CSCE 5290/4290:
Natural Language Processing Project

Names:
Oriana Borges-Salinas, Zachary Chenausky, Anas Mohammed Nayeem, Dagar Rehan

Project Title:
Research Paper Translation and Summarization

Github:
https://github.com/oriAleph/Research-Translation-Summarization

Project Proposal:
The project's primary goal is to translate and summarize research papers. The project will be divided into two sections. The first section focuses on research paper translation from one language to another. The second section focuses on summarizing the document to provide the user with the most important information. Four deep-learning models will be trained using the same training set and epochs for both portions of the project. There will be four models in total, two for translation and two for summarization. Furthermore, comparisons will be made to show which models performed well and which did not, and ideas for improving the models will be tested.

TL;DR: Train and compare four deep-learning models for the translation and summarization of research papers.

Description:

1. **Motivation**
   The goal of the project is to develop software that will enable someone who doesn't speak the original language to comprehend the significant ideas of a research article. This comprises translating the paper's content from the original tongue into English and summarizing it for easier user comprehension. All the models will also be compared to one another to see which combination performed the best and how to make the models better.

2. Significance

This project has many practical applications and is a key component of natural language processing. A wide range of businesses and organizations use translation and summarization, two of the most explored topics in computer science. With this study, we will concentrate on Czech and English research syntax and semantics because various languages have distinct ways of describing information. Due to the project's NLP components, we apply what we learned in class to use by designing a tool for cross-cultural research comprehension. This serves as an excellent learning tool for our team.

3. Objectives

- Train two models that can translate from the native language to English
- Train two models that can summarize English text and give key points
- Compare which combination of models gives the best results
- Try to improve the models to give better results

4. Features

- Natural Language Toolkit (NLTK)
- Hugging Face
- spaCy
- TextBlob
- English datasets
- Czech datasets
- REGEX
- GPT Corpus
- Collections
- TensorFlow
- Keras
- Numpy

5. Task Visualization

Name	Task
Dagar	Summarization - T5-small model
Zachary	Translation - RNN model with Attention
Anas	Summarization - Abstractive roBERTa model
Oriana	Czech to English NMT with a Transformer and Keras

Related Work (Background):

Translation:

The task of machine translation involves converting a sentence from one target language to another, which has lately significantly advanced thanks to encoder-decoder attention-based systems like BERT. Rule-based, statistical, hybrid and neural-based methods can all be used for machine translation.

The rule-based approach, or RBMT, to machine translation, generates sentences using both the source language and the target language's grammatical rules. However, this method has the drawback of heavily relying on dictionaries and extensive editing.

The second approach is models based on statistics. The best part about statistical models is that they can be quickly translated from one language to another without taking context into account.

A hybrid machine translation system, or HMT, combines statistical models and RBMT. This performs better than prior tests conducted by other researchers, but it takes a lot more resources to develop a model like this than it does to use the other methods.

Using neural networks and their capacity to learn when given enough data, neural machine translation is a translation method that can independently pick up linguistic rules and iteratively learn how to connect the source and target languages. The issue with this method is that it requires a sizable dataset for training. NMT was used for this project; it is used by many popular translation sources, such as Google Translate.

Summarization:

NLP summarization is the task of breaking down large amounts of text into a paragraph or a few sentences to provide key information. As the field of NLP developed, summarization was split into two different subfields Extractive and Abstractive.

Extractive Summarization is the process of extracting key information from the text and using techniques like frequency to determine how relevant that information is. If the frequency is high enough or meets certain criteria then it will be added to the summarization paragraph and displayed to the user. This type of summarization does not involve generating new sentences, instead, sentences that appeared in the original text will appear in the summary as well. This has led to limited use of such a method even though useful it lacks the flexibility to make its predictions.

Abstractive Summarization is the process of generating a summary of text with sentences that are generated by the language model instead of just grabbing key information from the initial text. The model in Abstractive Summarization uses deep-learning models such as T5 or Bert to generate sentences that have not appeared in the text before. The models use supervised learning where you have a dataset with the actual text and a summary written by a human and the model tries to learn to make connections from text to summary.

The project is based on Abstractive Summarization.

Models such as T5 and Bert come pre-trained in which the weights and biases have already been trained on a huge dataset but for this project, the modes were initialized with the default weights of the model before the training took place. The models are trained on the dataset we have chosen.

Overall:

The combination of translation and summarization has already been combined in the past. This project focuses on the translation and summarization of research papers from all different types of fields and shall be fine-tuned for Computer Science related papers from the NIPS dataset.

Datasets:

Translation Datasets

WMT is the main event for machine translation and research. The conference is held yearly in conjunction with bigger natural language processing conferences [1].

The **wmt19_translate** dataset is based on the data from statmt.org, versions exist for various years utilizing a combination of several data sources [2].

Czech to English parallel sentences [3].

[1]“WMT,” *Machine Translate*, Sep. 2022. [Online]. Available: https://machinetranslate.org/wmt. [Accessed: Nov. 06, 2022]

[2]“wmt19_translate | TensorFlow Datasets,” *TensorFlow*, Oct. 2022. [Online]. Available: https://www.tensorflow.org/datasets/catalog/wmt19_translate. [Accessed: Nov. 06, 2022]

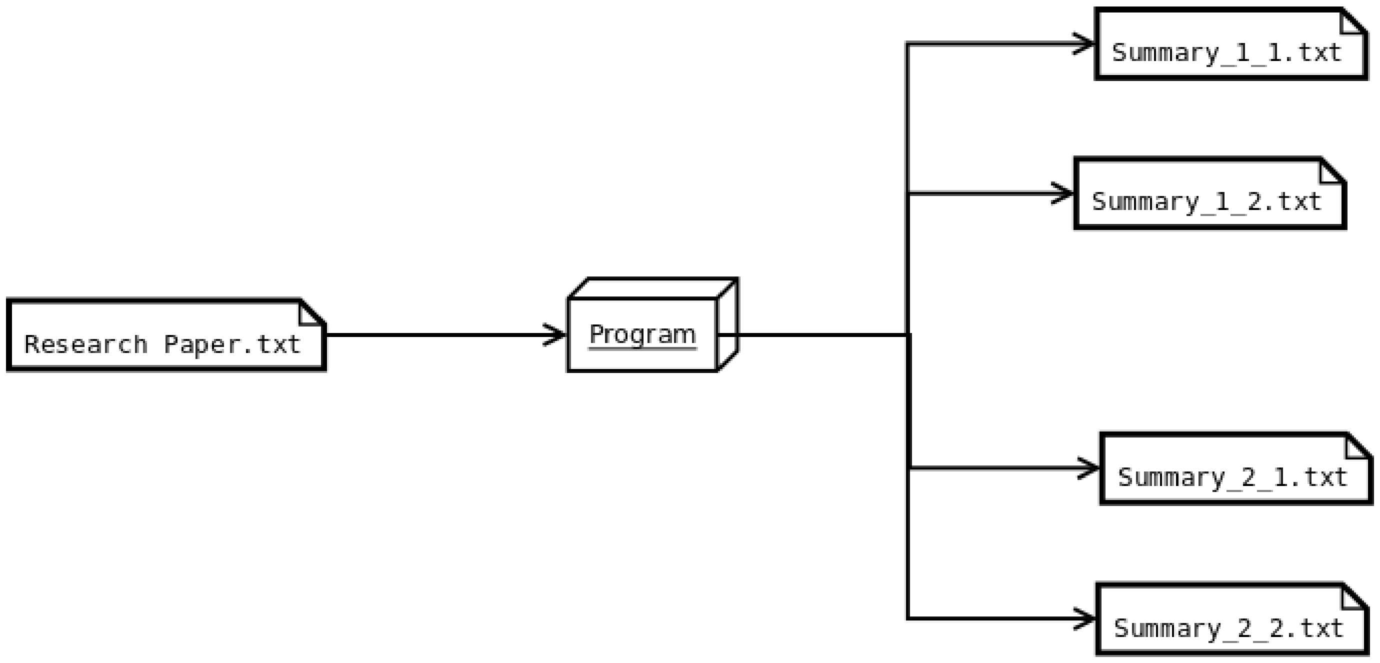
[3]“Tab-delimited Bilingual Sentence Pairs from the Tatoeba Project (Good for Anki and Similar Flashcard Applications),” *ManyThings*, 2022. [Online]. Available: http://www.manythings.org/anki/. [Accessed: Nov. 06, 2022]

