Index

Introduction

In this project, I am going to use CNN daily mail datasets to build a text summarization model. Text summarization task means generating a short organized summary from a long article, such as news, through extracting and abstracting.

This will be a research project in which I investigate recent trends and deepen my knowledge of Deep Learning, which goes beyond the contents of CM3015, and a text summarization model is eventually built.

In addition, this project supposes an organization or a group is the user, such as a (web) system development company with little machine-learning knowledge. Therefore, the deliverable report must be scientific and convincing to explain why each parameter is adopted and be knowledgeable for its developers. The deliverable code and model must run easily on common devices, such as a web server. Mobile devices, on the other hand, might not be able to run the large model because they have a smaller amount of memory.

What is interesting and motivative (Rapid Evolution)

In recent years, various types of text representations and neural network models have been developed, especially since the Transformer model was invented.

Text representation

In addition to the One-hot-encoding introduced in CM3015, I am going to use other techniques, such as word2vec and GloVe, that can appropriately handle the meanings of each word.

The neural network is a vector transformation. It is worth investigating how well each representation of embedding layers can capture the characteristics of text data.

Transformer

The transformer is one of the most common and game-changing models for sequential data, such as text. In addition, it is also used in text-to-image generative artificial intelligence (AI), large language models (LLM), and so on. There are a lot of derived models based on the Transformer architecture, such as BERT, which outperforms others in natural language processing. [10] Whereas another model based on the Transformer, named Vision Transformer, has been used for image recognition tasks to overcome CNN's weaknesses recently. [9] Moreover, there is a more efficient model based on the Transformer model, which is called the Retentive Network. Because all these architectures are based on the Transformer model, deepening its knowledge leads to the understanding of whole machine learning. Additionally, even in the future, a number of new architectures based on the Transformer models will be released.

What is interesting and motivative (from the aspect of efficiency and ecology)

Measuring the performances of various models is one of the exciting and essential points.

Advanced models do not always achieve the best performance. For example, when the Recurrent Neural Network model, which is one of the advanced models, is used for a text classification task, even though it does not always work better than a dense model, it mostly consumes greater computing resources. [11] That is, it completely wastes resources. I am going to analyze the strengths and weaknesses of each model in terms of practical uses.

As recent investigations indicate, building LLM models indirectly emits a significant amount of CO2. That is, the investigation of efficiency that leads to the ecology is socially meaningful, even in the environmental aspect. However, since there are not enough computing resources to build an LLM model, this project does not target building an LLM itself.

Literature Review

I aim to explore the current understanding of machine learning model architectures and their performances, focusing on the key technology "Transformer". Moreover, in this project, I investigate the effectiveness of the Transformer and finally build a text

summarization model.

Transformer

Vaswani et al. (2017) proposed a new simple architecture, named Transformer, based on attention mechanisms without recurrence and convolutions. [1]

The architecture of the Transformer, which consists of both encoders and decoders, is introduced with a simple figure.

The paper indicates the algorithms and architecture of the Transformer with formulas, comparing the former architectures, such as the recurrent neural network model and the gate recurrent unit model. How to reduce the amount of calculation for longer input sequences, which is one of the defects, is also introduced kindly. However, because of the nature of the paper, the concrete implementation is not shown on it. Instead, it provides a link to a GitHub repository, "tensor2tensor," [2] where some implementations with Python code are introduced. However, since the repository has not been updated by the owner, it is pretty uncertain whether they can run on the current Python environment or not.

In addition, because the Transformer was initially developed for the machine translation task, various tasks except translation are not mentioned in this paper, written in 2017. Thereby, the effectiveness of the Transformer is not investigated for a wide range of NLP tasks here.

As mentioned at the beginning, hyperparameter tunings are obviously required to investigate effectiveness. The section "6.2 Model Variations" describes the parameters such as the number of heads, the parameters of the optimizer, and so on that exist in the Transformer model, and the same indicators will be used to compare the performance in this project. And the experiment environment, which the paper disclosed, is useful to the experiments in this project.

I have obtained the following beneficial knowledge, even though it is not directly related to my project.

- The number of heads is neither too many nor too few
- Interpretable models

The paper does not include the following items, which are required for this project.

- Sufficient benchmarks for various NLP tasks/applications
- Minimum viable and runnable code in the modern environment

RetNet

Yutao et al. (2023) proposed the Retentive Network (RetNet), which is a strong

successor to Transformer for large language models.[3]

One of the drawbacks of the Transformer, which is introduced in this paper, is the slow inference and inefficiency due to multi-head attention. The performance comparison between the RetNet and the Transformer has been executed for large language models and represents the improvement. They are executed and compared for large language models and that difference might not be remarkable on smaller models such as text summarization tasks.

The section "3 Experiments" mentions the following:

> Perplexity decreases along with scaling up the model size. We empirically observe that RetNet tends to outperform Transformer when the model size is larger than 2B. [3]

When the model size is larger than 2.7B, with more than 2.7 billion parameters, the advantages of the RetNet model might finally be detected.

The section "3.3 Training Cost" shows the specific environment where the evaluation of this paper was executed as follows.

> We evaluate the results with eight Nvidia A100-80GB GPUs, because FlashAttention is highly optimized for A100. Tensor parallelism is enabled for 6.7B and 13B models. [3]

Building a text summarization model in this project is mainly executed on the local macOS machine, which has an M2 CPU, which has a significantly different architecture from Nvidia GPUs.

> Moreover, without relying on specific kernels, it is easy to train RetNet on other platforms efficiently. For example, we train the RetNet models on an AMD MI200 cluster with decent throughput. It is notable that RetNet has the potential to further reduce cost via advanced implementation, such as kernel fusion. [3]

Even though the above quote mentioned the performance on the other device, the environment is highly parallelized, and it is uncertain if this project can derive beneficial knowledge through experimental comparisons.

Though this does not matter directly to this project, as the sections "2.3 Overall Architecture of Retention Networks" and "3.4 Inference Cost" mention, the inference cost of the RetNet model is remarkably $O(1)$. [3] In the case where the quick response for longer sequences/tokens is required, the RetNet model can be one of the realistic choices.

Neural Text Summarization: A Critical Evaluation

Wojciech et al. (2019) introduced and evaluated the current shortcomings of text summarization.[4] The methodology of the evaluation should be notable in this paper. The previously introduced 2 papers are not for text summarization tasks, and the BLEU metric is used, which is for the evaluation of translation task performance. This is not for summarization tasks. Though loss values probably evaluate the performances, they cannot clearly consider sentences' structures for evaluations. Therefore, following this paper, the ROUGE metric is used in this project. Fortunately, KerasNLP has the following 2 metrics as standard.

- ROUGE-N
- ROUGE-L

Note that the ROUGE-N metric uses n-gram to refer to the overlap, and the ROUGE-L metric uses the longest common subsequence that can detect the sentence level structure. [8]

Though the following does not matter directly, some shortcomings of the current summarization model are introduced here.

- Even though a summarization task depends on the reader's expectations and prior knowledge, the current models are not provided as additional information.
- Even though the ROUGE metric is widely used, it does not factually and consistently examine summarized text, but just lexically.
- The bias and diversity of the dataset.

