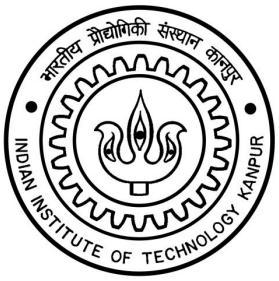
Abstractive Text Summarisation

Dhruv Aggarwal(231080033)
Ishi Jain(220464)
Naman Manchanda(231080061)
Nandini Bhattad(220693)
Shantaprasad Kamat(231080081)



A project report fulfilling the MTH209 requirements for BS and MS students at the Department of Mathematics and Statistics, Indian Institute of Technology Kanpur.

April 2024

Supervisor Dr. Subhajit Dutta

Contents

1 Acknowledegement **1**

2 Abstract **1**

3 Introduction **2**

4 Previous Work **3**

5 Methodology **4**

5.1 Input: . . . 4
5.2 Text cleaning: . . . 4
5.3 Tokenisation: . . . 4
5.4 Encoder: . . . 5
5.5 Function: . . . 5
5.6 LSTM: . . . 6
5.7 Decoder: . . . 7
5.8 Model Architecture . . . 8
5.9 Training the model: . . . 9

6 Model metrics and Evaluation **11**

6.1 Understanding the distribution of the sequences . . . 11
6.2 Some Experimentation . . . 12

7 Future Scope **13**

8 Conclusion **13**

1 Acknowledegement

We would like to express our gratitude to Mr. Subhajit Dutta for his invaluable guidance and support throughout the duration of this project. His expertise and insightful feedback greatly contributed to the success of our endeavors. We thank him for providing me with the knowledge and skills necessary to undertake this endeavor.
We also extend our sincere appreciation to the esteemed institution, Indian Institute of Technology Kanpur, for providing us with the necessary resources and environment conducive to our research and learning.
We acknowledge the assistance received from both Mr. Dutta and IIT Kanpur, which has been instrumental in shaping the outcome of this project.

2 Abstract

This project delves into the realm of text summarization, an essential task in natural language processing aimed at condensing lengthy texts into concise summaries while preserving key information. The report explores abstractive approaches to text summarization, highlighting their respective methodologies and applications. It discusses prominent models and techniques used in text summarization, including TextRank, Gensim's Summarization Module, and Sequence-to-Sequence (Seq2Seq) models. The report then delves into the methodology employed for abstractive text summarization, covering text cleaning, tokenization, word embedding using Word2Vec, and the use of the Universal Approximation Theorem for function approximation. It further discusses the Long Short-Term Memory (LSTM) architecture, a pivotal development in addressing the issues of exploding and vanishing gradients in recurrent neural networks (RNNs) for sequence-to-sequence modeling tasks. The report elucidates the model architecture and training process, along with potential future enhancements such as incorporating attention mechanisms, increasing training dataset size, and utilizing Bi-Directional LSTM layers. Through experimentation and analysis, the report concludes with insights into the effectiveness and scope for improvement in text summarization models, paving the way for further advancements in the field.

3 Introduction

Text summarization is the process of condensing a piece of text, such as an article, document, or a larger body of text, into a shorter version while retaining its key points and main ideas. It aims to provide a concise representation of the original text, making it easier and quicker to comprehend. Summarization can be done manually by humans or automatically using algorithms and natural language processing techniques. Text summarisation can be classified as abstractive and extractive. Extractive text summarization, on the other hand, involves selecting and assembling existing sentences or phrases directly from the original text to create a summary. It identifies and extracts the most relevant sentences based on criteria such as importance, relevance, or frequency of occurrence. Extractive summarization typically preserves the original wording, making it simpler to implement but potentially limiting the coherence and fluency of the summary.

Abstractive text summarization involves generating a summary that may contain new phrases or sentences not present in the original text, rephrasing the content to convey the main ideas concisely while maintaining coherence. It employs natural language generation techniques to produce summaries that are more human-like and can potentially offer a more comprehensive understanding of the text.

