In [1]:
```
import pandas as pd
import string
from string import digits
import re
import os
from numpy import array, argmax, random, take
from sklearn.model_selection import train_test_split
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding, RepeatVector,Tir
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceL
from tensorflow.keras.models import load_model
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt
%matplotlib inline
# pd.set_option('display.max_colwidth', 200)
```

In [2]:
```
project_path = "/content/language_translator"
```

In [3]:
```
# read phrases from english_telugu_data.txt file
english_sentences = []
telugu_sentences = []
with open("english_telugu_data.txt", mode='rt', encoding='utf-8') as fp:
    for line in fp.readlines():
        eng_tel = line.split("++++$++++")
        english_sentences.append(eng_tel[0])
        telugu_sentences.append(eng_tel[1])
```

In [4]:
```
data = pd.DataFrame({"english_sentences":english_sentences,"telugu_sentences
```

In [5]:
```
data.head(9)
```

Out[5]:

	english_sentences	telugu_sentences
0	His legs are long.	అతని కాళ్ళు పొడవుగా ఉన్నాయి.\n
1	Who taught Tom how to speak French?	టామ్ ఫ్రెంచ్ మాట్లాడటం ఎలా నేర్పించారు?\n
2	I swim in the sea every day.	నేను ప్రతి రోజు సముద్రంలో ఈత కొడతాను.\n
3	Tom popped into the supermarket on his way hom...	టామ్ కొంచెం పాలు కొనడానికి ఇంటికి వెళ్ళేటప్పుడ...
4	Smoke filled the room.	పొగ గదిని నింపింది.\n
5	Tom and Mary understood each other.	టామ్ మరియు మేరీ ఒకరినొకరు అర్థం చేసుకున్నారు.\n
6	Many men want to be thin, too.	చాలా మంది పురుషులు కూడా సన్నగా ఉండాలని కోరుకుం...
7	We need three cups.	మాకు మూడు కప్పులు అవసరం.\n
8	I warned Tom not to come here.	టామ్‌ను ఇక్కడికి రానివ్వమని హెచ్చరించాను.\n

In [6]:
```
data.shape
```

Out[6]:
```
(155798, 2)
```

In [7]:
```
data = data.iloc[:70000,:]
```

In [8]:
```
contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cann
                           "didn't": "did not",  "doesn't": "does not", "don
                           "he'd": "he would","he'll": "he will", "he's": "h
                           "I'd": "I would", "I'd've": "I would have", "I'll
                           "i'd've": "i would have", "i'll": "i will",  "i'l
                           "it'd've": "it would have", "it'll": "it will", '
                           "mayn't": "may not", "might've": "might have","mi
                           "mustn't": "must not", "mustn't've": "must not ha
                           "oughtn't": "ought not", "oughtn't've": "ought no
                           "she'd": "she would", "she'd've": "she would have
                           "should've": "should have", "shouldn't": "should
                           "this's": "this is","that'd": "that would", "that
                           "there'd've": "there would have", "there's": "the
                           "they'll": "they will", "they'll've": "they will
                           "wasn't": "was not", "we'd": "we would", "we'd've
                           "we've": "we have", "weren't": "were not", "what
                           "what's": "what is", "what've": "what have", "whe
                           "where've": "where have", "who'll": "who will", '
                           "why's": "why is", "why've": "why have", "will've
                           "would've": "would have", "wouldn't": "would not'
                           "y'all'd": "you all would","y'all'd've": "you all
                           "you'd": "you would", "you'd've": "you would have
                           "you're": "you are", "you've": "you have"}
```

In [9]:
```
# clean english sentances
def clean_eng(text):
    # Lowercase all characters
    text = text.lower()
    # map contractions
    text = ' '.join([contraction_mapping[w] if w in contraction_mapping else
    # Remove quotes
    text = re.sub("'", '', text)
    # Remove all the special characters
    exclude = set(string.punctuation) # Set of all special characters
    text = ''.join([c for c in text if c not in exclude])
    # Remove all numbers from text
    remove_digits = str.maketrans('', '', digits)
    text = text.translate(remove_digits)
    # Remove extra spaces
    text= text.strip()

    return text
```

