NLP Assignmnet 4

Name : Krishna Kant Verma
Roll No: 2211cs19
Name : Gourab Chatterjee
Roll No: 2211cs08

`Importing libraries Used for this Assignmnemt`

```
import numpy as np
import tensorflow as tf
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
import keras.backend as K
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
from keras.layers import Dense,SimpleRNN
from keras.models import Sequential
```

`Reading text dataset file`

```
with open("names.txt",'r') as f:
    data=f.readlines()
data=[text.strip('\n')+'.' for text in data]
```

Creating Dataset

```
def createDataset(data,ngram):
    X=[]
    Y=[]
    for text in data:
        pointer=0
        while pointer+ngram<len(text):
            X.append(text[pointer:pointer+ngram])
            Y.append(text[pointer+ngram])
            pointer+=1
    ctoi={char:ind for ind,char in enumerate(sorted(set(Y)))}

    X = [[to_categorical(ctoi[charector],27) for charector in  text_data] for text_data in X]
    X = np.array(X)
    Y = [to_categorical(ctoi[charector],27) for charector in Y]
    Y = np.array(Y)
    X = X.reshape((X.shape[0],X.shape[1]*X.shape[2]))
    fd=np.concatenate([X,Y],axis=1)
    np.random.shuffle(fd)
    part1=int(fd.shape[0]*0.9)
    part2=part1+int(fd.shape[0]*0.05)
    return np.split(fd,[part1,part2]),ctoi
```

```
def createModel(ngram):
    def perplexityLoss(y_true, y_pred):
        crossEntropyError = K.categorical_crossentropy(y_true, y_pred)
        perplexity = K.pow(np.e, crossEntropyError)
        return perplexity

    model = Sequential([
        Dense(128,input_shape=(27*ngram,),activation='relu'),
        Dense(64,activation='relu'),
        Dense(27,activation='softmax'),
    ])

    model.compile(optimizer='adam',loss=perplexityLoss,metrics=['accuracy'])
    return model
```

Bi-Gram Model

```
earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='auto', restore_best_weights=True)
(train,test,val),ctoi=createDataset(data,2)

model_2gram = createModel(2)
gram2His = model_2gram.fit(train[:,:27*2],train[:,27*2:],
                  validation_data=[val[:,:27*2],val[:,27*2:]],
                  epochs=30,callbacks=[earlyStopping])
```

```
Epoch 1/30
4615/4615 [==============================] - 17s 3ms/step - loss: 14.9830 - accuracy: 0.2858 - val_loss: 14.3642 - val_accuracy: 0.3005
Epoch 2/30
4615/4615 [==============================] - 17s 4ms/step - loss: 14.0387 - accuracy: 0.3018 - val_loss: 14.1682 - val_accuracy: 0.3079
Epoch 3/30
4615/4615 [==============================] - 15s 3ms/step - loss: 13.7875 - accuracy: 0.3064 - val_loss: 14.1574 - val_accuracy: 0.3130
Epoch 4/30
4615/4615 [==============================] - 16s 3ms/step - loss: 13.6542 - accuracy: 0.3084 - val_loss: 14.1320 - val_accuracy: 0.3069
Epoch 5/30
4615/4615 [==============================] - 16s 3ms/step - loss: 13.5396 - accuracy: 0.3087 - val_loss: 15.4833 - val_accuracy: 0.3044
Epoch 6/30
4615/4615 [==============================] - 17s 4ms/step - loss: 13.4668 - accuracy: 0.3084 - val_loss: 21.6408 - val_accuracy: 0.3103
Epoch 7/30
4615/4615 [==============================] - 15s 3ms/step - loss: 13.4078 - accuracy: 0.3103 - val_loss: 21.3248 - val_accuracy: 0.3117
Epoch 8/30
4615/4615 [==============================] - 16s 3ms/step - loss: 13.3562 - accuracy: 0.3101 - val_loss: 27.8426 - val_accuracy: 0.3118
Epoch 9/30
4615/4615 [==============================] - 16s 3ms/step - loss: 13.3168 - accuracy: 0.3108 - val_loss: 80.6986 - val_accuracy: 0.3116
Epoch 10/30
4615/4615 [==============================] - 16s 4ms/step - loss: 13.2953 - accuracy: 0.3111 - val_loss: 50.0324 - val_accuracy: 0.3156
Epoch 11/30
4615/4615 [==============================] - 16s 3ms/step - loss: 13.2676 - accuracy: 0.3111 - val_loss: 114.3568 - val_accuracy: 0.3133
Epoch 12/30
4615/4615 [==============================] - 16s 3ms/step - loss: 13.2340 - accuracy: 0.3107 - val_loss: 75.4421 - val_accuracy: 0.3133
Epoch 13/30
4615/4615 [==============================] - 17s 4ms/step - loss: 13.2142 - accuracy: 0.3123 - val_loss: 280.6599 - val_accuracy: 0.3055
Epoch 14/30
4612/4615 [============================>.] - ETA: 0s - loss: 13.1940 - accuracy: 0.3128Restoring model weights from the end of the best epoch: 4.
4615/4615 [==============================] - 16s 3ms/step - loss: 13.1951 - accuracy: 0.3128 - val_loss: 1316.8237 - val_accuracy: 0.3118
Epoch 14: early stopping
```