This transformative technology opens up a world of possibilities across various domains. In journalism, it facilitates the rapid distillation of lengthy articles into informative headlines or bullet points, aiding readers in quickly grasping the main points of a story. In academia, researchers can use abstractive summarization to efficiently summarize vast amounts of literature, enabling them to stay abreast of the latest developments in their field. Moreover, in the business world, executives can leverage this capability to extract actionable insights from extensive reports, enabling faster decision-making and driving innovation.This project is focused on **abstractive text summarisation**.

4 Previous Work

Several models have been previously used for text summarization, both abstractive and extractive. Here are a few prominent ones:

- **TextRank:** Inspired by Google's PageRank algorithm, TextRank [1] is an extractive summarization technique that assigns scores to sentences based on their importance within the document's graph structure. Latent Semantic Analysis (LSA): LSA is a statistical technique that analyzes relationships between terms and concepts in a body of text, aiming to capture the underlying structure of meaning. It can be used for both extractive and abstractive summarization.
- **Gensim's Summarization Module:** Gensim [2] is a Python library for topic modeling and document similarity analysis. Its summarization module offers an extractive summarization approach based on text ranking and sentence scoring.
- **Sequence-to-Sequence (Seq2Seq) Models:** Originally developed for machine translation, Seq2Seq models [3], particularly using recurrent neural networks (RNNs), have been adapted for abstractive text summarization by generating summaries based on encoder-decoder architectures.
- **SummaRuNNer:** SummaRuNNer[4] is a neural network-based model that incorporates a hierarchical architecture to capture both local and global dependencies in the text, aiming for improved extractive summarization performance. These are just a few examples, and the field of text summarization continues to evolve with the development of new models and techniques.

5 Methodology

To perform abstractive text summarisation the following methodology was implemented:

5.1 Input:

The input is taken in the form of a paragraph on which abstractive text summarisation is to be performed.

5.2 Text cleaning:

Text cleaning helps in text summarization by removing irrelevant or noisy elements from the input text, ensuring that only essential information is retained. Some essential components of text cleaning include:

- **Removing Special Characters and Punctuation:** This involves eliminating non-alphanumeric characters, punctuation marks, and symbols that do not contribute to the semantic meaning of the text.
- **Lowercasing:** Converting all letters to lowercase ensures consistency in the text and helps in standardizing the vocabulary.
- **Handling Contractions and Abbreviations:** Replacing contractions (e.g., "can't" to "cannot") and expanding abbreviations (e.g., "Dr." to "Doctor") helps in improving text readability and comprehension.
- **Removing Stopwords:** Stopwords are common words (e.g., "the," "is," "and") that occur frequently but carry little semantic meaning. Removing stopwords reduces noise in the text.

5.3 Tokenisation:

Tokenisation involves splitting the text into individual words or tokens. This step is essential for further analysis, such as counting word frequencies or creating word embeddings. This process facilitates subsequent analysis and processing of the text. Here's how tokenization helps:

5.4 Encoder:

For encoding the tokens into vectors, we have used Word2Vec algorithm. **Word2Vec** [5] is an algorithm that learns distributed representations (embeddings) of words by capturing semantic and syntactic similarities from large text corpora. It offers two main architectures: Continuous Bag of Words (CBOW) and Skip-gram. CBOW predicts a target word based on its context, while Skip-gram predicts context words given a target word. Both architectures utilize shallow neural networks to train word embeddings, adjusting them to minimize a loss function. Once trained, these embeddings encode semantic relationships between words, enabling their use in various natural language processing tasks such as sentiment analysis and machine translation. Word2Vec's effectiveness, simplicity, and efficiency have made it a cornerstone of modern NLP systems.

