In [0]:
```
## Assignment 3 Implementation
```

Part 1 using RNN with TBTT

In [0]:
```
import os
import re

import nltk
import numpy as np
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Embedding, Dense, SimpleRNN
from keras.preprocessing.text import Tokenizer

base_path = os.path.abspath('English Literature.txt')

with open('/content/English Literature.txt', 'r') as f:
  sample= f.read()


def format_data(input_string):

    clean_string = re.sub(r'\((\d+)\)', r'', input_string)

    clean_string = re.sub(r'\s\s', ' ', clean_string)
    return clean_string


formatted_string1 = format_data(sample)
print(len(formatted_string1))

regular_exp = nltk.RegexpTokenizer(r"\w+")
formatted_string = regular_exp.tokenize(formatted_string1)
tokenizer = Tokenizer()
tokenizer.fit_on_texts([formatted_string])

encoded = tokenizer.texts_to_sequences([formatted_string])[0]
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

# create word -> word sequences
sequences = []
for i in range(1, len(encoded)):
    sequence = encoded[i - 1:i + 1]
    sequences.append(sequence)
print('Total Sequences: %d' % len(sequences))
```

```
sequences = np.array(sequences)
X, y = sequences[:, 0], sequences[:, 1]

print(len(X), len(y))

# one hot encode outputs
y = to_categorical(y, num_classes=vocab_size)


#Model Architecture
embedding_size = 100
units = 500
model = Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=1))
model.add(SimpleRNN(units=units, input_shape=(1, 100), activation='sigmo
model.add(Dense(vocab_size, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics
print(model.summary())
model.fit(X, y, epochs=10, verbose=1)
model.save('my_simple_model.h5')
```

```
1108158
Vocabulary Size: 11457
Total Sequences: 208529
208529 208529
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1, 100)            1145700
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 500)               300500
_________________________________________________________________
dense_2 (Dense)              (None, 11457)             5739957
=================================================================
Total params: 7,186,157
Trainable params: 7,186,157
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
208529/208529 [==============================] - 514s 2ms/step - loss:
6.7969 - acc: 0.0460
Epoch 2/10
208529/208529 [==============================] - 524s 3ms/step - loss:
6.2229 - acc: 0.0799
Epoch 3/10
208529/208529 [==============================] - 527s 3ms/step - loss:
5.9780 - acc: 0.0944
Epoch 4/10
```

```
208529/208529 [==============================] - 525s 3ms/step - loss:
5.8018 - acc: 0.1021
Epoch 5/10
208529/208529 [==============================] - 537s 3ms/step - loss:
5.6542 - acc: 0.1091
Epoch 6/10
208529/208529 [==============================] - 526s 3ms/step - loss:
5.5255 - acc: 0.1148
Epoch 7/10
208529/208529 [==============================] - 522s 3ms/step - loss:
5.4044 - acc: 0.1186
Epoch 8/10
208529/208529 [==============================] - 529s 3ms/step - loss:
5.2867 - acc: 0.1217
Epoch 9/10
208529/208529 [==============================] - 524s 3ms/step - loss:
5.1770 - acc: 0.1235
Epoch 10/10
208529/208529 [==============================] - 522s 3ms/step - loss:
5.0774 - acc: 0.1240
```

