Text Summarization

Konuralp Dalkılınç - 23COMP5004

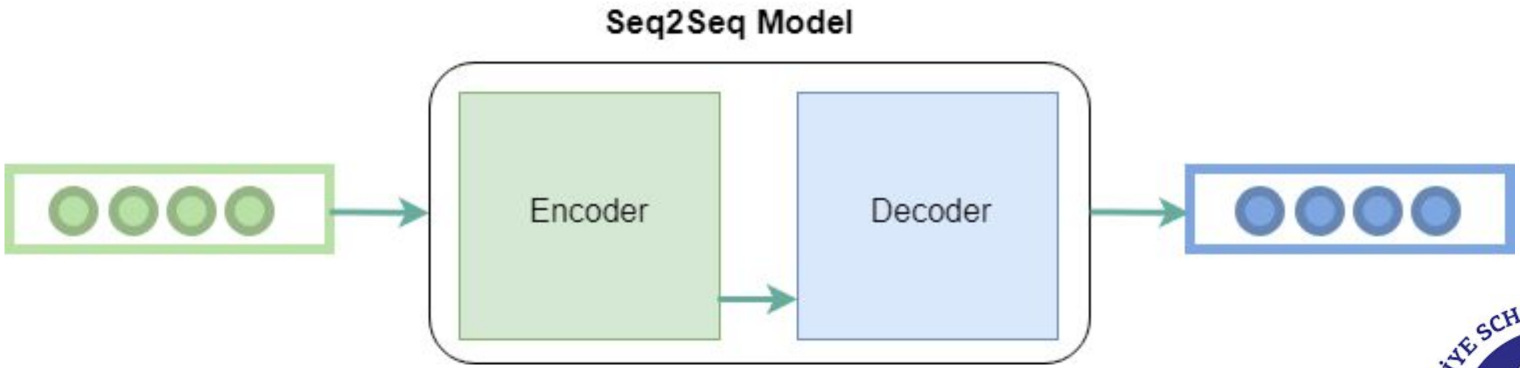
Isik University Faculty of Engineering and Natural Sciences

Project Description

- A model to convert paragraphs into condensed summaries. Capturing key information in an easily understandable and shorter format.

- Main architecture is Seq2Seq model with encoder-decoder pair.

- Implementing the fundamental principles covered in class to practice.



