1) Pre-processing Custom Emoji Dataset

In [5]:

```
import numpy as np
import pandas as pd
import emoji
from keras.utils import to_categorical
from gensim.models import KeyedVectors
```

In [4]:

```
data_train = pd.read_csv("Datasets/train_emoji.csv", header=None)
data_test = pd.read_csv("Datasets/test_emoji.csv", header=None)

X_train = data_train.values[:,0]
Y_train = data_train.values[:,1]

X_test = data_test.values[:,0]
Y_test = data_test.values[:,1]

print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(132,) (132,)
(56,) (56,)
```

In [6]:

```
# Required functions and dictionaries from previous notebook:- 01
emoji_dictionary = {
    "0": "\u2764\uFE0F",
    "1": ":baseball:",
    "2": ":grinning_face_with_big_eyes:",
    "3": ":disappointed_face:",
    "4": ":fork_and_knife:"
}

word_vec = KeyedVectors.load_word2vec_format("Datasets/word2vec_from_glove.6B.50d.txt")

def getOutputEmbeddings(X):

    emb_matrix = np.zeros((X.shape[0], 10, 50))
    for sentences in range(X.shape[0]): # iterating over each sentence
        words = X[sentences].split() # Breaking each sentence into respective words
        for word in range(len(words)):
            emb_matrix[sentences][word] = word_vec[words[word].lower()]

    return emb_matrix
```

In [7]:
```
# Processing the Testing & Training data:

embedding_train = getOutputEmbeddings(X_train)
embedding_test = getOutputEmbeddings(X_test)

Y_train = to_categorical(Y_train, num_classes=5)
Y_test = to_categorical(Y_test, num_classes=5)

print(embedding_train.shape, Y_train.shape)
print(embedding_test.shape, Y_test.shape)
```

```
(132, 10, 50) (132, 5)
(56, 10, 50) (56, 5)
```

2) Building Stacked LSTM Model

In this model we unroll the LSTM cells and we use output(y) from each cell and pass it onto the layer above it to the respective cells and the activation state vector(a) calculated from the second LSTM layer and pass it into a Dense layer with 5 outputs and a softmax activation.

In [8]:
```
from keras.models import Sequential
from keras.layers import *
```

In [48]:
```
model = Sequential()
model.add(LSTM(64, input_shape=(10,50), return_sequences=True)) # return_sequences=True for sending
# signal from each cell to the cell above it
model.add(Dropout(0.4))
model.add(LSTM(64))
model.add(Dropout(0.3))
model.add(Dense(5))
model.add(Activation("softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])
model.summary()

# We are not training our embedding layer and using transfer learning(pre trained glove vectors)
# because then just the embedding layer would add 5,00,000 parameters to train just for a dictionary of
# 10,000 words and the dataset we have is too small for training these many parameters
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_23 (LSTM)               (None, 10, 64)            29440
_________________________________________________________________
dropout_23 (Dropout)         (None, 10, 64)            0
_________________________________________________________________
lstm_24 (LSTM)               (None, 64)                33024
_________________________________________________________________
dropout_24 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 5)                 325
_________________________________________________________________
activation_12 (Activation)   (None, 5)                 0
=================================================================
Total params: 62,789
Trainable params: 62,789
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

In [49]:

```
# Training our model

earlystop = EarlyStopping(patience=20, monitor='val_acc')
checkpoint = ModelCheckpoint("best_weights_2.h5", save_best_only=True, save_weights_only=True)

hist = model.fit(embedding_train, Y_train, batch_size=64, epochs=140, shuffle=True, validation_split=0.1,
callbacks=[earlystop, checkpoint])
```

