CSC 591 - ADBI
Capstone Project

Text Classification: Comparative Analysis of Different Deep Neural Network Architectures

Team:

a) Venkata Pasumarty
b) LV Raju Nadimpalli
c) Krishnachaitanya Pullakandam

CNN: Convolutional Neural Networks

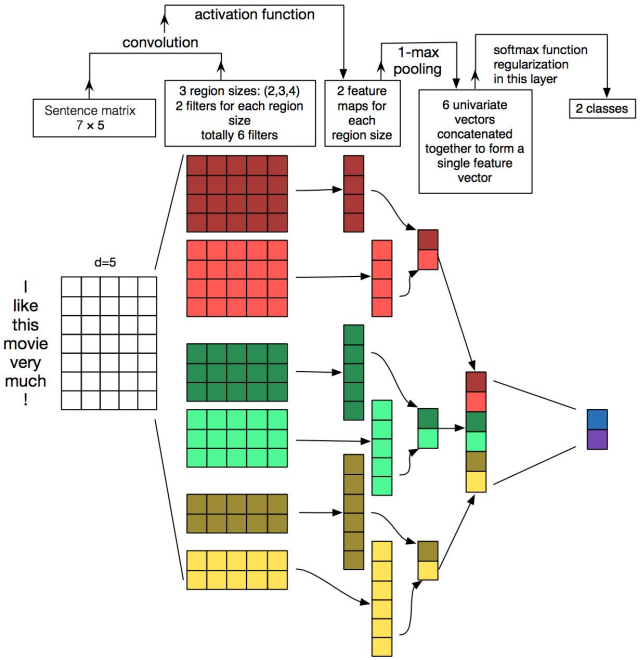
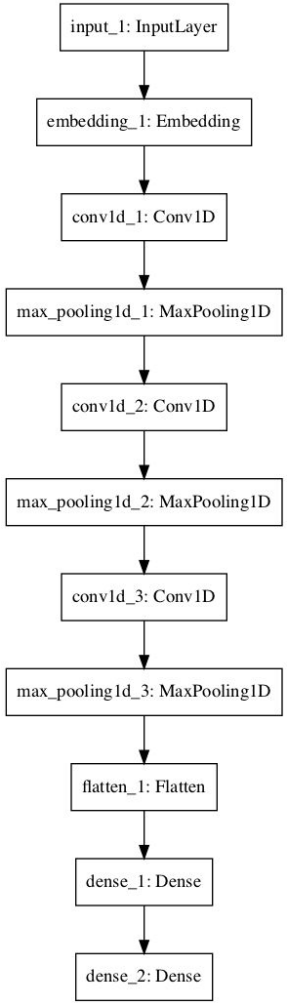