In [10]:
```
# clean telugu sentances
def clean_tel(text):
    # Lowercase all characters
    text = text.lower()
    # Remove quotes
    text = re.sub("'", '', text)
    # Remove all the special characters
    exclude = set(string.punctuation) # Set of all special characters
    text = ''.join([c for c in text if c not in exclude])
    # Remove all numbers from text
    remove_digits = str.maketrans('', '', digits)
    text = text.translate(remove_digits)
    # Remove Telugu numbers from text
    text = re.sub("[౦౧౨౩౪౫౬౭౮౯]", '', text)
    # Remove extra spaces
    text= text.strip()

    return text
```

In [11]:
```
data_df = data.copy()
data_df["english_sentences"] = data_df["english_sentences"] .apply(lambda x
data_df["telugu_sentences"] = data_df["telugu_sentences"] .apply(lambda x: 
```

In [12]:
```
data_df.head()
```

Out[12]:

	english_sentences	telugu_sentences
0	his legs are long	అతని కాళ్ళు పొడవుగా ఉన్నాయి
1	who taught tom how to speak french	టామ్ ఫ్రెంచ్ మాట్లాడటం ఎలా నేర్పించారు
2	i swim in the sea every day	నేను ప్రతి రోజు సముద్రంలో ఈత కొడతాను
3	tom popped into the supermarket on his way hom...	టామ్ కొంచెం పాలు కొనడానికి ఇంటికి వెళ్ళేటప్పుడ...
4	smoke filled the room	పొగ గదిని నింపింది

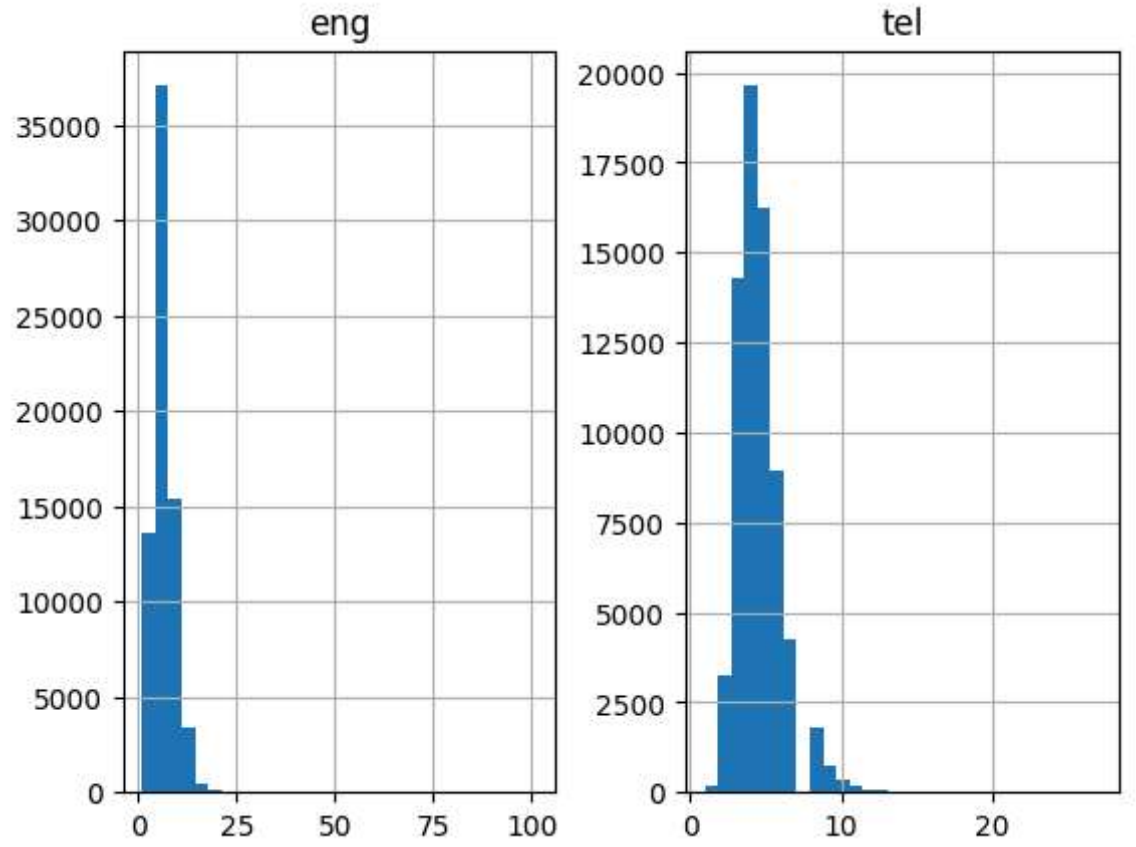
In [13]:
```
# empty lists
eng_l = []
tel_l = []

# populate the lists with sentence lengths
for i in data_df["english_sentences"].values:
      eng_l.append(len(i.split()))

for i in data_df["telugu_sentences"].values:
      tel_l.append(len(i.split()))

length_df = pd.DataFrame({'eng':eng_l, 'tel':tel_l})

length_df.hist(bins = 30)
plt.show()
```