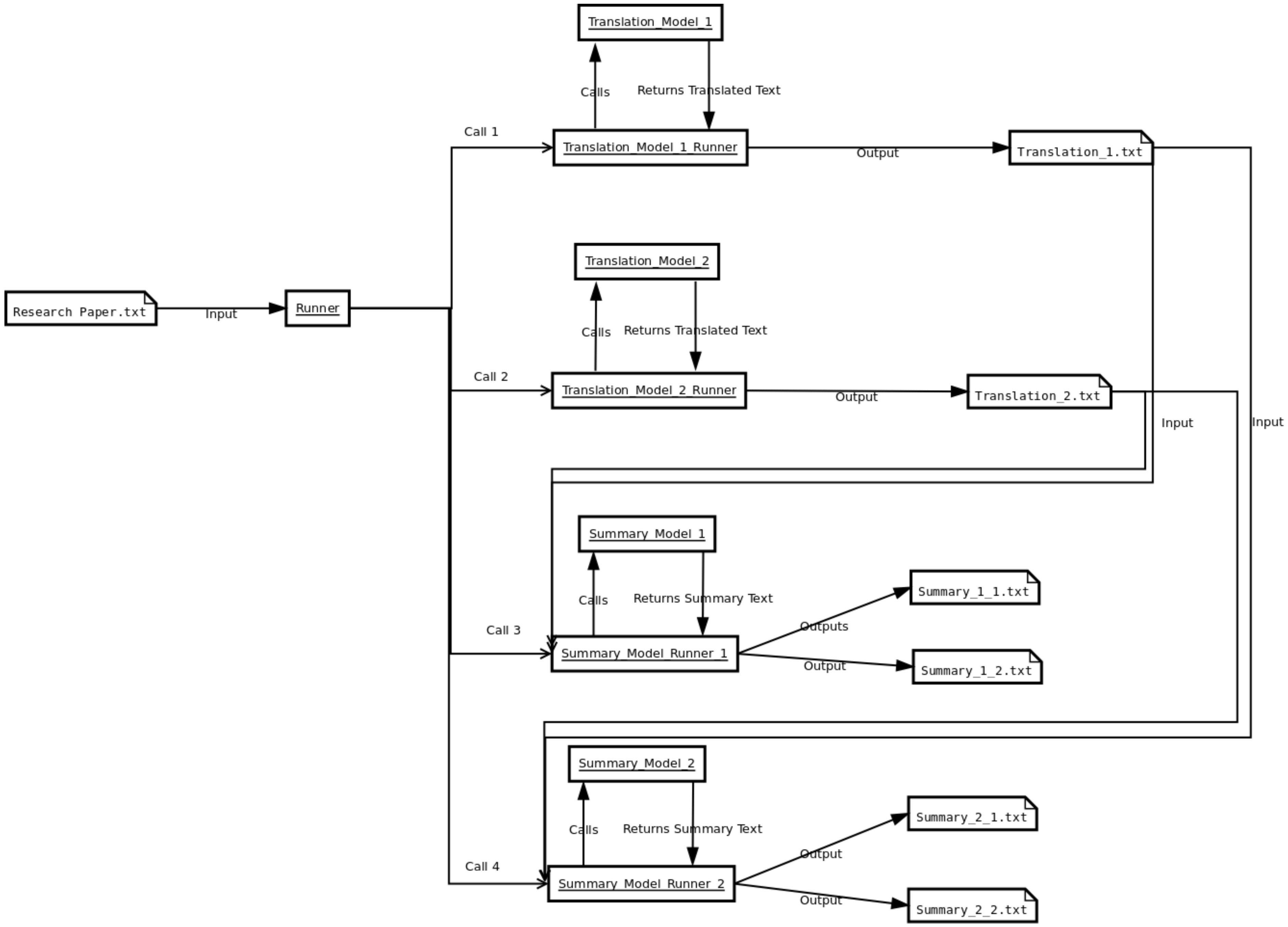
Summarization Datasets

For Training: scientific_papers subset: **arxiv** - contains papers that are not related to medicine.

For Tuning: All NeurIPS (NIPS) Papers

Detail Design of Features:




Research Paper.txt
Input
Runner
Call 1
Call 2
Call 3
Call 4
Translation_Model_1
Calls
Returns Translated Text
Translation_Model_1_Runner
Output
Translation_1.txt
Translation_Model_2
Calls
Returns Translated Text
Translation_Model_2_Runner
Output
Translation_2.txt
Input
Input
Summary_Model_1
Calls
Returns Summary Text
Summary_Model_Runner_1
Outputs
Summary_1_1.txt
Output
Summary_1_2.txt
Summary_Model_2
Calls
Returns Summary Text
Summary_Model_Runner_2
Output
Summary_2_1.txt
Output
Summary_2_2.txt

Analysis:

The project's goal is to make a piece of software that can take a research paper from a different language and translate it to English and then summarize the content to key points within the paper. This required the combination of two major NLP techniques such as machine translation and abstractive summarization. This combination of techniques has been done before but the focus is on research paper summarization and not a general summarization model. Furthermore, the model will be tuned for Computer Science paper summarization using the NIPS dataset.

Translation:

Transformer with Self-Attention Model [Oriana]:

A neural network is a model containing at least one hidden layer, which holds a set of neurons performing specific tasks [1].

