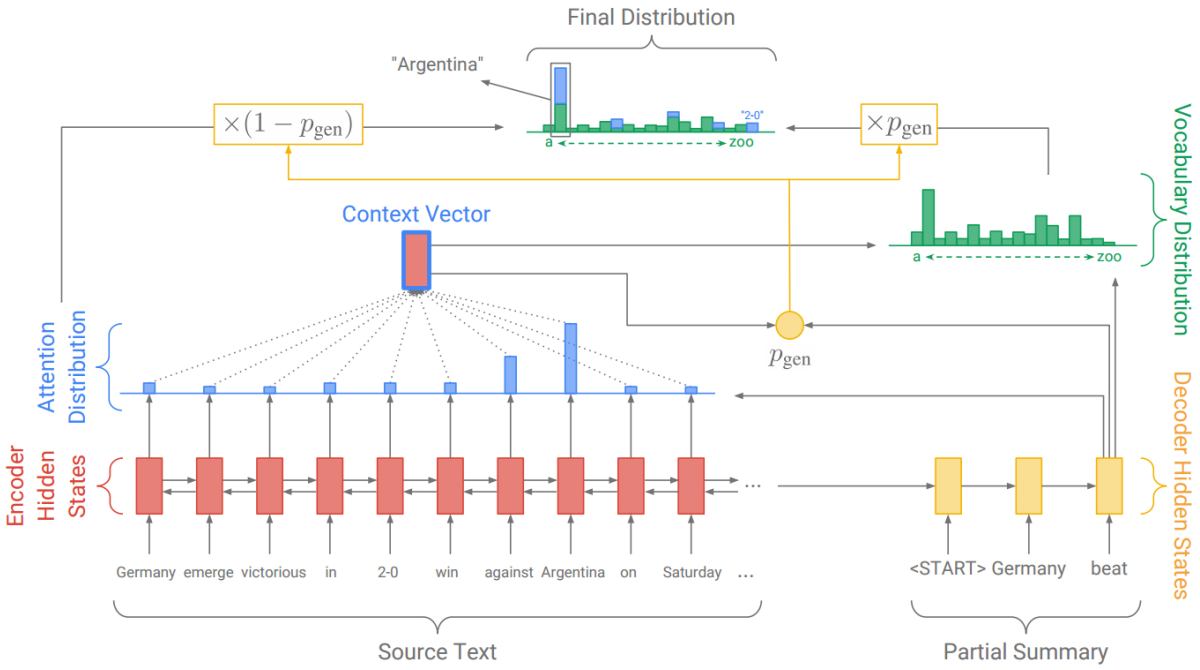


FIGURE 3.10: Pointer Generator Model source: [SEE ET AL., 2017]

Pointer generator mechanism gives us the advantage to use smaller vocabularies for training our models and still generate good summaries. As clear from figure above the decoder output can now be new word which is generated from the original limited vocab or it can be source word directly copied to the output.

3.3.3 Coverage

Another common problem with RNN based s2s models is the ability to generate same word multiple times. In Machine Translation this problem is known as coverage. A suggested solution [SEE ET AL., 2017] for this problem uses attention probabilities to penalize the repeated word generation. Using attention naturally leads to the problem of repeating same words again as the decoder input during testing is model generated, which itself was generated using attention mechanism. So a general observation is that these models do tend to have repeated outputs. So coverage mechanism acts to tame the attention from

getting wild. A coverage vector c_t at decode step t is defined as

$$c_t = \sum_{t'=1}^{t-1} \alpha^{t'} \tag{3.13}$$

where $\alpha^{t'}$'s is the attention weight at step t as was defined in eq 2.9. The coverage vector is used to calculate the updated attention scores, updating equation 2.8 to

$$e_i^t = v^T \tanh(W_{h_e} h_i + W_{h_d} h_s + w_c c_i^t + b_{attn}) \tag{3.14}$$

where W_{h_e}, W_{h_d}, w_c, b_attn are all learnable model parameters. The coverage loss at decoding step t is defined as

$$covloss_t = \sum_{i=1}^{encsteps} min(\alpha_i, c_i^t) \tag{3.15}$$

where again α_i is attention weight for encoder state i and c_t is coverage as in eq 3.13. Basically the idea is if the attention model is giving higher preference to already generated word which is supposed to have a higher coverage than it should be penalised. The total loss becomes

$$loss\_total_t = -log(P_{vocab})(w) + covloss_t \tag{3.16}$$

where P_{vocab} is the normal cross-entropy seq loss.

3.4 Extractive + Abstractive

We tested two methods to use a mixed approach for summarization. First using dependency parsing and second using Latent Semantic Analysis to get a reduced document.