In [14]:
```
# function to build a tokenizer
def tokenization(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

In [15]:
```
# prepare english tokenizer
eng_tokenizer = tokenization(data_df["english_sentences"])
eng_vocab_size = len(eng_tokenizer.word_index) + 1

eng_length = 43
print('English Vocabulary Size: %d' % eng_vocab_size)
```

```
English Vocabulary Size: 10315
```

In [16]:
```
# prepare Telugu tokenizer
tel_tokenizer = tokenization(data_df["telugu_sentences"])
tel_vocab_size = len(tel_tokenizer.word_index) + 1

tel_length = 26
print('Telugu Vocabulary Size: %d' % tel_vocab_size)
```

```
Telugu Vocabulary Size: 26680
```

In [17]:
```
def encode_sequences(tokenizer, length, lines):
    # integer encode sequences
    seq = tokenizer.texts_to_sequences(lines)
    # pad sequences with 0 values
    seq = pad_sequences(seq, maxlen=length, padding='post')
    return seq
```

In [18]:
```
train, test = train_test_split(data_df, test_size=0.2, random_state = 12)
```

In [19]:
```
trainX = encode_sequences(eng_tokenizer, eng_length, train["english_sentence
trainY = encode_sequences(tel_tokenizer, tel_length, train["telugu_sentences

# prepare validation data
testX = encode_sequences(eng_tokenizer, eng_length, test["english_sentences"
testY = encode_sequences(tel_tokenizer, tel_length, test["telugu_sentences"]
```

In [20]:
```
trainX.shape,trainY.shape,testX.shape,testY.shape
```

Out[20]:
```
((56000, 43), (56000, 26), (14000, 43), (14000, 26))
```

In [21]:
```
# build NMT model
def define_model(in_vocab,out_vocab, in_timesteps,out_timesteps,units):
    model = Sequential()
    model.add(Embedding(in_vocab, units, input_length=in_timesteps, mask_zer
    model.add(LSTM(units))
    model.add(RepeatVector(out_timesteps))
    model.add(LSTM(units, return_sequences=True))
    model.add(TimeDistributed(Dense(out_vocab, activation='softmax')))
    return model
```

In [22]:
```
# model compilation
model = define_model(eng_vocab_size,tel_vocab_size,eng_length,tel_length, 51
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 43, 512)           5281280

 lstm (LSTM)                 (None, 512)               2099200

 repeat_vector (RepeatVecto  (None, 26, 512)           0
 r)

 lstm_1 (LSTM)               (None, 26, 512)           2099200

 time_distributed (TimeDist  (None, 26, 26680)         13686840
 ributed)

=================================================================
Total params: 23166520 (88.37 MB)
Trainable params: 23166520 (88.37 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