Part 2 - BTT

In [0]:

```
import os
import re
import nltk
import numpy as np
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Embedding, LSTM, Dense, SimpleRNN, GRU, Masking
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
nltk.download('punkt')

base_path = os.path.abspath('English Literature.txt')

with open('/content/English Literature.txt', 'r') as f:
  sample = f.read()


def format_data(input_string):
    clean_string = re.sub(r'\((\d+)\)', r'', input_string)

    clean_string = re.sub(r'\s\s', ' ', clean_string)
    return clean_string


formatted_string = format_data(sample)
```

```
# integer encode text
regular_exp = nltk.RegexpTokenizer(r"\w+")
sent = regular_exp.tokenize(formatted_string)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sent)
encoded = tokenizer.texts_to_sequences([sent])[0]

sentence_tokenize = nltk.tokenize.sent_tokenize(formatted_string)
print(sentence_tokenize)

sent_len = 16
i = 0
input_X = []
output_y = []

X = []
y1 = []
padded_sent = []
for i in range(len(sentence_tokenize)):
    rem_exp = nltk.RegexpTokenizer(r"\w+")
    sentences = rem_exp.tokenize(sentence_tokenize[i])
    encoded = tokenizer.texts_to_sequences([sentences])[0]
    if (len(encoded) < sent_len) or (len(encoded) > sent_len):
        padded_sent = pad_sequences([encoded], maxlen=sent_len, dtype='i
                                    value=0.0)
    input_X = padded_sent[0][0:(sent_len - 1)]
    output_y = padded_sent[0][1:sent_len]
    X.append(input_X)
    y1.append(output_y)


# integer encode text
X = np.array(X)
num_y = np.array(y1)
print(X, num_y)
max_words = 10


vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

# one hot encode outputs
y = to_categorical(num_y, num_classes=vocab_size)


# define the network architecture: a embedding followed by LSTM
embedding_size = 100
rnn_size = 50
model1 = Sequential()
model1.add(Embedding(vocab_size, embedding_size, input_length=15))
model1.add(Masking(mask_value=0.0))
```

```
model1.add(Masking(mask_value 0.0));
model1.add(SimpleRNN(rnn_size, return_sequences=True))
model1.add(TimeDistributed(Dense(vocab_size, activation='softmax')))
model1.compile(loss='categorical_crossentropy', optimizer='adam', metric
print(model1.summary())

# fit network
model1.fit(X, y, epochs=10, verbose=1, batch_size=1)
model1.save("my_rnn_model.h5")
```

Explanation for Part 2

Epoch 1/10

- 211s - loss: 4.5670 - accuracy: 0.3560

Epoch 2/10

- 218s - loss: 4.1274 - accuracy: 0.3811

Epoch 3/10

- 218s - loss: 3.9648 - accuracy: 0.3925

Epoch 4/10

- 228s - loss: 3.8438 - accuracy: 0.4017

Epoch 5/10

- 219s - loss: 3.7456 - accuracy: 0.4108

Epoch 6/10

- 224s - loss: 3.6586 - accuracy: 0.4193

Epoch 7/10

- 217s - loss: 3.5825 - accuracy: 0.4279

Epoch 8/10

- 215s - loss: 3.5133 - accuracy: 0.4348

Epoch 9/10

- 215s - loss: 3.4493 - accuracy: 0.4412

Epoch 10/10

- 216s - loss: 3.3964 - accuracy: 0.4468

Process finished with exit code 0

The document has been split into sentences using nltk.sentence_tokenize function and model has been trained with various minibatches of words in each sentence.

There is a gradual decrease in the loss function using the BTT function compared to TBTT. The accuracy seems to be increasing and better than the first one. However training the model to more epochs can bring more accuracy to the model.