Image Reference : http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/



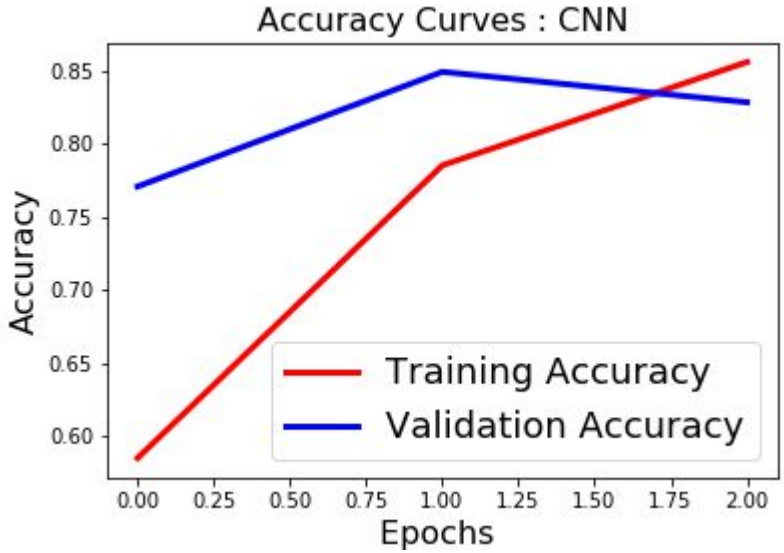
CNN Architecture for this Project

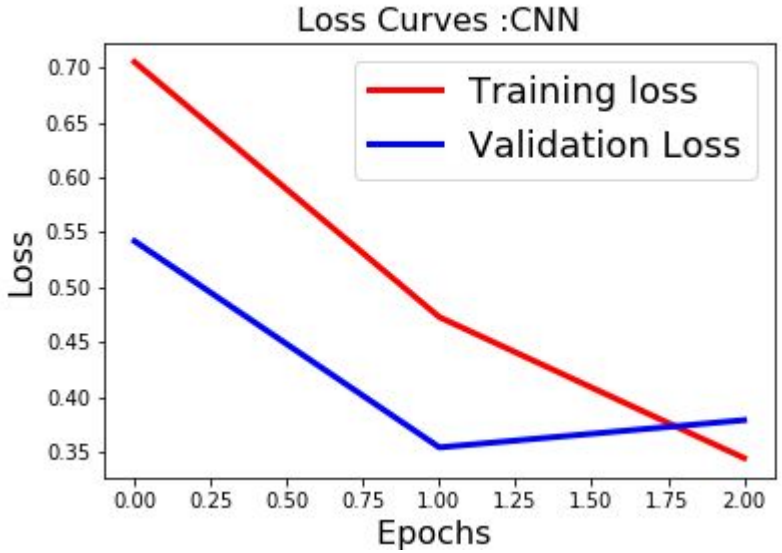
```
def clean_str(string):
    string = re.sub(r"\\", "", string)
    string = re.sub(r"\'", "", string)
    string = re.sub(r"\"", "", string)
    return string.strip().lower()

texts = [];labels = []

for i in range(df.message.shape[0]):
    text = BeautifulSoup(df.message[i])
    texts.append(clean_str(str(text.get_text().encode())))

for i in df['class']:
    labels.append(i)
```

```
data_train = pd.read_csv('labeledTrainData.tsv', sep='\t')
print (data_train.shape)

texts = []
labels = []

for idx in range(data_train.review.shape[0]):
    text = BeautifulSoup(data_train.review[idx], "html.parser")
    texts.append(clean_str(text.get_text().encode('ascii','ignore')))
    labels.append(data_train.sentiment[idx])


tokenizer = Tokenizer(nb_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]

print('Number of positive and negative reviews in traing and validation set ')
print (y_train.sum(axis=0))
print (y_val.sum(axis=0))
```

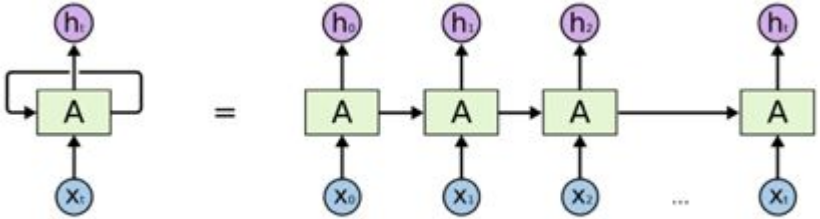
```
# ---------------------------------------------------------------------------------------------------------
# Loading GloVe Word Vectors Pretrained on Wikipedia data - 6B tokens, 400K vocab, uncased, 100d vectors
# ---------------------------------------------------------------------------------------------------------

GLOVE_DIR = ""
embeddings_index = {}
f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Total %s word vectors in Glove 6B 100d.' % len(embeddings_index))

# ------------------------------------------------
# Embedding Layer CNN Model
# ------------------------------------------------
embedding_matrix = np.random.random((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=True)
```

```
# ---------------------------------------------------
# Train / Test CNN Model
# ---------------------------------------------------

# Build
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
l_cov1= Conv1D(128, 5, activation='relu')(embedded_sequences)
l_pool1 = MaxPooling1D(5)(l_cov1)
l_cov2 = Conv1D(128, 5, activation='relu')(l_pool1)
l_pool2 = MaxPooling1D(5)(l_cov2)
l_cov3 = Conv1D(128, 5, activation='relu')(l_pool2)
l_pool3 = MaxPooling1D(35)(l_cov3)  # global max pooling
l_flat = Flatten()(l_pool3)
l_dense = Dense(128, activation='relu')(l_flat)
preds = Dense(2, activation='softmax')(l_dense)

model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])
```

