Problem 6 - Sentiment Analysis using recurrent models 20 points

In this problem, you will compare the performance of RNN, LSTM, GRU and BiLSTM for the task of sentiment analysis. You'll use the IMDB sentiment analysis dataset for this task - Sentiment Analysis of IMDB Movie Reviews. For each model, use a single cell, and keep the number of units fixed to 256. Train each model for 10 epochs using the Adam optimizer, batch size of 256, and a learning rate of 0.01.

In [ ]:
```
from spacy.tokenizer import Tokenizer
from spacy.lang.en import English

def process_tokens(text):
    """
    function to process tokens, replace any unwanted chars
    """
    preprocessed_text = text.lower().replace(",", "").replace(".", "").replac
    preprocessed_text = ''.join([i for i in preprocessed_text if not preproce
    return preprocessed_text

def preprocessing(data):
    """
    preprocessing data to list of tokens
    """
    nlp = English()
    tokenizer = Tokenizer(nlp.vocab)
    preprocessed_data = []
    for sentence in data:
        sentence = process_tokens(sentence)
        tokens = tokenizer(sentence)
        tlist = []
        for token in tokens:
            tlist.append(str(token))
        preprocessed_data.append(tlist)
    return preprocessed_data
```

1. Import the dataset and convert it into vector form using Bag of Words technique.(2)

In [ ]:
```
!gdown 1o5Hu9mOZsXxhIbPEov80LsxxCs5A2_7W -O './imdb.csv'
```

```
Downloading...
From: https://drive.google.com/uc?id=1o5Hu9mOZsXxhIbPEov80LsxxCs5A2_7W
To: /content/imdb.csv
100% 66.2M/66.2M [00:02<00:00, 32.8MB/s]
```

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

df = pd.read_csv("imdb.csv", usecols=["review", "sentiment"], encoding='latin
## 1 - positive, 0 - negative
df.sentiment = (df.sentiment == "positive").astype("int")
df.head()

val_size = int(df.shape[0] * 0.15)
test_size = int(df.shape[0] * 0.15)


def train_val_test_split(df=None, train_percent=0.7, test_percent=0.15, val_p
```

```
    df = df.sample(frac=1)
    train_df = df[: int(len(df)*train_percent)]
    test_df = df[int(len(df)*train_percent)+1 : int(len(df)*(train_percent+test_
    val_df = df[int(len(df)*(train_percent + test_percent))+1 : ]
    return train_df, test_df, val_df

train_df, test_df, val_df = train_val_test_split(df, 0.7, 0.15, 0.15)
train_labels, train_texts = train_df.values[:,1], train_df.values[:,0]
val_labels, val_texts = val_df.values[:,1], val_df.values[:,0]
test_labels, test_texts = test_df.values[:,1], test_df.values[:,0]
print(len(train_df), len(test_df), len(val_df))
print(len(train_texts), len(train_labels), len(val_df))


train_data = preprocessing(train_texts)
val_data = preprocessing(val_texts)
test_data = preprocessing(test_texts)
```

```
35000 7499 7499
35000 35000 7499
```

In [ ]:

```
import numpy as np
import itertools

## Creating a vectorizer to vectorize text and create matrix of features
## Bag of words technique
class Vectorizer():
    def __init__(self, max_features):
        self.max_features = max_features
        self.vocab_list = None
        self.token_to_index = None

    def fit(self, dataset):
        word_dict = {}
        for sentence in dataset:
            for token in sentence:
                if token not in word_dict:
                    word_dict[token] = 1
                else:
                    word_dict[token] += 1
        word_dict = dict(sorted(word_dict.items(), key=lambda item: item[1], 
        end_to_slice = min(len(word_dict), self.max_features)
        word_dict = dict(itertools.islice(word_dict.items(), end_to_slice))
        self.vocab_list = list(word_dict.keys())
        self.token_to_index = {}
        counter = 0
        for token in self.vocab_list:
            self.token_to_index[token] = counter
            counter += 1


    def transform(self, dataset):
        data_matrix = np.zeros((len(dataset), len(self.vocab_list)))
        for i, sentence in enumerate(dataset):
            for token in sentence:
                if token in self.token_to_index:
                    data_matrix[i, self.token_to_index[token]] += 1
        return data_matrix