PART 3.1 - Implementing GRU

In [0]:
```
import os
import re
import nltk
import numpy as np
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Embedding, LSTM, Dense, SimpleRNN, GRU, Masking
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
nltk.download('punkt')

base_path = os.path.abspath('English Literature.txt')

with open(base_path, encoding='utf-8') as f:
    sample = f.read()


def format_data(input_string):
    clean_string = re.sub(r'\((\d+)\)', r'', input_string)

    clean_string = re.sub(r'\s\s', ' ', clean_string)
    return clean_string


formatted_string = format_data(sample)

# integer encode text
regular_exp = nltk.RegexpTokenizer(r"\w+")
sent = regular_exp.tokenize(formatted_string)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sent)
encoded = tokenizer.texts_to_sequences([sent])[0]

sentence_tokenize = nltk.tokenize.sent_tokenize(formatted_string)
print(sentence_tokenize)
```

```
sent_len = 16
i = 0
input_X = []
output_y = []

X = []
y1 = []
padded_sent = []
for i in range(len(sentence_tokenize)):
    rem_exp = nltk.RegexpTokenizer(r"\w+")
    sentences = rem_exp.tokenize(sentence_tokenize[i])
    encoded = tokenizer.texts_to_sequences([sentences])[0]
    if (len(encoded) < sent_len) or (len(encoded) > sent_len):
        padded_sent = pad_sequences([encoded], maxlen=sent_len, dtype='i
                                    value=0.0)
    input_X = padded_sent[0][0:(sent_len - 1)]
    output_y = padded_sent[0][1:sent_len]
    X.append(input_X)
    y1.append(output_y)


# integer encode text
X = np.array(X)
num_y = np.array(y1)
print(X, num_y)
max_words = 10


vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

# one hot encode outputs
y = to_categorical(num_y, num_classes=vocab_size)


# define the network architecture: a embedding followed by LSTM
embedding_size = 100
rnn_size = 50
model1 = Sequential()
model1.add(Embedding(vocab_size, embedding_size, input_length=15))
model1.add(Masking(mask_value=0.0))
model1.add(GRU(rnn_size, return_sequences=True))
model1.add(TimeDistributed(Dense(vocab_size, activation='softmax')))
model1.compile(loss='categorical_crossentropy', optimizer='adam', metric
print(model1.summary())

# fit network
model1.fit(X, y, epochs=10, verbose=1, batch_size=1)
model.save('my_gru_model.h5')
```

Explanation 3.1

Epoch 1/10

20ms/step - loss: 4.1253 - accuracy: 0.3828

263s 21ms/step - loss: 3.9559 - accuracy: 0.3934

267s 21ms/step - loss: 3.8300 - accuracy: 0.4033

219ms/step - loss: 3.4546 - accuracy: 0.4418

250s 20ms/step - loss: 3.3084 - accuracy: 0.4585

245s 20ms/step - loss: 3.2458 - accuracy: 0.4666

....

Epoch 10/10

260s 21ms/step - loss: 3.0573 - accuracy: 0.4887

Process finished with exit code 0

After comparing the losss with Part 1 and Part 2. We can see that the GRU model performs better than the other two models.

As we can see here in the GRU model, the loss is steeply reducing and the accuracy seems to be better than the RNN model. However, increasing the epochs will increase the accuracy.

Part 3.2 Experimenting the generated word sequence for all the 3 models

Generating the word sequence for Part1

In [0]:
```
import os
import re

import nltk
import numpy as np
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Embedding, Dense, SimpleRNN
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
nltk.download('punkt')

base_path = os.path.abspath('English Literature.txt')
```

```
with open(base_path, encoding='utf-8') as f:
    sample = f.read()


def format_data(input_string):
    clean_string = re.sub(r'\((\d+)\)', r'', input_string)

    clean_string = re.sub(r'\s\s', ' ', clean_string)
    return clean_string


formatted_string = format_data(sample)

regular_exp = nltk.RegexpTokenizer(r"\w+")
token_sent = regular_exp.tokenize(formatted_string)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(token_sent)

outputs = []


def results(model, tokenizer, max_length, input_vector, n_words):
    for i in range(len(n_words)):
        input_vector = n_words[i]
        #print(len(nltk.word_tokenize(input_vector)))
        while len(nltk.word_tokenize(input_vector)) <= max_length:
            encoded = tokenizer.texts_to_sequences([input_vector])[0]
            encoded = pad_sequences([encoded], maxlen=max_length, dtype=
            predicted_words = model.predict_classes(encoded)
            # reverse_word_map = dict(map(reversed, tokenizer.word_index
            out_word = ''
            for word, index in tokenizer.word_index.items():
                if index == predicted_words:
                    out_word = word
                    break
            input_vector = " ".join((input_vector, out_word))
        outputs.append([input_vector])
    return outputs


n_words = ['good', 'first', 'citizen', 'second', 'talking', 'poor', 'bea
model = load_model('my_simple_model.h5')
generated_words = results(model, tokenizer, 1, 'love', n_words)
print("The generated word sequences are:", generated_words)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
The generated word sequences are: [['good lord'], ['first citizen'], [
```

```
'citizen we'], ['second murderer'], ['talking with'], ['poor soul'], [
'bear the'], ['With the'], ['what is'], ['More than']]
```

All the predicted words after a given input gives the user an understandable next english word.