3.4.1 Using Dependency Parsing

A Dependency Parse of a sentence is tree with labelled edges such that the main verb or the focused noun is the root and edges are the relations between words in a sentence. We used an approximation and considered a window of size L around the root word. Assuming the important context words generally are the neighbours of the root word, a sentence of size n is reduced to a sentence of size $2 * L + 1$. This approach reduced the source article

sizes to almost half thus making training faster. For example the sentence "Delhi is the capital of India , with lots of people." has the following dependency parse Here capital is

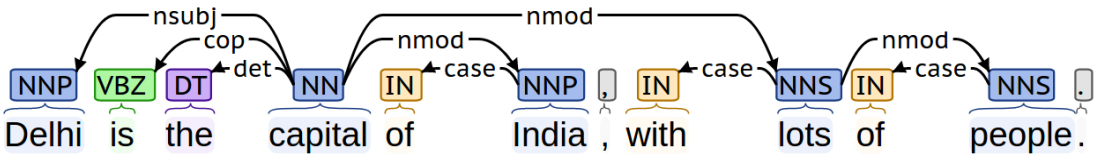


FIGURE 3.11: Dependency Parse Example

the root word and if we set L to be 3 we are able to extract the main fact in the sentence "Delhi is the capital of India ,". This helps to reduce the article lengths in our dataset and also eliminate a lot of words from vocabulary. We used value of L = 7 in our experiments.

3.4.2 Latent Semantic Analysis

Here we used the extractive summarization technique discussed in last chapter to get the most relevant sentences from the original document and do an abstractive summarization using the extracted sentences. The *CNNDailymail* dataset has on average 30+ sentences per document. Using such a large input makes this task quite challenging. We started with setting the number of extractive sentences to 5 initially but on qualitative analysis there was a huge mismatch between facts in derived sentences and the original human summaries. Only the first sentence from the summary seemed to have a match in most cases. A value of 10 seemed to have decent coverage so we did our training and testing using top 10 extractive sentences. We also tested with a value of 15. Higher values do seem to get higher scores in our testing.

Chapter 4

Results and Analysis

This chapter includes the results from models described in last chapter along with experimental settings along with qualitative analysis.

4.1 Rouge or Recall-Oriented Understudy for Gisting Evaluation

Non-human evaluation metrics for summarization and machine translation are still very far from ideal. But the problem is inherently difficult. In principle, we can have two perfectly good summaries (or translations) of the same text with very little overlap in terms of words occurring in the text of the summary except for named entities and perhaps a few other words that are likely to occur in both summaries. So, using the degree of overlap, in whatever form, between two summaries is not the best metric to determine how good an automatically produced summary is. However, current metrics use some form of overlap between the algorithmically produced summary and the human produced summary to evaluate summarization algorithms. These are the class of ROUGE metrics discussed in more detail below. For evaluation purpose we are using Rouge metric. Rouge [Lin, 2004] is basically a recall based metric which basically counts the number of overlapping n-grams or word sequences between the candidate and the reference summaries.

4.1.1 Rouge-N

Rouge-N is a n-gram based similarity measure given by

$$Rouge_N = \frac{\sum_{s \in (reference-summaries)} Count_{match_{ngram}} s}{\sum_{s \in (reference-summaries)} Count_{all_{ngram}} s} \tag{4.1}$$

where $Count_{match_{ngram}} s_i$ is the maximum number of n-grams co-occurring in the candidate summary s_c and the i^{th} reference summary s_i.

we will only consider $Rouge-1$ and $Rouge-2$ for our analysis. As an example of $Rouge-1$ consider the following suppose we have the candidate or system generated summary of length 7 as

```
the cat was found under the bed
```

and the reference or gold summary or human generated summary of length 6 as

```
the cat was under the bed
```

there are 6 unigram or word matches between the above sentences, which results in a recall value of $\frac{6}{6} = 1$ and a precision value of $\frac{6}{7} = 0.86$, thus we see that the length of generated summaries effects the Rouge value quite drastically. Normally we give recall more preference when evaluating summarization in limited settings where the length of the generated summaries are limited. Whereas in unbounded settings we will want our summaries to be concise and not include unnecessary terms. We will use the middle ground and report all precision, recall and F1-values. $Rouge-2$ is bi-gram based model and is a better measure of grammatical structure of the generated output.

4.1.2 Rouge-L

Rouge-L measures the Longest Common Sub-sequence or LCS overlap between the candidate and reference summaries. A LCS is defined as the longest sequence of matching words such that the order for words in both sentences is preserved. Formally if $x_{i_1}, \cdots, x_{i_n}$ is a subsequence of Candidate S_c and $y_{i_1}, \cdots, y_{i_n}$ is subsequence of Reference S_r such that $x_{i_k} = y_{i_k}$ than $i_1 < i_2 < \cdots < i_n$ and $j_1 < j_2 < \cdots < j_n$. Given reference summary X of length m and candidate summary Y of length n, Rouge-L is defined by the following formulas

$$R_{LCS} = \frac{LCS(X,Y)}{m} \tag{4.2}$$

$$P_{LCS} = \frac{LCS(X,Y)}{n} \tag{4.3}$$

$$Rouge - L \quad F_{LCS} = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS} + \beta^2 P_{LCS}} \tag{4.4}$$

where $LCS(X,Y)$ is the longest common subsequence of text sequences X and Y and β is variable which decides weighting for LCS Precision P_{LCS} and LCS Recall R_{LCS}. We used $\beta = 1$ in our evaluations.

Rouge-L F value is bounded by the unigram *Rouge-1 F* value as no LCS sub-sequence will have longer match than the number of uni-gram matches in two sentences. Also Rouge-L metric in general is better at capturing sentence meaning, ex consider the reference summary

```
S1: police killed the gunman
```

and two candidate summaries

```
S2: police kill the gunman
S3: the gunman kill police
```

with $\beta = 1$ S2 has $Rouge - L$ of 0.75 and S3 has $Rouge - L$ of 0.5, whereas the both sentences would have the same score for $Rouge - 2$. So we see Rouge-L is better at capturing meaning of the generated text.