Tri-Gram-Model

```
(train,test,val),ctoi=createDataset(data,3)
m3gram = createModel(3)
```

```
gram3His = m3gram.fit(train[:,:27*3],train[:,27*3:],
                 validation_data=[val[:,:27*3],val[:,27*3:]],
                 epochs=30,callbacks=[earlyStopping])
```

```
Epoch 1/30
3714/3714 [==============================] - 14s 4ms/step - loss: 13.5096 - accuracy: 0.3314 - val_loss: 12.4584 - val_accuracy: 0.3592
Epoch 2/30
3714/3714 [==============================] - 13s 3ms/step - loss: 12.0642 - accuracy: 0.3642 - val_loss: 12.1334 - val_accuracy: 0.3686
Epoch 3/30
3714/3714 [==============================] - 13s 3ms/step - loss: 11.5901 - accuracy: 0.3745 - val_loss: 11.9448 - val_accuracy: 0.3788
Epoch 4/30
3714/3714 [==============================] - 13s 3ms/step - loss: 11.2514 - accuracy: 0.3808 - val_loss: 11.6643 - val_accuracy: 0.3801
Epoch 5/30
3714/3714 [==============================] - 13s 4ms/step - loss: 11.0154 - accuracy: 0.3826 - val_loss: 11.9007 - val_accuracy: 0.3765
Epoch 6/30
3714/3714 [==============================] - 13s 3ms/step - loss: 10.8030 - accuracy: 0.3848 - val_loss: 11.8727 - val_accuracy: 0.3736
Epoch 7/30
3714/3714 [==============================] - 13s 3ms/step - loss: 10.6454 - accuracy: 0.3880 - val_loss: 12.0161 - val_accuracy: 0.3827
Epoch 8/30
3714/3714 [==============================] - 13s 3ms/step - loss: 10.5077 - accuracy: 0.3899 - val_loss: 12.1218 - val_accuracy: 0.3803
Epoch 9/30
3714/3714 [==============================] - 13s 3ms/step - loss: 10.3686 - accuracy: 0.3897 - val_loss: 12.5413 - val_accuracy: 0.3803
Epoch 10/30
3714/3714 [==============================] - 13s 3ms/step - loss: 10.2623 - accuracy: 0.3919 - val_loss: 12.9072 - val_accuracy: 0.3845
Epoch 11/30
3714/3714 [==============================] - 12s 3ms/step - loss: 10.1693 - accuracy: 0.3937 - val_loss: 13.7097 - val_accuracy: 0.3842
Epoch 12/30
3714/3714 [==============================] - 13s 3ms/step - loss: 10.0795 - accuracy: 0.3931 - val_loss: 15.5663 - val_accuracy: 0.3794
Epoch 13/30
3714/3714 [==============================] - 13s 3ms/step - loss: 9.9991 - accuracy: 0.3945 - val_loss: 14.5424 - val_accuracy: 0.3850
Epoch 14/30
3705/3714 [============================>.] - ETA: 0s - loss: 9.9450 - accuracy: 0.3947Restoring model weights from the end of the best epoch: 4.
3714/3714 [==============================] - 13s 3ms/step - loss: 9.9446 - accuracy: 0.3947 - val_loss: 19.7349 - val_accuracy: 0.3888
Epoch 14: early stopping
```

