JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

TEXT SUMMARIZATION

Shravani Soma. Stanley College of engineering and technology for women, Hyderabad, India
Kacharla Navya. Stanley College of engineering and technology for women, Hyderabad, India
K. Manilekha. Stanley College of engineering and technology for women, Hyderabad, India

Abstract A vast quantity of text data is generated on the internet, be it social media articles, news articles, etc. Manual creation of summaries is time-consuming, and therefore a need for automatic text summaries has arisen. Abstractive Text Summarization is the task of generating a short and concise summary that captures the salient ideas of the source text. The generated summaries potentially contain new phrases and sentences that may not appear in the source text. In this work, we model abstractive text summarization using Attentional Encoder Decoder using transformers, and show that they achieve state-of-the-art performance on two different corpora. We studied on several novel models that address critical problems in summarization that are not adequately modelled by the basic architecture, such as modelling key-words, capturing the hierarchy of sentence-to word structure, and emitting words that are rare or unseen at training time. Our work shows that our proposed model contribute to further improvement in performance. In future, we will also try to propose a new dataset consisting of multisentence summaries, and establish performance benchmarks for further research.

1. INTRODUCTION

1.1 Introduction

With the present explosion of data circulating the digital space, which is mostly nonstructured textual data, there is a need to develop automatic text summarization tools that allow people to get insights from them easily. Currently, we enjoy quick access to enormous amounts of information. However, most of this information is redundant, insignificant, and may not convey the intended meaning. For example, if you are looking for specific information from an online news article, you may have to dig through its content and spend a lot of time weeding out the unnecessary stuff before getting the information you want. Therefore, using automatic text summarizers capable of extracting useful information that leaves out inessential and insignificant data is becoming vital. Implementing summarization can enhance the readability of documents, reduce the time spent in researching for information, and allow for more information to be fitted in a particular area.

1.2 Problem Statement

Techniques like multi-layer preceptorn (MLP) work well your input data is vector and convolutional neural networks works very well if your input data is an image. What if my input x is a sequence? What if x is a sequence of words? In most languages sequence of words matters a lot. We need to somehow preserve the sequence of words. The core idea here is if output depends on a sequence of inputs, then we need to build a new type of neural network which gives importance to sequence information, which somehow retains and leverages the sequence information

1.3 Objective

We can build a Seq2Seq model on any problem which involves sequential information. In our case, our objective is to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a summary (which is a sequence as well). So, we can model this as a Many-to-Many Seq2Seq problem. A many to many seq2seq model have two building blocks- Encoder and Decoder. The Encoder-Decoder BE VIII-Sem IT Dept Stanley College of Engineering & Technology for Women 3 architecture is mainly used to solve the sequence-to-sequence (Seq2Seq) problems where the input and output sequences are of different lengths. Generally, variants of Recurrent Neural Networks (RNNs), i.e., Gated Recurrent Neural Network (GRU) or Long ShortTerm Memory (LSTM), are preferred as the encoder and decoder components. This is because they are capable of capturing long term dependencies by overcoming the problem of vanishing gradient.

.

2. Literature Survey

Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond

In this work, Ramesh Nallapati had modelled an abstractive text summarization using Attentional Encoder Decoder Recurrent Neural Networks, and shown that they achieve state-of-the-art performance on two different corpora. He proposed several novel models that address critical problems in summarization that are not adequately modelled by the basic architecture, such as modelling key-words, capturing the hierarchy of sentence-to word structure, and emitting words that are rare or unseen at training time. Their work shows that many of their proposed models contribute to further improvement in performance. They also proposedd a new dataset consisting of multisentence summaries, and establish performance benchmarks for further research.

Deep Reinforcement Learning for Sequence-to-Sequence Models

In recent times, sequence-to-sequence (seq2seq) models have gained a lot of popularity and provide state-of the-art performance in a wide variety of tasks such as machine translation, headline generation, text summarization, speech to text conversion, and image caption generation. The underlying framework for all these models is usually a deep neural network comprising an encoder and a decoder. Although simple encoder decoder models produce competitive results, many researchers have proposed additional improvements over these seq2seq models, e.g., using an attention-based model over the input, pointer generation models, and self-attention models. However, such seq2seq models suffer from two common problems: 1) exposure bias and 2) inconsistency between train/test measurement. Recently, a completely novel point of view has emerged in addressing these two problems in seq2seq models, leveraging methods from reinforcement learning (RL). In this survey, Yaser Keneshloo considered seq2seq problems from the RL point of view and provide a formulation combining the power of RL methods in decision-making with seq2seq models that enable remembering long-term memories. They presented some of the most recent frameworks that combine concepts from RL and deep neural networks. Their work aims to provide BE VIII-Sem IT Dept Stanley College of Engineering & Technology for Women 8 insights into some of the problems that inherently arise with current approaches and how they can address them with better RL models. They also provided the source code for implementing most of the RL models discussed in this paper to support the complex task of abstractive text summarization and provide some targeted experiments for these RL models, both in terms of performance and training time