Text Summarization with Pretrained Encoders

Yang et al. (2019) indicated the effectiveness of the BERT, which is an abbreviation of Bidirectional Encoder Representations from Transformers, for text summarization, which is one of the NLP tasks. [5]

In this paper, the Bidirectional Encoder Representations from Transformers(BERT) model, where a pre-trained model and a scratch model are combined, is compared with other typical models for text summarization tasks, and it is concluded that the BERT model outperformed others. However, it seems slightly unclear whether the pre-trained model actually works better because there is not a comparison of the same architecture model between pre-trained and scratch.

As the section "Introduction" mentions, the BERT model, one of the Transformer models, has affected word and sentence representation. Thus, this model may become a research target in this project.

> When fine-tuning for a specific task, unlike ELMo whose parameters are usually fixed, parameters in BERT are jointly fine-tuned with additional task- specific parameters. [5]

Whereas, as the above is written, since the pretrained BERT model goes through the

fine-tuning phase, it might take longer time for training than fixed parameters' models such as word2vec and GloVe.

The table of the section "3.3 Abstractive Summarization" holds the various types of datasets. Though this might not matter to the paper, apart from CNN DailyMail, NYT, and XSum are used as the datasets. In this project, the CNN dailymail will be utilized as a dataset. However, they are kept as sub plans, preparing for unexpected situations.

> Each token w_i is assigned three kinds of embeddings: token embeddings indicate the meaning of each token, segmentation embeddings are used to discriminate between two sentences (e.g., during a sentence-pair classification task) and position embeddings indicate the position of each token within the text sequence. [5]

If the BERT model is used, differences in embeddings must be addressed.

facebook/bart-large-cnn

This is a text summarization model developed by Facebook that is available on Hugging Face. [30]

Particularly noteworthy is the model size and its performance. The model size, the number of parameters, is 406M, and the results of ROUGE-1, ROUGE-2, and ROUGE-L are 42.949, 20.815, and 30.619 respectively.

In addition, pre-trained BART is used in this text summarization model.

> BART is a transformer encoder-encoder (seq2seq) model with a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder. BART is pre-trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text. [30]

> BART is particularly effective when fine-tuned for text generation (e.g. summarization, translation) but also works well for comprehension tasks (e.g. text classification, question answering). This particular checkpoint has been fine-tuned on CNN Daily Mail, a large collection of text-summary pairs. [30]

The concrete specifications are as follows. [30]

Parameter	Value
Self-attention heads	16
Encoder layers	12
Feed forward network dim	4096

Dropout	0.1
Max length	142
Max position emneddings	1024
Min length	56

google/pegasus-cnn_dailymail

This is another text summarization model developed by Google that is available on Hugging Face. [31]

This model has been trained not only with the CNN dailymail dataset but also with many others. It has 568M parameters and consists of 12 Transformer encoder layers and 12 Transformer decoder layers. The sinusoidal method is used for the positional embedding. The number of vocabulary is 50265.

In addition, the paper "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization" shows that the results of ROUGE-1, ROUGE-2, and ROUGE-L are 44.16, 21.56, and 41.30 respectively. [32]

The concrete specifications of the Pegasus base model are as follows. [32]

Parameter	Value
Self-attention heads	12
Encoder layers	12
Feed forward network dim	3072

Conclusion

In conclusion, the superiority and effectiveness of the Transformer and its self-attention have been shown, compared with the other architectures such as the recurrent neural network and convolutional neural network. Whereas performances, tunings, other NLP tasks, and architectures, which determine if encode or/and decode is used, have not been shown. And that is what I aim to do.

I have slightly changed my mind while reading papers. Specifically, instead of regarding the RetNet model as the main topic, hyperparameter tunings of the Transformer model are going to be mainly researched. The RetNet model will be researched optionally. The reason is as follows.

- The RetNet paper shows that it is effective only for large language models.
- The experiments on the ResNet paper were executed on the highly parallelized environment, which is different from the local machine that is used in this project.

- The training to build a text summarization model takes a considerable amount of time.

Therefore, because a beneficial report may not be provided if most resources are invested in experiments with little chance of seeing differences, I am going to make the experiment about the RetNet model optional. Instead, clearly beneficial hyperparameter tunings are executed in the first half to ensure users' benefits.

Moreover, the comparison between the pre-trained models and non-pre-trained models will also be one of the interesting topics.

Finally, it might be very difficult to self-build a text summarization model and mark a better score of ROUGE metrics on the current local machine because the models of Google and Facebook are trained with a number of layers and parameters using high computational GPUs. However, it might be possible to build a smaller model that is beneficial for a specific environment. Since KerasNLP provides an easy way to use Bidirectional Autoregressive Transformer (BART), the performance might be able to get close to the Google or Facebook results if it is used. [33]

Design

Project Overview

I have chosen the template "Deep Learning on a public dataset" of CM3015 "Machine Learning and Neural Networks". In this project, I am going to build a text summarization model with the Cable News Network(CNN) dailymail datasets.

Unlike other templates of other courses, the deliverables are the machine learning model and its documents, and this project does not have user interfaces for end users. However, I assume to deliver the machine learning model to an organization that has a plan where the text summarization model is integrated into a system, which has actual end users.

There is also the aspect of a research project where I investigate recent trends and deepen my knowledge of Deep Learning that was not sufficiently covered in the CM3015 course. Day by day and month by month, the field and market of deep learning technology for natural language processing is rapidly growing. The deliverable document is going to be beneficial for developers.

Domain and Users

As I mentioned above, unlike other project templates, the project template on CM3015 does not have a specific user. However, since it is difficult to determine what

is beneficial and how beneficial a feature is without specific users, I assume that users are developers outside the organization who want to install a machine learning feature and model into a currently running system. Specifically, that is as follows.

They are not familiar with deep learning but with server-side applications of Python, and they want to adopt a text summarization feature into a current running system. My job is solving the problem and providing text summarization models and their knowledgeable and beneficial documents to them.

Justify features based on domain and users

Generally, end users do not mind if a model is highly advanced or not. Instead, they mind the time and performance of response.

Why is the Transformer model used?

In this project, text summarization models are built. That is, the neural network learns sequences as the input to output another sequences as the output. The Transformer model is superior to the Dense model to process sequences data. That is the reason why the Transformer model is used for the text summarization model.

On text classification tasks, well-tuned dense layer models overcome advanced models, such as a Transformer model, an Recurrent Neural Network(RNN) model and so on, under certain conditions.

However, text generation tasks, such as text summarization, could not achieve satisfactory outcomes for a long time. Encoder-decoder models, text embeddings, and Transformer, which can handle sequences well have made them possible. That is why the Transformer model is used.

Why is hyperparameter tuning necessary?

There are a number of hypterparameters in a neural network model, and there does not exist a formula that can derive optimized their values. They depend on each dataset and architecture. The best parameter set must be found with hyperparameter tunings.

Prior to testing advanced models, the hyperparameter tunings should be executed sufficiently. Even if an advanced model such as RetNet outperforms a traditional model, it is impossible to compare the advanced model and the traditional model fairly without hyperparameter tunings.

Fairly comparisons derive better models that are beneficial for users. That is the reason why hypter parameter tunings are necessary.

Why is the RetNet model reseached optionally?

Optionally, the other cutting-edge models, such as the RetNet model, are examined. Generally, end users do not mind if a model is highly advanced model or not. Instead, they mind the time and performance of response. Whereas, it might be able to efficiently use resources, and that might lead to the reduction of the cost for users. This is one of the reason why the research of the RetNet model is made optional.

