LOVELY PROFESSIONAL UNIVERSITY

PROJECT REPORT ON

TWITTER SENTIMENT ANALYSIS

SUBJECT:	**INT-248**
SUBMITTED BY:	**ANMOL BAUNTHIYAL**
REGISTRATION NO:	**11908907**
SECTION:	**KM053**
UNDER THE GUIDANCE OF:	**MS. JASPREET KAUR**

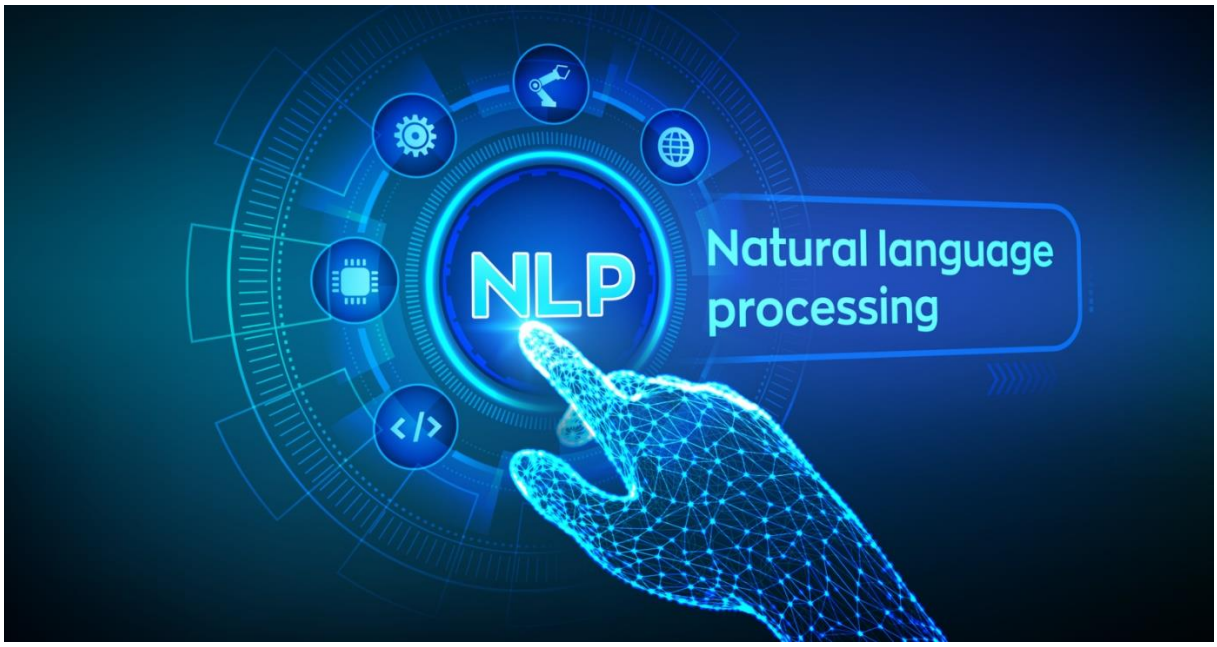
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

LOVELY PROFESSIONAL UNIVERSITY

Contents

Introduction

Sentiment analysis (also known as opinion mining) is a type of natural language processing (NLP) technique that determines whether data is positive, negative, or neutral. Sentiment analysis on textual data is frequently used to assist businesses in monitoring brand and product sentiment in customer feedback and understanding customer needs.



Sentiment analysis is widely popular among business to extract the customer behavior. It can be used on various platforms but in the project, we will focus on Twitter Sentiment Analysis.