A deep neural network design, a model with more than one hidden layer, called a **transformer** was created at Google and relies on self-attention processes. It can be thought of as a stack of self-attention layers that includes either a decoder, an encoder, or both [1].

A series of embeddings is transformed into another by a **self-attention** layer [1]. Attention can be any of a vast variety of methods that collect information from a collection of inputs in a dependent manner [1]. An example of a typical attention mechanism is a weighted sum over a collection of inputs, where the weight for each input is determined by a different component. Self-attention refers to the process of paying *attention* to itself rather than another context [1].

Transformers are effective for modeling sequential data because they can be parallelized, therefore, computations can be done concurrently [2]. Additionally, transformers are capable of capturing far-reaching contexts because attention enables each place to access the entirety of the input at each given layer [2]. Comparatively, the information must go through numerous processing steps in RNNs and CNNs in order to travel a long distance, which makes learning more challenging. A Transformer model is similar to a Recurrent Neural Network (RNN) model, but instead of RNN layers, a Transformer uses self-attention layers [2].

I had to develop a subword vocabulary and a BERT tokenizer for this model. A subword tokenizer's key benefit is that it dynamically adjusts between tokenization that is word-based and that is character-based, meaning it can return to word fragments and individual characters for unknown words [3].

According to the International Standardization Organization’s (ISO) 639 Language codes “cs” stands for Czech and “en” stands for English [4].

[1] “Machine Learning Glossary | Google Developers,” *Google Developers*, Oct. 2022. [Online]. Available: https://developers.google.com/machine-learning/glossary. [Accessed: Nov. 06, 2022]

[2] M. Daoust, “Neural machine translation with a Transformer and Keras | Text | TensorFlow,” *TensorFlow*, Nov. 2022. [Online]. Available: https://www.tensorflow.org/text/tutorials/transformer. [Accessed: Nov. 06, 2022]

[3] M. Daost, “Subword tokenizers | Text | TensorFlow,” *TensorFlow*, Jul. 2022. [Online]. Available: https://www.tensorflow.org/text/guide/subwords_tokenizer. [Accessed: Nov. 06, 2022]

[4] “ISO 639 — Language codes,” *ISO*, 2022. [Online]. Available: https://www.iso.org/iso-639-language-codes.html. [Accessed: Nov. 06, 2022]

RNN with Attention Model [Zachary]:

The recurrent neural network (RNN) model is based on the attention-based neural machine translation. The primary idea of training the model includes using sequence-to-sequence where the model converts one sequence in a certain language to another sequence in another language. The RNN encoder and decoder map the source language vector to its representation vector in its target sequence. This process works fine for smaller and simpler sentences however, as the length and contextual complexity of the source sequences increases it becomes more difficult for the model to translate. Attention allows for more accurate translations with RNN because of the fact that the decoder has the functionality to look back at previous translations.

roBERTa Abstractive summarization Model [Anas]:

The roBERTa model is based on the BERT model from hugging-face. roBERTa is an acronym for a robustly optimized BERT approach. It employs some enhancements like modifying the key hyperparameters and removing BERT’s next sentence pertaining to objectives. roBERTa uses more intricate tokens that go deeper down to characters. This helps in extracting text and reconstructing an abstractive summary for the model. When we say abstractive summary we target a summary that is concise and precise. Through its deeper learning techniques roBERTa leverages more intricate training sessions and a higher accuracy than that of a BERT summarizer. It is solely for these advantageous properties that roBERTa was employed in this project. Although BERT travels through 1M steps in total. The detailed architecture of roBERTa allows it to perform a higher quality grade training in just within 500K steps.

T5-small Model [Dagar]:

The T5 model is an encoder-decoder model that has been pre-trained on a huge dataset with supervised and unsupervised learning and can be finetuned downstream for specific tasks. In the case of the project, the weights were initialized to default to get rid of any of the training and were only trained on the science paper dataset. This has led to a significant impact on the summarization of general articles but should have better performance for science paper-related summaries. In addition, the T5-small model converts all NLP problems to Text-to-Text format for better performance.

Implementation:

Translation:

Transformer with Self-Attention Model [Oriana]:

This model follows and adapts the Neural machine translation with a Transformer and Keras TensorFlow tutorial. The Tensorflow lesson shows you step-by-step how to build and hone a Transformer model that can convert Portuguese into English by using the TED Talk HRLR transcripts from the Tensorflow Dataset Collection. However, this model converts Czech into English by using the WMT 2019 dataset from the Tensorflow Dataset Collection and statmt.org. For the exact dataset paths, look into wmt.py from the TensorFlow Datasets Translate Github. Both the original TensorFlow lesson and this notebook use the Subword Tokenizers TensorFlow tutorial to create a subword vocabulary and build a tokenizer for BERT, exported by using tf.saved_model so it can be used by other notebooks.

< —- Brief Overview of Set Up —->

Installation:

```
# Google Colab:
# !apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
!pip uninstall -y -q tensorflow keras tensorflow-estimator tensorflow-text
!pip install -q tensorflow_datasets
!pip install -q -U tensorflow-text tensorflow
```

Import modules:

```
import logging
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow as tf
import tensorflow_text
```

If on Google Colab:

```
# Google Colab:
# !unzip dev.zip
# !unzip training-parallel-nc-v13.zip
```

Download Dataset:

```
config = tfds.translate.wmt.WmtConfig(
      version="0.0.1",
      language_pair=("cs", "en"),
      subsets={
      tfds.Split.TRAIN: ["newscommentary_v13"],
      tfds.Split.VALIDATION: ["newstest2018"],
      },
)
builder = tfds.builder("wmt_translate", config=config)
```

```
print(builder.info.splits)
builder.download_and_prepare()
datasets = builder.as_dataset(as_supervised=True)
print('datasets are {}'.format(datasets))
```

```
train_examples = datasets['train']
val_examples = datasets['validation']
```

If on Google Colab:

```
# Google Colab:
# !unzip wmt19_translate_cs_en_converter.zip
```

Set up Tokenizer:

```
model_name = 'wmt19_translate_cs_en_converter'
tokenizers = tf.saved_model.load(model_name)
```

```
# Tokenizer methods
[item for item in dir(tokenizers.en) if not item.startswith('_')]
```

RNN with Attention Model [Zachary]:

The datasets used to train the model were gathered from manyThings.org/anki. The data sets were structured as side-by-side sentences, left-side English, and right-side Czech. Every Czech sentence was imported into its own text file, the same as the English sentences. Then the files were stored as arrays in order to create a TensorFlow.data dataset for the model.