Structure of the project

The main deliverable is a model for the text summarization task and its Jupiter notebook document. The model is provided with minimum viable code to run it. Multiple models are possibly provided because the difference in the size of each model affects the machine's requirements where it runs. The notebook's headings will be as follows.

- Abstract
- Introduction
- Background
  - Mathematic explanation
  - Scientific description
  - Technical description
- Experiments/Methodology
- Conclusion
  - Best model
  - Verification of hypotheses
  - Future work
- Citation/Reference

Key technologies and methods

I introduce libraries and technologies here. To be correct and objective, I explain them in my own words as little as possible, citing public documents, websites, and their Wikipedia page. Instead, I explain how each technology and library is used in my own words for the project.

TensorFlow

This is a machine learning library, which is mainly used in this project.

> An end-to-end open source machine learning platform for everyone. Discover TensorFlow's flexible ecosystem of tools, libraries and community resources. [12]

> TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural

> networks. [13]

In this project, the TensorFlow library is used not only for building text summarization models but also for most other tasks from the beginning to the end.

Keras

This is an OSS neural network library. It depends on the library version, it can switch Tensorflow to other machine learning library.

> Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. Keras also gives the highest priority to crafting great documentation and developer guides. [14]

> Keras is an open-source library that provides a Python interface for artificial neural networks. Keras was first independent software, then integrated into TensorFlow library, and later supporting more. [15]

This library is used throughout the whole project to manipulate the TensorFlow library.

KerasNLP

> KerasNLP is a natural language processing library that works natively with TensorFlow, JAX, or PyTorch. Built on Keras 3, these models, layers, metrics, and tokenizers can be trained and serialized in any framework and re-used in another without costly migrations. [16]

In this project, the Transformer model, which consists of the Transformer encoder layer and the Transformer decoder layer, is used to build text summarization models. It is possible to self-implement them by following the paper and books. However, it might unintentionally include bugs. Then, the model and its documents lose the reliability for readers. Therefore, the KerasNLP, which is a part of the widely known library "Keras" and contains the Transformer layer classes, is utilized here to secure the reliability.

Moreover, a lot of hyperparameter tunings are executed in this project. Their classes in KerasNLP is well-architected so that hyperparameters are easily injected to each layer. This is another reason why the KerasNLP is used here.

PyTorch

This is another OSS machine learning library. PyTorch is introduced on Wikipedia as follows.

> PyTorch is a machine learning library based on the Torch library, used

> for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation umbrella. It is recognized as one of the two most popular machine learning libraries alongside TensorFlow, offering free and open-source software released under the modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface. [17]

It is not directly used for the project development. However, some text summarization models that I have introduced has been developed with this PyTorch library.

Transformers

This is a software library of Hugging Face, Inc. The official repository of GitHub introduces as follows.

> Transformers provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio. [18]

> Transformers provides APIs to quickly download and use those pretrained models on a given text, fine-tune them on your own datasets and then share them with the community on our model hub. At the same time, each python module defining an architecture is fully standalone and can be modified to enable quick research experiments. [18]

> Transformers is backed by the three most popular deep learning libraries — Jax, PyTorch and TensorFlow — with a seamless integration between them. It's straightforward to train your models with one before loading them for inference with the other. [18]

I have investigated the following 2 text summarization models as a part of competitive research.

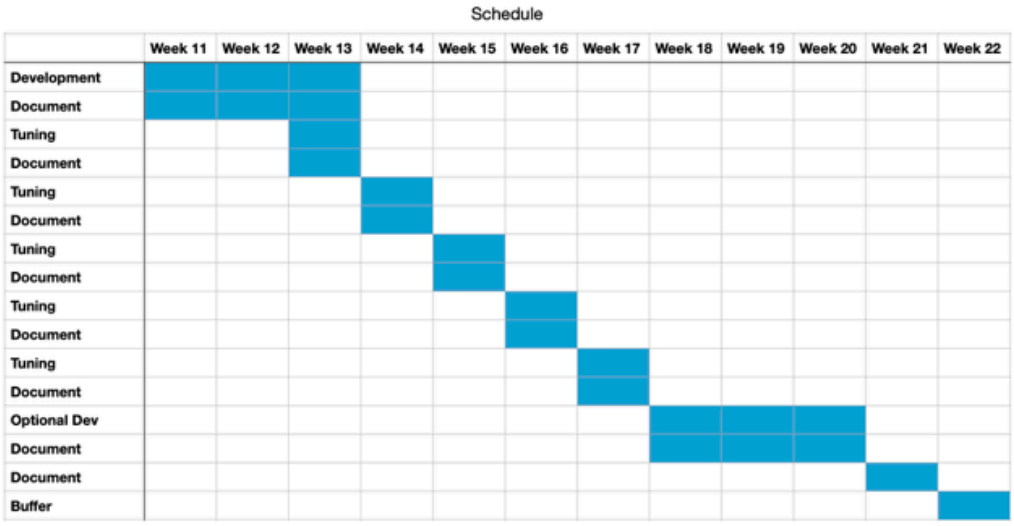
- facebook/bart-large-cnn [19]
- google/pegasus-cnn_dailymail [20]

They are distributed through huggingface and the Transformers library is necessary to use these models.

Work Plan (Gantt Chart)

The first 2 to 3 weeks are used to develop the basic Transformer model. Its part of the document is written at the same time. After that, hyperparameter tunings are repeatedly executed with small cycles so that they are efficient. The uncertainty of this phase is the length of training time. The optional implementation for the RetNet model keeps aside the time of 3 weeks. At last, there is a time for the documentation

and the buffer.



Evaluation Plan

Normal testing, which is for machine learning models, and hypothesis verification are executed to evaluate the project. However, the human tester is not used to check performance because it is difficult to organize the tester team for this project free of charge.

Testing

For the test dataset, the following metrics are used to measure and test model performance. This is executed to numerically prove how better a generated model is.

- loss
- ROUGE-N
- ROUGE-L

Verification of hypotheses

The following hypotheses are verified.

- Hyperparameter tunings are significantly effective even on a Transformer model.
- Transformer models can generate text more sophisticatedly than dense layer models.
- The encoder-decoder model of the Transformer works better for text summarization tasks than the only-decoder model.

- Pre-trained models improve the text summarization performance.

The following hypothesis is verified as a result of an optional implementation, if possible.

- The RetNet architecture for a small model does not make a difference.

Whereas, it is seemingly difficult to verify the following hypotheses in this project, even though they are interesting topics. The first one is a thought during the prototype implementation. A huge Transformer model is essential to verify the first hypothesis, and it is impossible for the current local environment to run. And no one has solved the second one as far as I have read a number of papers. That is, the task of text summarization is strongly related to human prior knowledge and understanding of the things that are summarized. Thus, I am going to exclude them from the verification of the project temporarily.

- An excessive number of units and layers for the number of dataset entries overfits.
- Prior knowledge and bias cannot be solved on the current architecture.

Feature Prototype

First, there are three types of Transformer models: Transformer encoder-only model, Transformer decoder-only model, and Transformer encoder-decoder model. Here are three types of different machine-learning tasks with the Transformer for each type.