4.2 Beam Search

When decoding in s2s learning models we just don't take the first sequence the decoder produces instead we try to generate multiple sequences and finally return the one with the highest score. Beam search algorithm defined below is used for this purpose we have used the beam size of 4 in our experiments, saving best 4 summaries at any time step. In s2s beam search is defined as below

```
results = [] # results list
beam_size = 4 # size of beam to be used
dec_steps = k # number of max dec steps
cur_step = 1 # step counter
hyps = [(start_state, <s>, 0)] * beam_size # current hypothesis state
while cur_step <= dec_steps:
 next_states, next_symbols = get_top_k(run_decoder_for_one_step(hyps[:][-1]),
         2*beam_size) # get top 2*beam_sized states, symbols
 if any hyps contains the </s> symbol add it to the results
 new_hyps = updated current hyps using the new states and next symbols
    ignoring </s> symbols
 hyps = new_hyps[:beam_size]
 cur_step += 1
results.append(hyps)
return sort_by_score(results)[:beam_size]
```

Listing 4.1: Beam Search

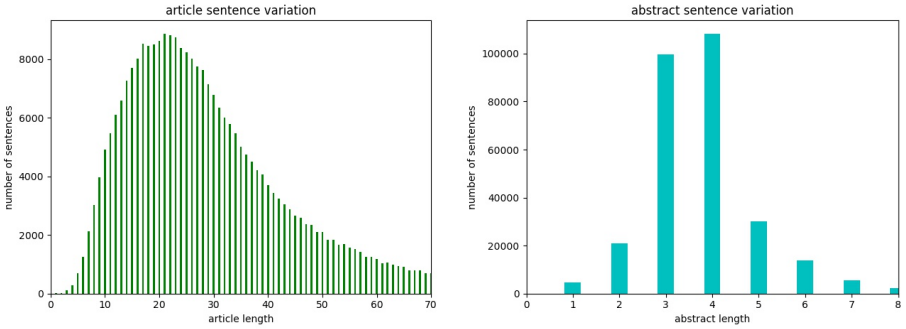
4.3 Datasets

We primarily used the *CNN/Dailymail* dataset. This dataset consists of news articles from *CNN* and *Dailymail* along with human summaries which are marked as highlights in the dataset. The training, test and validation split is as follows

Train	287,226
Test	11,490
Validation	13,368

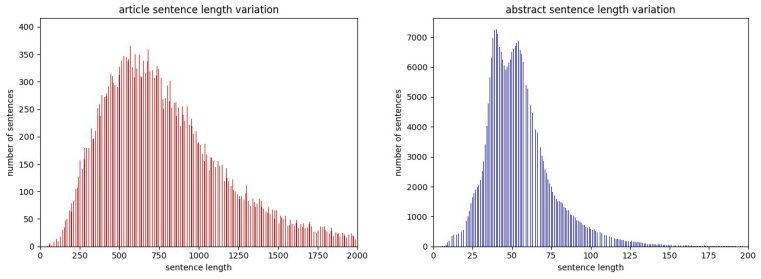
Table 4.1: CNN/Dailymail split

Following plots shows the data stats for sentence length variation and number of sentences per article on average



(a) Number of Sentences in article (b) Number of Sentences in summary

FIGURE 4.1: Number of sentences in article/summary



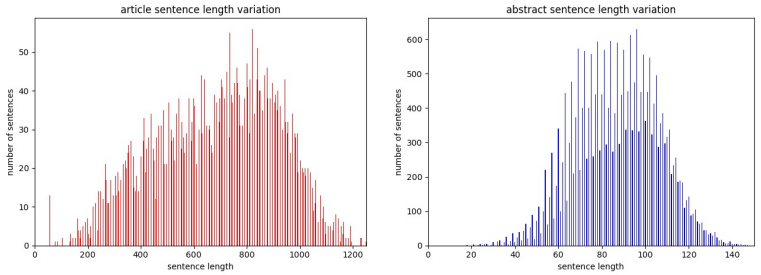
(a) Number of words in article (b) Number of words in summary

FIGURE 4.2: Number of words in article/summary

average number of sentences per article	31
average number of sentences per summary	3
average number of words per article	790
average number of words per summary	55

TABLE 4.2: CNN/Dailymail average stats for Training sets

The large average article length was primarily due to some articles being very long. These values were used to decide the hyper-parameter values for the seq2seq learning model. In general we used value of $350 - 400$ for the decoder and value of *max* 100 for the encoder.



(a) Number of words in article (b) Number of words in summary

FIGURE 4.3: Number of words in article/summary for AMR Data

Further using BPE model the average article length on complete CNN/Dailymail dataset increased to 818 with $50,000$ BPE merge operations.