Generating the word sequernce for Part2

In [0]:
```
import os
import re

import nltk
import numpy as np
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Embedding, Dense, SimpleRNN
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

base_path = os.path.abspath('English Literature.txt')

with open(base_path, encoding='utf-8') as f:
    sample = f.read()


def format_data(input_string):
    clean_string = re.sub(r'\((\d+)\)', r'', input_string)

    clean_string = re.sub(r'\s\s', ' ', clean_string)
    return clean_string


formatted_string = format_data(sample)

regular_exp = nltk.RegexpTokenizer(r"\w+")
token_sent = regular_exp.tokenize(formatted_string)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(token_sent)
encoded = tokenizer.texts_to_sequences([token_sent])[0]

outputs = []


def results(model, tokenizer, max_length, input_vector, n_words):
    for i in range(len(n_words)):
        input_vector = n_words[i]
        #print(len(nltk.word_tokenize(input_vector)))
        while len(nltk.word_tokenize(input_vector)) <= max_length:
```

```
            encoded = tokenizer.texts_to_sequences([input_vector])[0]
            encoded = pad_sequences([encoded], maxlen=max_length, dtype=
            predicted_words = model.predict_classes(encoded)
            # reverse_word_map = dict(map(reversed, tokenizer.word_index
            out_word = ''
            for word, index in tokenizer.word_index.items():
                if index == predicted_words[0][-1]:
                    out_word = word
                    break
            input_vector = " ".join((input_vector, out_word))
        outputs.append([input_vector])
    return outputs


n_words = ['what', 'first', 'then', 'second', 'my', 'poor', 'the', 'Upon
model = load_model('my_rnn_model.h5', compile=False)
generated_words = results(model, tokenizer, 15, 'love', n_words)
print("The generated RNN word sequences are:", generated_words)
```

```
The generated RNN word sequences are: [['what is the matter for what i
s the world that it was which from the crown'], ['first citizen you si
r i am not so much on my life s face for this'], ['then be gone and i
ll tell you as i am a man of my lord'], ['second citizen i ll tell the
e i say i ll be your good night i ll'], ['my horse is this the king an
d kill d him to the people and yet i'], ['poor boy hence i will not st
ay me to the king of york and death s'], ['the heavens have done to ha
nd in this way to be thine own with him for'], ['Upon a man of my lord
i ll tell you i am not so much on'], ['And what s the matter for t you
of me to the tower and with a'], ['but what is not so that i ll tell y
ou all for i am sure of']]
```

Ten sentences are generated and the generated RNN word sequence predicts and gives the user some better understandable english words which likely matching to the training data.