Methodology

a. **Data Preparation**

b. **Model Creation**

c. **Model Training**

d. **Evaluation**

Data Preparation

Multi-XScience Dataset

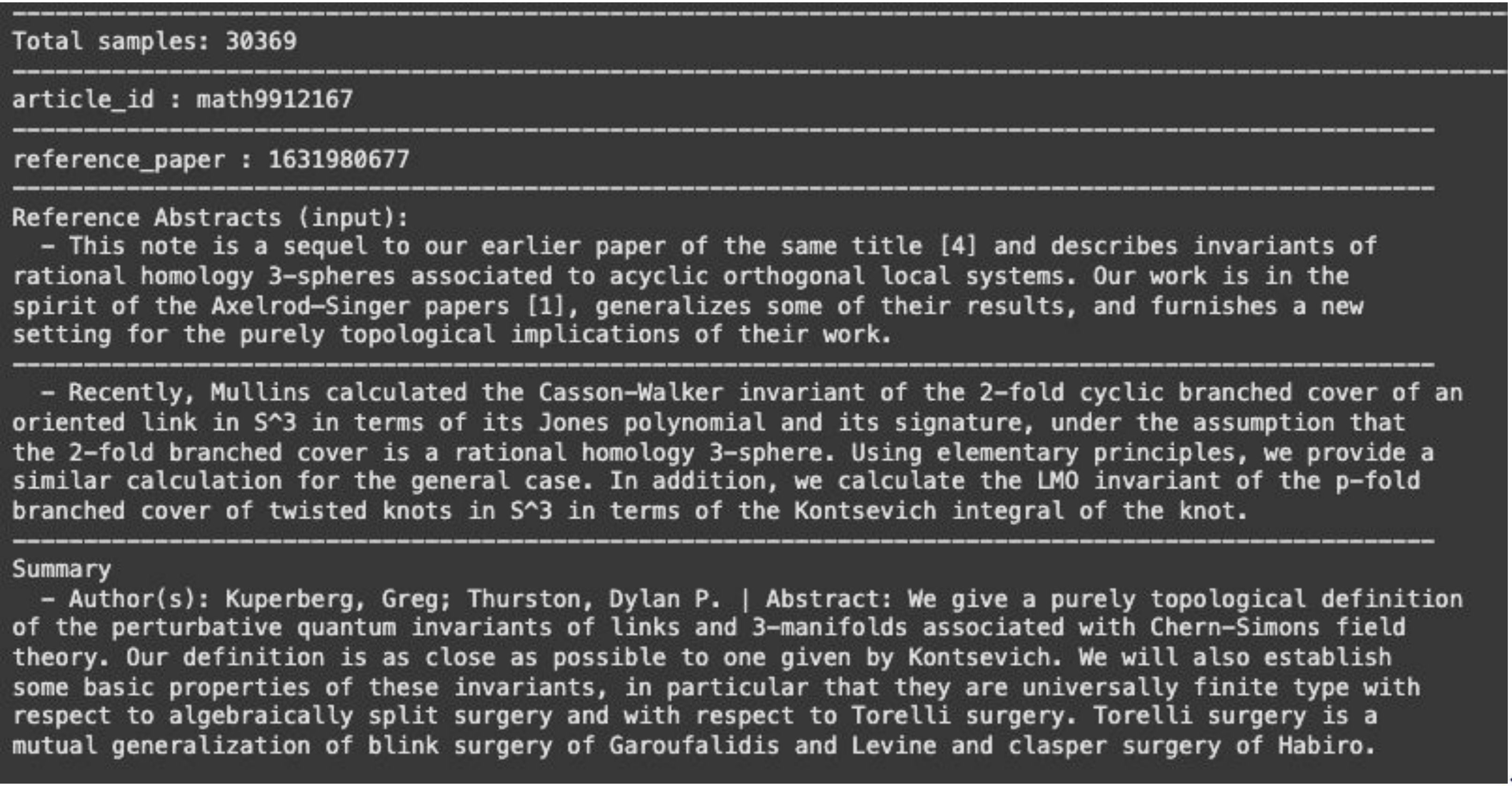
- Dataset consists of source abstracts, source titles, the target summary, and the target paper title.

- Dataset split

    - 30,019 training set
    - 3,814 validation set
    - 3,769 testing set

- Why this dataset is chosen

    - Providing realistic multi-document scientific summaries with abstract and related work section, making it ideal for abstractive summarization.



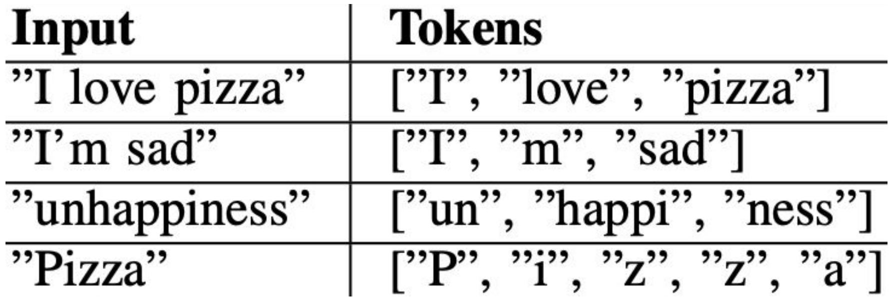
```
------------------------------------------------------------------------------------------------------------
Total samples: 30369
------------------------------------------------------------------------------------------------------------
article_id : math9912167
-------------------------------------------------------------------------------------------------------
reference_paper : 1631980677
-------------------------------------------------------------------------------------------------------
Reference Abstracts (input):
  - This note is a sequel to our earlier paper of the same title [4] and describes invariants of
rational homology 3-spheres associated to acyclic orthogonal local systems. Our work is in the
spirit of the Axelrod-Singer papers [1], generalizes some of their results, and furnishes a new
setting for the purely topological implications of their work.
-------------------------------------------------------------------------------------------------------
  - Recently, Mullins calculated the Casson-Walker invariant of the 2-fold cyclic branched cover of an
oriented link in S^3 in terms of its Jones polynomial and its signature, under the assumption that
the 2-fold branched cover is a rational homology 3-sphere. Using elementary principles, we provide a
similar calculation for the general case. In addition, we calculate the LMO invariant of the p-fold
branched cover of twisted knots in S^3 in terms of the Kontsevich integral of the knot.
-------------------------------------------------------------------------------------------------------
Summary
  - Author(s): Kuperberg, Greg; Thurston, Dylan P. | Abstract: We give a purely topological definition
of the perturbative quantum invariants of links and 3-manifolds associated with Chern-Simons field
theory. Our definition is as close as possible to one given by Kontsevich. We will also establish
some basic properties of these invariants, in particular that they are universally finite type with
respect to algebraically split surgery and with respect to Torelli surgery. Torelli surgery is a
mutual generalization of blink surgery of Garoufalidis and Levine and clasper surgery of Habiro.
```