4.4 Results

4.4.1 S2S results on CNN/Dailymail

This section reports results on the CNN/Dailymail dataset with various model approaches discussed in previous chapters. We used *tensorflow* [TEAM, 2015] for all our code. We used a single *GeForce GTX TITAN X* with 12 *GB* GPU memory for all our training. Training on full dataset takes on average $5-6$ days per model. The hyper-parameters for the model are

hyperparameter	**value**
LSTM state size	256
Word embedding dimensions	128
AMR embedding dimensions	128
POS embedding dimensions	128
batch size	16
encoder steps length	350-400
decoder steps length	60-100
beam size	4
learning rate for adagrad	0.15
maximum gradient norm	2.5
poss loss weight	0.25

TABLE 4.3: hyper-parameters for s2s model

R1-f	Rouge-1 F score
R1-p	Rouge-1 Precision
R1-r	Rouge-1 Recall
R2-f	Rouge-2 F score
R2-p	Rouge-2 Precision
R2-r	Rouge-2 Recall
RL-f	Rouge-L F score ($\beta = 1$)
RL-p	Rouge-L Precision
RL-r	Rouge-L Recall

TABLE 4.4: metric short-names mapping

bpe-no-cov	byte pair encoding model without coverage loss
bpe-cov	byte pair encoding model with coverage loss
pos-weighted-loss	pos model with weighted loss we are using $\lambda = 0.25$
text-no-cov	s2s on just plain text without augmentation without coverage loss
text-cov	s2s on just plain text without augmentation with coverage loss
text-200	s2s on just plain text using glove vectors with dim 200 to initialize word-embeddings
text-200-cov	s2s on just plain text using glove vectors with dim 200 to initialize word-embeddings with coverage loss
text-100	s2s on just plain text using glove vectors with dim 100 to initialize word-embeddings
text-100-cov	s2s on just plain text using glove vectors with dim 100 to initialize word-embeddings with coverage loss
pos-full	pos model with loss weight 1 for POS dist and text dist
pos-full-cov	pos model with loss weight 1 for POS dist and text dist with coverage

TABLE 4.5: Model Description

The results are reported in table below, infact our best model is getting better Rouge-2-F score than both [SEE ET AL., 2017] (17.28) and [PAULUS ET AL., 2017] (15.82).

Model	R1-f	R1-re	R1-p	R2-f	R2-re	R2-p	RL-f	RL-re	RL-p
bpe-no-cov	35.39	39.46	34.04	15.53	17.36	14.94	32.31	35.99	31.11
bpe-cov	36.31	37.3	37.52	15.69	16.11	**16.26**	**33.63**	34.52	**34.79**
pos-wt-loss*	35.46	42.79	31.47	16.54	19.99	14.71	31.83	38.37	28.27
text-no-cov	31.4	32.3	32.6	11.6	11.9	12	27.2	28.03	28.09
text-cov	33.19	33.07	35.27	12.38	12.27	13.27	28.12	27.98	29.93
text-200	34.66	41.69	30.84	16.23	19.52	14.50	30.96	37.12	27.57
text-200-cov	37.18	44.45	33.22	17.41	20.82	15.63	32.68	39.03	28.24
text-100	34.73	41.53	31.03	16.51	19.74	14.82	31.31	37.43	28.01
text-100-cov	37.54	45.03	33.46	**17.74**	21.27	15.88	33.37	39.96	29.78
pos-full*	35.76	43.15	31.72	16.9	20.43	15.04	31.86	38.4	28.29
pos-full-cov*	**37.96**	**45.67**	**33.76**	17.71	**21.32**	15.83	33.23	**39.96**	29.59

TABLE 4.6: *cov* indicates model is using coverage loss, $no - cov$ indicated model is not using coverage loss, *pos-wt-loss with $\lambda = 0.25$ model in eq 3.10, pos-full combines loss for word generation and POS tag generation with equal weight of 1 model in eq 3.9, all models use pointer copying mechanism by default except the bpe model, text-100 indicates glove vectors of size 100 were used to initialize the embeddings and text-200 indicates glove vectors of size 200 being used to initialize the embeddings

We can see that POS based models do perform better in most cases. Also initializing with pre-trained vectors resulted in better scores than randomly initialized vectors indicating the model does benefit from semantic word relations and is somehow unable to capture that information on its own might be due to lower vocab size. BPE based models perform quite well even without the *copying* mechanism as there are very few words which gets mapped to *UNK* also the lower vocab with increased word freq indicated model is able to learn good word representations for words in bpe.

4.4.2 Models using AMRs

As obtaining AMRs is a costly operation we used 25k examples for training, along with 5k for validation and 3k for testing.

4.4.2.1 AMR Augmentation

Here we used Hierarchical attention network to get embedding for AMR document, which was combined with encoding state for the text sequence.

aug-no-cov	amr augmented hierarchical model without coverage loss
aug-cov	amr augmented hierarchical model with coverage loss

TABLE 4.7: Model Description

Model	R1-f	R1-re	R1-p	R2-f	R2-re	R2-p	RL-f	RL-re	RL-p
aug-no-cov	30.18	30.23	31.71	11	11	11.64	26.52	26.6	27.91
aug-cov	**34.53**	**35.82**	**35.13**	**13.22**	**13.67**	**13.52**	**29.21**	**30.23**	**29.79**

TABLE 4.8: AMRs as Augmented Data

Even with much smaller dataset AMRs based augmentation achieved results comparable to the full dataset which was trained on much larger dataset of 200k training examples. Visualizing sentence embeddings does shows that similar amr sentences do get mapped closer to each other even after being trained on just 25k examples. For visualizing fig 4.4 we used PCA to reduce the dimensions to 2 and plotted along those axis.