The installation libraries:

```
!pip install "tensorflow-text>=2.10"
!pip install einops
```

And imports :

```
import numpy as np
import typing
from typing import Any, Tuple
import einops
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import tensorflow as tf
import tensorflow_text as tf_text
```

Next creating the TensorFlow dataset from the imported text files. In order to train the model the use of text vectorization for text processing is the next step. Text processing includes removing punctuation from the datasets as well as setting all characters to lowercase for Unicode normalization.

```
text = tf_text.normalize_utf8(text, 'NFKD')
text = tf.strings.lower(text)
# Keep space, a to z, and select punctuation.
text = tf.strings.regex_replace(text, '[^ a-z.?!,¿]', '')
# Add spaces around punctuation.
text = tf.strings.regex_replace(text, '[.?!,¿]', r' \0 ')
# Strip whitespace.
text = tf.strings.strip(text)
```

```
text = tf.strings.strip(text)
```

Extracting vocabulary from each language is important for the language model. The TensorFlow Keras text vectorization layer handles the vocabulary extraction for both the Czech and English datasets. The text vectorization function reads one epoch of the training set and initializes the layer based on each data to determine the language vocabulary.

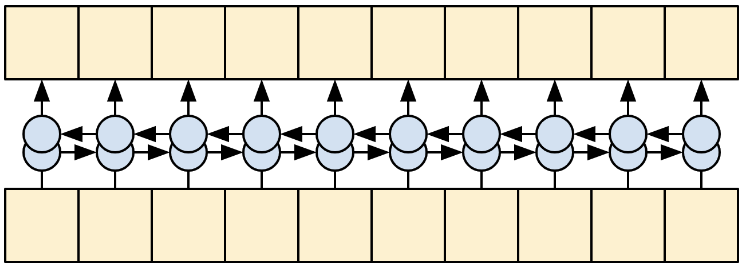
```
context_text_processor.adapt(train_raw.map(lambda context, target:
context))

# Here are the first 10 words from the vocabulary:
context_text_processor.get_vocabulary()[:10]
```

```
target_text_processor = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_vocab_size,
    ragged=True)

target_text_processor.adapt(train_raw.map(lambda context, target: target))
target_text_processor.get_vocabulary()[:10]
```

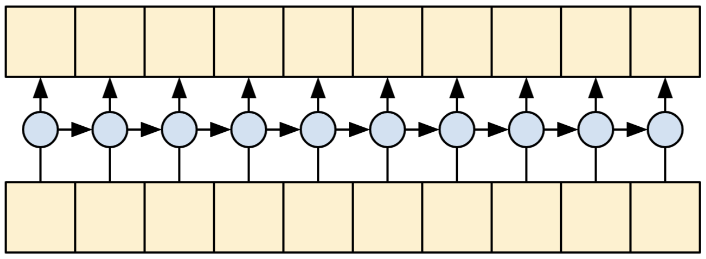
These layers can now convert the batch of strings into a batch of token IDs.

The TensorFlow datasets created previously are converted into 0-padded tensors of token IDs in the process_text function. In order to use Keras modle.fit the input must be the context(English) and the target(Czech) with the target in and target out labels.

The encoder is used to process the context sequence into a sequence of vectors so that the decoder may use that information to predict the output at each timestep. The sequence is constant so the model uses a bidirectional RNN.



The decoder will predict tokens at each target sequence. The decoder uses the RNN to process the target sequence and attention to keep track of each generated prediction. The model while training predicts the next word at each location one word at a time.



Given the context and target tokens, each target token, predicts the next target token. The model components include the RNN, attention setup, encoder, and decoder functionalities. After each section is implemented the model can be trained on the datasets.

```
 def __init__(self, units,
              context_text_processor,
              target_text_processor):
   super().__init__()
   # Build the encoder and decoder
   encoder = Encoder(context_text_processor, units)
   decoder = Decoder(target_text_processor, units)

   self.encoder = encoder
   self.decoder = decoder

  def call(self, inputs):
   context, x = inputs
   context = self.encoder(context)
   logits = self.decoder(context, x)
```

An example input after translation:

```
inputs = [
    'Je tu opravdu zima.', # "It's really cold here."
    'Tohle je můj život.', # "This is my life."
    'Jeho pokoj je nepořádek.' # "His room is a mess"
]
```

```
for t in inputs:
  print(model.translate([t])[0].numpy().decode())
print()
its really cold
this is my life doesnt belong to
his room is overconfident
```

The model works fine on short sentences, but when the input increases, the model loses focus and is unable to provide reasonable predictions. One way this could be improved would be to implement a re-learning of its predicted outputs. The model as is using a teacher-forcing method where each correct token is learned and passed through the model. The model makes predictions and learns the datasets regardless of whether the previous predictions were correct or not. More advanced systems are able to learn from predictions.

Summarization:

roBERTa Model [Anas]:

roBERTa is a hugging face module that was built on the architecture of BERT. BERT is a language network architecture that employs text to encode,segment,extract data. It is a part of a hugging-face module . SInce roBERTa has more deeply created tokens and a longer pertaining duration we can leverage its intricate structure to develop a more optimized text summarizer.
The roBERTa model was implemented relating very closely to the T5-small model, the objective was to implement the training model with the same dataset as with the T5 model so as to compare the summarization behavior between the two models.
roBERTa in its name stands for Robust Optimized BERT Approach, unlike BERT that utilizes word piece tokenization we will leverage roBERTa's character level byte pair encoding tokenizers to enhance the number of word pieces used.
A dataset was downloaded from the Hugging Face database namely "scientific_papers".
However the design implementation was a bit different,

The dataset was first downloaded and saved and then fed into the trainer.
The libraries installed were:

```
!pip install transformers
!pip install datasets
!pip install rogue_score
```

The libraries imported are:

```
import datasets
import transformers
import pandas as pd
from datasets import Dataset
from transformers import RobertaTokenizerFast
from transformers import EncoderDecoderModel
from transformers import TrainingArguments

from transformers.trainer_seq2seq import Seq2SeqTrainer
from transformers.training_args_seq2seq import Seq2SeqTrainingArguments
from dataclasses import dataclass, field as dataclassfield
from typing import Optional
from datasets import load_dataset, load_metric
from transformers import RobertaConfig, RobertaModel
from transformers import EncoderDecoderModel
from transformers import RobertaConfig, RobertaModel
from transformers import DataCollatorForSeq2Seq
```

We use RobertaConfig to tokenize our dataset into default biased tokens.
One important feature of RobertaConfig is that it helps us create weightless parameters such that there is no distinct bias in our tokens,
The same code can be found under the Github repository under the function name weightlessModel()

```
weightless_model(model_checkpoint,tokenizer):
   #make a model that is not pretrained
   config=RobertaConfig()
   model=RobertaModel(config)
   model.init_weights()

   data_collator=DataCollatorForSeq2Seq(tokenizer=tokenizer,model=model)
   return(model,data_collator)
```

This is part of the preprocessing steps that we take, meaning in order for us to feed the data into the model it should be in a way that the model is able to read and learn from it. By not biasing the data we are ensuring a more cleaner and stable learning algorithm that is able to understand word parameters and summarize.

We will not be jumping into other details of preprocessing so as to avoid any instances of duplication.