Tokenizer

- Converts text into smaller units called tokens.
- Tokens can be words, sub-words, or characters.
- It's the first step in text processing for NLP models.
- Built in encoder method to tokenize words.

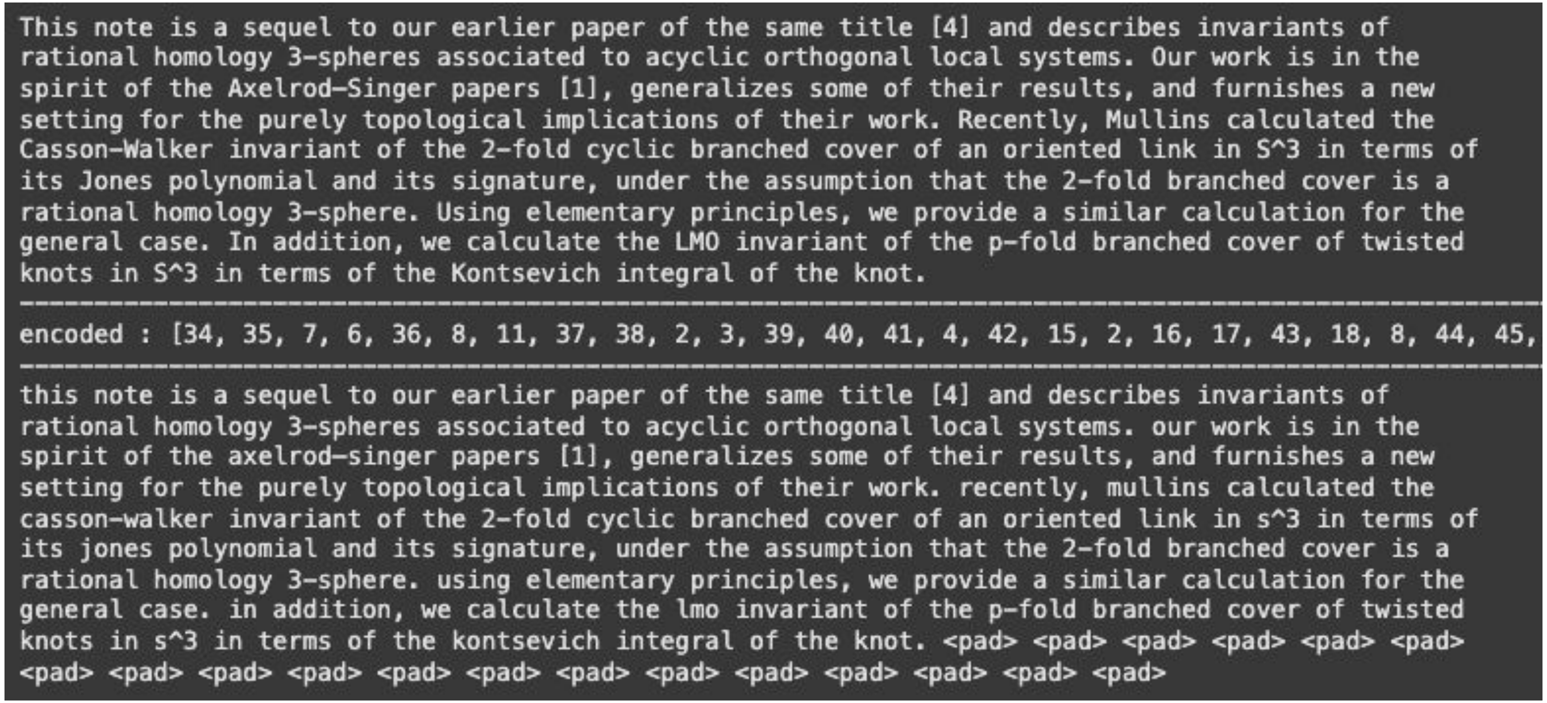
```
   def build_vocab(self):
       word_counter = Counter()
       for item in self.data:
           for text in item["ref_abstract"]["abstract"]:
               word_counter.update(text.lower().split())

word_counter.update(item["abstract"].lower().split())
       most_common_words =
word_counter.most_common(self.vocab_limit)
       self.word2idx = {"<pad>": 0, "<unk>": 1}
       for idx, (word, _) in
enumerate(most_common_words, start=2):
           self.word2idx[word] = idx
```