4.4.2.2 AMR to AMR s2s

Here we trained the s2s model to generate amr summaries from amr text. We tested on two datasets, first the CNN/Dailymail with 25k training samples and other a subset of the gigaword headline generation dataset with 20k training examples. Gigaword headline generation is much simpler task as compared to CNN/Dailymail summary generation as we need to generate a single one sentence summary from a 2-3 sentence articles as compared to 30+ sentence articles with 3-4 lines summaries for CNN/Dailymail. Although we are training with full linearized amrs the Rouge scores are only reported using the generated AMR concepts/nodes discarding the edge relations.

Model	R1-f	R1-re	R1-p	R2-f	R2-re	R2-p	RL-f	RL-re	RL-p
s2s-cnn-1	17.97	17.44	19.88	6.31	5.91	7.21	17.35	16.83	19.18
s2s-cnn-2	25.60	26.06	27.14	8.31	8.35	8.96	25.01	25.46	25.60
s2s-cnn-3	**31.96**	**28.08**	**40.70**	**10.71**	**9.30**	**13.92**	**29.12**	**25.57**	**37.11**

TABLE 4.9: 1: cnn no pointer gen 2: cnn with pointer gen 3: cnn with cov and pointer gen

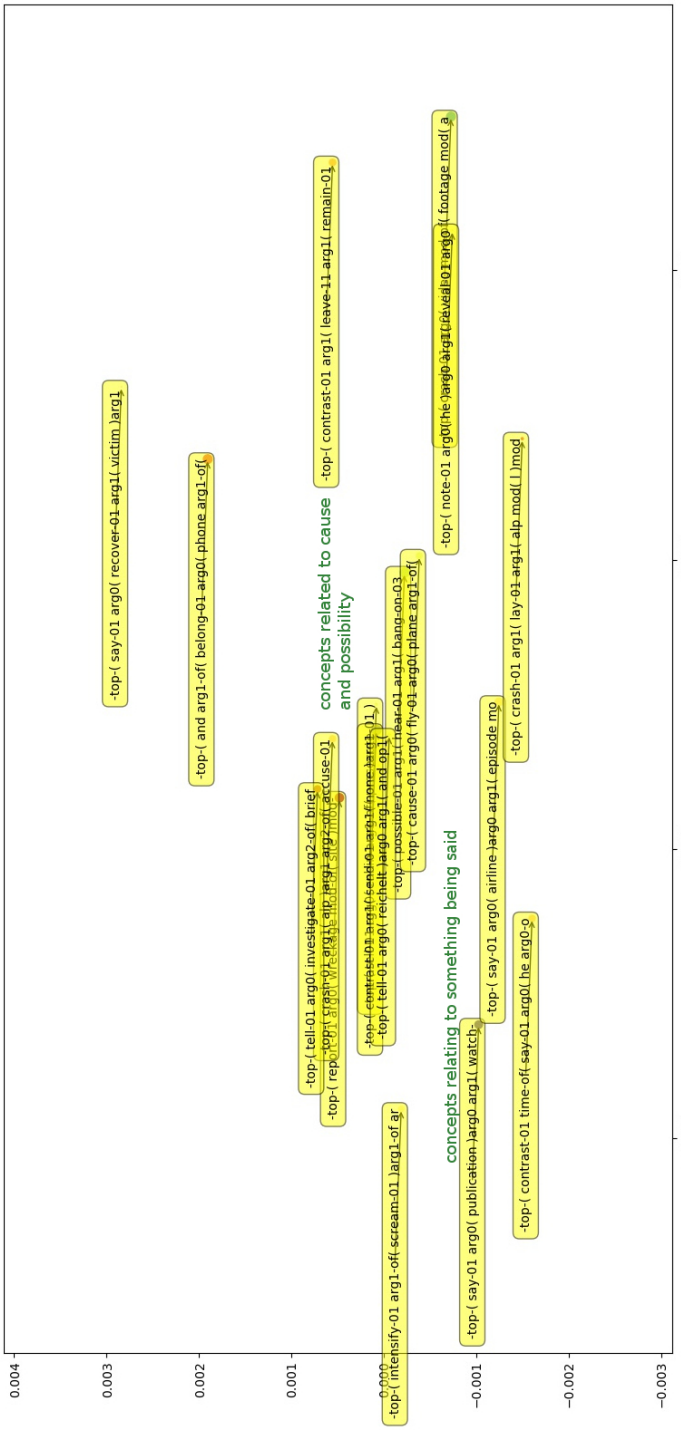


FIGURE 4.4: AMR PCA sentence embeds

Model	R1-f	R1-re	R1-p	R2-f	R2-re	R2-p	RL-f	RL-re	RL-p
s2s-giga-1	26.36	38.88	20.93	10.82	16.6	8.45	23.47	34.93	18.56

TABLE 4.10: 1: giga headline generation dataset with pgen-cov

The poor performance on the giga word dataset can be explained due to the fact that it is a highly processed dataset missing a lot of grammatical structure and a lot of words in the datset have been replaced by *UNK* resulting in poor JAMR parses. Also we notice that amr-amr generation is much harder task using s2s learning because the amr sequences are very large and the model is still struggling to capture the full amr semantics using the current representation.

4.4.2.3 Graph based summarization

Here we report results from the first approach discussed in chapter 3 in section 3.1.1. During training the number of nodes/edges to be selected is decided on the basis of gold summary graph and while testing we limit the model to select a fraction of nodes and edges which is equivalent to limiting L in equation 3.6. As the generated summaries are just *BOWs* obtained using word-node alignments, only Rouge-1 is reported.