Generating the word sequence for Part3

In [0]:
```
import os
import re

import nltk
import numpy as np
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Embedding, Dense, SimpleRNN
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

base_path = os.path.abspath('English_Literature.txt')
```

```
base_path = os.path.abspath('English_Literature.txt')

with open(base_path, encoding='utf-8') as f:
    sample = f.read()


def format_data(input_string):
    clean_string = re.sub(r'\((\d+)\)', r'', input_string)

    clean_string = re.sub(r'\s\s', ' ', clean_string)
    return clean_string


formatted_string = format_data(sample)

regular_exp = nltk.RegexpTokenizer(r"\w+")
token_sent = regular_exp.tokenize(formatted_string)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(token_sent)
encoded = tokenizer.texts_to_sequences([token_sent])[0]

outputs = []


def results(model, tokenizer, max_length, input_vector, n_words):
    for i in range(len(n_words)):
        input_vector = n_words[i]
        #print(len(nltk.word_tokenize(input_vector)))
        while len(nltk.word_tokenize(input_vector)) <= 15:
            encoded = tokenizer.texts_to_sequences([input_vector])[0]
            encoded = pad_sequences([encoded], maxlen=max_length, dtype=
            predicted_words = model.predict_classes(encoded)
            # reverse_word_map = dict(map(reversed, tokenizer.word_index
            out_word = ''
            for word, index in tokenizer.word_index.items():
                if index == predicted_words[0][-1]:
                    out_word = word
                    break
            input_vector = " ".join((input_vector, out_word))
        outputs.append([input_vector])
    return outputs


n_words = ['King', 'first', 'Till', 'bring', 'talking', 'You', 'bear', '
model = load_model('my_gru_model.h5', compile=False)
generated_words = results(model, tokenizer, 15, 'love', n_words)
print("The generated GRU word sequences are:", generated_words)
```

```
The generated GRU word sequences are: [['King richard iii what is the
matter now sir he is dead and all his name'], ['first musician ay what
s a woman of this house of york as thou hast need'], ['Till then be ki
```

```
ng and now to me and you are at home to be rid'], ['bring forth the wa
y to me and thou shalt know the king and so then to'], ['talking with
her with her to prison and welcome home the blood of death hath made']
, ['You re welcome home to her in heaven bless thee on the queen s wif
e for'], ['bear me the king s death shall be so full of you are all th
e matter'], ['We ll not stay him for the people and that i have heard
of them and'], ['when you are here at the least is there is a man that
would never were'], ['I am not sir of you and you are enough to you an
d your daughter sir']]
```

Extract a word representation from a trained RNN

In [0]:

```
from keras.models import load_model
import os
import re
import nltk
import numpy as np
from keras.preprocessing.text import Tokenizer
from numpy import dot

base_path = os.path.abspath('English Literature.txt')

with open(base_path, encoding='utf-8') as f:
    sample = f.read()


def format_data(input_string):
    clean_string = re.sub(r'\((\d+)\)', r'', input_string)

    clean_string = re.sub(r'\s\s', ' ', clean_string)
    return clean_string


formatted_string = format_data(sample)

# integer encode text
regular_exp = nltk.RegexpTokenizer(r"\w+")
sent = regular_exp.tokenize(formatted_string)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sent)
encoded = tokenizer.texts_to_sequences([sent])[0]

model1 = load_model('my_simple_model.h5', compile=False)
model2 = load_model('my_rnn_model.h5', compile=False)
model3 = load_model('my_gru_model.h5', compile=False)

models = [model1, model2, model3]

model1_name = 'Simple model'
```

```
model2_name = 'RNN model'
model3_name = 'GRU model'

def get_embedding(embeddings):
    words_embeddings = {word: embeddings[index] for word, index in token
    return words_embeddings


def check_similarity(string1, string2):
    return dot(string1, string2) / (np.linalg.norm(string1) * np.linalg.


values = []
for i in range(len(models)):
    model = models[i]
    embeddings = model.layers[0].get_weights()[0]
    list_embedding = get_embedding(embeddings)
    string1 = 'have'
    string2 = 'had'
    a = list_embedding[string1]
    b = list_embedding[string2]
    word_similarity = check_similarity(a, b)
    values.append(word_similarity)

print("The cosine similarity for {} the two words is {}:".format(model1_
print("The cosine similarity for {} the two words is {}:".format(model2_
print("The cosine similarity for {} the two words is {}:".format(model3_
```

```
The cosine similarity for Simple model the two words is 0.704890012741
0889:
The cosine similarity for RNN model the two words is 0.692296028137207
:
The cosine similarity for GRU model the two words is 0.569222629070282
:
```

Part5- Learning an RNN model that predicts document categories given its content (text classification)