After we have winder up the preprocess filters we will pass them through our training_args.
Interestingly we've always been curious about what exactly training_args is?
Well, training arguments are tweaks that we make in our dataset so as to compatibilitate it with our training model. The training model we will be using is the Seq2Seq trainer. Training

arguments help set parameters like the batch size of the trainer, CUDA operations, steps required for eval_process and warmup steps.

```
training_args = Seq2SeqTrainingArguments(
   output_dir="./",
   per_device_train_batch_size=batch_size,
   per_device_eval_batch_size=batch_size,
   predict_with_generate=True,
   #evaluate_during_training=True,
   do_train=True,
   do_eval=True,
   logging_steps=2,
   save_steps=16,
   eval_steps=500,
   warmup_steps=500,
   overwrite_output_dir=True,
   save_total_limit=1,
   fp16=True,
)
```

Here is an example, as we see in the code (Also available in the GIT repo) we have a set of parameters that all relate to our dataset , the way it is to be handled by the trainer.

Finally, lets talk about the trainer. Just like training_args , imported from Transformers we will be using the Seq2Seq trainer. This trainer allows us to integrate our model with rogue benchmarks.

I would have explained in detail about the rogue benchmarks that we are using but you can find that in the T5-small model explanation.

Both models had been trained to run 5 epochs so as to ensure stability in comparison of their outcomes.

Although summary expectations were set higher , the results exited were not as ambitious as expected. As a research in development this is a great sign as it gives developers an opportunity to explore several windows of improvement, some of which will be eagerly displayed in increment-2.

T5-small Model [Dagar]:

The T5 model was trained on the science paper dataset that was uploaded to the Hugging Face database.

To install the libraries for training using pip:

```
pip install datasets transformers rouge-score nltk
```

The libraries required to train the model are:

- `from datasets import load_dataset, load_metric`
- `from transformers import AutoTokenizer`
- `from transformers import DataCollatorForSeq2Seq`
- `from transformers import AutoModelForSeq2SeqLM, Seq2SeqTrainingArguments, Seq2SeqTrainer`
- `from transformers import AutoModelForSeq2SeqLM, DataCollatorForSeq2Seq, Seq2SeqTrainingArguments, Seq2SeqTrainer`
- `from transformers import create_optimizer, AdamWeightDecay`
- `from transformers import AutoConfig`
- `from transformers import T5Model`
- `import nltk`
- `import numpy as np`

The t5-small tokenizer used was prebuilt and pre-trained as to parse the text from the corpus. The T5-small tokenizer was used as it pairs well with the t5-small summarizer. The tokenizer from the training data takes in the paper and the summary and truncates it to 1024 and 128 bytes for the model.

To initialize the t5-small summarization model to weights, biases, and any other factor from before the model was pre-trained it is important to use:

```
config = AutoConfig.from_pretrained(model_checkpoint)
model = AutoModelForSeq2SeqLM.from_config(config)
model.init_weights()
```

The auto-config generates a t5-small model that has not been trained.

In addition, the T5 model was trained for 5 epochs with the current hyperparameters set to:

```
(
     f"{model_name}-science-papers",
     evaluation_strategy = "epoch",
     learning_rate=2e-5,
     per_device_train_batch_size=batch_size,
     per_device_eval_batch_size=batch_size,
     weight_decay=0.01,
     save_total_limit=20,
     num_train_epochs=my_epochs,
     predict_with_generate=True,
     fp16=floating_point,
     push_to_hub=True,
 )
```

The batch_size was set to 16 to reduce the amount of memory consumption but the runtime increases.

The metric used to compare the models was Recall-Oriented Understudy (ROUGE):

```
metric = load_metric("rouge")
```

The score is measured by comparing the summary generated by the model with a summary written by a human ut ROUGE goes a step further and does:

Recall = Number_of_overlapping_words / Total_words_in_Refrece_summary

To calculate precision which tells how verbose the summary is:

Precision = Number_of_overlapping_words / Total_words_in_System_summary

Rouge1 measures unigrams

Rouge 2 measures bigrams

Rougel measures the longest matching sequence of words by using the LCS algorithm.

Having higher values for all Rouges is always better as it produces generally better summaries.

To install the libraries to run the model using pip:

```
pip install transformers
```

The libraries to import are:

- `import torch`
- `import json`
- `from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config`

- `from transformers import AutoModelForSeq2SeqLM, AutoTokenizer`

To get the custom trained model to use:

```
    model =
AutoModelForSeq2SeqLM.from_pretrained("Dagar/t5-small-science-papers")
```

This line of code gets the model from the online repository that is on Hugging Face:

https://huggingface.co/Dagar/t5-small-science-papers

This makes using the model easier as you can just get it from the repo instead of having the model in the git repo.

The tokenizer needs to be initialized as:

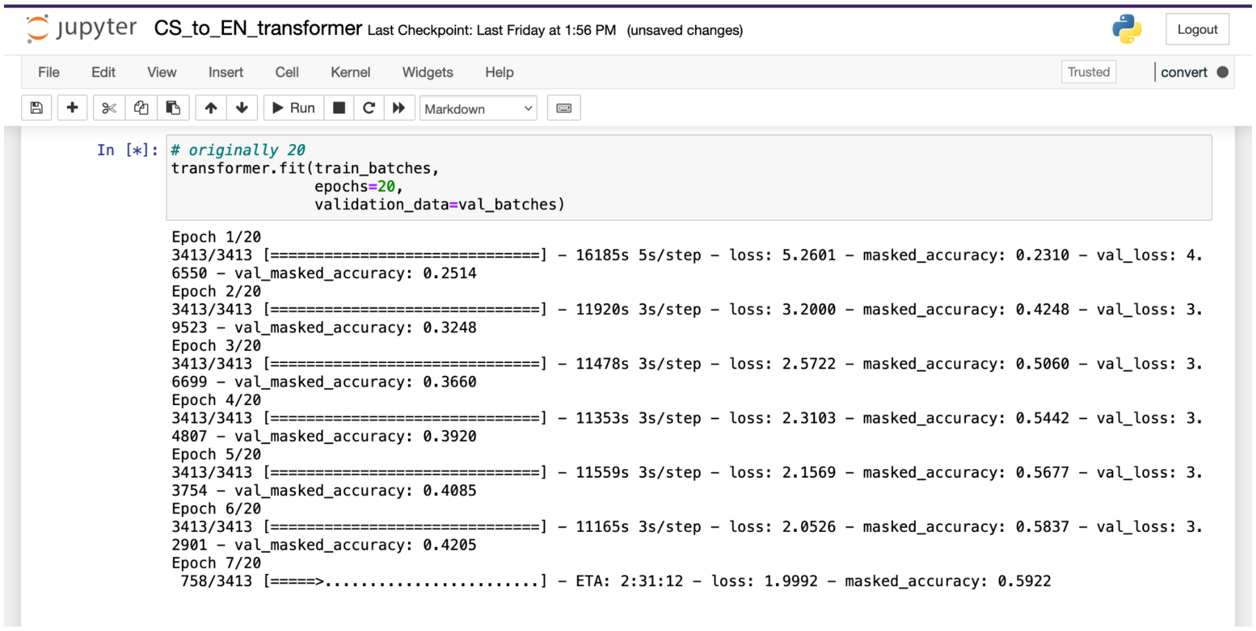
```
    tokenizer = AutoTokenizer.from_pretrained('t5-small')
```

The reason it needs to be a t5-small as it was the same tokenizer used to train the model previously.