1. The Transformer encoder-only model for the text classification task
2. The Transformer decoder-only model for the text generation task
3. The Transformer encoder-decoder model for the text summarization task

Generally, the encoder-decoder model of the Transformer is used for text summarization tasks. However, the decoder-only model might be able to be used. In any case, there is a necessity to verify whether each model and each API of libraries can actually work. This time, the KerasNLP library is used to add the encoder and decoder layers of the Transformer. The reasons why the KerasNLP library is utilized are as follows:

- Stability
- Concise of parameter experiments

The book "Deep Learning with Python, Second Edition" [7] introduces the simplified Transformer implementation that works in the local environment. However, it is not certain how widely it is used, and it might not pass any testing. In addition, even though it is good to understand how the Transformer works internally because it is simplified, it is difficult for us to experiment with various parameters.

In [ ]:
```
# Install KerasNLP
%pip install --upgrade keras-nlp rouge-score
```

Transformer text classification task

Firstly, a Transformer classification model is built so that the local environment determines if it can run or not.

In [ ]:
```
import platform

import keras
import keras_nlp
import tensorflow as tf
import tensorflow_datasets as tfds

# Hyperparameters
VOCAB_SIZE = 25000
BATCH_SIZE = 4096
NUM_HEADS = 4
INTERMEDIATE_DIM = 64
SEQ_LENGTH = 100
BUFFER_SIZE = 10000
```

In [ ]:
```
def load_ag_news_dataset(batch_size):
    """
    Load ag_news_subset dataset.
    :param batch_size: the number of batch size.
    :return: a dataset object.
    """
    dataset = tfds.load('ag_news_subset')
    train_dataset, test_dataset = dataset['train'], dataset['test']

    # Devide the dataset for training and validation in the ratio 8:2.
    partial_train_dataset = train_dataset.take(len(train_dataset) // 10 *
    val_dataset = train_dataset.skip(len(train_dataset) // 10 * 8)

    # For loading performance
    train_dataset = train_dataset.batch(batch_size).prefetch(tf.data.AUTO
    partial_train_dataset = partial_train_dataset.batch(batch_size).prefe
    val_dataset = val_dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE
    test_dataset = test_dataset.batch(batch_size).prefetch(tf.data.AUTOTU

    return train_dataset, partial_train_dataset, val_dataset, test_datase

def tuplize(x):
    """
    Transform a row from the dataset to learn.
    :param x: a single row of the dataset.
    :return: a tuple of the feature and the target.
    """
    return (
        x['title'] + ' ' + x['description'],# x: feature
        x['label']# y: target
    )

def build_model(
        vectorization_layer: keras.layers.TextVectorization,
        max_tokens=25000,
        embedding_dim=128,
        intermediate_dim=32,
        num_heads=4,
        sequence_length=50):
    """
    Build a Transformer encoder model for text classification task.
```

```
    :param vectorization_layer: the layer object where sentence is conver
    :param max_tokens: the number of token.
    :param embedding_dim: the number of dimension for embedding.
    :param intermediate_dim: the number of units.
    :param num_heads: the number of heads.
    :param sequence_length: the length of a sequence.
    :return: a sequential model.
    """
    if platform.system() == "Darwin" and platform.processor() == "arm":
        """
        Apple Silicon mac shows tht following warning.
        WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimiz
        please use the legacy Keras optimizer instead,
        located at `tf.keras.optimizers.legacy.Adam`
        Therefore, keras.optimizers.legacy.Adam is used.
        """
        optimizer = keras.optimizers.legacy.Adam()
    else:
        optimizer = keras.optimizers.Adam()
    inputs = keras.layers.Input(shape=(1,), dtype="string")
    x = vectorization_layer(inputs)
    x = keras_nlp.layers.TokenAndPositionEmbedding(
        vocabulary_size=max_tokens,
        sequence_length=sequence_length,
        embedding_dim=embedding_dim,
        mask_zero=True,
    )(x)
    x = keras_nlp.layers.TransformerEncoder(
        intermediate_dim=intermediate_dim,
        num_heads=num_heads
    )(inputs=x)
    x = keras.layers.GlobalMaxPooling1D()(x)
    x = keras.layers.Dropout(0.2)(x)
    outputs = keras.layers.Dense(4, activation="softmax")(x)
    model = keras.Model(
        inputs=inputs,
        outputs=outputs,
        name="transformer_text_classification_model"
    )
    model.compile(
        optimizer=optimizer,
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'],
    )
    return model
```

In [ ]:
```
train_dataset, partial_train_dataset, val_dataset, test_dataset = load_ag
    batch_size=BATCH_SIZE
)
vectorization_layer = keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    output_sequence_length=SEQ_LENGTH
)
vectorization_layer.adapt(train_dataset.map(lambda x: x['description'] +
model = build_model(
    vectorization_layer,
    max_tokens=VOCAB_SIZE,
    intermediate_dim=INTERMEDIATE_DIM,
    num_heads=NUM_HEADS,
    sequence_length=SEQ_LENGTH
)
model.summary()
history = model.fit(
    partial_train_dataset.map(tuplize, num_parallel_calls=tf.data.AUTOTUN
    validation_data=val_dataset.map(tuplize, num_parallel_calls=tf.data.A
    epochs=20,
    batch_size=BATCH_SIZE,
    verbose=1,
    callbacks =[
        keras.callbacks.EarlyStopping(monitor='val_loss', patience=5),
    ]
)
```

Transformer text generation task

Now, the following implementation, which is partially based on the book "Deep Learning with Python" [6], inspects how the Transformer decoder-only model works, and investigates how long a text generation task takes so that the time for the text summarization task is estimated. The points of difference from the above classification task are as follows:

- The number of units at the output layer.
- The number of layers and units at the hidden layers.
- The number of epochs.
- And so on.

In the generative task, such as text summarization, these numbers generally get larger than the text classification model. As a result, it takes a significant amount of time for training. In this case, it took over 8 hours. The number of parameters will be increased at least 15% and the training time will be 15% longer. That is one of the reasons why I have changed my plan, where the RetNet model is an optional topic. Even though the Retentive Network model will make the training 8.4 times faster in the aspect of the throughput, which is shown in the paper, the same Transformer model training must be executed for the comparison. [3] The experiments will still take a lot of time, and that is not realistic.

In [ ]:
```
import platform

import numpy as np

import tensorflow as tf
import keras
import keras_nlp

# Hyperparameters
BATCH_SIZE = 256
SEQUENCE_LENGTH = 50
MAX_TOKENS = 15000
EMBEDDING_DIM = 256
INTERMIDIATE_DIM = 2048
NUM_HEADS = 2
LEARNING_RATE = 2e-6 #  changed from 2e-5
```

In [ ]:
```
def prepare_dataset(text_batch):
    vectorized_sequences = text_vectorization(text_batch)
    x = vectorized_sequences[:, :-1]
    y = vectorized_sequences[:, 1:]
    return x, y

text_vectorization = keras.layers.TextVectorization(
    max_tokens=MAX_TOKENS,
    output_mode="int",
    output_sequence_length=SEQUENCE_LENGTH,
)

dataset = keras.utils.text_dataset_from_directory(
    directory="aclImdb",
    label_mode=None,
    batch_size=BATCH_SIZE
)
dataset = dataset.map(
    lambda x: tf.strings.regex_replace(x, "<br />", " ")
)

text_vectorization.adapt(dataset)
tokens_index = dict(enumerate(text_vectorization.get_vocabulary()))

dataset = dataset.map(
    prepare_dataset,
    num_parallel_calls=4
)
```