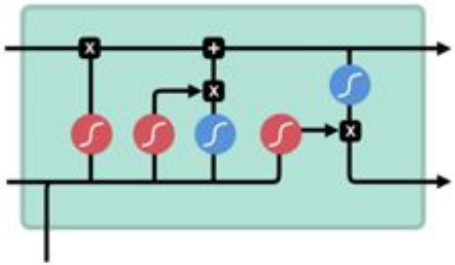

Accuracy Curves : CNN
Accuracy
Epochs
Training Accuracy
Validation Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

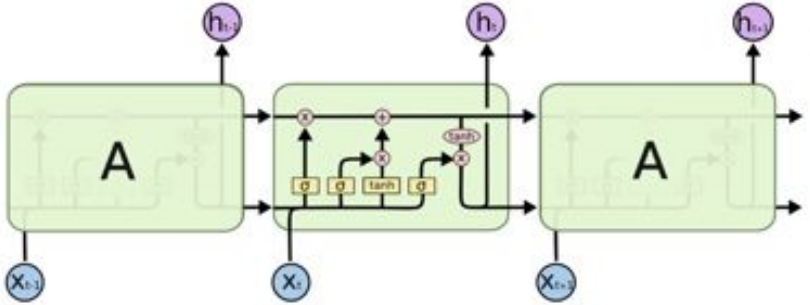

Loss Curves :CNN
Loss
Epochs
Training loss
Validation Loss
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

RNN: Recurrent Neural Networks



An unrolled recurrent neural network.

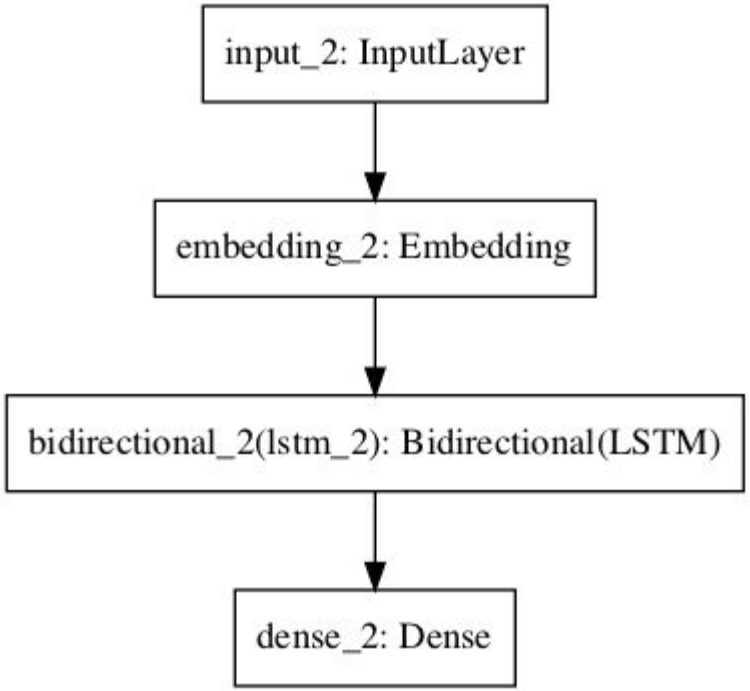




The repeating module in an LSTM contains four interacting layers.