Preliminary Results:

Transformer with Self-Attention Model [Oriana]:

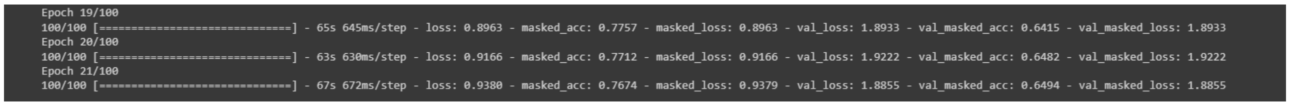
As you can see below, the model is taking a very long time to train since the data set is rather large. There are 20 epochs and each epoch needs about 3 hours to train, however, the accuracy seems to be increasing at a favorable rate.


jupyter
CS_to_EN_transformer Last Checkpoint: Last Friday at 1:56 PM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
convert
Run
Markdown

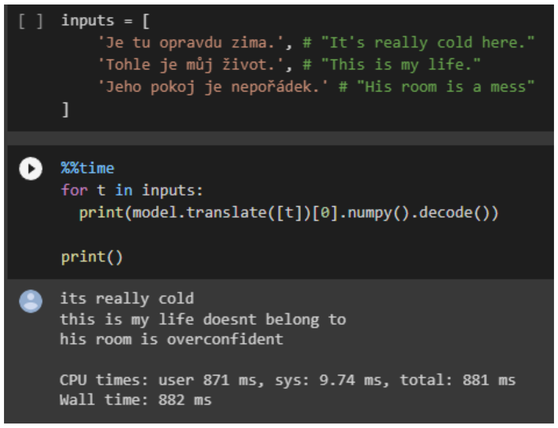
```
In [*]: # originally 20
        transformer.fit(train_batches,
                        epochs=20,
                        validation_data=val_batches)

        Epoch 1/20
        3413/3413 [==============================] - 16185s 5s/step - loss: 5.2601 - masked_accuracy: 0.2310 - val_loss: 4.
        6550 - val_masked_accuracy: 0.2514
        Epoch 2/20
        3413/3413 [==============================] - 11920s 3s/step - loss: 3.2000 - masked_accuracy: 0.4248 - val_loss: 3.
        9523 - val_masked_accuracy: 0.3248
        Epoch 3/20
        3413/3413 [==============================] - 11478s 3s/step - loss: 2.5722 - masked_accuracy: 0.5060 - val_loss: 3.
        6699 - val_masked_accuracy: 0.3660
        Epoch 4/20
        3413/3413 [==============================] - 11353s 3s/step - loss: 2.3103 - masked_accuracy: 0.5442 - val_loss: 3.
        4807 - val_masked_accuracy: 0.3920
        Epoch 5/20
        3413/3413 [==============================] - 11559s 3s/step - loss: 2.1569 - masked_accuracy: 0.5677 - val_loss: 3.
        3754 - val_masked_accuracy: 0.4085
        Epoch 6/20
        3413/3413 [==============================] - 11165s 3s/step - loss: 2.0526 - masked_accuracy: 0.5837 - val_loss: 3.
        2901 - val_masked_accuracy: 0.4205
        Epoch 7/20
         758/3413 [=====>........................] - ETA: 2:31:12 - loss: 1.9992 - masked_accuracy: 0.5922
```

RNN with Attention Model [Zachary]:

With short sentences the model is relatively accurate:



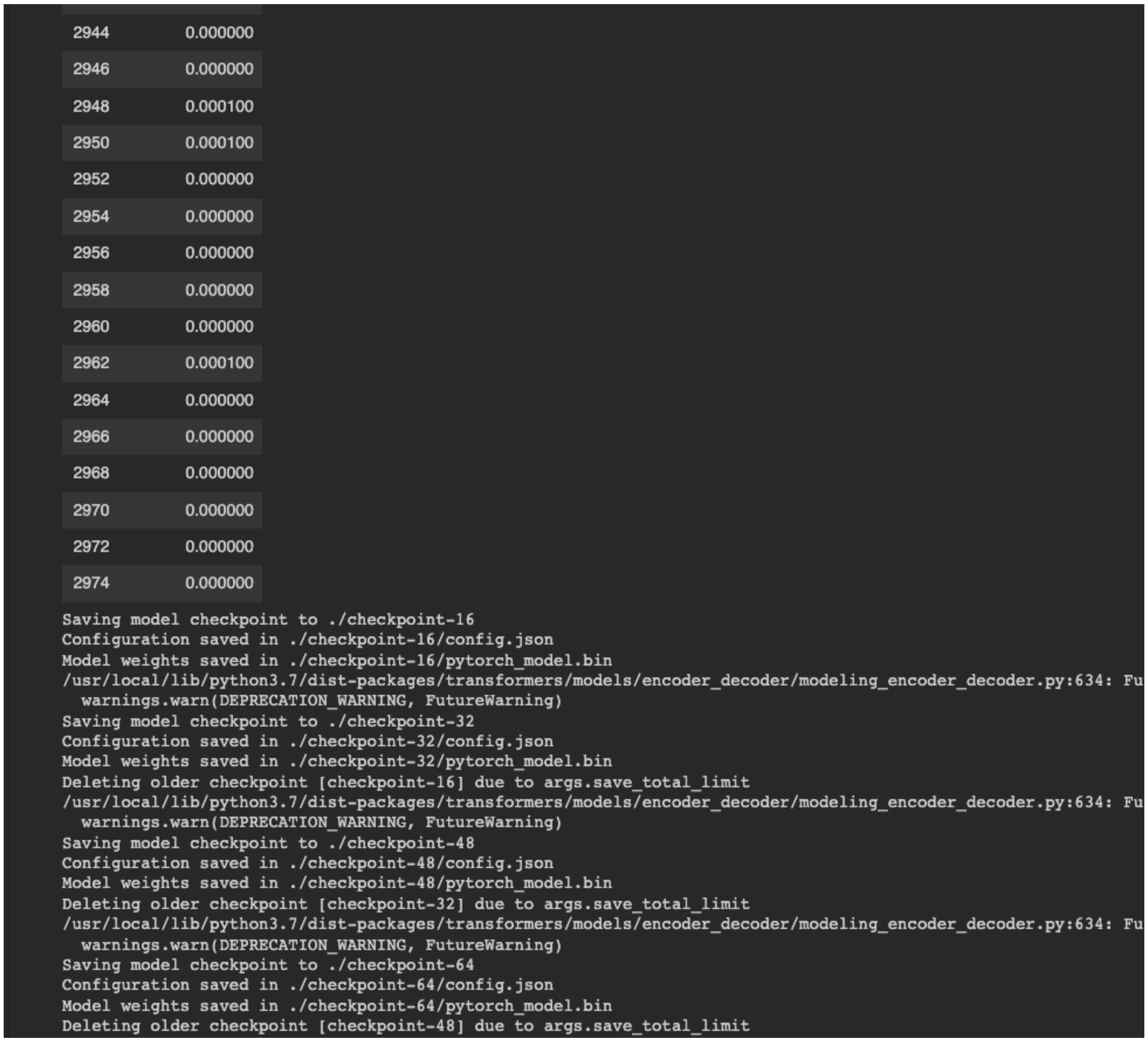
```
Epoch 19/100
100/100 [==============================] - 65s 645ms/step - loss: 0.8963 - masked_acc: 0.7757 - masked_loss: 0.8963 - val_loss: 1.8933 - val_masked_acc: 0.6415 - val_masked_loss: 1.8933
Epoch 20/100
100/100 [==============================] - 63s 630ms/step - loss: 0.9166 - masked_acc: 0.7712 - masked_loss: 0.9166 - val_loss: 1.9222 - val_masked_acc: 0.6482 - val_masked_loss: 1.9222
Epoch 21/100
100/100 [==============================] - 67s 672ms/step - loss: 0.9380 - masked_acc: 0.7674 - masked_loss: 0.9379 - val_loss: 1.8855 - val_masked_acc: 0.6494 - val_masked_loss: 1.8855
```



```
[ ] inputs = [
        'Je tu opravdu zima.', # "It's really cold here."
        'Tohle je můj život.', # "This is my life."
        'Jeho pokoj je nepořádek.' # "His room is a mess"
    ]

    %%time
    for t in inputs:
      print(model.translate([t])[0].numpy().decode())

    print()

    its really cold
    this is my life doesnt belong to
    his room is overconfident

    CPU times: user 871 ms, sys: 9.74 ms, total: 881 ms
    Wall time: 882 ms
```