In [ ]:

```
if platform.system() == "Darwin" and platform.processor() == "arm":
    """
    Apple Silicon mac shows tht following warning.
    WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.
    please use the legacy Keras optimizer instead,
    located at `tf.keras.optimizers.legacy.Adam`
    Therefore, keras.optimizers.legacy.Adam is used.
    """
    optimizer = keras.optimizers.legacy.Adam(learning_rate=LEARNING_RATE)
else:
    optimizer = keras.optimizers.Adam(learning_rate=LEARNING_RATE)

# Build model
inputs = keras.Input(shape=(None,), dtype="int64")
x = keras_nlp.layers.TokenAndPositionEmbedding(
    vocabulary_size=MAX_TOKENS,
    sequence_length=SEQUENCE_LENGTH,
    embedding_dim=EMBEDDING_DIM,
    mask_zero=True,
)(inputs)
x = keras_nlp.layers.TransformerDecoder(
    intermediate_dim=INTERMIDIATE_DIM,
    num_heads=NUM_HEADS
)(x, x)
outputs = keras.layers.Dense(
    MAX_TOKENS,
    activation="softmax"
)(x)
model = keras.Model(
    inputs,
    outputs,
    name="transformer_text_generation_model",
)
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=optimizer,
)
model.summary()
```

In [ ]:

```
def sample_next(predictions, temperature=1.0):
    """
    Choose the next token, using the temperature.
    :param predictions:
    :param temperature:
    :return: the next token index
    """
    predictions = np.asarray(predictions).astype("float64")
    predictions = np.log(predictions) / temperature
    exp_preds = np.exp(predictions)
    predictions = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, predictions, 1)
    return np.argmax(probas)

"""
Generate a sentence with the latest model on every epoch.
"""
class TextGenerator(keras.callbacks.Callback):
    def __init__(
            self,
            prompt,
            generate_length,
            model_input_length,
            temperatures=(1.,),
            print_freq=1):
        self.prompt = prompt
        self.generate_length = generate_length
        self.model_input_length = model_input_length
        self.temperatures = temperatures
        self.print_freq = print_freq

    def on_epoch_end(self, epoch, logs=None):
        if (epoch + 1) % self.print_freq != 0:
            return
        print("\n")
        for temperature in self.temperatures:
            sentence = self.prompt
            for i in range(self.generate_length):
                tokenized_sentence = text_vectorization([sentence])
                predictions = self.model(tokenized_sentence)
                next_token = sample_next(
                    predictions=predictions[0, i, :],
                    temperature=temperature
                )
                sampled_token = tokens_index[next_token]
                sentence += " " + sampled_token
            print(f"Temperature {temperature}: {sentence}")

prompt = "This movie"
text_gen_callback = TextGenerator(
    prompt,
    generate_length=50,
    model_input_length=SEQUENCE_LENGTH,
    temperatures=(0.2, 0.5, 0.7, 1., 1.5)
)
```

In [ ]:
```
model.fit(
    dataset,
    # Generally, 100 to 200 is used as the epoch number for generative mo
    # However, because this is a prototype, the number is intentionally s
    # epochs=200,
    epochs=20,
    callbacks=[
        text_gen_callback,
    ]
)
```

Transformer text summarization task

Finally, to understand the outline, the minimum viable implementation of text summarization task is below. Because the dataset contains only 2 entries, this model overfits 100%. This time, the encoder and decoder model of the Transformer is adopted. However, this is not a requirement of the text summarization model. The decoder-only model of the Transformer might be able to build the text summarization model and achieve a good performance. This will be an experiment in this report.

Moreover, there are various types of text representation/vectorization as follows, including pre-trained models.

- Static embeddings
  - word2vec [22]
  - fastText (more advanced than word2vec) [23]
  - GloVe [24]
- Dynamic embeddings
  - BERT

This is an important experiment. Because, if there exists an ultimate representation to express words and text sequence, the neural network holds the ability to fit the hyper-dimensions. That is, finding better vector representation is essential to improve the performance.

Furthermore, there are many hyperparameters. Some of them are shown at the top of the later code.

Embedding layer

The default Embedding layer of the Keras library is used in this project. [21] This layer vectorizes words in a sequence, and similar words are vectorized closely. An accurate description is on the official page, as follows.

> Word embeddings give us a way to use an efficient, dense representation in which similar words have a similar encoding. Importantly, you do not have to specify this encoding by hand. An

> embedding is a dense vector of floating point values (the length of the vector is a parameter you specify). Instead of specifying the values for the embedding manually, they are trainable parameters (weights learned by the model during training, in the same way a model learns weights for a dense layer). It is common to see word embeddings that are 8-dimensional (for small datasets), up to 1024-dimensions when working with large datasets. A higher dimensional embedding can capture fine-grained relationships between words, but takes more data to learn. [25]

This layer has the following tunable hyperparameters:

- The number of vocabulary
- The number of embedding dimension
- Whether the value 0 is masked or not as padding

PositionEmbedding layer

The following sample code uses the PositionEmbedding layer of the KerasNLP library. [26] The KerasNLP library also has some positional embedding layers, such as the SinePositionEncoding layer that was originally used in the thesis. [27] [1] Because the tunable hyperparameters are few, this project examines various types of positional embeddings here.

- The number of sequence length

TransformerEncoder layer

The default TransformerEncoder layer of the Keras library is used in this project. [28] This layer encodes text to the meaningful representation internally:

- The number of layers
- The number of hidden units
- The number of heads
- The dropout rate
- The epsilon value for normalization

TransformerDecoder layer

The default TransformerDecoder layer of the Keras library is used in this project. [29] This layer decodes the internally meaningful representation into the text.

- The number of layers
- The number of hidden units
- The number of heads
- The dropout rate
- The epsilon value for normalization

Model

- The number of epochs
- The type of optimizer
- The learning rate of the optimizer

Text summarization task of CNN dailymail dataset

In [ ]:
```
# Install KerasNLP, and so on
%pip install keras-nlp rouge-score tensorflow-datasets datasets
```

In [ ]:
```
import platform
import numpy as np

import tensorflow as tf
import tensorflow_datasets as tfds
import keras
import keras_nlp

# Hyperparameters
EMBEDDING_DIM = 64
NUM_HEADS = 8
INTERMIDIATE_DIM = 1024
VOCAB_SIZE = 15000
BATCH_SIZE = 64 # 1 does not work
NUM_EPOCHS = 10 # 1 100
```

In [ ]:
```
import pandas as pd
```

In [ ]:
```
df_train = pd.read_csv('data/train.csv')
df_validation = pd.read_csv('data/validation.csv')
df_test = pd.read_csv('data/test.csv')
```

In [ ]:
```
if platform.system() == "Darwin" and platform.processor() == "arm":
    args = {
        'trust_remote_code': False,
    }
else:
    """
    When 'trust_remote_code' is False, it does not work on AWS SageMaker.
    """
    args = {
    }
train_dataset, validation_dataset, test_dataset = tfds.load(
    'huggingface:ccdv__cnn_dailymail/3.0.0',
    split=['train', 'validation', 'test'],
    builder_kwargs=args,
)
```