5.5 Function:

After cleaning our paragraphs and mapping them to a vector space, we are assuming the existence of a function f that maps these vectors to shorter-length vectors, which represent the summaries of the original paragraphs. This function f essentially condenses the information contained in the paragraphs into more concise representations.
$y = f(x)$
y : the vector representing the summary of the original paragraph
x : the vector representing the original paragraph
Assumptions: We assume that the function f is continuous and bounded on a compact set K. However, in practice, finding such a function can be challenging or even impossible. Therefore, we resort to approximating it. One powerful tool for approximation is the **Universal Approximation Theorem** introduced in "Approximation by Superpositions of a Sigmoidal Function" by G. Cybenko (1989) [6], which states that a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy, given a sufficiently large number of neurons. By leveraging this theorem, we can construct neural networks capable of approximating complex functions, allowing us to approximate f effectively in our context.
Universal approximation theorem — Let $C(X, \mathbb{R}^m)$ denote the set of

continuous functions from a subset X of a Euclidean $\mathbb{R}^n$ space to a Euclidean space $\mathbb{R}^m$. Let $\sigma \in C(\mathbb{R}, \mathbb{R})$. Note that $(\sigma \circ x)_i = \sigma(x_i)$, so $\sigma \circ x$ denotes σ applied to each component of x. Then σ is not polynomial if and only if for every $n \in \mathbb{N}$, $m \in \mathbb{N}$, compact $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m)$, $\varepsilon > 0$ there exist $k \in \mathbb{N}$, $A \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$, $C \in \mathbb{R}^{m \times k}$ such that

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon$$

where $g(x) = C \cdot (\sigma \circ (A \cdot x + b))$.
Since we are dealing with sequences of words, we want to intoduce Seq2Seq model which are RNNs[7].

Problem with Recurrent Neural Network: The problem of exploding and vanishing gradients in recurrent neural networks (RNNs) [8] arises due to the nature of backpropagation through time (BPTT), where gradients are propagated backward through sequential time steps during training. When the gradient magnitudes become too large (exploding gradients), it can lead to unstable training, causing the model parameters to update excessively and diverge. Conversely, when the gradients vanish, becoming extremely small, the model fails to capture long-term dependencies effectively, hindering learning (vanishing gradients). This issue is particularly problematic in RNNs with many time steps or deep architectures, where gradients are multiplied over multiple layers or time steps. One of the seminal papers addressing this problem is "LONG SHORT-TERM MEMORY" by [9] Hochreiter et . al (1997), which introduces the Long Short-Term Memory (LSTM) architecture. LSTM mitigates the vanishing gradient problem by incorporating gating mechanisms that selectively retain or forget information over time, allowing for more effective learning of long-range dependencies in sequential data.

5.6 LSTM:

In 1997, a groundbreaking paper "LONG SHORT-TERM MEMORY" by [9] Hochreiter et . al (1997) introduced a novel architecture aimed at addressing the issue of exploding and vanishing gradients in recurrent neural networks (RNNs) when applied to sequence-to-sequence (Seq2Seq) modeling tasks. This architecture, known as the Long Short-Term Memory (LSTM) network, introduced specialized memory cells equipped with gating mechanisms to regulate the flow of information over time.