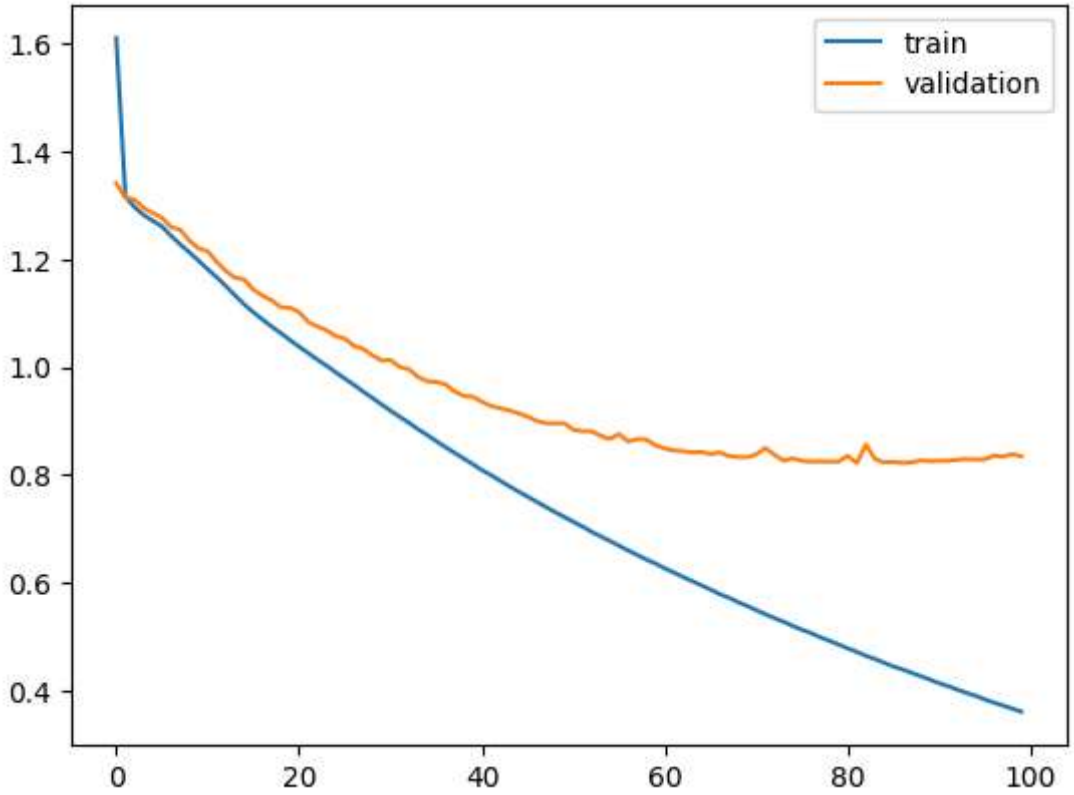
In [23]:
```
rms = optimizers.RMSprop()
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')
```

In [24]:
```
# Defining a helper function to save the model after each epoch
# in which the loss decreases
filepath = project_path+'NMT_model.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_b
# Defining a helper function to reduce the learning rate each time
# the learning plateaus
reduce_alpha = ReduceLROnPlateau(monitor ='val_loss', factor = 0.2, patience
# stop traning if there increase in loss
# es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,patience=2)
callbacks = [checkpoint, reduce_alpha]
```

In [25]:
```
history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1],
                    epochs=100, batch_size=128, validation_split = 0.2,callb
```

```
Epoch 65: val_loss did not improve from 0.84172
350/350 [==============================] - 62s 178ms/step - loss: 0.5951
- val_loss: 0.8424 - lr: 0.0010
Epoch 66/100
350/350 [==============================] - ETA: 0s - loss: 0.5871
Epoch 66: val_loss improved from 0.84172 to 0.83858, saving model to /co
ntent/language_translatorNMT_model.h5
350/350 [==============================] - 68s 193ms/step - loss: 0.5871
- val_loss: 0.8386 - lr: 0.0010
Epoch 67/100
350/350 [==============================] - ETA: 0s - loss: 0.5788
Epoch 67: val_loss did not improve from 0.83858
350/350 [==============================] - 67s 191ms/step - loss: 0.5788
- val_loss: 0.8417 - lr: 0.0010
Epoch 68/100
350/350 [==============================] - ETA: 0s - loss: 0.5720
Epoch 68: val_loss improved from 0.83858 to 0.83501, saving model to /co
ntent/language_translatorNMT_model.h5
350/350 [==============================] - 63s 179ms/step - loss: 0.5720
```