Model	R1-f
number of edges L <= nodes/2 - 1 (ref gold)	18.59
number of edges L <= nodes/3 -1 (ref gold)	**19.72**
number of edges L <= nodes/4 -1 (ref gold)	19.57
number of edges L <= nodes/2 -1 (ref *generated)	38.05
number of edges L <= nodes/3 -1 (ref *generated)	**44.72**
number of edges L <= nodes/4 -1 (ref *generated)	43.60

TABLE 4.11: ref-gold refers when we compare the generated summaries to the original gold summaries, *ref generated summaries were generated using word alignment using nodes in the summary graph

The lower score for ref gold summaries can be due to the fact that an AMR abstracts a lot of information and many nodes don't have any alignment in the original text.

4.4.3 Extractive + Abstractive

This section reports results using mixed approach for text summarization.

dep-no-cov	dependency parses used to reduce document size, no coverage loss
dep-cov	dependency parses used to reduce document size with coverage loss
lsa-10-no-cov	lsa used to extract 10 sentences and no coverage loss
lsa-10-cov	lsa used to extract 10 sentences and with coverage loss
lsa-15-no-cov	lsa used to extract 15 sentences and no coverage loss
lsa-15-cov	lsa used to extract 10 sentences and with coverage loss

TABLE 4.12: Model Description

Model	R1-f	R1-re	R1-p	R2-f	R2-re	R2-p	RL-f	RL-re	RL-p
dep-no-cov	25.89	25.85	27.52	8.72	8.67	9.32	23.81	25.34	23.75
dep-cov	30.53	31.67	31.33	10.26	10.6	10.61	28.06	29.08	28.85
lsa-10-no-cov	29.75	30.27	31.23	11.27	11.47	11.83	27.3	27.74	28.68
lsa-10-cov	32.9	**34.41**	33.57	12.17	12.73	12.44	**30.41**	**31.76**	31.08
lsa-15-no-cov	31.64	34.15	31.51	12.34	**13.32**	12.31	28.74	30.97	28.67
lsa-15-cov	**33.22**	32.57	**36.14**	**12.86**	12.62	**14.03**	30.11	29.5	**32.78**

TABLE 4.13: Comparing the mixed approach for summarization, Dependency based method used $L = 7$ as context window value

As can be seen the performance has reduced compared to other methods the main reason can be lower overall information content, thus missing on some relevant facts. *LSA-15* performing better than *LSA-10* proves this point. Dependency based method works poorer than LSA approach. A better text extraction technique from Dependency tree might be useful as the context window approach is not fully exploiting the tree structure and seems to be missing critical information.

4.5 Some Generated Summaries

- repetition
  - **Reference:** roseanne barr told the daily beast that she is slowly going blind barr said she has macular degeneration and glaucoma
  - **Generated:** roseanne barr said she is slowly going blind in interview with the daily beast barr said she suffers from macular degeneration and glaucoma [*she suffers from macular degeneration and glaucoma*] [*she suffers from macular degeneration and glaucoma*]

- less abstraction
  - **Reference:** batman superman dawn justice trailer leaked thursday yanked offline film will released march 25 2016 stars ben affleck henry cavill
  - **Generated:** batman superman dawn justice [*leaked handheld camera phone user uploaded handheld camera phone capture trailer spanish*] superman dawn justice will released march 25 2016 [*stars affleck batman henry cavill superman film also stars amy adams*] much costs
- facts redundancy
  - **Reference:** coast guard says about 50 people were rescued from mobile bay more than 100 sailboats took part in the dauphin island race an annual event
  - **Generated:** [*sailboats took part in the dauphin island race*] and as many as 50 [*sailboats took part in the dauphin island race*] and as many as 50 people in all were rescued from water [*the coast guard says this is the 57th year*] for the event [*the club says this is the 57th year*] for the event

4.6 Attention Visualization

The following figures shows the attention heatmap produced during decoding step. This is an visualization of the attention weights which decides the preference of which encoder state to attend to when producing next word. The encoder input sequence is on the x-axis and the decoded sequence is on the y-axis. Darker color indicates higher attention weight value.

The figure 4.7 shows the words with high generation probability i.e $p_gen > 0.5$ in eq 3.11 highlighted, non marked words have $p_gen < 0.5$