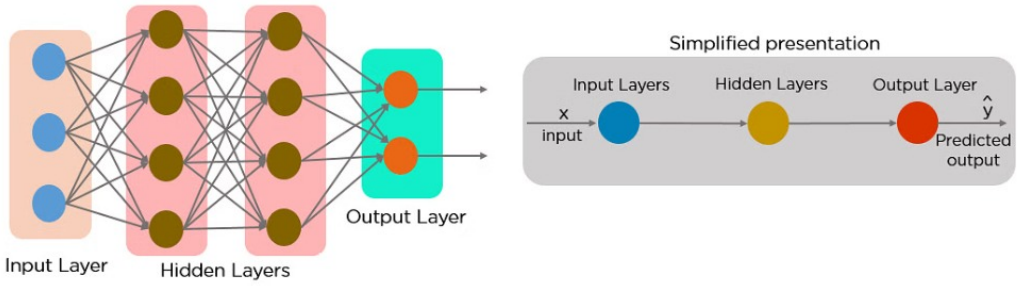


Figure 1: Neural Network

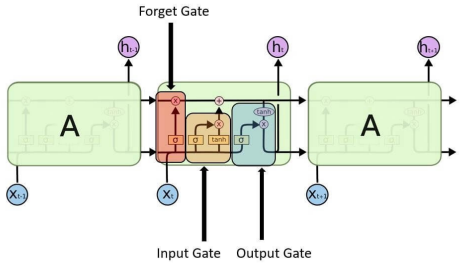


Figure 2: LSTM Architecture

By selectively retaining or forgetting information, LSTMs effectively mitigated the vanishing gradient problem by maintaining long-term dependencies in sequential data while preventing the explosion of gradients during training. This breakthrough enabled more effective modeling of sequential data, revolutionizing various natural language processing tasks such as machine translation, speech recognition, and text generation.

5.7 Decoder:

The decoder takes the input sequence from the encoder and transforms it into a fixed-dimensional vector representation using a series of LSTM layers in the encoder part. This vector serves as the initial state for the decoder LSTM layer. The decoder LSTM layer then takes the target sequence (decoder inputs) and the initial state from the encoder, processing the input sequence step by step while simultaneously updating its internal state. The output

from each time step is passed through a dense layer with a softmax activation function, producing a probability distribution over the vocabulary for each word in the output sequence. This probability distribution indicates the likelihood of each word being the next word in the sequence. Overall, the decoder generates a sequence of words by iteratively predicting the next word given the context encoded by the encoder and the previously generated words.

5.8 Model Architecture

Our model architecture represents a sequence-to-sequence (seq2seq) model with an encoder-decoder architecture, which is commonly used for tasks like machine translation, text summarization, and question answering. The model takes as input a sequence of words (`encoder_inputs`), representing a source language sentence or a textual input, and produces a sequence of words (`decoder_outputs`), representing a target language translation or a summary.

In the encoder part of the model, the input sequence is first passed through an embedding layer (`enc_emb`), which converts each word index into a dense vector representation. This embedding layer is trainable, meaning the word vectors are learned during the training process. The embedded sequence then goes through multiple layers of LSTM (Long Short-Term Memory) cells (`encoder_lstm1`, `encoder_lstm2`, `encoder_lstm3`). LSTM cells are a type of recurrent neural network (RNN) cell that can capture sequential information effectively. These LSTM layers process the input sequence, progressively extracting and encoding its features into a fixed-size context vector, represented by the states (`state_h`, `state_c`) of the final LSTM layer.

In the decoder part of the model, another input sequence (`decoder_inputs`) is similarly passed through an embedding layer (`dec_emb`) to obtain dense vector representations. These embeddings are then fed into another LSTM layer (`decoder_lstm`), which is initialized with the final states of the encoder LSTM layers. This enables the decoder to use the context vector obtained from the encoder to guide its generation of the output sequence. Finally, a dense layer (`decoder_dense`) with a softmax activation function is applied to produce the probability distribution over the vocabulary for each word in the output sequence. The model is trained to minimize the cross-entropy loss between the predicted

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 80)	0	-
embedding (Embedding)	(None, 80, 500)	25,886,500	input_layer[0][0]
lstm (LSTM)	[(None, 80, 500), (None, 500), (None, 500)]	2,002,000	embedding[0][0]
input_layer_1 (InputLayer)	(None, None)	0	-
lstm_1 (LSTM)	[(None, 80, 500), (None, 500), (None, 500)]	2,002,000	lstm[0][0]
embedding_1 (Embedding)	(None, None, 500)	7,073,000	input_layer_1[0][0]
lstm_2 (LSTM)	[(None, 80, 500), (None, 500), (None, 500)]	2,002,000	lstm_1[0][0]
lstm_3 (LSTM)	[(None, None, 500), (None, 500), (None, 500)]	2,002,000	embedding_1[0][0], lstm_2[0][1], lstm_2[0][2]
time_distributed (TimeDistributed)	(None, None, 14146)	7,087,146	lstm_3[0][0]

Total params: 48,054,646 (183.31 MB)

Trainable params: 48,054,646 (183.31 MB)

Non-trainable params: 0 (0.00 B)

Figure 3: Model Architecture

and actual sequences using the backpropagation algorithm.