```
Train on 118 samples, validate on 14 samples
Epoch 1/140
118/118 [==============================] - 6s 55ms/step - loss: 1.6044 - acc: 0.1864 - val_loss: 1.6221 - val_acc: 0.0714
Epoch 2/140
118/118 [==============================] - 0s 481us/step - loss: 1.5673 - acc: 0.3729 - val_loss: 1.6278 - val_acc: 0.1429
Epoch 3/140
118/118 [==============================] - 0s 403us/step - loss: 1.5450 - acc: 0.3475 - val_loss: 1.6401 - val_acc: 0.1429
Epoch 4/140
118/118 [==============================] - 0s 426us/step - loss: 1.5057 - acc: 0.3729 - val_loss: 1.6565 - val_acc: 0.2857
Epoch 5/140
118/118 [==============================] - 0s 433us/step - loss: 1.5101 - acc: 0.3814 - val_loss: 1.6813 - val_acc: 0.2857
Epoch 6/140
118/118 [==============================] - 0s 427us/step - loss: 1.4965 - acc: 0.3729 - val_loss: 1.7013 - val_acc: 0.2857
Epoch 7/140
118/118 [==============================] - 0s 402us/step - loss: 1.4711 - acc: 0.4237 - val_loss: 1.7028 - val_acc: 0.2143
Epoch 8/140
118/118 [==============================] - 0s 389us/step - loss: 1.4534 - acc: 0.4237 - val_loss: 1.6837 - val_acc: 0.2143
Epoch 9/140
118/118 [==============================] - 0s 422us/step - loss: 1.4364 - acc: 0.3983 - val_loss: 1.6662 - val_acc: 0.1429
Epoch 10/140
118/118 [==============================] - 0s 396us/step - loss: 1.4141 - acc: 0.4153 - val_loss: 1.6356 - val_acc: 0.1429
Epoch 11/140
118/118 [==============================] - 0s 383us/step - loss: 1.3795 - acc: 0.4576 - val_loss: 1.6047 - val_acc: 0.1429
Epoch 12/140
118/118 [==============================] - 0s 444us/step - loss: 1.3551 - acc: 0.4237 - val_loss: 1.5814 - val_acc: 0.1429
Epoch 13/140
118/118 [==============================] - ETA: 0s - loss: 1.3192 - acc: 0.453 - 0s 375us/step - loss: 1.3078 - acc: 0.4831 - val_loss:
1.5465 - val_acc: 0.2143
Epoch 14/140
118/118 [==============================] - 0s 367us/step - loss: 1.2611 - acc: 0.5000 - val_loss: 1.5149 - val_acc: 0.2857
Epoch 15/140
118/118 [==============================] - 0s 444us/step - loss: 1.2041 - acc: 0.5169 - val_loss: 1.4960 - val_acc: 0.2857
Epoch 16/140
118/118 [==============================] - 0s 455us/step - loss: 1.1526 - acc: 0.5254 - val_loss: 1.4835 - val_acc: 0.2857
Epoch 17/140
118/118 [==============================] - 0s 377us/step - loss: 1.1075 - acc: 0.5678 - val_loss: 1.4712 - val_acc: 0.2857
Epoch 18/140
118/118 [==============================] - 0s 426us/step - loss: 1.0381 - acc: 0.6186 - val_loss: 1.4325 - val_acc: 0.2857
Epoch 19/140
118/118 [==============================] - 0s 454us/step - loss: 0.9866 - acc: 0.6102 - val_loss: 1.4072 - val_acc: 0.3571
Epoch 20/140
118/118 [==============================] - 0s 425us/step - loss: 0.9520 - acc: 0.6186 - val_loss: 1.3666 - val_acc: 0.3571
Epoch 21/140
118/118 [==============================] - 0s 427us/step - loss: 0.8636 - acc: 0.7458 - val_loss: 1.2602 - val_acc: 0.4286
Epoch 22/140
118/118 [==============================] - 0s 450us/step - loss: 0.8039 - acc: 0.7288 - val_loss: 1.3631 - val_acc: 0.3571
Epoch 23/140
118/118 [==============================] - 0s 439us/step - loss: 0.7588 - acc: 0.7458 - val_loss: 1.4408 - val_acc: 0.3571
Epoch 24/140
118/118 [==============================] - 0s 438us/step - loss: 0.6937 - acc: 0.7542 - val_loss: 1.2136 - val_acc: 0.7143
Epoch 25/140
118/118 [==============================] - 0s 372us/step - loss: 0.6269 - acc: 0.8136 - val_loss: 1.1660 - val_acc: 0.6429
Epoch 26/140
118/118 [==============================] - 0s 379us/step - loss: 0.6456 - acc: 0.8390 - val_loss: 1.2208 - val_acc: 0.5714
Epoch 27/140
118/118 [==============================] - 0s 381us/step - loss: 0.5876 - acc: 0.8305 - val_loss: 1.1360 - val_acc: 0.7143
Epoch 28/140
118/118 [==============================] - 0s 351us/step - loss: 0.4929 - acc: 0.8729 - val_loss: 1.4284 - val_acc: 0.5714
Epoch 29/140
118/118 [==============================] - 0s 431us/step - loss: 0.5109 - acc: 0.8559 - val_loss: 1.3028 - val_acc: 0.5714
Epoch 30/140
118/118 [==============================] - 0s 374us/step - loss: 0.4082 - acc: 0.8644 - val_loss: 1.0397 - val_acc: 0.6429
Epoch 31/140
118/118 [==============================] - 0s 340us/step - loss: 0.4023 - acc: 0.8898 - val_loss: 1.1056 - val_acc: 0.5714
Epoch 32/140
118/118 [==============================] - 0s 340us/step - loss: 0.4202 - acc: 0.8814 - val_loss: 1.2325 - val_acc: 0.5714
Epoch 33/140
118/118 [==============================] - 0s 382us/step - loss: 0.3447 - acc: 0.9237 - val_loss: 1.0746 - val_acc: 0.7143
Epoch 34/140
118/118 [==============================] - 0s 349us/step - loss: 0.3762 - acc: 0.8559 - val_loss: 1.0410 - val_acc: 0.6429
Epoch 35/140
118/118 [==============================] - 0s 355us/step - loss: 0.2948 - acc: 0.8814 - val_loss: 1.2342 - val_acc: 0.5714
Epoch 36/140
118/118 [==============================] - 0s 378us/step - loss: 0.3227 - acc: 0.9068 - val_loss: 1.0623 - val_acc: 0.5714
Epoch 37/140
118/118 [==============================] - 0s 357us/step - loss: 0.3035 - acc: 0.9153 - val_loss: 1.1036 - val_acc: 0.6429
Epoch 38/140
118/118 [==============================] - 0s 372us/step - loss: 0.3129 - acc: 0.8983 - val_loss: 1.4381 - val_acc: 0.5714
Epoch 39/140
118/118 [==============================] - 0s 393us/step - loss: 0.2565 - acc: 0.8983 - val_loss: 1.3645 - val_acc: 0.5000
Epoch 40/140
118/118 [==============================] - 0s 370us/step - loss: 0.1954 - acc: 0.9237 - val_loss: 0.9421 - val_acc: 0.7143
Epoch 41/140
```