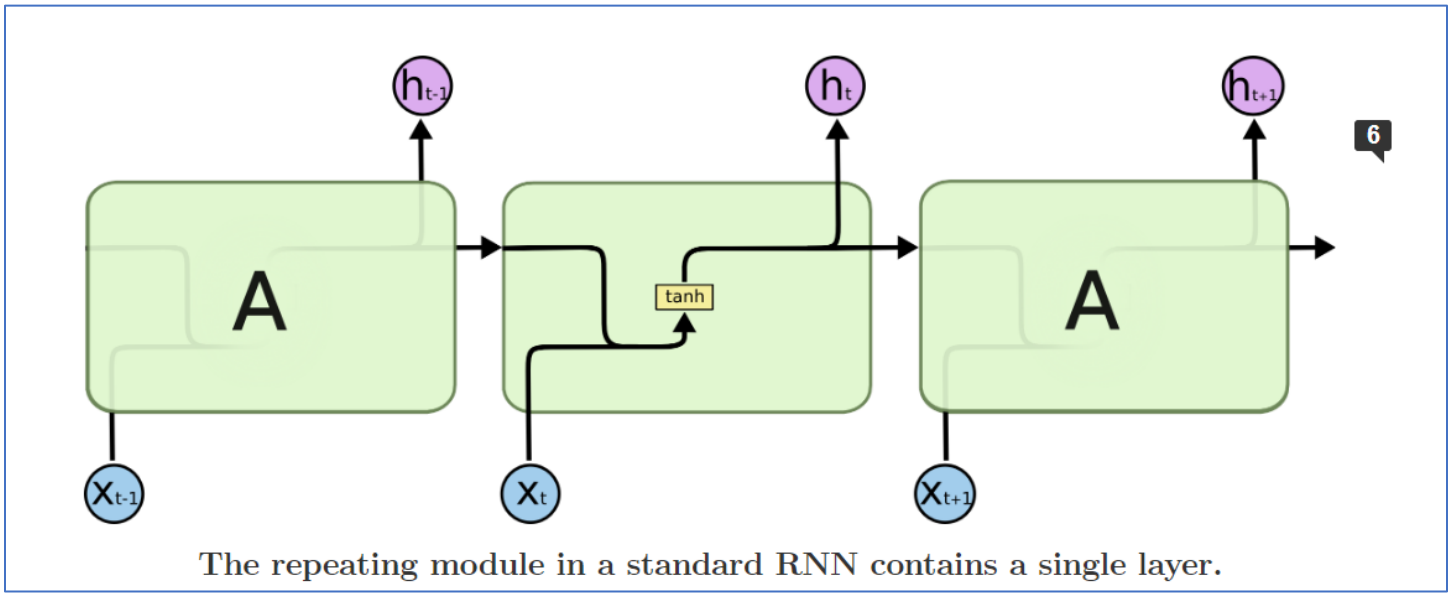
Though, this analysis can be performed using various techniques, but I have used LSTM (Long Short-Term Memory)

RNN (Recurrent Neural Networks)

Humans do not start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You do not throw everything away and start thinking from scratch again. Your thoughts have persistence.

Traditional neural networks cannot do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It is unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

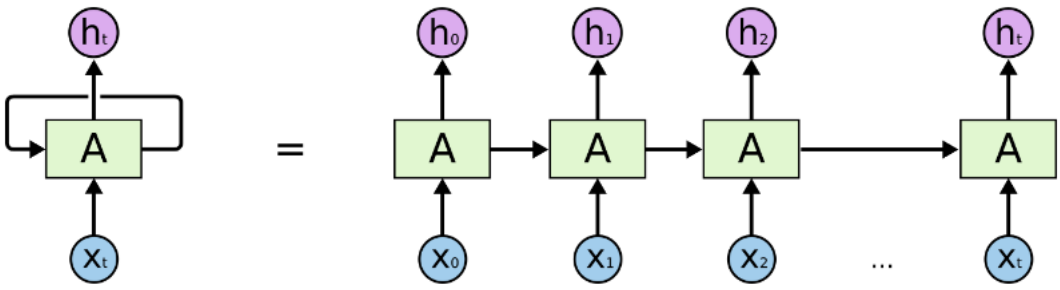
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



The repeating module in a standard RNN contains a single layer.

In the above diagram, a chunk of neural network, AA, looks at some input xtxt and outputs a value htht. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they are not all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



An unrolled recurrent neural network.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They are the natural architecture of neural network to use for such data.

And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on. I'll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy's excellent blog post, The Unreasonable Effectiveness of Recurrent Neural Networks. But they really are amazing.

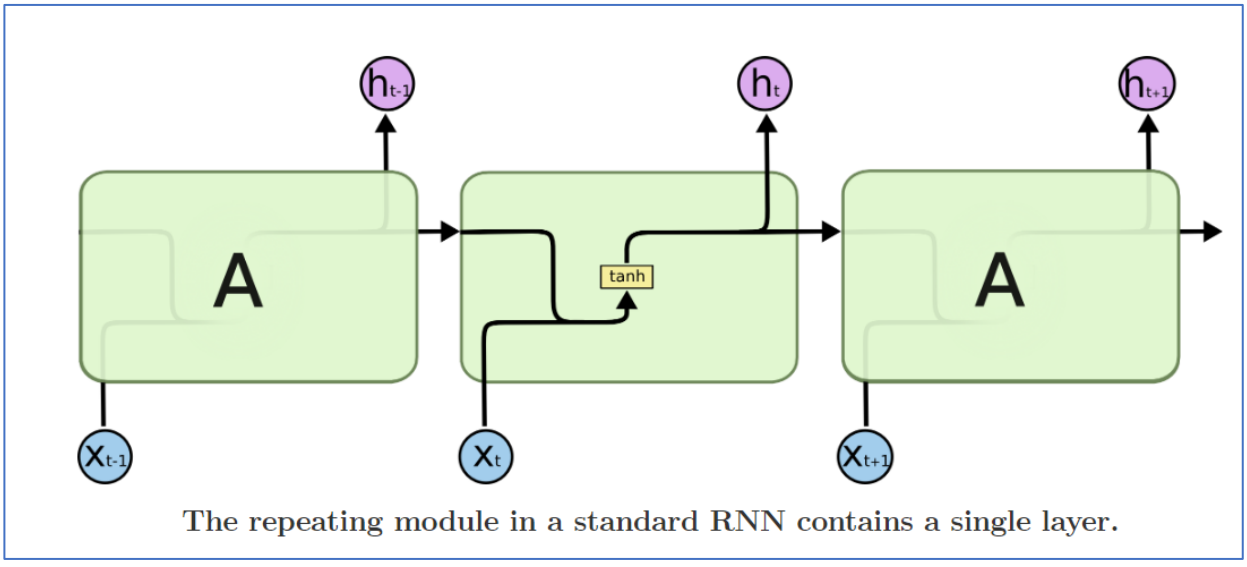
Essential to these successes is the use of "LSTMs," a very special kind of recurrent neural network which works, for many tasks, much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