5.9 Training the model:

For fitting the model, we have use Root Mean Squared Propagation (RMSprop) to minimize the Sparse Categorical Cross Entropy Loss function.

RMSprop[10] is an optimization algorithm used for training neural networks, particularly effective in handling problems with varying gradients. It calculates an exponentially decaying average of past squared gradients at each iteration, adjusting the learning rate for each parameter based on the ratio of the current gradient to the square root of the accumulated gradients. This adaptiveness enables RMSprop to effectively deal with sparse or noisy gradients encountered during training, leading to faster convergence and improved stability. By individually adjusting

learning rates for each parameter, RMSprop helps ensure that frequently occurring features receive smaller updates while less frequent features receive larger updates, resulting in efficient parameter updates and enhanced model performance.

Sparse Categorical Cross Entropy: Sparse Categorical Cross Entropy Loss function is commonly used for multi-class classification tasks where the target labels are provided as integers representing the class index. It calculates the cross-entropy loss between the predicted probability distribution and the true probability distribution of the classes. The mathematical formula for Sparse Categorical Cross Entropy Loss function is:

$$L(y_{\text{true}}, y_{\text{pred}}) = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{y_{\text{pred}}[i, y_{\text{true}}[i]]}}{\sum_{j=1}^{C} e^{y_{\text{pred}}[i,j]}}\right)$$

where,

N is the number of samples in the batch,

C is the number of classes,

y_{true} is the true class labels (represented as integers), and

y_{pred} is the predicted probability distribution over the classes. The formula computes the negative log likelihood of the true class labels, penalizing the model more when it makes predictions with low confidence on the correct class and less when it's more confident.

6 Model metrics and Evaluation

6.1 Understanding the distribution of the sequences

Here, our aim is to analyze the length of both the reviews and summaries to gain insight into the distribution of text lengths. This preliminary examination will guide us in determining the optimal maximum sequence length for our data.