However when the input data grows the model becomes less contextually accurate and thus the translation suffers.

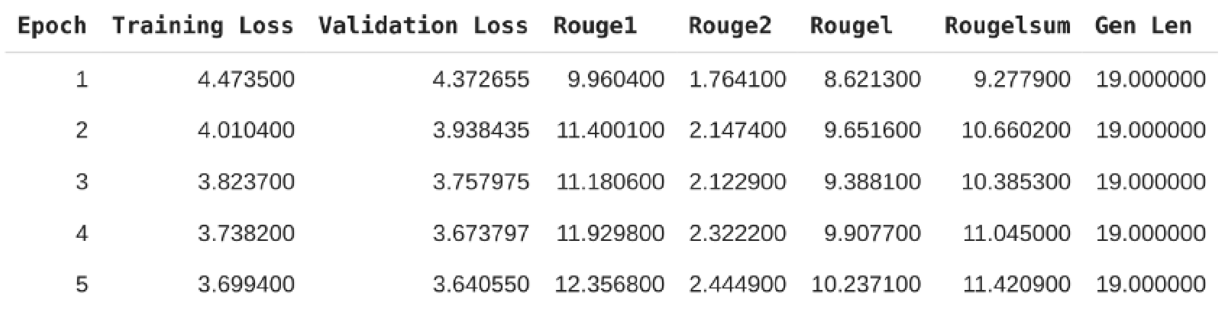
roBERTa Model [Anas]:
The platform had timed out by the time I was able to analyze the rogue outputs. Since the epochs take anywhere 3 to 4 hours to process . I will be displaying the training model results in hopes that they count as preliminary.



```
2944    0.000000
2946    0.000000
2948    0.000100
2950    0.000100
2952    0.000000
2954    0.000000
2956    0.000000
2958    0.000000
2960    0.000000
2962    0.000100
2964    0.000000
2966    0.000000
2968    0.000000
2970    0.000000
2972    0.000000
2974    0.000000
Saving model checkpoint to ./checkpoint-16
Configuration saved in ./checkpoint-16/config.json
Model weights saved in ./checkpoint-16/pytorch_model.bin
/usr/local/lib/python3.7/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:634: Fu
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
Saving model checkpoint to ./checkpoint-32
Configuration saved in ./checkpoint-32/config.json
Model weights saved in ./checkpoint-32/pytorch_model.bin
Deleting older checkpoint [checkpoint-16] due to args.save_total_limit
/usr/local/lib/python3.7/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:634: Fu
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
Saving model checkpoint to ./checkpoint-48
Configuration saved in ./checkpoint-48/config.json
Model weights saved in ./checkpoint-48/pytorch_model.bin
Deleting older checkpoint [checkpoint-32] due to args.save_total_limit
/usr/local/lib/python3.7/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:634: Fu
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
Saving model checkpoint to ./checkpoint-64
Configuration saved in ./checkpoint-64/config.json
Model weights saved in ./checkpoint-64/pytorch_model.bin
Deleting older checkpoint [checkpoint-48] due to args.save_total_limit
```

So by analyzing our data, we see that the more steps we incur the lesser are our training losses.

T5-small Model [Dagar]:



Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	4.473500	4.372655	9.960400	1.764100	8.621300	9.277900	19.000000
2	4.010400	3.938435	11.400100	2.147400	9.651600	10.660200	19.000000
3	3.823700	3.757975	11.180600	2.122900	9.388100	10.385300	19.000000
4	3.738200	3.673797	11.929800	2.322200	9.907700	11.045000	19.000000
5	3.699400	3.640550	12.356800	2.444900	10.237100	11.420900	19.000000

For the total of 5 epochs run the ROUGE1 score reached 12.35 on the 5th epoch. The ROUGE2 score remains at a 2.44 as the dataset used does not have enough data on research papers to generate consistent bigrams. Furthermore, the idea to increase epochs to more than 5 was discarded to overfit the data. The second dataset chosen for tuning the model should increase the ROUGE2 score. The ROUGEl score is at a surprising 10.23

For example given the sentence:

"This paper pertains to discussing the Schrodinger equation along with its historical background. The paper starts with an introduction to quantum mechanics and requirement of Schrodinger equation. The paper highlights the need of Schrodinger equation in light of progresses made in quantum mechanics. The paper also highlights the historical background of the development of Schrodinger equation after which the paper also briefly touches the development of the same through classical means. The paper discusses about the Schrodinger equation while presenting the same for a hydrogen atom. The paper ends with conclusion in which the importance of Schrodinger equation is highlighted."