In [26]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train','validation'])
plt.show()
```



In [27]:
```
# get 10 random ids of test samples
idx = random.randint(testX.shape[0], size=10)
# get 10 encoded english test samples
encoded_english_actual = testX[idx,:]
# get 10 actual english sentences
eng_actual = test["english_sentences"].values
eng_actual = eng_actual[idx]
# get 10 actual telugu sentences
actual = test["telugu_sentences"].values
actual = actual[idx]
```

In [28]:
```
# load model weights
# model.load_weights(filepath)
# predict english sentence to telugu sentence
# Predict probabilities for each class
pred_probs = model.predict(encoded_english_actual.reshape((encoded_english_a

# Get the class with the highest probability for each prediction
preds = pred_probs.argmax(axis=-1)
```

```
1/1 [==============================] - 2s 2s/step
```

In [29]:
```
def get_word(n, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == n:
            return word
    return None
```

In [30]:
```
preds_text = []
for i in preds:
    temp = []
    for j in range(len(i)):
        t = get_word(i[j], tel_tokenizer)
        if j > 0:
            if (t == get_word(i[j-1], tel_tokenizer)) or (t == None):
                temp.append('')
            else:
                temp.append(t)
        else:
            if(t == None):
                temp.append('')
            else:
                temp.append(t)

    preds_text.append(' '.join(temp))
```

In [31]:
```
pred_df = pd.DataFrame({'english_actual':eng_actual, 'telugu_actual' : actua
```

In [32]:
```
pred_df.head(10)
```

Out[32]:

	english_actual	telugu_actual	telugu_predicted
0	i assure you that is not the case	నేను మీకు భరోసా ఇస్తున్నాను	ఇది లేదని నేను ఇస్తున్నాను
1	if you trust such a fellow you will lose every...	మీరు అలాంటి తోటివారిని విశ్వసిస్తే మీ వద్ద ఉన్...	మీరు మీకు వాటిని ఉంచే దయచేసి మీరు వాటిని చేశార...
2	tom does not need to go there if he does not w...	టామ్ కోరుకోకపోతే అక్కడికి వెళ్లవలసిన అవసరం లేదు	టామ్ అక్కడికి వెళ్ళవలసిన అవసరం లేదు ...
3	you should marry tom	మీరు టామ్‌ను వివాహం చేసుకోవాలి	మీరు టామ్‌ను వివాహం చేసుకోవాలి ...
4	i thought that tom said he liked this kind of ...	టామ్ తనకు ఈ రకమైన సంగీతం నచ్చిందని చెప్పాడు	అతను తనకు రకమైన సంగీతం నచ్చిందని లేదని అభిప్రా...
5	i would hate to do that	నేను అలా చేయడాన్ని ద్వేషిస్తాను	నేను అలా
6	tom went straight to the post office	టామ్ నేరుగా పోస్టాఫీసుకు వెళ్ళాడు	టామ్ వెనుక నుండి దుకాణానికి వెళ్ళాడు ...
7	can you sail a boat	మీరు పడవలో ప్రయాణించగలరా	మీరు ఒక పడవ కావాలనుకుంటున్నారా ...
8	who did you talk to	మీరు ఎవరితో మాట్లాడారు	మీరు ఎవరితో మాట్లాడతారు
9	it all seems pointless	ఇదంతా అర్థంలేనిదిగా అనిపిస్తుంది	ఇదంతా భయంకరంగా అనిపిస్తుంది ...

In [ ]:
```

```