A Large-Scale Text Summarization Dataset

• Sequence-to-sequence models have recently gained the state-of-the-art performance in summarization. However, not too many large-scale high-quality datasets are available and almost all the available ones are mainly news articles with specific writing style. Moreover, abstractive human-style systems involving description of the content at a deeper level require data with higher levels of abstraction. In this paper, Mahnaz Koupaee presented WikiHow, a dataset of more than 230,000 article and summary pairs extracted and constructed from an online knowledge base written by different human authors. The articles span a wide range of topics and therefore represent high diversity styles. They evaluated the performance of the existing methods on WikiHow to present its challenges and set some baselines to further improve it.

3. OVERVIEW OF THESYSTEM

3.1 Existing System

• In the past few years, neural abstractive text summarization with sequence-tosequence (seq2seq) models have gained a lot of popularity. Many interesting techniques have been proposed to improve seq2seq models, making them capable of handling different challenges, such as saliency, fluency and human readability, and generate high-quality summaries. Generally speaking, most of these techniques differ in one of these three categories: network structure, parameter inference, and decoding/generation. There are also other concerns, such as efficiency and parallelism for training a model. In this paper, Yaser Keneshloo provided a comprehensive literature survey on different seq2seq models for abstractive text summarization from the viewpoint of network structures, training strategies, and summary generation algorithms. Several models were first proposed for language.

3.2 Proposed System

The sequence-to-sequence encoder-decoder architecture is the base for sequence transduction tasks. It essentially suggests encoding the complete sequence at once and then using this encoding as a context for the generation of decoded sequence or the target sequence. The seq2seq model consists of separate RNNs at encoder and decoder respectively. The encoded sequence is the hidden state of the RNN at the encoder network. Using this encoded sequence and (usually) word-level generative modelling, seq2seq generates the target sequence. Since encoding is at the word-level, for longer sequences it is difficult to preserve the context at the encoder, hence the well-known attention mechanism was incorporated with seq2seq to 'pay attention' at specific words in the sequence that prominently contribute to the generation of the target sequence. Attention is weighing individual words in the input sequence according to the impact they make on the target sequence generation.

3.3 Proposed System Design

In this project work, I used five modules and each module has own functions, such as:

1. The Dataset
2. Preprocessing
3. Utility Functions
4. The Model
5. Training the Model
6. Inference

3.3.1 Dataset

For this task, we have used Inshorts Dataset. Inshorts is a service that collects news from various sources and publishes them as a summary ('in short'). "Stay informed in 60 words." is the tagline of the service. For training, we will consider the headline as the summary of a news article and the original article will be the context. The columns which were unnecessary for this task (Source, Time, Date) are removed. The dataset has ~55k samples of news-headlines pairs. There are ~76k unique words in the news data and ~29k in the headlines. The sample with the longest news sequence consists of 469 words and the average length of news is 368 words, while that of the headlines is 96 and 63 words respectively.

3.3.2 Preprocessing

Here we perform basic preprocessing that is required to train any NLP model. For

recognizing the start and end of target sequences, we pad them with start ("<go>") and

end ("<stop>") tokens

We fit a Tokenizer on the sequences which essentially filters punctuation marks,

converts text to lower case, maintains a vocabulary dict which is ordered by frequency of occurrence of words and contains a mapping of words with their token equivalents and

finally, converts the textual data to tokens which can be directly inputted to the model.

Code for Tokenization:

```
# since < and > from default tokens cannot be removed
filters = '!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n'
oov_token = '<unk>'

document_tokenizer =
tf.keras.preprocessing.text.Tokenizer(oov_token=oov_token)
summary_tokenizer =
tf.keras.preprocessing.text.Tokenizer(filters=filters,
oov_token=oov_token)

document_tokenizer.fit_on_texts(document)
summary_tokenizer.fit_on_texts(summary)

inputs = document_tokenizer.texts_to_sequences(document)
targets = summary_tokenizer.texts_to_sequences(summary)
```

3.3.3 *Utility Functions*

We finally move on to the components of the Transformer which will directly contribute to the model.