In [ ]:
```
# max_input_length = max(len(row[0][0]) for ds in [preprocessed_train_dat
# max_target_length = max(len(row[0][1]) for ds in [preprocessed_train_da
# max_decoder_target_length = max(len(row[1]) for ds in [preprocessed_tra
# max_input_length, max_target_length, max_decoder_target_length

summarized_text_size = 256 #  1437 is the longest summarized text in data
min_summarized_text_size = 128

# @TODO The followings should programmatically be derived.
max_input_length = 2137
max_target_length = summarized_text_size + 1
max_decoder_target_length = summarized_text_size + 1
```

In [ ]:
```
def filter_data_frame(df):
    return df[(df['highlights'].str.len() <= summarized_text_size) & (min
df_train = filter_data_frame(df=df_train)
df_validation = filter_data_frame(df=df_validation)
df_test = filter_data_frame(df=df_test)
```

In [ ]:
```
def split_input_target(df):
    return df['article'].to_numpy(), df['highlights'].to_numpy()
split_input_target(df=df_train)
```

In [ ]:
```
# Wrong
# train_dataset = train_dataset.filter(lambda x: tf.strings.length(x['hig
# validation_dataset = validation_dataset.filter(lambda x: tf.strings.len
# test_dataset = test_dataset.filter(lambda x: tf.strings.length(x['highl

# for development with 1/10 entries
DEVELOPMENT = True
if DEVELOPMENT:
    if platform.system() == "Darwin" and platform.processor() == "arm":
        NUM_TAKE = 128 # 500
        train_dataset = train_dataset.take(NUM_TAKE)
        validation_dataset = validation_dataset.take(NUM_TAKE)
        test_dataset = test_dataset.take(NUM_TAKE)
    else:
        # Use 10% dataset.
        train_size = len(train_dataset) // 10 * 9
        validation_size = len(validation_dataset) // 10 * 9
        test_size = len(test_dataset) // 10 * 9
        train_dataset = train_dataset.skip(train_size)
        validation_dataset = validation_dataset.skip(validation_size)
        test_dataset = test_dataset.skip(test_size)
```

In [ ]:
```
# @see https://github.com/keras-team/keras-nlp/blob/50e041487b1d8b30b34c5
import string
import re

strip_chars = string.punctuation
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

def custom_standardization(input_string):
    lowercase = tf.strings.lower(input_string)
    return tf.strings.regex_replace(
        lowercase,
        "[%s]" % re.escape(strip_chars),
        "",
    )

vectorization_layer = keras.layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    ragged=True,
)
# Warning: adapt, which clear the already held data inside, must be calle
vectorization_layer.adapt(train_dataset.concatenate(validation_dataset).c
```

In [ ]:
```
input_vectorization_layer = keras.layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    # @TODO This should be programmatically obtained
    output_sequence_length=2137,
)
target_vectorization_layer = keras.layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    output_sequence_length=summarized_text_size + 1,
)
input_vectorization_layer.set_vocabulary(vectorization_layer.get_vocabula
target_vectorization_layer.set_vocabulary(vectorization_layer.get_vocabul
```

In [ ]:
```
vectorization_layer.vocabulary_size(), vectorization_layer.get_vocabulary
```

In [ ]:
```
# Must be False
assert not vectorization_layer(['[start]'])[0] == vectorization_layer(['s
```

In [ ]:
```
"""
vectorization_layer(['This is a pen', 'I am a software engineer'])
#vectorization_layer(['This is a pen', 'I am a software engineer']).row_l
# 2
rows = vectorization_layer(['This is a pen', 'I am a software engineer'])
vectorization_layer(['This is a pen', 'I am a software engineer']).to_ten
# .to_tensor()

RaggedTensor.to_tensor can make 0-filled Tensor
"""
def prepare_dataset(x):
    article = input_vectorization_layer(x['article'])
    highlights = tf.strings.join(['[start] ', x['highlights'], ' [end]'])
    h = vectorization_layer(highlights)
    rows = h.row_lengths().shape[0]
    sequences = h.to_tensor(shape=(rows, summarized_text_size + 1 + 1))
    highlights_decoder_input = sequences[:, :-1] # summarized_text_size -
    highlights_decoder_output = sequences[:, 1:] # summarized_text_size -
    return (
        (
            article, # encoder input
            highlights_decoder_input, # decoder input
        ),
        highlights_decoder_output, # decoder output
    )

preprocessed_train_dataset = train_dataset.batch(BATCH_SIZE).map(prepare_
preprocessed_validation_dataset = validation_dataset.batch(BATCH_SIZE).ma
preprocessed_test_dataset = test_dataset.batch(BATCH_SIZE).map(prepare_da
```

In [ ]:
```
for entry in train_dataset.take(1):
    print('article: ', entry['article'].numpy())
    print('highlights: ', entry['highlights'].numpy())
for entry in preprocessed_train_dataset.take(1):
    print('encoder input: ', entry[0][0].shape, entry[0][0][:10])
    print('decoder input: ', entry[0][1].shape, entry[0][1][:10])
    print('decoder output: ', entry[1].shape, entry[1][:10])
```

In [ ]:
```
learning_rate = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.001,
    decay_steps=20,
    decay_rate=0.99,
)
if platform.system() == "Darwin" and platform.processor() == "arm":
    """
    Apple Silicon mac shows tht following warning.
    WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.
    please use the legacy Keras optimizer instead,
    located at `tf.keras.optimizers.legacy.Adam`
    Therefore, keras.optimizers.legacy.Adam is used.
    """
    optimizer = keras.optimizers.legacy.Adam()
else:
    optimizer = keras.optimizers.Adam()

# Build model / Encoder & Decoder model
```

```
# The encoder encodes text and represents the feature vector.
# However, the decoder scheme contains this working, especially in its he
# That is, it is not certain whether the encoder is necessary for this ta
# There is value in the investigation.
encoder_inputs = keras.Input(
    shape=(max_input_length,),
    name="encoder_inputs"
)
encoder_embedding = keras_nlp.layers.TokenAndPositionEmbedding(
    vocabulary_size=VOCAB_SIZE,
    sequence_length=max_input_length,
    embedding_dim=EMBEDDING_DIM,
    mask_zero=True,
)(encoder_inputs)
encoder_outputs = keras_nlp.layers.TransformerEncoder(
    num_heads=NUM_HEADS,
    intermediate_dim=INTERMIDIATE_DIM,
)(encoder_embedding)
### <temporary>
encoder_outputs = keras_nlp.layers.TransformerEncoder(
    num_heads=NUM_HEADS,
    intermediate_dim=INTERMIDIATE_DIM,
)(encoder_outputs)
### </temporary>

decoder_inputs = keras.Input(
    shape=(max_target_length,),
    name="decoder_inputs"
)
decoder_embedding = keras_nlp.layers.TokenAndPositionEmbedding(
    vocabulary_size=VOCAB_SIZE,
    sequence_length=max_target_length,
    embedding_dim=EMBEDDING_DIM,
    mask_zero=True,
)(decoder_inputs)
decoder_outputs = keras_nlp.layers.TransformerDecoder(
    num_heads=NUM_HEADS,
    intermediate_dim=INTERMIDIATE_DIM,
)(decoder_embedding, encoder_outputs, use_causal_mask=True)
### <temporary>
decoder_outputs = keras_nlp.layers.TransformerDecoder(
    num_heads=NUM_HEADS,
    intermediate_dim=INTERMIDIATE_DIM,
)(decoder_outputs, encoder_outputs, use_causal_mask=True)
### </temporary>

outputs = keras.layers.Dense(
    VOCAB_SIZE,
    activation="softmax"
)(decoder_outputs)

model = keras.Model(
    [encoder_inputs, decoder_inputs],
    outputs,
    name="transformer_text_summarization_model",
)
# Note
```

```
# In the case that the dataset is large and the dimension is small,
# the learning rate of Adam needed to be smaller.
model.compile(
    optimizer=optimizer,
    loss="sparse_categorical_crossentropy",
    metrics=[
        keras.metrics.SparseCategoricalAccuracy()
        # 'accuracy', #  This should not be used.
        # "loss", # This is not necessarily specified.
        # keras_nlp.metrics.RougeL()
    ]
)
model.summary()
```