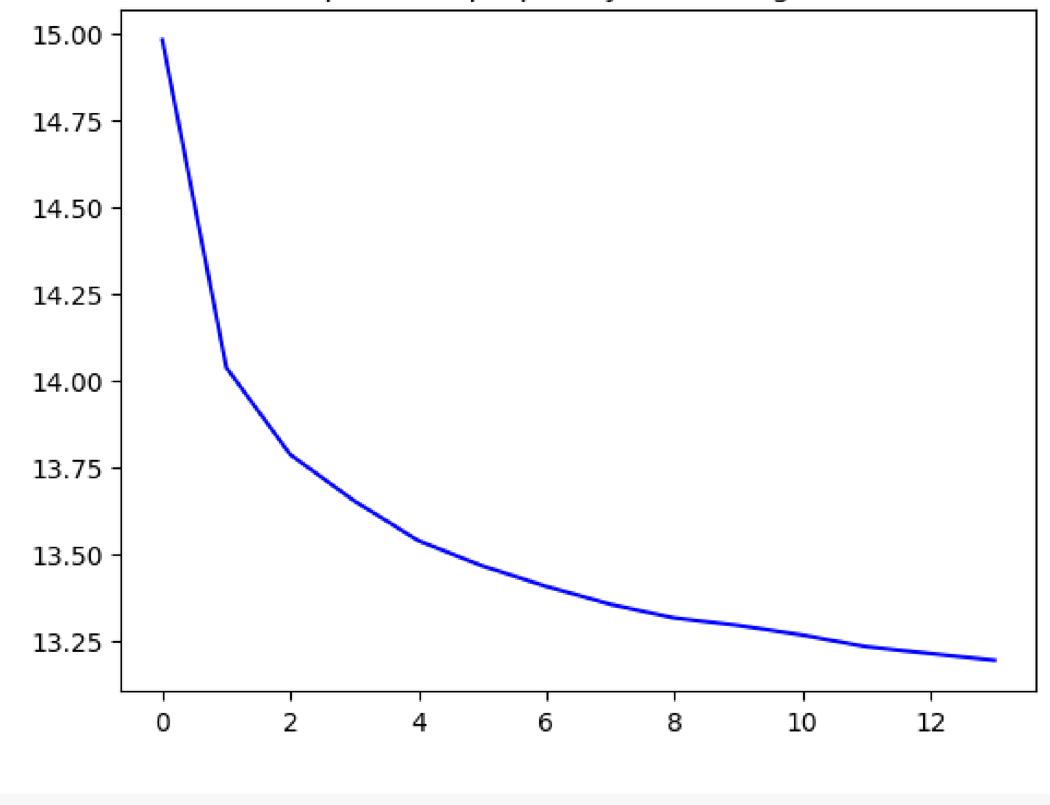
```
earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='auto', restore_best_weights=True)
def create_RNN_model(ngram):
    def perplexityLoss(y_true, y_pred):
        crossEntropyError = K.categorical_crossentropy(y_true, y_pred)
        perplexity = K.pow(np.e, crossEntropyError)
        return perplexity
    model = Sequential([
        SimpleRNN(128,input_shape=(ngram,27),activation='relu'),
        Dense(27,activation='softmax'),
    ])

    model.compile(optimizer='adam',loss=perplexityLoss,metrics=['accuracy'])
    return model
```

```
ngram = 3
(train,test,val),ctoi=createDataset(data,ngram)
m2RNNgram = create_RNN_model(ngram)
m2RNNgram.fit(train[:,:27*ngram].reshape(train.shape[0],ngram,27),train[:,27*ngram:],
          validation_data=[val[:,:27*ngram].reshape(val.shape[0],ngram,27),val[:,27*ngram:]],
          epochs=30,callbacks=[earlyStopping])
```