LSTM Cell and It's Operations

Image Reference : http://colah.github.io/posts/2015-08-Understanding-LSTMs/



RNN Architecture for this Model

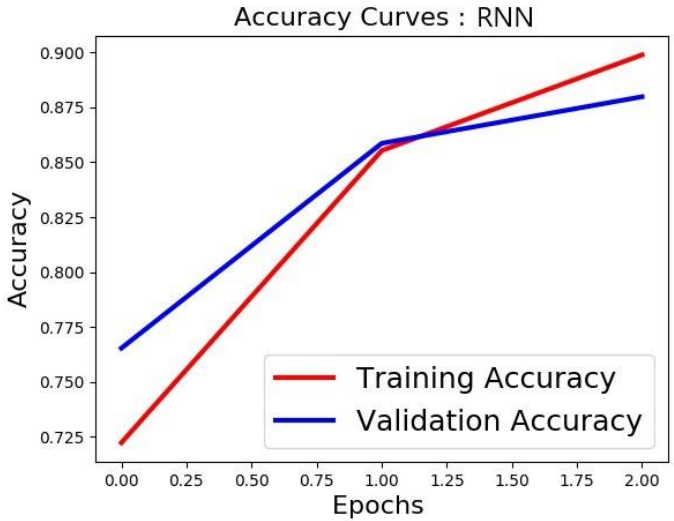
```
id	sentiment	review
"5814_8"	1	"With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary here and there, watched The Wiz and watched Moonwalker again. Maybe i just want to get a certain insight into this guy who i thought was really cool in the eighties just to maybe make up my mind whether he is guilty or innocent. Moonwalker is part biography, part feature film which i remember going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's feeling towards the press and also the obvious message of drugs are bad m'kay.<br /><br />Visually impressive but of course this is all about Michael Jackson so unless you remotely like MJ in anyway then you are going to hate this and find it boring. Some may call MJ an egotist for consenting to the making of this movie BUT MJ and most of his fans would say that he made it for the fans which if true is really nice of him.<br /><br />The actual feature film bit when it finally starts is only on for 20 minutes or so excluding the Smooth Criminal sequence and Joe Pesci is convincing as a psychopathic all powerful drug lord. Why he wants MJ dead so bad is beyond me. Because MJ overheard his plans? Nah, Joe Pesci's character ranted that he wanted people to know it is he who is supplying drugs etc so i dunno, maybe he just hates MJ's music.<br /><br />Lots of cool things in this like MJ turning into a car and a robot and the whole Speed Demon sequence. Also, the director must have had the patience of a saint when it came to filming the kiddy Bad sequence as usually directors hate working with one kid let alone a whole bunch of them performing a complex dance scene.<br /><br />Bottom line, this movie is for people who like MJ on one level or another (which i think is most people). If not, then stay away. It does try and give off a wholesome message and ironically MJ's bestest buddy in this movie is a girl! Michael Jackson is truly one of the most talented people ever to grace this planet but is he guilty? Well, with all the attention i've gave this subject....hmmm well i don't know because people can be different behind closed doors, i know this for a fact. He is either an extremely nice but stupid guy or one of the most sickest liars. I hope he is not the latter."
"2381_9"	1	"\"The Classic War of the Worlds\" by Timothy Hines is a very entertaining film that obviously goes to great effort and lengths to faithfully recreate H. G. Wells' classic book. Mr. Hines succeeds in doing so. I, and those who watched his film with me, appreciated the fact that it was not the standard, predictable Hollywood fare that comes out every year, e.g. the Spielberg version with Tom Cruise that had only the slightest resemblance to the book. Obviously, everyone looks for different things in a movie. Those who envision themselves as amateur \"critics\" look only to criticize everything they can. Others rate a movie on more important bases like being entertained, which is why most people never agree with the \"critics\". We enjoyed the effort Mr. Hines put into being faithful to H.G. Wells' classic novel, and we found it to be very entertaining. This made it easy to overlook what the \"critics\" perceive to be its shortcomings."
```