In [ ]:
```
# Training
history = model.fit(
    preprocessed_train_dataset,
    validation_data=preprocessed_validation_dataset,
    # Generally, 100 to 200 is used as the epoch number for generative mo
    # However, because this is a prototype, the number is intentionally s
    epochs=NUM_EPOCHS,
    # steps_per_epoch=2,
)
```

In [ ]:
```
model.save('text_classification.keras')
```

In [ ]:
```
test_loss = model.evaluate(
    preprocessed_test_dataset,
)
test_loss
```

In [ ]:
```
def summarize(text):
    """
    Summarize text
    :param text: original text
    :return: summarized text
    """
    table = vectorization_layer.get_vocabulary()
    input_sequence = input_vectorization_layer([text])


    start_token = vectorization_layer('[start]')[0].numpy()
    end_token = vectorization_layer('[end]')[0].numpy()
    decoded_sentence = [start_token]
    for i in range(max_target_length):
        decoder_inputs = tf.convert_to_tensor(
            [decoded_sentence],
            dtype="int64",
        )
        decoder_inputs = tf.concat(
            [
                decoder_inputs,
                tf.zeros(
                    [1, max_target_length - i - 1],
                    dtype="int64",
                ),
```

```
            ],
            axis=1,
        )
        predictions = model.predict(
            [input_sequence, decoder_inputs],
            verbose=0
        )
        predicted_token = np.argmax(predictions[0, i, :])
        decoded_sentence.append(predicted_token)
        if predicted_token == end_token:
            break

    detokenized_output = []
    for token in decoded_sentence:
        detokenized_output.append(table[token])
    return " ".join(detokenized_output)

#     print(start_token)
#     decoder_input_sequence = vectorization_layer(['[start]'])
#     rows = decoder_input_sequence.row_lengths().shape[0]
#     decoder_input_sequence = decoder_input_sequence.to_tensor(shape=(ro
#     print(decoder_input_sequence)

#     # article = input_vectorization_layer(x['article'])
#     # highlights = tf.strings.join(['[start] ', x['highlights'], ' [end
#     # h = vectorization_layer(highlights)
#     # rows = h.row_lengths().shape[0]
#     # sequences = h.to_tensor(shape=(rows, 1439))
#     # highlights_decoder_input = sequences[:, :-1] # 1438
#     # highlights_decoder_output = sequences[:, 1:] # 1438


#     summary = []
#     for _ in range(max_target_length):
#         predictions = model.predict(
#             [input_sequence, decoder_input_sequence],
#             verbose=0
#         )
#         #np.argmax(predictions[0, i, :])
#         next_token = tf.argmax(predictions[0, -1, :])
#         print(predictions[0, -1])
#         next_word = tokenizer.index_word.get(next_token.numpy(), '[UNK]
#         if next_word == '[end]':
#             break
#         summary.append(next_word)
#         decoder_input_sequence = tf.concat(
#             [decoder_input_sequence, tf.expand_dims([next_token], axis=
#             axis=-1
#         )
#     return ' '.join(summary)

# Sample
# sample_text = "Giant pig fell into the swimming pool at his home in Rin
# print("Original:", sample_text)
# print("Summary:", summarize(sample_text))
sample_text = """
(CNN) -- Usain Bolt rounded off the world championships Sunday by claimin
```

```
"""
print("Original:", sample_text)
print("Summary:", summarize(sample_text))

sample_text = """
Vice President Dick Cheney will serve as acting president briefly Saturda
"""
print("Original:", sample_text)
print("Summary:", summarize(sample_text))

sample_text = "There are two chickens in the garden."
print("Original:", sample_text)
print("Summary:", summarize(sample_text))

sample_text = "Two chickens fell into the swimming pool in the garden."
print("Original:", sample_text)
print("Summary:", summarize(sample_text))
```

In [ ]:

```
import tensorflow as tf
import keras
import keras_nlp

# Hyperparameters
EMBEDDING_DIM = 64
NUM_HEADS = 2
INTERMIDIATE_DIM = 256
START_TOKEN = '[start]'
END_TOKEN = '[end]'

# Sample dataset.
dataset = [
    (
        "Giant pig fell into the swimming pool at his home in Ringwood, H
        f"{START_TOKEN} Giant pig fell into the swimming pool.",
        f"Giant pig fell into the swimming pool. {END_TOKEN}",
    ),
    (
        "There are two chickens in the garden.",
        f"{START_TOKEN} There are chickens.",
        f"There are chickens. {END_TOKEN}",
    ),
]

# Preprocessing
input_texts, target_texts, decoder_target_text = zip(*dataset)

tokenizer = keras.preprocessing.text.Tokenizer(
    split=' ',
    filters='!"#$%&()*+,-./:;=?@\\^_`{|}~\t\n'
)
tokenizer.fit_on_texts(input_texts + target_texts + decoder_target_text)
vocabulary_size = len(tokenizer.word_index) + 1

input_sequences = tokenizer.texts_to_sequences(input_texts)
target_sequences = tokenizer.texts_to_sequences(target_texts)
decoder_target_sequences = tokenizer.texts_to_sequences(decoder_target_te
```

```
max_input_length = max(len(sequence) for sequence in input_sequences)
max_target_length = max(len(sequence) for sequence in target_sequences)
max_decoder_target_length = max(len(sequence) for sequence in decoder_tar

input_sequences = keras.preprocessing.sequence.pad_sequences(
    input_sequences,
    maxlen=max_input_length,
    padding='post'
)
target_sequences = keras.preprocessing.sequence.pad_sequences(
    target_sequences,
    maxlen=max_target_length,
    padding='post'
)
decoder_target_sequences = keras.preprocessing.sequence.pad_sequences(
    decoder_target_sequences,
    maxlen=max_decoder_target_length,
    padding='post'
)
# decoder_target_sequences = tf.expand_dims(decoder_target_sequences, axi

# Build model / Encoder & Decoder model
# The encoder encodes text and represents the feature vector.
# However, the decoder scheme contains this working, especially in its he
# That is, it is not certain whether the encoder is necessary for this ta
# There is value in the investigation.
encoder_inputs = keras.Input(
    shape=(max_input_length,),
    name="encoder_inputs"
)
encoder_embedding = keras_nlp.layers.TokenAndPositionEmbedding(
    vocabulary_size=vocabulary_size,
    sequence_length=max_input_length,
    embedding_dim=EMBEDDING_DIM,
    mask_zero=True,
)(encoder_inputs)
encoder_outputs = keras_nlp.layers.TransformerEncoder(
    num_heads=NUM_HEADS,
    intermediate_dim=INTERMIDIATE_DIM,
)(encoder_embedding)

decoder_inputs = keras.Input(
    shape=(max_target_length,),
    name="decoder_inputs"
)
decoder_embedding = keras_nlp.layers.TokenAndPositionEmbedding(
    vocabulary_size=vocabulary_size,
    sequence_length=max_target_length,
    embedding_dim=EMBEDDING_DIM,
    mask_zero=True,
)(decoder_inputs)
decoder_outputs = keras_nlp.layers.TransformerDecoder(
    num_heads=NUM_HEADS,
    intermediate_dim=INTERMIDIATE_DIM,
)(decoder_embedding, encoder_outputs)

outputs = keras.layers.Dense(
```

```
    vocabulary_size,
    activation="softmax"
)(decoder_outputs)

model = keras.Model(
    [encoder_inputs, decoder_inputs],
    outputs,
    name="transformer_text_summarization_model",
)
# Note
# In the case that the dataset is large and the dimension is small,
# the learning rate of Adam needed to be smaller.
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

# Training
model.fit(
    [input_sequences, target_sequences],
    decoder_target_sequences,
    # Generally, 100 to 200 is used as the epoch number for generative mo
    # However, because this is a prototype, the number is intentionally s
    # epochs=100,
    epochs=10,
)
```