The summary generated by the model would be:

"in this paper, we study the dynamics of a hamiltonian of the heisenberg equation of motion for atoms. we show that the nonlinear equations of state of two - dimensionality in the case of an atomic system are derived from the generalized eigenstates and the quantum system based on the time evolution equation for the system with arbitrary number of energy and interacting with respect to the energy of this system is characterized by the method of quantum equation with the same time in analytic approach to hilbert space of non"

It is not a good summary but with further tuning the summary should become more accurate towards these types of papers as the hyperparameters have yet to be tuned.

Project Management:

- Implementation Status Report **(Oriana Borges-Salinas)**
  - Work Completed
    - Description -
      I originally started working on a Chinese-to-English NMT Transformer model but it proved to be difficult. I couldn't get past the preprocessing steps when creating the tokenizer since the top-down subword approach provided by TensorFlow doesn't work for CJK languages because they don't have clear multi-character units. The creator of the tutorial responded to a question asked previously by someone else about this issue, and they answered that the simplest approach to this was to use a pre-trained Chinese segmentation model from the TensorFlow Hub such as zh_segmentation and bert_zh_preprocess. While I could find one example using the TF BERT segmentation process on Github here and read about other implementations in WMT conference papers and from this Standford study, I didn't want to use the same types of methods they all had taken. However, I couldn't figure out the zh_segmentation implementation in time so I switched over to Czech since I needed a better understanding of my model and a working piece for the overall product.

      I adapted the Portuguese to English Transformer Tensorflow tutorial to fit the Czech dataset of my choosing, and it's currently in the process of training the Czech to English model. Originally I had it running for half a day on Google Collab but it was taking 14 hours for the first epoch, which went blank on me simply because the Wifi went out. Considering the size of the large size of my data set and the lag have I from my computer and cloud, I switched over to Jupyter Notebooks. Each epoch now lasts about three hours and at the time of this writing, it has two hours left for its 7th epoch. I have about 40 hours left, therefore in a little less than 2 days the training will be complete and I can export the model for further use and analysis.

    - Responsibility - I am responsible for getting the Neural Machine Translation Transformer using the Self-Attention model working and properly exported, evaluated, and integrated with the project.

    - Contributions - I helped Zachary get set up with TensorFlow after switching over from OpenNMT and tried to preprocess the same dataset I used with his tutorial. Since the preprocessing steps are different between each model and the data set from his tutorial isn't originally part of TensorFlow, I converted my sets into text files and modified the first few pre-processing steps in order to better fit the data. It seemed to be working since I tested it out with Spanish to English. However, when converting the Czech to English text files into NumPy arrays, one of them was off by 1 in terms of shape and they both need to be the same shape in order to use the Dataset.from_tensor_slices method. We both decided on using the same language as of this moment until we figured out a way to make the preprocessing work for the same dataset.

- Work to be completed
    - Description - As mentioned above, once my training finishes I will test and evaluate it, then export it. Afterward, I would need to combine my model with the others to achieve the goal of the project: research paper translation and summarization, and then analyze those results.
    - Responsibility (Task, Person) - My responsibility is to help integrate the models with each other and perform the translation-summarization tasks on research papers.
    - Issues / Concerns - My biggest concern is how long it's taking to train each model. It's very long and taxing for my laptop. I am also concerned about getting the same data set to work for both translation models, but that would just take us asking for help and some time to figure out.

- Implementation Status Report **(Zachary Chenausky)**
    - Work Completed
        - Description - Started work originally on OpenNMT translation models with transformers. With a lack of good resources, online a decision was made to switch to TensorFlow RNN with attention translation models. The final project implementation was created using a TensorFlow RNN model with custom datasets.
        - (Task, Person)- Trained RNN model on Czech/English Datasets, found results from different input texts.
        - Contributions - uploaded results to groups GitHub as well as contributed to group meetings and discussions.
    - Work to be completed
        - Description - Potentially improve translation as well as add punctuation to translation output.
        - Responsibility (Task, Person) - Individually I will help the group merge the models into one for the overall research understanding of each model and comparison.
        - Issues / Concerns - One concern would be improving the model could either be an issue of datasets, where I may need to find better and larger dataset resources which could take longer for the model to train on, or that this models implementation is just an outdated model where the transformer translator model would likely excel in the benchmark tests over the RNN model.

- Implementation Status Report **(Anas Mohammed Nayeem)**
    - Work Completed
        - Description - Started work initially on BERT summarizer for an extractive summary, realized expectations were directed towards a more abstractive inclined model rather than an extractive model of summarization. Focused research towards roBERTa optimization and trained a model to implement abstractive and ran a couple tests to analyze outputs.
        - Responsibility (Task, Person)- The summarizers are of 2 types. A T5 model and a roBERTa model.
        - Contributions - Worked on implementing the roBERTa model from scratch and training it.
    - Work to be completed
        - Description - Integration of translators implemented with roBERTa summarizer. Fine tuning the system to aim for a more accurate output. Record down outputs from the train data.
        - Responsibility (Task, Person) - Aim to integrate both RNN with attention and Transformer Self attention model with roBERTa summarizer.
        - Issues / Concerns - Regarding normalization of datasets. The integration can be a bit tricky though. Essentially we wanna take the output of the translators as our inputs. But we want to implement that by maintaining as much of a close relationship with the two summarizers as possible.
- Implementation Status Report **(Dagar Rehan)**
    - Work Completed
        - Description - Work on one of the summarization models to train and implement it. Get statistics to compare initial results for the model chosen. Work on the overall design of how to implement the models after training them all.
        - Responsibility (Task, Person) - Train the T5 model for summarization. Implement the T5 model for summarization. Get results using the ROUGE benchmark.
        - Contributions - Trained T5 model from scratch. Made diagrams for the system. Implemented T5 model. Helped with the Background, Related Work and Design section of the doc.
    - Work to be completed
        - Description - Finish tuning the T5 model to the NIPS dataset for better results regarding research papers.
        - Responsibility (Task, Person) - I will help with merging the modules together and fine-tuning the T5 model for better performance on the NIPS dataset. I will also convert the NIPS dataset to the T5 model format
        - Issues / Concerns - I don’t know if we should train our models on some general article to help the model learn better sentence structure. I think fine-tuning the summarization models on general articles and the NIPS dataset would be better. I don’t know if you can only tune with the NIPS

dataset or add a general dataset as well? To fine-tune the dataset to NIPS we need to convert the dataset to our own format with each model.

Resources:

- Papers:

A Survey on NLP-based Text Summarization for Summarizing Product Reviews
Summarising company announcements
Automatic Summarizing the News from Inform.kz by Using Natural Language Processing Tools
Analysis of Learning Approaches for Machine Translation Systems
Intro to ROUGE

Tensorflow:
Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo,
Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis,
Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,
Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia,
Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster,
Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens,
Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker,
Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas,
Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke,
Yuan Yu, and Xiaoqiang Zheng.
TensorFlow: Large-scale machine learning on heterogeneous systems,
2015. Software is available from tensorflow.org.