LSTM Networks

Long Short-Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.[1] They work tremendously well on a large variety of problems, and are now widely used.

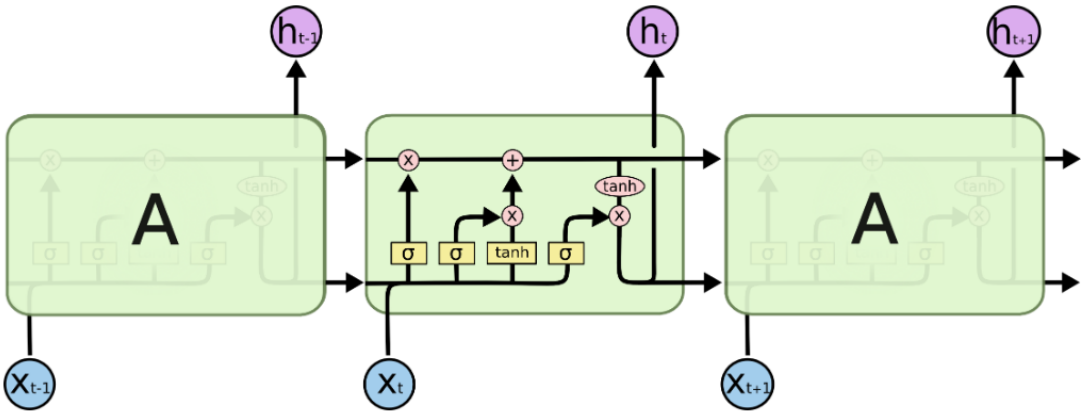
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

LSTMs were a big step in what we can accomplish with RNNs. It is natural to wonder: is there another big step? A common opinion among researchers is: "Yes! There is a next step and its attention!" The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, *et al.* (2015) do exactly this – it might be a fun starting point if you want to explore attention! There has been several really exciting results using attention, and it seems like a lot more are around the corner.

Use-Cases

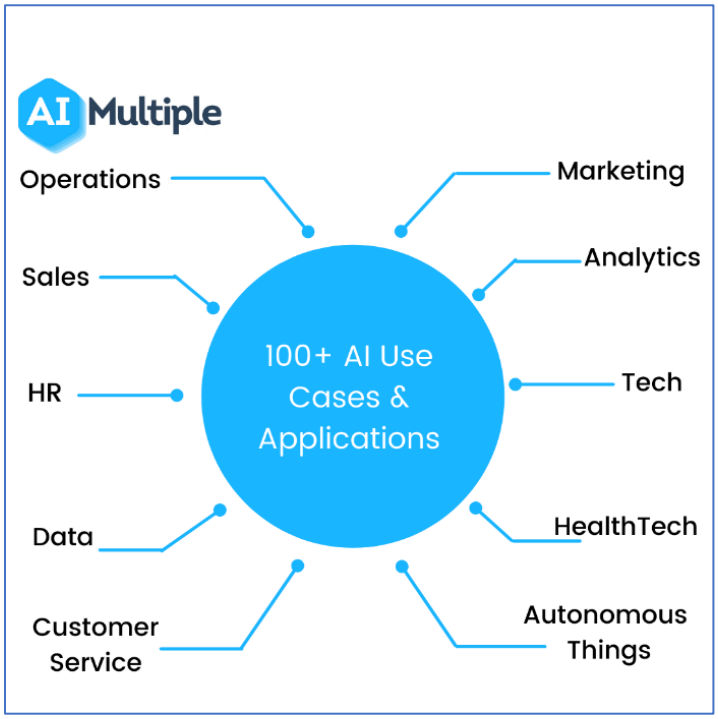
Social media services such as Facebook and Twitter have grown at an exponential rate over the last decade.