Code snippet for Positional Encoding:

```
def get_angles(position, i, d_model):
    angle_rates = 1 / np.power(10000, (2 * (i // 2)) /
np.float32(d_model))
    return position * angle_rates

def positional_encoding(position, d_model):
    angle_rads = get_angles(
        np.arange(position)[:, np.newaxis],
        np.arange(d_model)[np.newaxis, :],
        d_model
    )

    # apply sin to even indices in the array; 2i
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

    # apply cos to odd indices in the array; 2i+1
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

    pos_encoding = angle_rads[np.newaxis, ...]

    return tf.cast(pos_encoding, dtype=tf.float32)
```

```
>>> a = tf.constant([0.6, 0.2, 0.3, 0.4, 0, 0, 0, 0, 0, 0])
>>> tf.nn.softmax(a)
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([0.15330984, 0.10276665, 0.11357471, 0.12551947, 0.08413821,
0.08413821, 0.08413821, 0.08413821, 0.08413821, 0.08413821],
dtype=float32)>
>>> b = tf.constant([0.6, 0.2, 0.3, 0.4, -1e9, -1e9, -1e9, -1e9, -
1e9, -1e9])
>>> tf.nn.softmax(b)
<tf.Tensor: shape=(10,), dtype=float32, numpy= array([0.3094101 ,
0.20753784, 0.22936477, 0.25368732, 0. ,0. , 0. , 0. , 0.
],dtype=float32)>
```

3.3.4 The Model

We arrive at the construction of the Transformer model

Code snippet for scaled dot product:

```
def scaled_dot_product_attention(q, k, v, mask):
    matmul_qk = tf.matmul(q, k, transpose_b=True)

    dk = tf.cast(tf.shape(k)[-1], tf.float32)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

    if mask is not None:
        scaled_attention_logits += (mask * -1e9)

    attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-
1)

    output = tf.matmul(attention_weights, v)
    return output, attention_weights
```

This is a regular two-layered Feed-Forward Network which is used after almost every sub-layer and is used identically.

Code snippet:

```
def point_wise_feed_forward_network(d_model, dff):
    return tf.keras.Sequential([
        tf.keras.layers.Dense(dff, activation='relu'),
        tf.keras.layers.Dense(d_model)
    ])
```

The Encoder and Decoder Blocks:

These are the fundamental units of encoder and decoder respectively. These may be expanded into *N*, encoder/decoder layers while tuning the model.

Code snippet for Encoder Block:

```
class EncoderLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads, dff, rate=0.1):
        super(EncoderLayer, self).__init__()

        self.mha = MultiHeadAttention(d_model, num_heads)
        self.ffn = point_wise_feed_forward_network(d_model, dff)

        self.layernorm1 =
tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 =
tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)

    def call(self, x, training, mask):
        attn_output, _ = self.mha(x, x, x, mask)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(x + attn_output)

        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        out2 = self.layernorm2(out1 + ffn_output)

        return out2
```

Stacking the Layers in the 'Model':

Code snippet:

```
class Transformer(tf.keras.Model):
    def __init__(self, num_layers, d_model, num_heads, dff,
input_vocab_size, target_vocab_size, pe_input, pe_target, rate=0.1):
        super(Transformer, self).__init__()

        self.encoder = Encoder(num_layers, d_model, num_heads, dff,
input_vocab_size, pe_input, rate)

        self.decoder = Decoder(num_layers, d_model, num_heads, dff,
target_vocab_size, pe_target, rate)

        self.final_layer = tf.keras.layers.Dense(target_vocab_size)

    def call(self, inp, tar, training, enc_padding_mask,
look_ahead_mask, dec_padding_mask):
        enc_output = self.encoder(inp, training, enc_padding_mask)

        dec_output, attention_weights = self.decoder(tar, enc_output,
training, look_ahead_mask, dec_padding_mask)

        final_output = self.final_layer(dec_output)

        return final_output, attention_weights
```

Finally, we stack all the intermediate layers in a Custom Model class inherited from tf.keras.Model class.We select the appropriate model or the degree of the polynomial (if using regression model only) by minimizing the error on the cross-validation set.