Input	Tokens
”I love pizza”	[”I”, ”love”, ”pizza”]
”I’m sad”	[”I”, ”m”, ”sad”]
”unhappiness”	[”un”, ”happi”, ”ness”]
”Pizza”	[”P”, ”i”, ”z”, ”z”, ”a”]

Tokenizer



```
This note is a sequel to our earlier paper of the same title [4] and describes invariants of
rational homology 3-spheres associated to acyclic orthogonal local systems. Our work is in the
spirit of the Axelrod-Singer papers [1], generalizes some of their results, and furnishes a new
setting for the purely topological implications of their work. Recently, Mullins calculated the
Casson-Walker invariant of the 2-fold cyclic branched cover of an oriented link in S^3 in terms of
its Jones polynomial and its signature, under the assumption that the 2-fold branched cover is a
rational homology 3-sphere. Using elementary principles, we provide a similar calculation for the
general case. In addition, we calculate the LMO invariant of the p-fold branched cover of twisted
knots in S^3 in terms of the Kontsevich integral of the knot.
------------------------------------------------------------------------------------------------------
encoded : [34, 35, 7, 6, 36, 8, 11, 37, 38, 2, 3, 39, 40, 41, 4, 42, 15, 2, 16, 17, 43, 18, 8, 44, 45,
------------------------------------------------------------------------------------------------------
this note is a sequel to our earlier paper of the same title [4] and describes invariants of
rational homology 3-spheres associated to acyclic orthogonal local systems. our work is in the
spirit of the axelrod-singer papers [1], generalizes some of their results, and furnishes a new
setting for the purely topological implications of their work. recently, mullins calculated the
casson-walker invariant of the 2-fold cyclic branched cover of an oriented link in s^3 in terms of
its jones polynomial and its signature, under the assumption that the 2-fold branched cover is a
rational homology 3-sphere. using elementary principles, we provide a similar calculation for the
general case. in addition, we calculate the lmo invariant of the p-fold branched cover of twisted
knots in s^3 in terms of the kontsevich integral of the knot. <pad> <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
```

