Module 4: Recurrent Neural Network

To write legible answers you will need to be familiar with both Markdown and Latex

Before you turn this problem in, make sure everything runs as expected. First, restart the kernel (in the menubar, select Kernel→→Restart) and then run all cells (in the menubar, select Cell→→Run All).

Make sure you fill in any place that says "YOUR CODE HERE" or "YOUR ANSWER HERE", as well as your name below:

In [ ]:

```
NAME = ""
STUDENT_ID = ""
```

Some imports we will need:

In [ ]:

```
import tensorflow as tf
from tensorflow.keras.datasets import reuters
import matplotlib.pyplot as plt
import numpy as np
import string
import textwrap
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Embedding, Dense, Dropout, Input, LSTM, GRU, Bi
from tensorflow.keras.models import Model
```

We will be using the Reuters newswire classification dataset, which has text paired with 46 topics as labels. You can see what these labels represent here.

Let's load the data:

In [ ]:

```
(X_train, y_train), (_, _) = reuters.load_data()
```

In [ ]:

```
# https://stackoverflow.com/questions/42821330/restore-original-text-from-keras-s-i
# Needed to encode our own reviews later

word_dict = reuters.get_word_index()
word_dict = {k:(v+3) for k,v in word_dict.items()}
word_dict["<PAD>"] = 0
word_dict["<START>"] = 1
word_dict["<UNK>"] = 2
word_dict["<UNUSED>"] = 3

vocab_size = len(word_dict.keys())
```

In [ ]:

```
# Needed to decode training data into readable text

inverse_word_dict = {value:key for key,value in word_dict.items()}
```

In [ ]:

```
X_train = np.array(X_train)
X_train = pad_sequences(X_train)

max_sequence_len = X_train[0].shape[0]
print('Padded to longest sequence length: ', max_sequence_len)
```

In [ ]:

```
y_train = to_categorical(y_train, 46)
y_train = np.array(y_train)
```

In [ ]:

```
print('Number of words in vocabulary: ', vocab_size)
```

In [ ]:

```
def encode_text(text, word_dict, maxlen):
  encoded_text = []
  for raw_word in text.split(' '):
    word = raw_word.strip().strip(string.punctuation).lower()
    if word is '' or word is '\n':
      continue
    try:
      encoded_text.append(word_dict[word])
    except KeyError as e:
      # raise KeyError(f'{e} not in word dictionary, text not encoded.')
      continue
  return pad_sequences(np.array(encoded_text).reshape(1,-1), maxlen=maxlen)

def decode_text(encoded_text, inverse_word_dict):
  sentence = []
  for encoded_word in encoded_text:
    if encoded_word == 0:
      continue
    sentence.append(inverse_word_dict[encoded_word])
  w = textwrap.TextWrapper(width=120,break_long_words=False,replace_whitespace=Fals
  return '\n'.join(w.wrap(' '.join(sentence)))
```

Let's take a look at an article in our training data:

In [ ]:

```
idx = 144

print(decode_text(X_train[idx], inverse_word_dict), end='\n\n')

print('Topic: ', y_train[idx])
```

Question 1

Create a model using an RNN layer (LSTM or GRU, unidirectional or bidirectional) to achieve at least 60% validation accuracy in 10 epochs or less:

In [ ]:
```
### YOUR CODE HERE ###

# Reminder: We have 46 categories. What final activation do we need to use?
```

Compile your model and display the summary:

In [ ]:
```
loss = ### YOUR CODE HERE ###

opt = ### YOUR CODE HERE ###

metrics = ### YOUR CODE HERE ###

reuters_model.compile(loss=loss,
              optimizer=opt,
              metrics=metrics)

reuters_model.summary()
```

Train your model:

In [ ]:
```
batchsize = ### YOUR CODE HERE ###

history = reuters_model.fit(X_train, y_train, batch_size=batchsize, epochs=10, vali
```

Plot the training and validation losses and accuracies:

In [ ]:
```
### YOUR CODE HERE ###
```

Question 2

We've seen both LSTM and GRU cells as building blocks for RNNs.

Here is a reminder of each of their corresponding architectures:

i) What are the major differences between each?

ii) What are the major advantages of each?

iii) What are the major disadvantages of each?

YOUR ANSWERS HERE