3.3.5 Training Model:

Following are the hyper-parameter values that we have used while training the summarizer:

```
# hyper-params
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
EPOCHS = 20

BUFFER_SIZE = 20000
BATCH_SIZE = 64

encoder_maxlen = 400
decoder_maxlen = 70

encoder_vocab_size, decoder_vocab_size = 76362, 29681
```

Following is a snippet of the verbose of a few final epochs of training:

```
Epoch 18 Batch 0 Loss 1.6277
Epoch 18 Batch 429 Loss 1.7935
Epoch 18 Batch 858 Loss 1.8485
Epoch 18 Loss 1.8489
Time taken for 1 epoch: 298.76677560806274 secsEpoch 19 Batch
0 Loss 1.6240
Epoch 19 Batch 429 Loss 1.7632
Epoch 19 Batch 858 Loss 1.8245


Epoch 19 Loss 1.8252
Time taken for 1 epoch: 298.52965450286865 secsEpoch 20 Batch
0 Loss 1.7538
Epoch 20 Batch 429 Loss 1.7412
Epoch 20 Batch 858 Loss 1.8018
Saving checkpoint for epoch 20 at checkpoints/ckpt-8
Epoch 20 Loss 1.8020
Time taken for 1 epoch: 299.6963185787201 secs
```

3.3.6 INFERENCE

we need to write a custom inference step as the model is used to receiving target sequence as an input at the decoder (Teacher Forcing). However, while testing, we need to use the previous predictions as input at the decoder.

Code snippet:

```
def evaluate(input_document):
    input_document =
document_tokenizer.texts_to_sequences([input_document])
    input_document =
tf.keras.preprocessing.sequence.pad_sequences(input_document,
maxlen=encoder_maxlen, padding='post', truncating='post')

    encoder_input = tf.expand_dims(input_document[0], 0)

    decoder_input = [summary_tokenizer.word_index["<go>"]]
    output = tf.expand_dims(decoder_input, 0)

    for i in range(decoder_maxlen):
        enc_padding_mask, combined_mask, dec_padding_mask =
create_masks(encoder_input, output)

        predictions, attention_weights = transformer(
            encoder_input,
            output,
            False,
            enc_padding_mask,
            combined_mask,
            dec_padding_mask
        )

        predictions = predictions[: ,-1:, :]
        predicted_id = tf.cast(tf.argmax(predictions, axis=-1),
tf.int32)

        if predicted_id == summary_tokenizer.word_index["<stop>"]:
            return tf.squeeze(output, axis=0), attention_weights

        output = tf.concat([output, predicted_id], axis=-1)

    return tf.squeeze(output, axis=0), attention_weights
```

4 Architecture

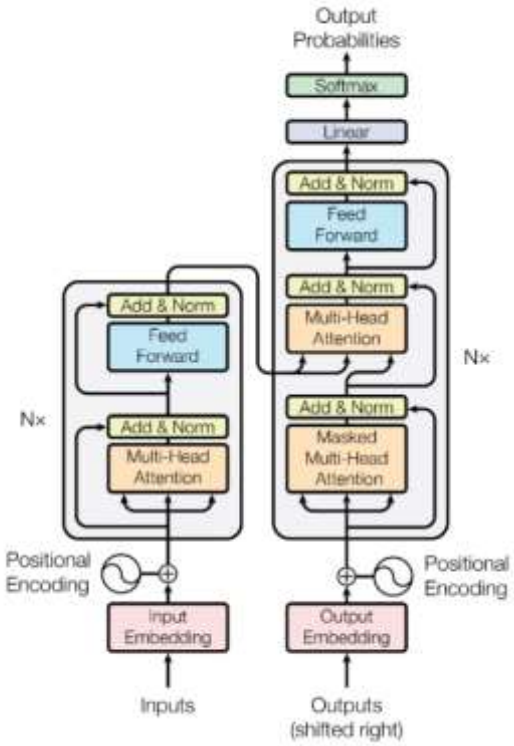


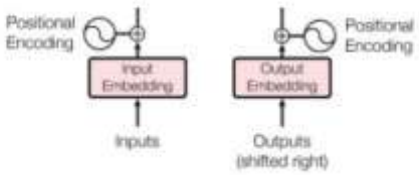
Fig 1: Transformer Architecture

Above architecture diagram shows three stages of data flow form one module to another module. Back-end storage use data to store in cloud server proxy server manages data by handling container data share package and share cache data, Client-side uploads data and manage data in effective way which satisfies container management system.

Algorithm:

Positional Encodings: These functions are responsible for obtaining the positional encodings for the input sequences. Positional Encodings basically induces a notion of ordering among the input words, since the self-attention mechanism used in the Transformer has no regard for it.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$



Here, i is the dimension and pos is the position of the word. We use sine for even values (2i) of dimensions and cosine for odd values (2i + 1). There are several choices for positional encodings — learned or fixed. This is the fixed

way as the paper states learned as well as fixed methods achieved identical results.