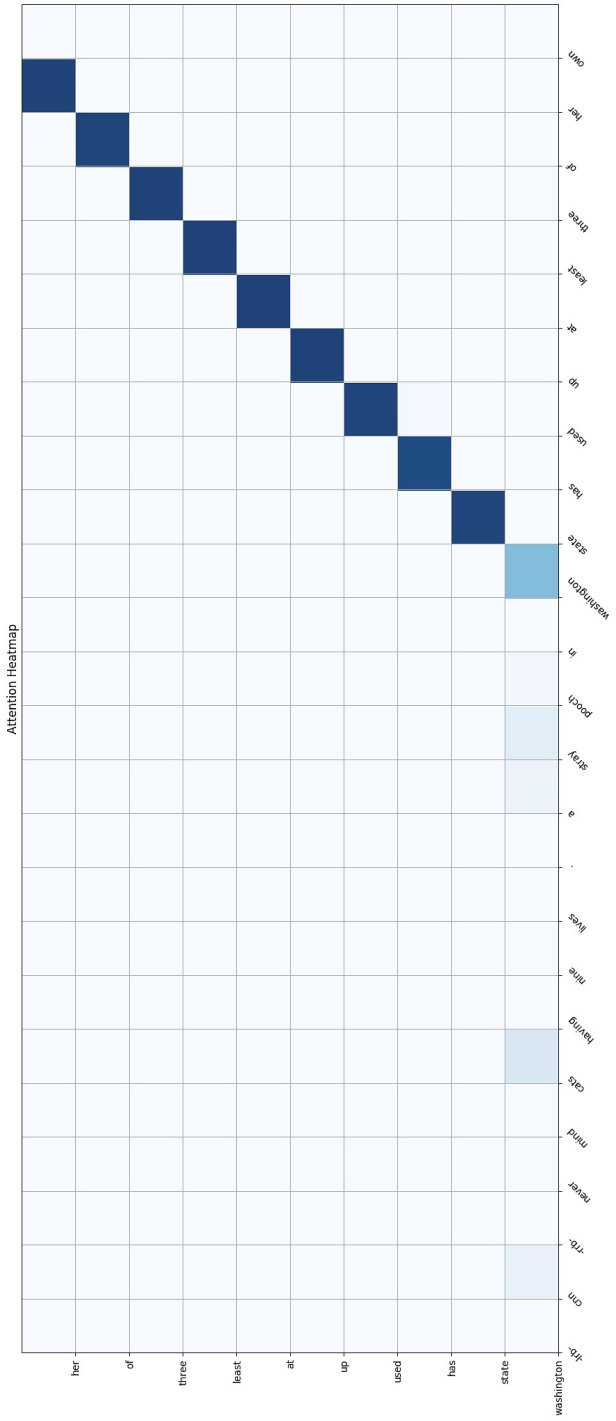


FIGURE 4.5: Attention heatmap 1

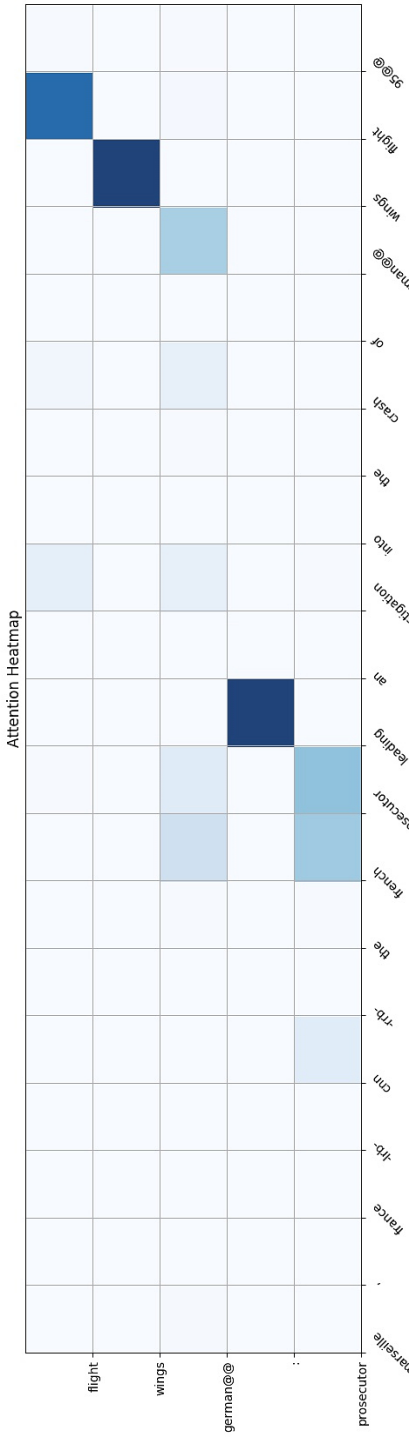


FIGURE 4.6: Attention heatmap 2

Article

-lrb- cnn -rrb- governments around the world are using the threat of terrorism -- real or perceived -- to advance executions , amnesty international alleges in its annual report on the death penalty . `` the dark trend of governments using the death penalty in a futile attempt to tackle real or imaginary threats to state security and public safety was stark last year , '' said sali@@ l she@@ tty , amnesty 's secretary general in a release . `` it is shameful that so many states around the world are essentially playing with people 's lives -- putting people to death for ` terrorism ' or to quell internal instability on the ill-@@ conceived premise of deterrence . '' the report , `` death sentences and executions 2014 , '' cites the example of pakistan lifting a six-year moratorium on the execution of civilians following the horrific attack on a school in peshawar in december . china is also mentioned , as having used the death penalty as a tool in its `` strike hard '' campaign against terrorism in the restive far-@@ western province of xinjiang . the annual report catalo@@ gs the use of state-@@ sanctioned killing as a punitive measure across the globe , and this year 's edition contains some mixed findings . on one hand , the number of executions worldwide has gone down by almost 22 % on the previous year . at least 60@@ 7 people were executed around the world in 2014 , compared to 7@@ 78 in 2013 . amnesty 's figures do not include statistics on executions carried out in china , where information on the practice is regarded as a state secret . belarus and vietnam , too , do not release data on death penalty cases . `` the long-term trend is definitely positive -- we are seeing a decrease in the number of executions -lrb- worldwide -rrb- , '' audrey gau@@ gh@@ ran , amnesty 's director of global issues , told cnn . `` a number of countries are closer to abolition , and

Reference summary

amnesty 's annual death penalty report catalo@@ gs encouraging signs , but setbacks in numbers of those sentenced to death . organization claims that governments around the world are using the threat of terrorism to advance executions . the number of executions worldwide has gone down by almost 22 % compared with 2013 , but death sentences up by 28 % .