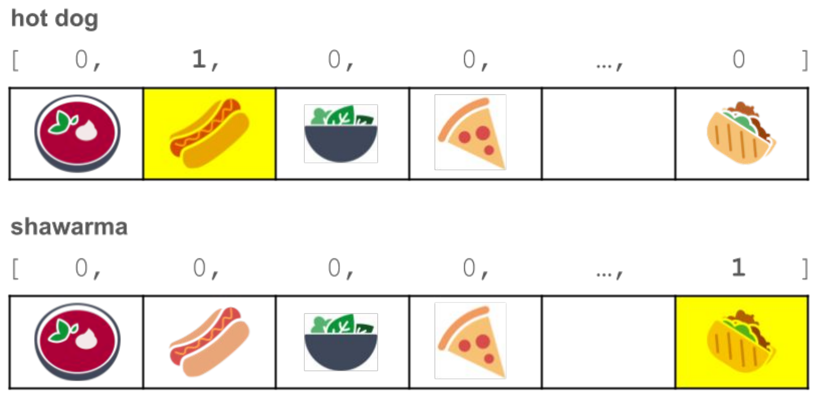
Model Architecture

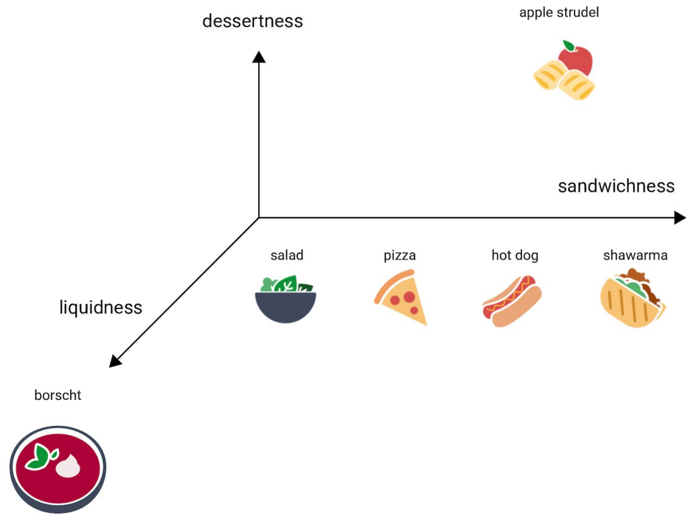
- Seq2Seq neural network architecture is used.
- It transforms one sequence into another.
- Embedding words to dense vectors to understands the context of the input.
- It consists of two main parts: an encoder and a decoder.
- Encoder process the input.
- Decoder generates output.

Model Architecture

```
class Seq2Seq(nn.Module):
  def __init__(self, encoder, decoder, device):
     super().__init__()
     self.encoder = encoder
     self.decoder = decoder
     self.device = device
  def forward(self, src, trg):
     batch_size = src.shape[0]
     trg_len = trg.shape[1]
     vocab_size = self.decoder.fc_out.out_features
     outputs = torch.zeros(batch_size, trg_len, vocab_size).to(self.device)
     encoder_outputs, hidden, cell = self.encoder(src)
     input_token = trg[:, 0].unsqueeze(1)
     for t in range(1, trg_len):
       output, hidden, cell = self.decoder(input_token, hidden, cell)
       outputs[:, t] = output.squeeze(1)
       input_token = trg[:, t].unsqueeze(1)
     return outputs
```

Word Embeddings





Vector representation and Word Embeddings

- Encoders convert tokens into vectors, these vectors capture semantic meanings and properties.
- Each token is mapped to a unique vector, these embeddings are learned during training.
- Word embeddings represent semantic relationships, words with similar meanings have similar vector representations.
- Contextual relationships are captured, helps the model understand how words relate to each other in different contexts.

Encoder-Decoder Pair

- The encoder processes the input sequence.
- It compresses the input into a context vector called hidden state.
- The decoder uses this context vector to generate the output sequence.
- The decoder works one token at a time, predicting each step based on the hidden state.

Encoder

```
import torch.nn as nn
class Encoder(nn.Module):
  def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers):
    super().__init__()
    self.embedding = nn.Embedding(vocab_size, embed_dim)
    self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers)
  def forward(self, input_ids):
    embedded = self.embedding(input_ids)
    outputs, (hidden, cell) = self.lstm(embedded)
    return outputs, hidden, cell
```

Using Encoder class, each layer is created.

- vocab size : number of unique words.
- embed_dim : size of the embedding vector, bigger value capture more semantic information.
- hidden_dim : number of features in hidden state of LSTM, affecting models capacity.
- num_layers : stack LSTM layer.

As result a words turn into vectors with dimension of embed_dim.

Decoder

```
class Decoder(nn.Module):
 def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers):
   super().__init__()
   self.embedding = nn.Embedding(vocab_size, embed_dim)
   self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers, batch_first=True)
   self.fc_out = nn.Linear(hidden_dim, vocab_size)
 def forward(self, input_ids, hidden, cell):
   embedded = self.embedding(input_ids)
   outputs, (hidden, cell) = self.lstm(embedded, (hidden, cell))
   predictions = self.fc_out(outputs)
   return predictions, hidden, cell
```

- Similar structure to encoder.
- Fully connected output layer (fc_out) layer maps the hidden state outputs from LSTM to predictions.
- Using softmax function, probabilities turn into words.