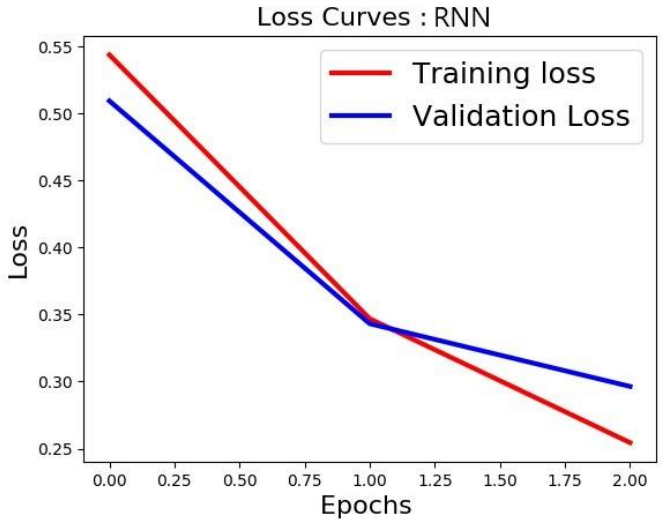
```
#!/usr/bin/env python

"""cnn.py: Text classification using cnn"""
# __author__      = "Venkata Pasumarty"

import os
import re
import sys
import pickle
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
from collections import defaultdict

os.environ['KERAS_BACKEND']='theano'

from keras.preprocessing.text import Tokenizer
from keras.utils.np_utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

from keras.models import Model
from keras.layers import Embedding
from keras.layers import Dense, Input, Flatten
from keras.layers import Conv1D, MaxPooling1D, Embedding, Dropout

# Maximum words per sentence
MAX_SEQUENCE_LENGTH = 1000
# Max vocabulary size
MAX_NB_WORDS = 20000
# Dimension of GloVe
EMBEDDING_DIM = 100
# Training - Validation split
VALIDATION_SPLIT = 0.2
```

```
def clean_str(string):
    """
    Tokenization/string cleaning for dataset
    Every dataset is lower cased except
    """
    string = string.decode('utf-8')
    string = re.sub(r"\\", "", string)
    string = re.sub(r"\'", "", string)
    string = re.sub(r"\"", "", string)
    return string.strip().lower()

# ---------------------------------------------------
# Load Kaggle IMDB Dataset
# ---------------------------------------------------
data_train = pd.read_csv('labeledTrainData.tsv', sep='\t')
print (data_train.shape)

texts = []
labels = []

for idx in range(data_train.review.shape[0]):
    text = BeautifulSoup(data_train.review[idx],"html.parser")
    texts.append(clean_str(text.get_text().encode('ascii','ignore')))
    labels.append(data_train.sentiment[idx])


tokenizer = Tokenizer(nb_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])
```

```
x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]

print('Traing and validation set number of positive and negative reviews')
print (y_train.sum(axis=0))
print (y_val.sum(axis=0))


# ---------------------------------------------------------------------------------------------------------------
# Loading GloVe Word Vectors Pretrained on Wikipedia data - 6B tokens, 400K vocab, uncased, 100d vectors
# ---------------------------------------------------------------------------------------------------------------
GLOVE_DIR = ""
embeddings_index = {}
f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Total %s word vectors.' % len(embeddings_index))


# -------------------------------------------------
# Embedding Layer RNN Model
# -------------------------------------------------
embedding_matrix = np.random.random((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=True)
```

```
# ----------------------------------------------------
# Train / Test RNN Model
# ----------------------------------------------------
# Build
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
l_lstm = Bidirectional(LSTM(100))(embedded_sequences)
preds = Dense(2, activation='softmax')(l_lstm)
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])
```


Accuracy Curves : RNN
Accuracy
Epochs
Training Accuracy
Validation Accuracy
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Loss Curves : RNN
Loss
Epochs
Training loss
Validation Loss
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