Generated summary (highlighted = high generation probability)

amnesty international says death sentences and executions 2014 . report cites the example of pakistan lifting a six-year moratorium on the execution of civilians . china is also mentioned , as having used the death penalty as a tool in its `` strike hard '' campaign .

FIGURE 4.7: Attention Probability for decoding

Chapter 5

Conclusion and Future work

5.1 Conclusion

In this work we explored the efficacy of some of the recent techniques for text summarization. We looked both into the some more recent extractive algorithms and the latest work using deep neural nets with attention. We tried to study how the combination of both methods affects the quality of summaries. Extractive methods always degrades the summaries due to lack of complete information but requires lower training times. We adopted a very generic approach of selecting neighbouring words, exploiting dependency tree structure hoping to get better results. We looked into AMRs and their applicability to the task of summarization and their effectiveness on a smaller data set. It looks promising but we would need to develop better ways to get richer embeddings for graphical data. Still the lack of complete graph embeddings presents some issues which need to be solved. With the recent advances in AMR parsing [Buys and Blunsom, 2017], there is a lot of potential in this direction. Techniques like pointer based copying are effective but we do loose on the abstractiveness of the generated text. Encoding techniques like BPE do allow us to use larger vocabularies but the task becomes harder for the sequence based models to learn and also it is not clear what should be the vocabulary threshold to use. We also saw that using *POS* tags based information does give some structure to the generated text by getting better Rouge-2 scores.

5.2 Future work

- There has been a recent work using Reinforcement learning [Paulus et al., 2017] for summarization which has produced encouraging results. So this is one direction we must look into.
- Another area still not fully explored is of embedding graph based structures using deep learning. There has been some work like the deepwalk [Perozzi et al., 2014] for node embeddings but it still is fairly local in scope, an appropriate global graph embedding method is still to be discovered.
- Data augmentation techniques like using POS tags does seem improve performance, but they also increase the model complexity. Searching for alternate ways to use the extra information in deep learning models is another area to look into.
- We saw if AMRs can be used to represent text meaning then we can exploit their structure but they seem to be too abstract. There is no easy way to connect between the AMR of one sentence to the AMR of another sentence in the same document. A better way to compose the meaning representation of a document from the meaning representations of individual sentences in the document can be a major area to look into which combines the power of AMRs with better cross referencing.
- We also need to invent better ways to evaluate summaries. The current ROUGE class of metrics which completely depend on overlaps are not adequate. They miss out on grammaticality and readablity of the summary. It seems possible to produce high ROUGE-1 and ROUGE-2 scores algorithmically by directly optimizing for such metrics without necessarily getting proper summaries.

Bibliography

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

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2012). Abstract meaning representation (amr) 1.0 specification. In *Parsing on Freebase from Question-Answer Pairs. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. Seattle: ACL*, pages 1533–1544.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bonial, C., Babko-Malaya, O., Choi, J. D., Hwang, J., and Palmer, M. (2010). Propbank annotation guidelines. *Center for Computational Language and Edu-cation Research Institute of Cognitive Science Uni-versity of Colorad at Boulder*.

Buys, J. and Blunsom, P. (2017). Robust incremental neural semantic graph parsing. *arXiv preprint arXiv:1704.07092*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Erkan, G. and Radev, D. R. (2004). Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479.

Flanigan, J., Thomson, S., Carbonell, J. G., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the abstract meaning representation.

Gimpel, K. and Smith, N. A. (2012a). Structured ramp loss minimization for machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL

HLT '12, pages 221–231, Stroudsburg, PA, USA. Association for Computational Linguistics.

Gimpel, K. and Smith, N. A. (2012b). Structured ramp loss minimization for machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231. Association for Computational Linguistics.

Gong, Y. and Liu, X. (2001). Generic text summarization using relevance measure and latent semantic analysis. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–25. ACM.

Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Langkilde, I. and Knight, K. (1998). Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 704–710. Association for Computational Linguistics.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. Insight.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain.

Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. (2015). Toward abstractive summarization using semantic representations.

Martins, A. F. and Smith, N. A. (2009). Summarization with a joint model for sentence extraction and compression. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*, pages 1–9. Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer.

Paulus, R., Xiong, C., and Socher, R. (2017). A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.

Peng, X., Wang, C., Gildea, D., and Xue, N. (2017). Addressing the data sparsity issue in neural amr parsing. *arXiv preprint arXiv:1702.05053*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM.

Schneider, N., Flanigan, J., and O'Gorman, T. (2015). The logic of amr: Practical, unified, graph-based sentence semantics for nlp.

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

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Steinberger, J. and Jezek, K. (2004). Using latent semantic analysis in text summarization and summary evaluation. In *Proc. ISIM'04*, pages 93–100.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Team, T. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems.(2015). *URL http://tensorflow. org/. Software available from tensorflow. org*.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*, pages 1480–1489.