## max features - top k words to consider only
max_features = 2000

vectorizer = Vectorizer(max_features=max_features)
vectorizer.fit(train_data)
```

```
## Checking if the len of vocab = k
X_train = vectorizer.transform(train_data)
X_val = vectorizer.transform(val_data)
X_test = vectorizer.transform(test_data)

y_train = np.array(train_labels)
y_val = np.array(val_labels)
y_test = np.array(test_labels)

vocab = vectorizer.vocab_list
```

In [ ]:
```
vocab
```

1. Define an RNN model and train it on the dataset (4)

In [ ]:
```
from tensorflow.keras.utils import to_categorical

y_train = y_train.astype('int')
y_val = y_val.astype('int')
y_test = y_test.astype('int')

y_train = to_categorical(y_train, 2)
y_test = to_categorical(y_test, 2)
y_val = to_categorical(y_val, 2)

X_train = X_train.reshape(-1, 1, X_train.shape[1])
X_val = X_val.reshape(-1, 1, X_val.shape[1])
X_test = X_test.reshape(-1, 1, X_test.shape[1])

y_train = y_train.reshape(-1, 2)
y_val = y_val.reshape(-1, 2)
y_test = y_test.reshape(-1, 2)

print(f'X_train.shape: {X_train.shape}, y_train.shape: {y_train.shape}')
```

```
X_train.shape: (35000, 1, 2000), y_train.shape: (35000, 2)
```

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import SimpleRNN, Dropout
from tensorflow.keras.optimizers import Adam

model = None
model = Sequential()
model.add(SimpleRNN(128, input_shape=(1, max_features)))
model.add(Dense(2, activation='softmax'))

optimizer = Adam()
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
              metrics=['accuracy'])
print(model.summary())
history = model.fit(X_train, y_train,
          batch_size=256,
          validation_data=(X_val, y_val),
          epochs=10)
print(history.history.keys())
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_5 (SimpleRNN)    (None, 128)               272512
```

```
 dense_10 (Dense)            (None, 2)                 258

=================================================================
Total params: 272,770
Trainable params: 272,770
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
137/137 [==============================] - 2s 10ms/step - loss: 0.3651 - accur
acy: 0.8378 - val_loss: 0.2882 - val_accuracy: 0.8813
Epoch 2/10
137/137 [==============================] - 1s 8ms/step - loss: 0.2696 - accura
cy: 0.8896 - val_loss: 0.2863 - val_accuracy: 0.8789
Epoch 3/10
137/137 [==============================] - 1s 8ms/step - loss: 0.2510 - accura
cy: 0.8963 - val_loss: 0.2906 - val_accuracy: 0.8797
Epoch 4/10
137/137 [==============================] - 1s 8ms/step - loss: 0.2383 - accura
cy: 0.9038 - val_loss: 0.2944 - val_accuracy: 0.8767
Epoch 5/10
137/137 [==============================] - 1s 8ms/step - loss: 0.2198 - accura
cy: 0.9124 - val_loss: 0.2981 - val_accuracy: 0.8756
Epoch 6/10
137/137 [==============================] - 1s 8ms/step - loss: 0.2096 - accura
cy: 0.9166 - val_loss: 0.3001 - val_accuracy: 0.8767
Epoch 7/10
137/137 [==============================] - 1s 8ms/step - loss: 0.1922 - accura
cy: 0.9253 - val_loss: 0.3069 - val_accuracy: 0.8734
Epoch 8/10
137/137 [==============================] - 1s 8ms/step - loss: 0.1749 - accura
cy: 0.9343 - val_loss: 0.3147 - val_accuracy: 0.8718
Epoch 9/10
137/137 [==============================] - 1s 8ms/step - loss: 0.1570 - accura
cy: 0.9438 - val_loss: 0.3205 - val_accuracy: 0.8748
Epoch 10/10
137/137 [==============================] - 1s 8ms/step - loss: 0.1381 - accura
cy: 0.9518 - val_loss: 0.3276 - val_accuracy: 0.8734
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:

```
score, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score)
print('Test accuracy:', acc)
```

```
Test loss: 0.34322747588157654
Test accuracy: 0.8715828657150269
```

1. Define a LSTM model and train it on the dataset (4)

In [ ]:

```
from tensorflow.keras.layers import LSTM

model = None
model = Sequential()
model.add(LSTM(128, input_shape=(1, max_features)))
model.add(Dense(2, activation='softmax'))

optimizer = Adam()
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
              metrics=['accuracy'])
print(model.summary())
history = model.fit(X_train, y_train,
          batch_size=256,
          validation_data=(X_val, y_val),
          epochs=10)
print(history.history.keys())
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_5 (LSTM)               (None, 128)               1090048

 dense_11 (Dense)            (None, 2)                 258

=================================================================
Total params: 1,090,306
Trainable params: 1,090,306
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
137/137 [==============================] - 5s 28ms/step - loss: 0.3614 - accur
acy: 0.8454 - val_loss: 0.2905 - val_accuracy: 0.8792
Epoch 2/10
137/137 [==============================] - 3s 23ms/step - loss: 0.2613 - accur
acy: 0.8927 - val_loss: 0.2959 - val_accuracy: 0.8767
Epoch 3/10
137/137 [==============================] - 3s 23ms/step - loss: 0.2235 - accur
acy: 0.9099 - val_loss: 0.2928 - val_accuracy: 0.8765
Epoch 4/10
137/137 [==============================] - 3s 22ms/step - loss: 0.1909 - accur
acy: 0.9258 - val_loss: 0.2933 - val_accuracy: 0.8788
Epoch 5/10
137/137 [==============================] - 3s 22ms/step - loss: 0.1546 - accur
acy: 0.9432 - val_loss: 0.3123 - val_accuracy: 0.8744
Epoch 6/10
137/137 [==============================] - 3s 22ms/step - loss: 0.1186 - accur
acy: 0.9602 - val_loss: 0.3279 - val_accuracy: 0.8765
Epoch 7/10
137/137 [==============================] - 3s 22ms/step - loss: 0.0852 - accur
acy: 0.9760 - val_loss: 0.3423 - val_accuracy: 0.8767
Epoch 8/10
137/137 [==============================] - 3s 22ms/step - loss: 0.0593 - accur
acy: 0.9860 - val_loss: 0.3640 - val_accuracy: 0.8751
Epoch 9/10
137/137 [==============================] - 3s 22ms/step - loss: 0.0412 - accur
acy: 0.9923 - val_loss: 0.3821 - val_accuracy: 0.8772
Epoch 10/10
137/137 [==============================] - 3s 22ms/step - loss: 0.0279 - accur
acy: 0.9957 - val_loss: 0.4091 - val_accuracy: 0.8761
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:

```
score, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score)
print('Test accuracy:', acc)
```

```
Test loss: 0.4387073218822479
Test accuracy: 0.8751833438873291
```