HAN: Hierarchical Attention Networks

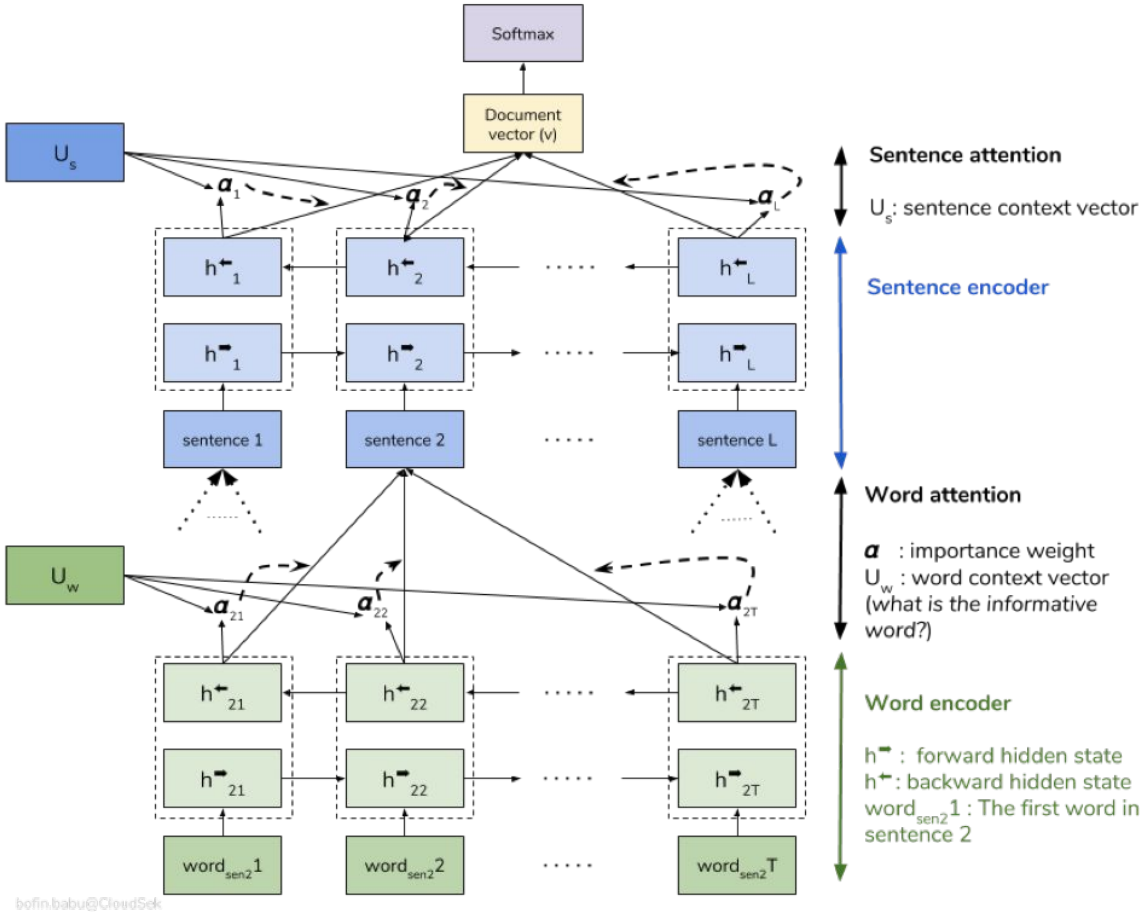
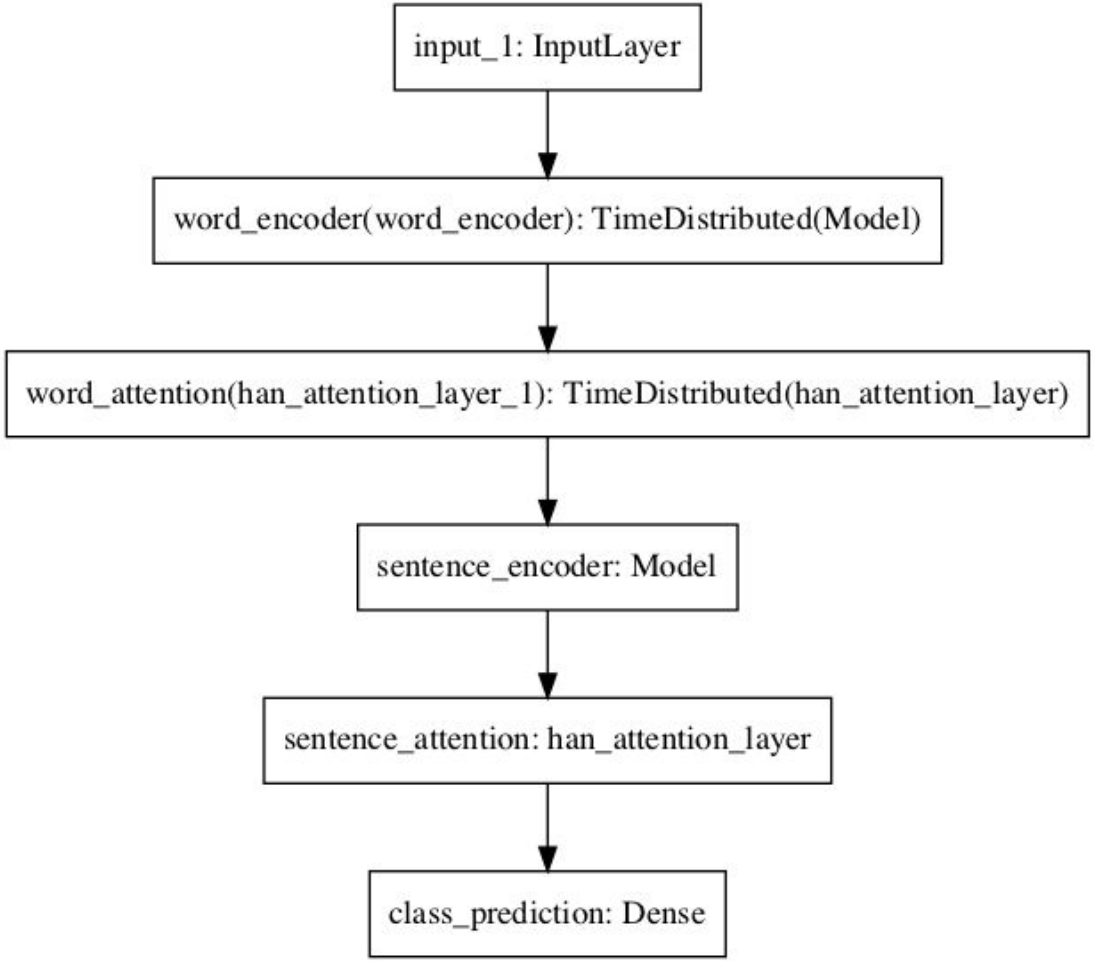
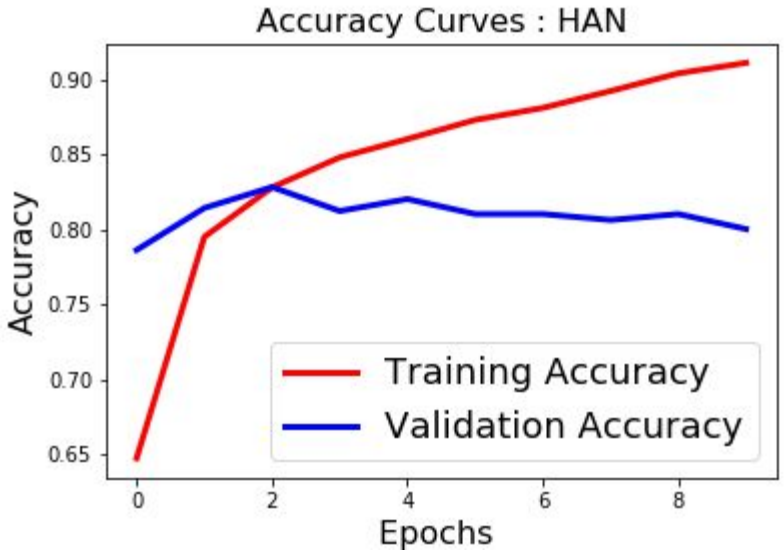