Masking:

Padding mask is responsible for ignoring the external padding added to sequences that are shorter than maxlen. Look-ahead mask is responsible for ignoring the words that occur after a given current word in the target sequence from contributing to the prediction of the current word.

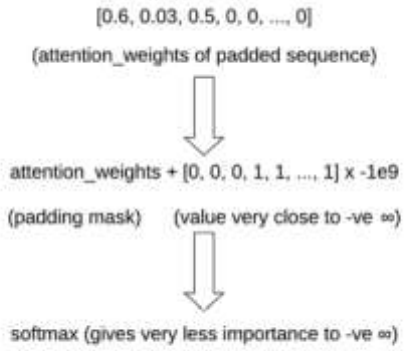


Fig 3.4: Working of a Padding Mask

Padding Mask: The input vector of the sequences is supposed to be fixed in length. Hence, a max_length parameter defines the maximum length of a sequence that the transformer can accept. All the sequences that are greater in length than max_length is truncated while shorter sequences are padded with zeros. The zero-paddings, however, are not supposed to contribute to the attention calculation nor in the target sequence generation. The working of padding mask is explained in the adjacent figure. This is an optional operation in the Transformer. Look-ahead Mask: While generating target sequences at the decoder, since the Transformer uses self-attention, it tends to include all the words from the decoder inputs. But practically this is incorrect. Only the words preceding the current word may contribute to the generation of the next word. Masked Multi-Head Attention ensures this

5 RESULTS

Here, we obtain the output for the first word, append it to the first word and jointly obtain the third word as the output for the sequence of the first and the second word, again appending this output to the decoder inputs and repeating until either the length of output sequence reaches maxlen, or the predicted word is “”..

Fig 80-20 split GBDT distribution graph

Fig 5.1: Inferring on Test data

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Table 6.1: The Transformer achieves better BLEU scores than previous state-of-the-art models

In Table 3 rows (A), we vary the number of attention heads and the attention key and value dimensions, keeping the amount of computation constant. To evaluate the importance of different components of the Transformer, we varied our base model in different ways, measuring the change in performance on English-to-German translation on the development set.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Table 6.2: Variations on the Transformer architecture.

6. CONCLUSION

In this work, we apply the attentional encoder-decoder for the task of abstractive summarization using transformer with very promising results, outperforming state-ofthe-art results significantly on two different datasets. Our proposed novel model addresses a specific problem in abstractive summarization, yielding further improvement in performance. As part of our future work, we plan to focus our efforts on the data and build more robust model for summaries consisting of multiple documents. We’ve seen the issues with RNN-based approaches in sequence transduction tasks and how the revolutionary, Transformer model addresses them. Later,

we studied the mechanisms underlying the Transformer and their working. Finally, we saw how various components and the underlying mechanisms work with the Transformer. We've successfully implemented the well-know Transformer model using the Tensor Flow API. We have also addressed a fairly difficult use case in deep learning and obtained decent results on it..

✓

7. References

[1] AAIC [2] A Srikanth, AS Umasankar, S Thanu... - 2020 5th International 2020- ieeexplore.ieee.org

[3] https://arxiv.org/abs/1706.03762

[4]https://towardsdatascience.com/text-summarization-using-deep-learning6e379ed2e89c

[5]https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-textsummarization-using-deep-learning-python/

[6]https://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machinelearning/

[7] https://www.hindawi.com/journals/mpe/2020/9365340/

[7] https://www.thesmartcube.com/ai-lab/experiments/7424/

[8] L Huang, S Cao, N Parulian, H Ji, L Wang - arXiv preprint arXiv..., 2021 - arxiv.org

[9] S Subramanian, R Li, J Pilault, C Pal - arXiv preprint arXiv:1909.03186, 2019 - arxiv.org

[10] T Cai, M Shen, H Peng, L Jiang, Q Dai - CCF International Conference on ..., 2019 - Springer

[11] U Khandelwal, K Clark, D Jurafsky, L Kaiser - arXiv preprint arXiv..., 2019 - arxiv.org BE VIII-Sem IT Dept Stanley College of Engineering & Technology for Women 30

[12] www.analyticsvidhya.com

[13] Y Liu, M Lapata - arXiv preprint arXiv:1905.13164, 2019-arxiv.org

[14] Y Liu, M Lapata - arXiv preprint arXiv:1908.08345, 2019 - arxiv.org

[15] Y Iwasaki, A Yamashita, Y Konno... - ... on Technologies and ..., 2019 - ieeexplore.ieee.org [16] Y Zhao, M Saleh, PJ Liu- arXiv preprint arXiv:2006.10213, 2020 - arxiv.or