```
118/118 [==============================] - 0s 372us/step - loss: 0.2530 - acc: 0.9237 - val_loss: 0.9734 - val_acc: 0.7143
Epoch 42/140
118/118 [==============================] - 0s 380us/step - loss: 0.2503 - acc: 0.8983 - val_loss: 1.1281 - val_acc: 0.6429
Epoch 43/140
118/118 [==============================] - 0s 372us/step - loss: 0.1855 - acc: 0.9322 - val_loss: 1.3988 - val_acc: 0.5714
Epoch 44/140
118/118 [==============================] - 0s 358us/step - loss: 0.1777 - acc: 0.9492 - val_loss: 1.3468 - val_acc: 0.5714
```

In [55]:

```
model.load_weights("best_weights_2.h5")
print(model.metrics_names)
print(model.evaluate(embedding_test, Y_test))

# We can see that the accuracy has increased but with very less percentage
```

```
['loss', 'acc']
56/56 [==============================] - 0s 195us/step
[1.5573524066380091, 0.660714294229235]
```

3) Predicting Emojis

In [58]:

```
y_pred = model.predict_classes(embedding_test)
print(y_pred.shape)
print(y_pred[:5])
```

```
(56,)
[4 3 2 0 2]
```

In [61]:

```
for i in range(embedding_test.shape[0]):
    print("Sentence: "+X_test[i])
    print("Actual Emoji: "+emoji.emojize(emoji_dictionary[str(np.argmax(Y_test[i]))]))
    print("Predicted Emoji: "+emoji.emojize(emoji_dictionary[str(y_pred[i])])+"\n")
```

```
Sentence: I want to eat
Actual Emoji: 🍴
Predicted Emoji: 🍴

Sentence: he did not answer
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: he got a raise
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: she got me a present
Actual Emoji: ♡
Predicted Emoji: ♡

Sentence: ha ha ha it was so funny
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: he is a good friend
Actual Emoji: ♡
Predicted Emoji: 😀