Facebook, Twitter, LinkedIn, and numerous other sites. Companies and organizations have recognized the value of these platforms as a source of information that allow them to interact with their customers and learn more about them However because of the large number of users, posts, comments, messages, and other form.

Tracking a user's overall appreciation of a brand can be extremely useful in terms of interaction is complex. Twitter, for example, is one of the most popular social media platforms.

Today has approximately 350 million active users who post approximately 500 million tweets per day.

Deep Learning and other Machine Learning (ML) techniques, however, can provide an automated method to assist in the processing and extraction of useful data from these a great deal of information.



Dataset

The dataset which is used in this project is taken from Kaggle.

It is a labelled dataset and have over 17000 rows and 2 columns after preprocessing it.

The dataset looks as follows:

	non_punctuated_tweets	sentiment
0	put hours game last year still holds well blas...	positive
1	story great terrible	positive
2	jaedo event bought tickets nd quarter finally ...	negative
3	year going super dynamic bc im graduating uni ...	positive
4	literally tough year none plan going right pre...	positive
...	...	...
17769	really rough one mental health truly feel many...	positive
17770	happy new year everyone except boy dated coupl...	positive
17771	crazy bought car moved shot porn butt plug bum...	negative
17772	think descent art year despite able use tablet...	neutral
17773	hi active moment wanted say happy new year acc...	positive

17774 rows × 2 columns

Code

```
# -*- coding: utf-8 -*-

Mounting the Google Drive for fetching the train/test data.
"""

from google.colab import drive
drive.mount('/content/drive')

"""Reading data from Drive."""

df = pd.read_csv("/content/drive/MyDrive/twitter/Tweets.csv")

# Importing required libraries
import nltk
import numpy as np
import pandas as pd
from nltk.corpus import stopwords
from textblob import Word
from sklearn.preprocessing import LabelEncoder
from collections import Counter
import wordcloud
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from tensorflow.keras.callbacks import EarlyStopping
!pip install tensorflow-addons==0.8.3
!pip install tensorflow==2.2.0-rc3
import tensorflow_addons as tfa
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

Data = df[['non_punctuated_tweets','sentiment']]

Data

"""Tokenization"""

Data['tokenized_tweets'] = Data.apply(lambda row :
nltk.word_tokenize(str(row['non_punctuated_tweets'])),axis = 1)

Data
```

```
"""Lemmatization"""

def lemma(data):
    return " ".join([Word(word).lemmatize() for word in data])
Data['lemmatized_tweets']= Data['tokenized_tweets'].apply(lambda x: lemma(x))

Data

"""Converting Lemmatized Tweets to Vectors.

"""

# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 500
# Max number of words in each tweets.
MAX_SEQUENCE_LENGTH = 50
# This is fixed.
EMBEDDING_DIM = 100

tokenizer = Tokenizer(num_words=MAX_NB_WORDS, split=' ')
tokenizer.fit_on_texts(Data.lemmatized_tweets.values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

X = tokenizer.texts_to_sequences(Data.lemmatized_tweets.values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)

"""One Hot Encoding of the Sentiment Column.

"""

Y = pd.get_dummies(Data.sentiment).values
print('Shape of label tensor:', Y.shape)

Y

"""Train Test Split."""

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)

"""Modelling."""

from numpy.random import seed
seed(1)
import tensorflow
tensorflow.random.set_seed(2)
```

```
f1 = tfa.metrics.F1Score(36,'micro' or 'macro')
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy',f1])

epochs = 7
batch_size = 64

history_1 = model.fit(X_train, Y_train, epochs=epochs,
batch_size=batch_size,validation_split=0.1,callbacks=[EarlyStopping(monitor='val_loss',
patience=3, min_delta=0.0001)])

accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))

plt.figure(figsize=(8, 8))
plt.title('Loss')
plt.plot(history_1.history['loss'], label='train')
plt.plot(history_1.history['val_loss'], label='test')
plt.legend()
plt.show()

plt.figure(figsize=(8, 8))
plt.title('Accuracy')
plt.plot(history_1.history['accuracy'], label='train')
plt.plot(history_1.history['val_accuracy'], label='test')
plt.legend()
plt.show()
tweet = [""]
seq = tokenizer.texts_to_sequences(tweet)
padded = pad_sequences(seq, maxlen=MAX_SEQUENCE_LENGTH)
pred = model.predict(padded)
labels = ['negative','neutral','positive']
print(pred, labels[np.argmax(pred)])




##Today was the worst day of my life as i got dreched while going for my interview. -ve
##Hello, how you doinnn?? =
##Wow what a lovely day it is +ve
```

References

1. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
2. https://en.wikipedia.org/wiki/Recurrent_neural_network
3. https://builtin.com/data-science/recurrent-neural-networks-and-lstm#:~:text=Long%20short%2Dterm%20memory%20(LSTM)%20networks%20are%20an%20extension,enough%20to%20impact%20the%20output.