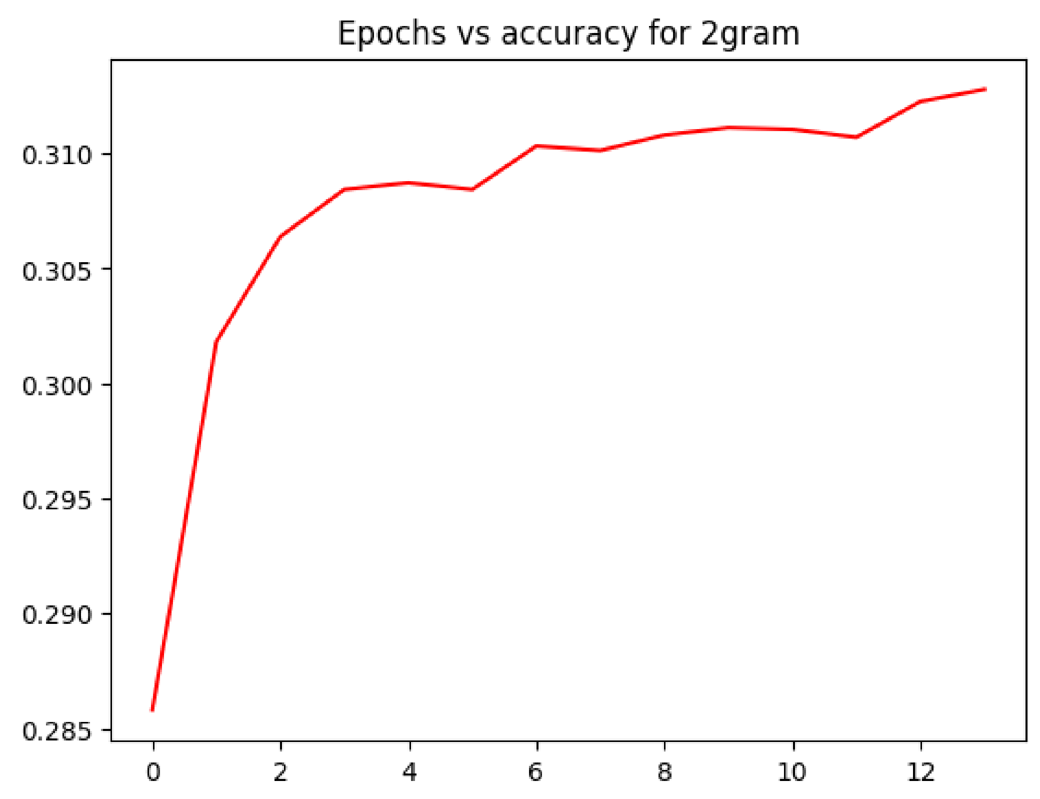
```
Epoch 1/30
3714/3714 [==============================] - 24s 6ms/step - loss: 13.6401 - accuracy: 0.3287 - val_loss: 12.9733 - val_accuracy: 0.3562
Epoch 2/30
3714/3714 [==============================] - 22s 6ms/step - loss: 12.3990 - accuracy: 0.3565 - val_loss: 12.3381 - val_accuracy: 0.3638
Epoch 3/30
3714/3714 [==============================] - 21s 6ms/step - loss: 11.8176 - accuracy: 0.3687 - val_loss: 12.3693 - val_accuracy: 0.3623
Epoch 4/30
3714/3714 [==============================] - 22s 6ms/step - loss: 11.5572 - accuracy: 0.3731 - val_loss: 12.2064 - val_accuracy: 0.3760
Epoch 5/30
3714/3714 [==============================] - 22s 6ms/step - loss: 11.2724 - accuracy: 0.3783 - val_loss: 13.0541 - val_accuracy: 0.3816
Epoch 6/30
3714/3714 [==============================] - 22s 6ms/step - loss: 11.0756 - accuracy: 0.3823 - val_loss: 12.8082 - val_accuracy: 0.3797
Epoch 7/30
3714/3714 [==============================] - 21s 6ms/step - loss: 10.8910 - accuracy: 0.3849 - val_loss: 12.9368 - val_accuracy: 0.3824
Epoch 8/30
3714/3714 [==============================] - 22s 6ms/step - loss: 10.7443 - accuracy: 0.3858 - val_loss: 12.6300 - val_accuracy: 0.3845
Epoch 9/30
3714/3714 [==============================] - 22s 6ms/step - loss: 10.6279 - accuracy: 0.3884 - val_loss: 12.9676 - val_accuracy: 0.3815
Epoch 10/30
3714/3714 [==============================] - 22s 6ms/step - loss: 10.5113 - accuracy: 0.3894 - val_loss: 14.1492 - val_accuracy: 0.3829
Epoch 11/30
3714/3714 [==============================] - 21s 6ms/step - loss: 10.4062 - accuracy: 0.3908 - val_loss: 14.5524 - val_accuracy: 0.3847
Epoch 12/30
3714/3714 [==============================] - 22s 6ms/step - loss: 10.3238 - accuracy: 0.3914 - val_loss: 13.3902 - val_accuracy: 0.3863
Epoch 13/30
3714/3714 [==============================] - 22s 6ms/step - loss: 10.2419 - accuracy: 0.3923 - val_loss: 14.4990 - val_accuracy: 0.3889
Epoch 14/30
3714/3714 [==============================] - ETA: 0s - loss: 10.1754 - accuracy: 0.3930Restoring model weights from the end of the best epoch: 4.
3714/3714 [==============================] - 22s 6ms/step - loss: 10.1754 - accuracy: 0.3930 - val_loss: 16.5194 - val_accuracy: 0.3841
Epoch 14: early stopping
<keras.callbacks.History at 0x7f96cd6003a0>
```