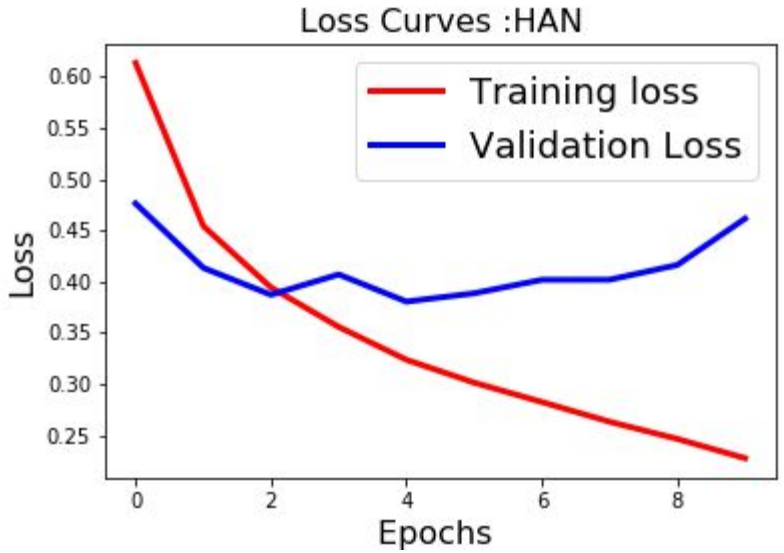
Image Reference :
https://arxiv.org/pdf/1506.01057v2.pdf