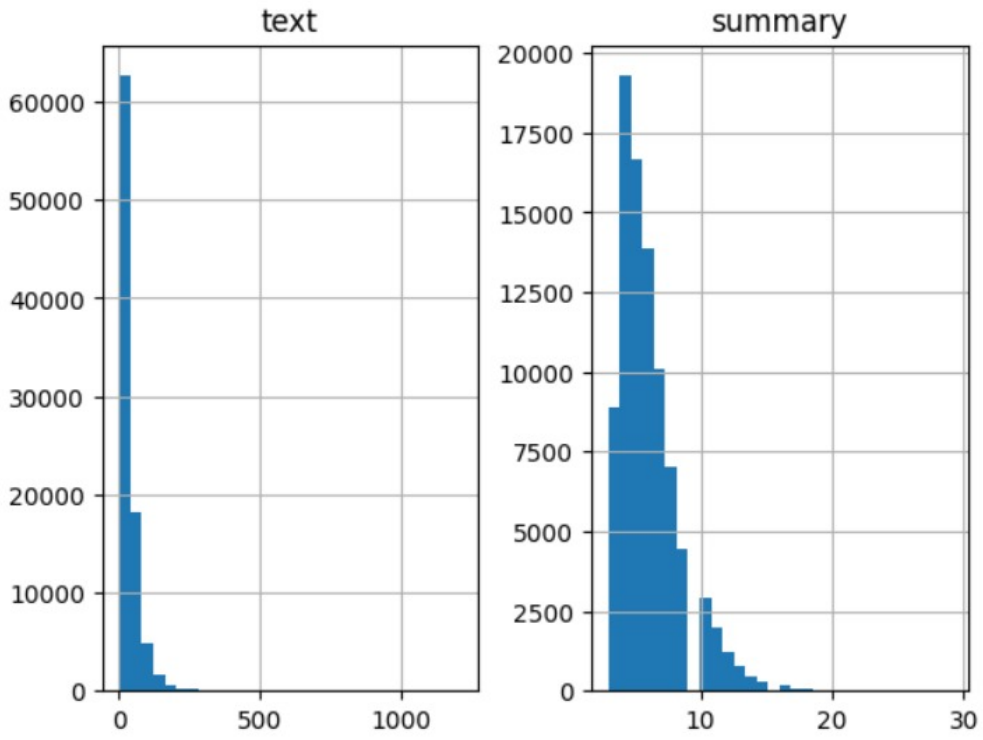


Figure 4: Distribution of Summary and Text Length

6.2 Some Experimentation

We gave our model some paragraphs to see its response from the testing dataset.

```
Review: dog loves lbs regular size perfect seems help teeth
Original summary: great
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 56ms/step
Predicted summary:  my dog loves these


Review: dogs love treats chews keep dogs teeth cleaned love know get home work every night get treat excited highly recommend plus much cheaper amazon com local petsmart petco
Original summary: my dogs love these
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
Predicted summary:  my dog loves these


Review: suffering post nasal drip congestion many years tea made big difference couple days drink times per day feel much better breathe easier little post nasal drip throughout day
Original summary: this tea works feel so much better
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
Predicted summary:  great


Review: got samples green mountain really liked fact pretty much instantly downside drink really feel kick lot caffeine great like taste coffee coffee sake
Original summary: smooth taste but not very
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
Predicted summary:  great for the best
```

7 Future Scope

Learning does not stop here! There's a lot more we can do to play around and experiment with the model:

- Try Attention Mechanism , introduced by Ashish Vaswani et . al (2017)[11]
- We could increase the training dataset size and build the model. The generalization capability of a deep learning model enhances with an increase in the training dataset size, however we would need higher computational power.
- Instead of using traditional LSTM layers , we could use Bi-Directional LSTM which is capable of capturing the context from both the directions and results in a better context vector.

8 Conclusion

In conclusion, text summarization is a vital task in natural language processing, aiming to condense large amounts of text into concise summaries while preserving key information. This project focuses on abstractive text summarization, utilizing techniques such as text cleaning, tokenization, word embeddings with Word2Vec, and the Universal Approximation Theorem to approximate the summarization function. The LSTM architecture, introduced by Hochreiter et al. (1997), addresses the challenges of exploding and vanishing gradients in recurrent neural networks, making it suitable for sequence-to-sequence modeling tasks like text summarization. The model architecture involves an encoder-decoder framework, with the encoder extracting features from the input sequence and the decoder generating the summary based on the encoded context. Future work could involve incorporating attention mechanisms, increasing the training dataset size, or exploring alternative architectures like Bi-Directional LSTM to further improve the model's performance and generalization capabilities. Overall, this project lays the groundwork for leveraging deep learning techniques to automate the text summarization process, facilitating faster comprehension and decision-making across various domains.

References

[1] TextRank
https://aclanthology.org/W04-3252.pdf

[2] Text summarization with Gensim
https://rare-technologies.com/text-summarization-with-gensim/

[3] Seq2Seq Model architecture
https://arxiv.org/pdf/1409.3215.pdf

[4] SummaRuNNer
https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=
https://ojs.aaai.org/index.php/AAAI/article/view/10959/10818&ved=
2ahUKEwiX1aXi7qyFAxVQoGMGHUYwDfIQFnoECCAQAQ&usg=
AOvVaw2NomyeCYh0lfdRash6Egcp

[5] Word2Vec
https://arxiv.org/pdf/1301.3781.pdf

[6] Universal Approximation Theorem
https://web.njit.edu/~usman/courses/cs675_fall18/10.1.1.441.7873.pdf

[7] RNNs
https://dokumen.pub/
beyond-regression-new-tools-for-prediction-andanalysis-in-the-behavioral-sciences.
html

[8] Problems in RNN
https://arxiv.org/pdf/1211.5063.pdf

[9] Long Short Term Memory
https://www.bioinf.jku.at/publications/older/2604.pdf

[10] RMSProp
https:
//www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[11] Attention is all you need
https://arxiv.org/pdf/1706.03762.pdf