1. Define a GRU model and train it on the dataset (4)

In [ ]:

```
from tensorflow.keras.layers import GRU

model = None
model = Sequential()
model.add(GRU(128, input_shape=(1, max_features)))
model.add(Dense(2, activation='softmax'))

optimizer = Adam()
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
              metrics=['accuracy'])
print(model.summary())
history = model.fit(X_train, y_train,
```

```
                batch_size=256,
                validation_data=(X_val, y_val),
                epochs=10)
print(history.history.keys())

score, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score)
print('Test accuracy:', acc)
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 gru_1 (GRU)                 (None, 128)               817920    
                                                                 
 dense_12 (Dense)            (None, 2)                 258       
                                                                 
=================================================================
Total params: 818,178
Trainable params: 818,178
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
137/137 [==============================] - 5s 20ms/step - loss: 0.3542 - accur
acy: 0.8476 - val_loss: 0.2924 - val_accuracy: 0.8795
Epoch 2/10
137/137 [==============================] - 2s 17ms/step - loss: 0.2587 - accur
acy: 0.8946 - val_loss: 0.3112 - val_accuracy: 0.8692
Epoch 3/10
137/137 [==============================] - 2s 18ms/step - loss: 0.2311 - accur
acy: 0.9069 - val_loss: 0.2892 - val_accuracy: 0.8779
Epoch 4/10
137/137 [==============================] - 2s 17ms/step - loss: 0.2020 - accur
acy: 0.9200 - val_loss: 0.2975 - val_accuracy: 0.8771
Epoch 5/10
137/137 [==============================] - 2s 17ms/step - loss: 0.1712 - accur
acy: 0.9342 - val_loss: 0.3044 - val_accuracy: 0.8771
Epoch 6/10
137/137 [==============================] - 2s 18ms/step - loss: 0.1402 - accur
acy: 0.9495 - val_loss: 0.3187 - val_accuracy: 0.8771
Epoch 7/10
137/137 [==============================] - 2s 18ms/step - loss: 0.1102 - accur
acy: 0.9629 - val_loss: 0.3327 - val_accuracy: 0.8746
Epoch 8/10
137/137 [==============================] - 2s 18ms/step - loss: 0.0817 - accur
acy: 0.9753 - val_loss: 0.3558 - val_accuracy: 0.8755
Epoch 9/10
137/137 [==============================] - 2s 18ms/step - loss: 0.0598 - accur
acy: 0.9840 - val_loss: 0.3772 - val_accuracy: 0.8714
Epoch 10/10
137/137 [==============================] - 3s 21ms/step - loss: 0.0422 - accur
acy: 0.9913 - val_loss: 0.4027 - val_accuracy: 0.8738
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
Test loss: 0.44197070598602295
Test accuracy: 0.8702493906021118
```

In [ ]:

```
# check predictions
from tensorflow.keras.backend import argmax

y_pred = model.predict(X_test)
for i in range(5):
  print(f'Label predicted: {argmax(y_pred[i]).numpy()}, Actual label: {argmax
  print(f'text: {test_texts[i]}')
```

```
235/235 [==============================] - 1s 2ms/step
Label predicted: 1, Actual label: 1
text: Love Jones cleverly portrays young African-American men and women in a c
lear, positive, realistic sense. I feel that all of the actors and actresses w
```

ere magnificent and really did a great job at capturing the mood. Nia Long and Larenz Tate worked well together and I hope to see more work from the two of them. As a matter of fact all of the actors/actresses did such a fine job it would be great to see another romantic-comedy from them. This movie can be compared to most any well-written, romantic comedy. If you have not seen this movie already I strongly recommend that you do, it can definitely give you another perspective on life and love.
Label predicted: 0, Actual label: 0
text: This version is very painful to watch. All of the acting is very stilted but especially that of Norma Shearer who is still acting as though she were in a silent movie instead of a talkie. Check out the 1937 version with Joan Crawford, Robert Montgomery and William Powell which is much more entertaining.
Label predicted: 0, Actual label: 0
text: An EXTREMELY fast paced,exhilarating, interesting, detail rich book. Its a huge shame that the film had none of these qualities. not only was Tom Hanks' mild mannered portrayal or Robert Langdon Laughable, but the name changes to key characters, huge deviances from the original story line, and poor Irish/Italian accent from Carmalengo Played by Ewan Mcgregor, made for the worst book to film EVER.

As a huge fan of A&D the book, i had high hopes for a more lavish, true to book detailed movie, where it would start and finish just as the book did - leaving me wanting more.