Plots

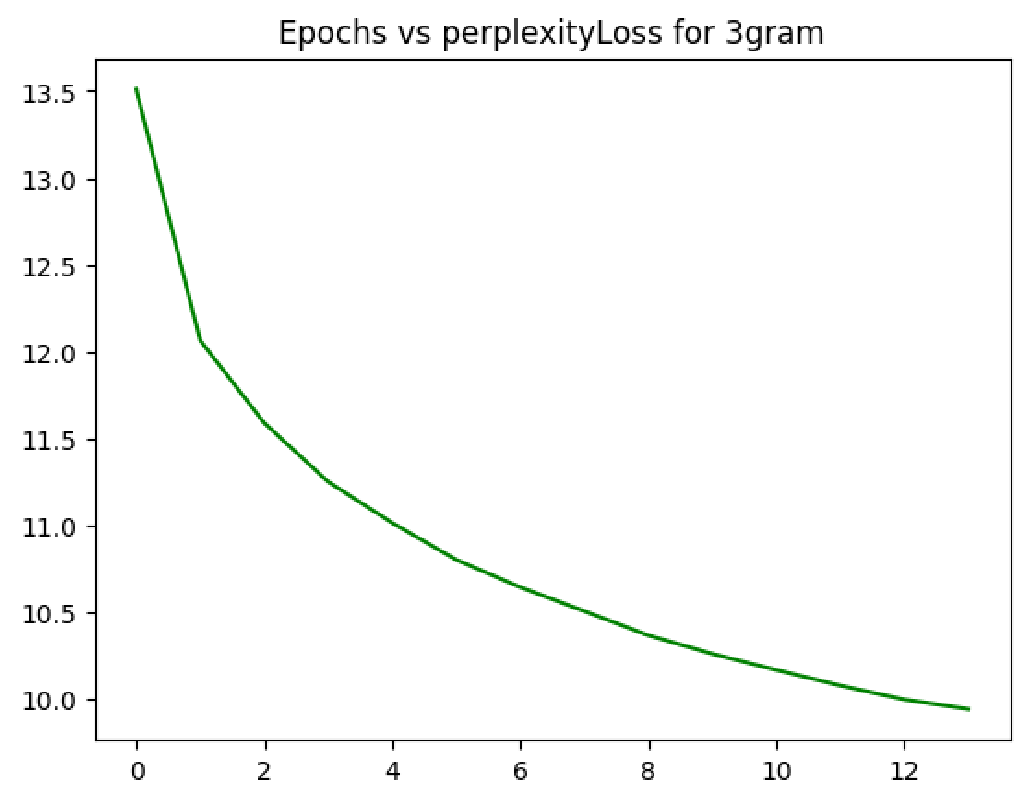
```
import matplotlib.pyplot as plt
plt.plot([x for x in range(len(gram2His.history['loss']))],gram2His.history['loss'],c='b')
plt.title("Epochs vs perplexityLoss for 2gram")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
```