Training

```
vocab_size = simple_tokenizer.get_vocab_size()
embed_dim = 256
hidden_dim = 512
num_layers = 2
encoder = Encoder(vocab_size, embed_dim, hidden_dim, num_layers)
decoder = Decoder(vocab_size, embed_dim, hidden_dim, num_layers)
model = Seq2Seq(encoder, decoder, device).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(ignore_index=simple_tokenizer.word2idx["<pad>"])
num_of_epochs = 15
for epoch in range(num_of_epochs):
    ...
```

- Encoder, decoder and LSTM class is initialized.
- Optimizer is initialized.
- Loss function is initialized (ignoring padding).

Training

```
for epoch in range(num_of_epochs):
  model.train()
  epoch_loss = 0
  for i, batch in enumerate(train_loader):
    inputs = batch["input_ids"].to(device)
    target = batch["labels"].to(device)
    optimizer.zero_grad()
    output = model(inputs, target)
    output_dim = output.shape[-1]
    output = output[:, 1:].reshape(-1, output_dim)
    target = target[:, 1:].reshape(-1)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
    epoch_loss += loss.item()
  avg_loss = epoch_loss / len(train_loader)
       if i % 10 == 0: print(f"Epoch [{epoch+1}/{num_of_epochs}], Loss: {avg_loss:.4f}")
```

- Inputs are fed through the network.
- Using backpropagation gradients are computed.
- Using Adam optimizer, model weights and biases are updated to minimize loss.
- Loss is computed and tracked.

Parameters and Hyperparameters

Vocab Size

- How many unique tokens are present.
- Due to gpu ram limitations balanced value is chosen, considering performance and computation cost.

Embedding Dimension

- Dimensional space words are embedded, larger values capture more details.
- A balanced value between model’s performance and computation cost.

Hidden Dimension

- The size of the hidden state in LSTM, can be considered as memory of LSTM.
- A balanced value between model’s performance and computation cost.

Number of Layers

- Represent the layers in LSTM.

Optimizer

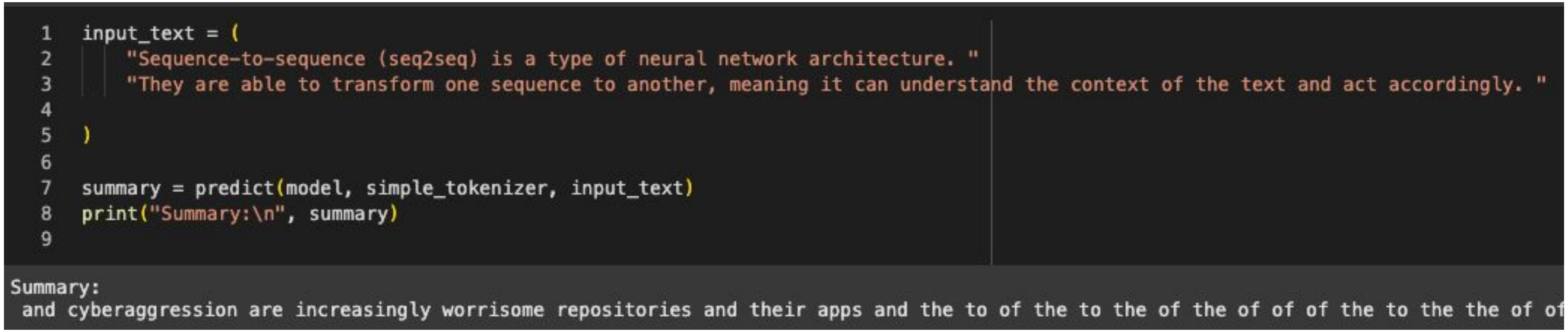
- An optimizer adjusts the model’s weights and biases to minimize the loss function during training.
- Adam optimizer is adaptive and converge fast with less hyperparameters.

Loss Function

- A loss function measures how wrong your model’s predictions are.
- Considering summarization is type of classification, CrossEntrophy is chosen.

Learning Rate

- Control size of each step for updating weights. Determines the speed of the model.
- 0.001 is common value for adam optimizer.