All the film really did was depress me within the 1st 10 minutes.

what was impressive was how the... sorry! i couldn't even finish that sentence without laughing.

in short - Vittoria was the token hottie, a very second to Audrey Tatou and there were some very nice Alfa Romeos.

i would recommend reading the book to understand that, if Ron Howard must insist on making ANOTHER book to film, i would be happy saving my Â£6.40 for a KFC zinger meal and some chicken wings - far more entertaining and deeply more satisfying!
Label predicted: 1, Actual label: 1
text: Beforehand Notification: I'm sure someone is going to accuse me of playing the race card here, but when I saw the preview for this movie, I was thinking "Finally!" I have yet to see one movie about popular African-influenced dance (be it popular hip hop moves, breaking, or stepping) where the main character was a Black woman. I've seen an excessive amount of movies where a non-Black woman who knew nothing about hip hop comes fresh to the hood and does a mediocre job of it (Breakin, Breakin 2, Save the Last Dance, Step Up), but the Black women in the film are almost nonexistent. That always bothered me considering so much of hip hop, African-influenced dance, and breaking was with Blacks and Latinos in massive amounts in these particular sets and it wasn't always men who performed it, so I felt this movie has been a long time coming. However, the race does not make the film, so I also wanted it to carry a believable plot; the dancing be entertaining; and interesting to watch.

Pros: I really enjoyed this film bringing Jamaican culture. I can't recall ever seeing a popular, mainstream film where all the main characters were Jamaican; had believable accents; and weren't stereotypical with the beanies. The steppers, family, friends, and even the "thugs" were all really intelligent, realistic people who were trying to love, live, and survive in the neighborhood they lived in by doing something positive. Even when the audience was made aware that the main character's sister chose an alternate lifestyle, it still didn't make the plot stereotypical. I was satisfied with the way it was portrayed. I LOVED the stepping; the romantic flirty relationship going on between two steppers; the trials that the main character's parents were going through; and how she dealt with coming back to her old neighborhood and dealing with Crabs in a Barrel. I respected that she was so intelligent and active at the same time, and so many other sistas in the film were handling themselves in the step world. They were all just as excellent as the fellas. I don't see that in too many movies nowadays, at least not those that would be considered Black films.

Cons: I'm not quite sure why the directors or whoever put the movie together did this, but I question whether they've been to real step shows. Whenever the steppers got ready to perform, some hip hop song would play in place of the steppers' hand/feet beats. At a real step show, there is zero need for music, other than to maybe entertain the crowds in between groups. And then when hip hop songs were played, sometimes the beat to the song was off to the beat of the steppers' hands and feet. It was awkward. I was more impressed with the stepping in this movie versus "Stomp the Yard" (another great stepping movie) because the women got to represent as fierce as the guys (in "Stomp the Yard," Meagan Good got all of a few seconds of some prissy twirl and hair flip and the (Deltas?) let out a chant and a few steps and were cut immediately). Even when there were very small scenes, the ladies tore it up, especially in the auto sho

```
p, and it was without all that music to drown out their physical music. I know
soundtracks have to be sold, but the movie folks could've played the music in
other parts of the film.<br /><br />I'm not a Keyshia Cole fan, so every time
I saw her, all I kept thinking was "Is it written in the script for her to con
stantly put her hand on her hip when she talks?" She looked uncomfortable on s
creen to me. I thought they should've used a host like Free or Rocsi instead.
Deray Davis was funny as usual though. Also, I groaned when I found out that t
he movie was supposed to be in the ghetto, like stepping couldn't possibly hap
pen anywhere else. Hollywood, as usual. However, only a couple of people were
portrayed as excessively ignorant due to their neighborhood and losers, which
mainstream movies tend to do.<br /><br />I would've given this movie five star
s, but the music playing killed it for me. I definitely plan to buy it when it
comes out and hopefully the bonus scenes will include the actual step shows wi
thout all the songs.
Label predicted: 1, Actual label: 1
text: "Atlantis: The Lost Empire" was everything the previews indicated it wou
ld be. It is not often you find that. Most of the time, the previews show only
the best parts and then the rest of the movie is terrible. Not so with this on
e. I was pleased with the original plot, even though the sub-plots were not. T
he animation was not break through like "Shrek" but it was good, none the les
s. The plot and the story line were well presented and there were only a few s
low spots in them. This keeps you interested. I found myself enjoying this on
e. "Atlantis" gets and keeps your attention. You also have to think a little b
it, but not too much. Once you think about it a little, you can figure out wha
t needs to happen but you really don't know for sure how it is going to happe
n.<br /><br />The casting was also good. Michael J. Fox, as Milo was an excell
ent choice. His personality fits nicely. The gruff natured Commander Rourke wa
s also well chosen with James Garner. His character reminded me of his perform
ance in "Maverick" which I also liked. I really liked the casting of Claudia C
hristian as Helga Sinclair. Her ability to play a no nonsense personality make
s the film more interesting. It's just too bad she is a villain.<br /><br />Ov
er all, definitely worth you while (8 out of 10).
```