Sentence: I am upset
Actual Emoji: ♡
Predicted Emoji: 😞

Sentence: We had such a lovely dinner tonight
Actual Emoji: ♡
Predicted Emoji: 😀

Sentence: where is the food
Actual Emoji: 🍴
Predicted Emoji: 🍴

Sentence: Stop making this joke ha ha ha
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: where is the ball
Actual Emoji: ⚾
Predicted Emoji: ⚾

Sentence: work is hard
Actual Emoji: 😞
Predicted Emoji: 😀

Sentence: This girl is messing with me
Actual Emoji: 😞
Predicted Emoji: ♡

Sentence: are you serious ha ha
Actual Emoji: 😀
Predicted Emoji: 😞

Sentence: Let us go play baseball
Actual Emoji: ⚾
Predicted Emoji: ⚾

Sentence: This stupid grader is not working
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: work is horrible
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: Congratulation for having a baby
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: stop messing around
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: any suggestions for dinner
Actual Emoji: 🍴
Predicted Emoji: 🍴

Sentence: I love taking breaks
Actual Emoji: ♡
Predicted Emoji: ♡
```

```
Sentence: you brighten my day
Actual Emoji: 😀
Predicted Emoji: ♡

Sentence: I boiled rice
Actual Emoji: 🍴
Predicted Emoji: 🍴

Sentence: she is a bully
Actual Emoji: 😞
Predicted Emoji: ♡

Sentence: Why are you feeling bad
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: I am upset
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: I worked during my birthday
Actual Emoji: 😞
Predicted Emoji: 😀

Sentence: My grandmother is the love of my life
Actual Emoji: ♡
Predicted Emoji: ♡

Sentence: enjoy your break
Actual Emoji: 😀
Predicted Emoji: ⚾

Sentence: valentine day is near
Actual Emoji: ♡
Predicted Emoji: 😀

Sentence: I miss you so much
Actual Emoji: ♡
Predicted Emoji: ♡

Sentence: throw the ball
Actual Emoji: ⚾
Predicted Emoji: ⚾

Sentence: My life is so boring
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: she said yes
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: will you be my valentine
Actual Emoji: ♡
Predicted Emoji: ♡

Sentence: he can pitch really well
Actual Emoji: ⚾
Predicted Emoji: ⚾

Sentence: dance with me
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: I am starving
Actual Emoji: 🍴
Predicted Emoji: 😞

Sentence: See you at the restaurant
Actual Emoji: 🍴
Predicted Emoji: 🍴

Sentence: I like to laugh
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: I will go dance
Actual Emoji: 😀
Predicted Emoji: 😀

Sentence: I like your jacket
Actual Emoji: 😀
Predicted Emoji: ♡
```

```
Sentence: i miss her
Actual Emoji: ♡
Predicted Emoji: ♡

Sentence: what is your favorite baseball game
Actual Emoji: ⚾
Predicted Emoji: ⚾

Sentence: Good job
Actual Emoji: 😃
Predicted Emoji: 😃

Sentence: I love to the stars and back
Actual Emoji: ♡
Predicted Emoji: 😃

Sentence: What you did was awesome
Actual Emoji: 😃
Predicted Emoji: 😞

Sentence: ha ha ha lol
Actual Emoji: 😃
Predicted Emoji: 😃

Sentence: I want to joke
Actual Emoji: 😃
Predicted Emoji: ♡

Sentence: go away
Actual Emoji: 😞
Predicted Emoji: ⚾

Sentence: yesterday we lost again
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: family is all I have
Actual Emoji: ♡
Predicted Emoji: ♡

Sentence: you are failing this exercise
Actual Emoji: 😞
Predicted Emoji: 😞

Sentence: Good joke
Actual Emoji: 😃
Predicted Emoji: 😃

Sentence: You totally deserve this prize
Actual Emoji: 😃
Predicted Emoji: 😞

Sentence: I did not have breakfast
Actual Emoji: 😞
Predicted Emoji: ♡
```

In [ ]:

```
y_test = np.zeros(y_pred.shape[0],)
for i in range(y_pred.shape[0]):
    y_test[i] = np.argmax(Y_test[i])
print(y_test)
```

In [77]:

```
print("Model Accuracy: {0:.2f}%".format(np.sum(y_pred == y_test)/y_pred.shape[0]*100))
```

```
Model Accuracy: 66.07%
```

Add confusion matrix in this file

https://www.geeksforgeeks.org/confusion-matrix-machine-learning/ (https://www.geeksforgeeks.org/confusion-matrix-machine-learning/)