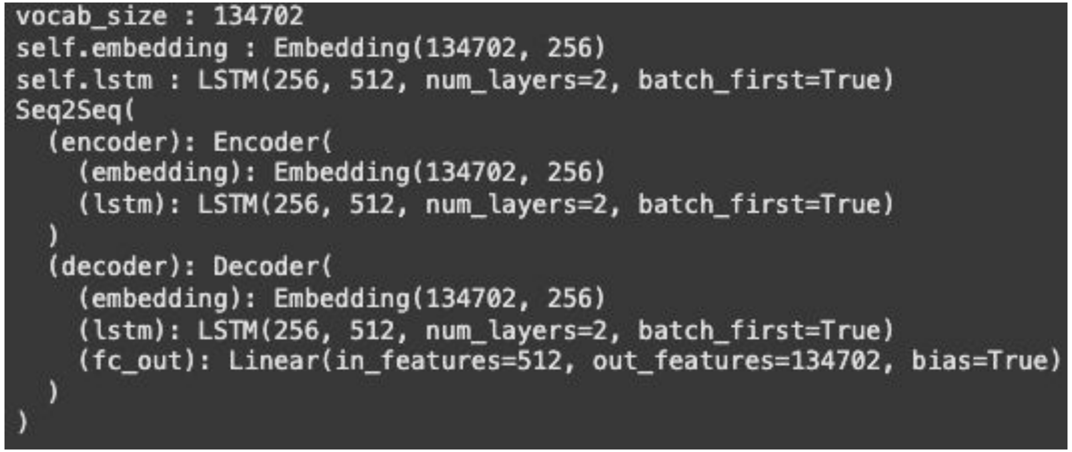
```
input_text = (
    "Sequence-to-sequence (seq2seq) is a type of neural network architecture. "
    "They are able to transform one sequence to another, meaning it can understand the context of the text and act accordingly. "

)

summary = predict(model, simple_tokenizer, input_text)
print("Summary:\n", summary)

```

```
Summary:
 and cyberaggression are increasingly worrisome repositories and their apps and the to of the to the of the of of of the to the the of of
```



```
vocab_size : 134702
self.embedding : Embedding(134702, 256)
self.lstm : LSTM(256, 512, num_layers=2, batch_first=True)
Seq2Seq(
  (encoder): Encoder(
    (embedding): Embedding(134702, 256)
    (lstm): LSTM(256, 512, num_layers=2, batch_first=True)
  )
  (decoder): Decoder(
    (embedding): Embedding(134702, 256)
    (lstm): LSTM(256, 512, num_layers=2, batch_first=True)
    (fc_out): Linear(in_features=512, out_features=134702, bias=True)
  )
)
```

Troubleshooting and Tuning

No Preprocessing of Input Text

- 134,702 unique tokens.

Assuming First Model Would Succeed

- Trial-and-Error Hyperparameter Changes without looking at results first.
    - Learning Rate
    - Hidden Dimension
    - Embedding Dimension

Resulting

- Multiple inconsistent models
- No clear improvement
- Output consist of repetitive tokens like 'of', 'the', 'to'

Troubleshooting and Tuning

Bottleneck Architecture

- Bottleneck structure is created.
    - Embedding dimension = 256
    - Hidden dimension = 128
- To reduce overfitting and improve generalization.

Limiting Vocabulary Size

- A vocabulary limit(25000) is added.
- According to most used words.
- Reducing the unnecessary complexity.

Validation Loss Check

- Early stopping added to stop training when validation loss stops improving.
- Helps select the best possible model without overtraining.

Graphs

- Added graph visualization tools.
- Graph-based approach is applied to better understand model's "black-box" nature

Conclusion

- The goal was to build a text summarization model using a seq2seq LSTM-based encoder-decoder architecture, mostly implemented from scratch.

- Bottleneck structure was designed to reduce overfitting and improve generalization.

- A vocabulary size limit was applied to eliminate unnecessary complexity.

- Early stopping was used to prevent overtraining based on validation loss.

- Graph visualization tools were added to better understand model performance and structure.

- Despite upgrading to the paid version of Google Colab to address resource limitations, challenges persisted, and due to time constraints, the model was unable to generate meaningful summaries.

- Key NLP fundamentals like tokenization, embeddings, semantic meaning, and sequence modeling were implemented.