1. Define a BiLSTM model and train it on the dataset (4)

In [ ]:
```python
from tensorflow.keras.layers import Bidirectional

model = None
model = Sequential()
model.add(Bidirectional(LSTM(256, return_sequences=True), input_shape=(1, max
model.add(Bidirectional(LSTM(256)))
model.add(Dense(2, activation='softmax'))

optimizer = Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
              metrics=['accuracy'])
print(model.summary())
history = model.fit(X_train, y_train,
          batch_size=256,
          validation_data=(X_val, y_val),
          epochs=10)
print(history.history.keys())
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 1, 512)           4622336
 l)

 bidirectional_1 (Bidirectio  (None, 512)              1574912
 nal)

 dense_7 (Dense)             (None, 2)                 1026

=================================================================
Total params: 6,198,274
```

```
Trainable params: 6,198,274
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
137/137 [==============================] - 24s 140ms/step - loss: 0.5976 - acc
uracy: 0.7325 - val_loss: 0.4370 - val_accuracy: 0.8644
Epoch 2/10
137/137 [==============================] - 18s 129ms/step - loss: 0.3787 - acc
uracy: 0.8784 - val_loss: 0.3599 - val_accuracy: 0.8684
Epoch 3/10
137/137 [==============================] - 19s 137ms/step - loss: 0.2935 - acc
uracy: 0.8982 - val_loss: 0.3275 - val_accuracy: 0.8661
Epoch 4/10
137/137 [==============================] - 18s 134ms/step - loss: 0.2489 - acc
uracy: 0.9126 - val_loss: 0.3137 - val_accuracy: 0.8765
Epoch 5/10
137/137 [==============================] - 17s 128ms/step - loss: 0.2046 - acc
uracy: 0.9326 - val_loss: 0.3347 - val_accuracy: 0.8726
Epoch 6/10
137/137 [==============================] - 18s 128ms/step - loss: 0.1666 - acc
uracy: 0.9464 - val_loss: 0.3510 - val_accuracy: 0.8617
Epoch 7/10
137/137 [==============================] - 18s 128ms/step - loss: 0.1275 - acc
uracy: 0.9632 - val_loss: 0.3588 - val_accuracy: 0.8741
Epoch 8/10
137/137 [==============================] - 18s 128ms/step - loss: 0.0937 - acc
uracy: 0.9760 - val_loss: 0.3863 - val_accuracy: 0.8705
Epoch 9/10
137/137 [==============================] - 18s 128ms/step - loss: 0.0726 - acc
uracy: 0.9834 - val_loss: 0.4315 - val_accuracy: 0.8618
Epoch 10/10
137/137 [==============================] - 18s 131ms/step - loss: 0.0580 - acc
uracy: 0.9872 - val_loss: 0.4358 - val_accuracy: 0.8681
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:

```python
score, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score)
print('Test accuracy:', acc)
```

```
Test loss: 0.3505217730998993
Test accuracy: 0.8695825934410095
```

1. Compare the performance of all the models. In which case do you get the best accuracy? (2)

Simple RNN: Test loss: 0.34322747588157654 Test accuracy: 0.8715828657150269

LSTM: Test loss: 0.4387073218822479 Test accuracy: 0.8751833438873291

GRU: Test loss: 0.44197070598602295 Test accuracy: 0.8702493906021118

Bi LSTM Test loss: 0.3505217730998993 Test accuracy: 0.8695825934410095

Best Accuracy was given by the simple RNN model.