Unlike the classification task, the first word is predicted as the decoder output, and it is used for the second word prediction as the decoder input. By repeating this until the decoder outputs the special end symbol, the complete summarized sentence is generated.

In [ ]:
```python
def summarize(text):
    """
    Summarize text
    :param text: original text
    :return: summarized text
    """
    input_sequence = tokenizer.texts_to_sequences([text])
    input_sequence = keras.preprocessing.sequence.pad_sequences(
        input_sequence,
        maxlen=max_input_length,
        padding='post'
    )
    idx = tokenizer.word_index[START_TOKEN]
    decoder_input_sequence = tf.constant(
        [[idx]],
        dtype=tf.int64
    )

    summary = []
    for i in range(max_target_length):
        predictions = model.predict(
            [input_sequence, decoder_input_sequence],
            verbose=0
        )
        next_token = tf.argmax(predictions[0, -1, :])
        next_word = tokenizer.index_word.get(next_token.numpy(), '[UNK]')
        if next_word == END_TOKEN:
            break
        summary.append(next_word)
        decoder_input_sequence = tf.concat(
            [decoder_input_sequence, tf.expand_dims([next_token], axis=-1
            axis=-1
        )
    return ' '.join(summary)

# Sample
sample_text = "Giant pig fell into the swimming pool at his home in Ringw
print("Original:", sample_text)
print("Summary:", summarize(sample_text))

sample_text = "There are two chickens in the garden."
print("Original:", sample_text)
print("Summary:", summarize(sample_text))

sample_text = "Two chickens fell into the swimming pool in the garden."
print("Original:", sample_text)
print("Summary:", summarize(sample_text))
```

References

- [1] Vaswani et al. (2017). Attention Is All You Need.https://doi.org/10.48550/arXiv.1706.03762
- [2] tensorflow. Tensor2tensor. Retrieved May 16, 2024, from

https://github.com/tensorflow/tensor2tensor
- [3] Yutao et al. (2023). Retentive Network: A Successor to Transformer for Large Language Models. https://doi.org/10.48550/arXiv.2307.08621
- [4] Wojciech et al. (2019). Neural Text Summarization: A Critical Evaluation. https://doi.org/10.48550/arXiv.1908.08960
- [5] Yang et al. (2019). Text Summarization with Pretrained Encoders. https://doi.org/10.48550/arXiv.1908.08345
- [6] Chollet, F. (2021). *Deep Learning with Python* (2nd ed., p. 266). Manning.
- [7] Chollet, F. (2021). *Deep Learning with Python* (2nd ed., p. 246). Manning.
- [8] ROUGE (metric). (2023, November 28). In Wikipedia. https://en.wikipedia.org/wiki/ROUGE_(metric)
- [9] Dosovitskiy, A. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. https://doi.org/10.48550/arXiv.2010.11929
- [10] BERT (language model). (2024, May 7). In Wikipedia. https://en.wikipedia.org/wiki/BERT_(language_model)
- [11] Chollet, F. (2021). *Deep Learning with Python* (2nd ed., p. 250). Manning.
- [12] Tensorflow. Retrieved June 6, 2024, from https://www.tensorflow.org
- [13] TensorFlow. (2024, May 26). In Wikipedia. https://en.wikipedia.org/wiki/TensorFlow
- [14] Keras: Deep Learning for humans. https://keras.io/
- [15] Keras. (2024, June 6). In Wikipedia. https://en.wikipedia.org/wiki/Keras
- [16] KerasNLP. Retrieved June 6, 2024, from https://keras.io/keras_nlp/
- [17] PyTorch. (2024, May 10). In Wikipedia. https://en.wikipedia.org/wiki/PyTorch
- [18] Hugging Face, Inc. Transformers. Retrieved June 6, 2024, from https://github.com/huggingface/transformers
- [19] (2020, February 22). facebook/bart-large-cnn. Retrieved May 1, 2024, from https://huggingface.co/facebook/bart-large-cnn
- [20] (2020, Aug 9). google/pegasus-cnn_dailymail. Retrieved May 1, 2024, from https://huggingface.co/google/pegasus-cnn_dailymail
- [21] Embedding layer. Keras. Retrieved June 4, 2024, from https://keras.io/api/layers/core_layers/embedding/
- [22] Word2vec. Tensorflow. Retrieved June 4, 2024, from https://www.tensorflow.org/text/tutorials/word2vec
- [23] Facebook Inc. FastText. Retrieved June 4, 2024, from https://fasttext.cc/
- [24] GloVe: Global Vectors for Word Representation. Retrieved June 4, 2024, from https://nlp.stanford.edu/projects/glove/
- [25] Google LLC. Word embeddings. Retrieved June 10, 2024, from https://www.tensorflow.org/text/guide/word_embeddings#word_embeddings_2
- [26] PositionEmbedding layer. Keras. Retrieved June 4, 2024, from https://keras.io/api/keras_nlp/modeling_layers/position_embedding/
- [27] Google LLC. SinePositionEncoding layer. Retrieved June 10, 2024, from https://keras.io/api/keras_nlp/modeling_layers/sine_position_encoding/
- [28] TransformerEncoder layer. Keras. Retrieved June 4, 2024, from https://keras.io/api/keras_nlp/modeling_layers/transformer_encoder/

- [29] TransformerDecoder layer. Keras. Retrieved June 4, 2024, from https://keras.io/api/keras_nlp/modeling_layers/transformer_decoder/
- [30] facebook (2020, February 22). Facebook/bart-large-cnn. Retrieved June 10, 2024, from https://huggingface.co/facebook/bart-large-cnn
- [31] Google (2020, August 9). Google/pegasus-cnn_dailymail. Retrieved June 10, 2024, from https://huggingface.co/google/pegasus-cnn_dailymail
- [32] Zhang, J. (2020). PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. https://doi.org/10.48550/arXiv.1912.08777
- [33] Google (2023, July 8). Abstractive Text Summarization with BART. Retrieved June 10, 2024, from https://keras.io/examples/nlp/abstractive_summarization_with_bart/