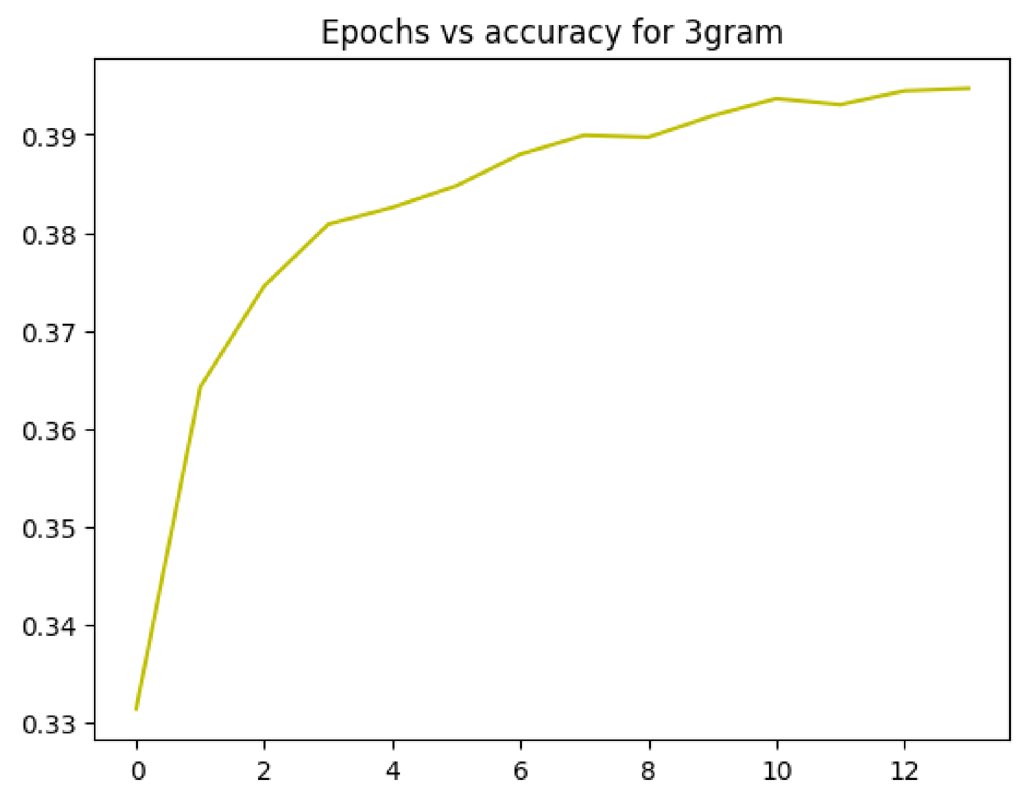
```
plt.plot([x for x in range(len(gram2His.history['accuracy']))],gram2His.history['accuracy'],c='r')
plt.title("Epochs vs accuracy for 2gram")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
```



```
plt.plot([x for x in range(len(gram3His.history['loss']))],gram3His.history['loss'],c='g')
plt.title("Epochs vs perplexityLoss for 3gram")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
```



```
plt.plot([x for x in range(len(gram3His.history['accuracy']))],gram3His.history['accuracy'],c='y')
plt.title("Epochs vs accuracy for 3gram")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
```



```
with open("logs.txt" , 'w') as f:
  for ind,history in enumerate([gram2His,gram3His]):
    for data in ['loss','accuracy']:
      for epoch,value in enumerate(history.history[data]):
        f.writelines(f"""For dataset_{ind+1}
                        \t\t\t For epoch_{epoch} \t{data}= {round(value,2)}\n""")
```

Thank You So Much

✓ 0s completed at 20:13