HAN Architecture for this Project

```
# ------------------------------------------------
# Create HAN Model
# ------------------------------------------------
class han(keras.models.Model):
    def __init__(self, max_words, max_sents, output_size, embed_matrix,
                 word_encode_dim=200, sent_encode_dim=200,
                 name="Hierarchical_Attention_Network"):
        self.max_words = max_words
        self.max_sents = max_sents
        self.output_size = output_size
        self.embed_matrix = embed_matrix
        self.word_encode_dim = word_encode_dim
        self.sent_encode_dim = sent_encode_dim

        in_tensor, out_tensor = self.build_network()

        super(han, self).__init__(inputs=in_tensor, outputs=out_tensor, name=name)

    def build_word_encoder(self, max_words, embed_matrix, encode_dim=200):
        vocab_size = embed_matrix.shape[0]
        embed_dim = embed_matrix.shape[1]
        embed_layer = keras.layers.Embedding(vocab_size, embed_dim, weights=[embed_matrix],
                                             input_length=max_words, trainable=False)
        sent_input = keras.layers.Input(shape=(max_words,), dtype="int32")
        embed_sents = embed_layer(sent_input)
        encode_sents = keras.layers.Bidirectional(keras.layers.GRU(int(encode_dim / 2)))(
            embed_sents)
        return keras.Model(inputs=[sent_input], outputs=[encode_sents], name="word_encoder")

    def build_sent_encoder(self, max_sents, summary_dim, encode_dim=200):
        text_input = keras.layers.Input(shape=(max_sents, summary_dim))
        encode_sents = keras.layers.Bidirectional(keras.layers.GRU(int(encode_dim / 2)))(
            text_input)
        return keras.Model(inputs=[text_input], outputs=[encode_sents], name="sentence_encoder")

    def build_network(self):
        in_tensor = keras.layers.Input(shape=(self.max_sents, self.max_words))
        word_encoder = self.build_word_encoder(self.max_words, self.embed_matrix, self.word_encode_dim)
        word_rep = keras.layers.TimeDistributed(word_encoder, name="word_encoder")(in_tensor)
        sentence_rep = keras.layers.TimeDistributed(han_attention_layer(), name="word_attention")(word_rep)
        doc_rep = self.build_sent_encoder(self.max_sents, self.word_encode_dim, self.sent_encode_dim)(sentence_rep)
        doc_summary = han_attention_layer(name="sentence_attention")(doc_rep)
        out_tensor = keras.layers.Dense(self.output_size, activation="softmax", name="class_prediction")(doc_summary)
        return in_tensor, out_tensor
```


Accuracy Curves : HAN
Accuracy
Epochs
Training Accuracy
Validation Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0
2
4
6
8


Loss Curves :HAN
Loss
Epochs
Training loss
Validation Loss
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
2
4
6
8