In [1]:
```
import os
import glob

import nltk
import numpy as np
from keras import Sequential
from keras.callbacks import EarlyStopping
from keras.regularizers import l2
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
```

```
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from keras.models import Sequential, load_model
from keras.layers import Embedding, LSTM, Dense, SpatialDropout1D, Simpl


def to_categories(name, cat=["politics", "rec", "comp", "religion"]):
    for i in range(len(cat)):
        if str.find(name, cat[i]) > -1:
            return (i)
    print("Unexpected folder: " + name)  # print the folder name which d
    return ("wth")


def data_loader(images_dir):
    categories = os.listdir(data_path)
    news = []  # news content
    groups = []  # category which it belong to

    for cat in categories:

        # print("Category:" + cat)
        for the_new_path in glob.glob(data_path + '/' + cat + '/*'):
            news.append(open(the_new_path, encoding="ISO-8859-1", mode='
            groups.append(cat)

    return news, list(map(to_categories, groups))


base_path = os.path.abspath('')
data_path = os.path.join(base_path, 'PyCharmProjects/RNN/datasets/20news
news, groups = data_loader(data_path)
print(news, groups)

#tokenized_sents = [nltk.word_tokenize(i) for i in news]

max_length = 200
embed_size = 100
tokenizer = Tokenizer()
tokenizer.fit_on_texts(news)
#word_index = tokenizer.word_index
#encoded = tokenizer.texts_to_sequences([news])[0]
vocab_size = len(tokenizer.word_index.items()) + 1

#word_tokenize = nltk.tokenize.sent_tokenize(news)

X1 = []
# X = tokenizer.texts_to_sequences([news])[0]
for i in range(len(news)):
    #tokenizer.fit_on_texts([news[i]])
    rem exp = nltk.RegexpTokenizer(r"\w+")
```

```
    sentences = rem_exp.tokenize(news[i])
    encoded = tokenizer.texts_to_sequences([sentences])[0]
    X = [pad_sequences([encoded], maxlen=max_length, dtype='int32', padd
    X1.append(X)

X = np.asarray(X1)
X = np.reshape(X, (13108, 200))

y = to_categorical(groups, num_classes=4)
print(y)

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.10
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
print(X.shape[1])
model = Sequential()
model.add(Embedding(vocab_size, embed_size, input_length=200))
model.add(SpatialDropout1D(0.2))
model.add(SimpleRNN(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(4, kernel_regularizer=l2(0.001), bias_regularizer=l2(0.0
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics
model.fit(X_train, Y_train, epochs=10, verbose=1, validation_data=(X_tes
model.save('my_text_classification_model.h5')
accuracy = model.evaluate(X_test, Y_test)
print('Testing Loss: {:0.5f}\n  Accuracy: {:0.5f}'.format(accuracy[0], a
```

```
Epoch 1/10
11797/11797 [==============================] - 69s 6ms/step - loss: 1.
3191 - accuracy: 0.3758 - val_loss: 1.2350 - val_accuracy: 0.4348
Epoch 2/10
11797/11797 [==============================] - 70s 6ms/step - loss: 1.
1012 - accuracy: 0.5462 - val_loss: 0.9762 - val_accuracy: 0.6301
Epoch 3/10
11797/11797 [==============================] - 71s 6ms/step - loss: 0.
8613 - accuracy: 0.6791 - val_loss: 0.8788 - val_accuracy: 0.6377
Epoch 4/10
11797/11797 [==============================] - 67s 6ms/step - loss: 0.
6919 - accuracy: 0.7525 - val_loss: 0.8413 - val_accuracy: 0.6598
Epoch 5/10
11797/11797 [==============================] - 67s 6ms/step - loss: 0.
6022 - accuracy: 0.7894 - val_loss: 0.8431 - val_accuracy: 0.6735
Epoch 6/10
11797/11797 [==============================] - 66s 6ms/step - loss: 0.
4707 - accuracy: 0.8389 - val_loss: 0.8531 - val_accuracy: 0.6903
Epoch 7/10
11797/11797 [==============================] - 68s 6ms/step - loss: 0.
```

****Part 5 Explanation****

The dataset has been downloaded and divided into training and testing with 90% and 10%.

****Report your accuracy results on the validation set****

I have trained the model by adding the L2 regularization to avoid the overfitting of the data. By proper splitting of the data and fine tuning the hyperparameters, I have achieved around a ****validation accuracy of 78** %** after running 15 epochs. However,increasing the no of epochs further and closely studying the data could improve the validation accuracy much better.