CAPSTONE PROJECT:TWITTER SENTIMENT ANALYSIS

ARYAMAAN PANDEY

JULY 17,2020

Abstract

In this report, we address the problem of sentiment classification on twitter dataset. We use a number of machine learning and deep learning methods to perform sentiment analysis. In the end, we use a majority vote with 4 of our best models to achieve the F1 score of 0.65 on Kaggle leaderboard.

1 : Problem Statement

Twitter is a popular social networking website where members create and interact with messages known as “tweets”. This serves as a mean for individuals to express their thoughts or feelings about different subjects. Various different parties such as consumers and marketers have done sentiment analysis on such tweets to gather insights into products or to conduct market analysis. Furthermore, with the recent advancements in machine learning algorithms, we are able improve the accuracy of our sentiment analysis predictions.

In this report, we will attempt to conduct sentiment analysis on “tweets” using various different machine learning algorithms. We attempt to classify the polarity of the tweet where it is either positive, negative or neutral. If the tweet has all positive, negative and neutral elements, the more dominant sentiment should be picked as the final label.

We use the dataset from Kaggle which was crawled and labelled positive/negative/neutral. The data provided comes with emoticons, usernames and hashtags which are required to be processed and converted into a standard form. We also need to extract useful features from the text such unigrams and bigrams which is a form of representation of the “tweet”.

We use various machine learning algorithms to conduct sentiment analysis using the extracted features. However, just relying on individual models did not give a high accuracy so we pick the top few models to generate a model ensemble. Finally, we report our experimental results and findings at the end.

2 :Data Description

The data given is in the form of a comma-separated values files with tweets and their corresponding sentiments. The training dataset is a csv file of type tweet_id , sentiment , tweet where the tweet_id is a unique integer identifying the tweet, sentiment is either positive , negative or neutral and tweet is the tweet enclosed in "". Similarly, the test dataset is a csv file of type tweet_id , tweet.

The dataset is a mixture of words, emoticons, symbols, URLs and references to people. Words and emoticons contribute to predicting the sentiment, but URLs and references to people don’t. Therefore, URLs and references can be ignored. The words are also a mixture of misspelled words, extra punctuations, and words with many repeated letters. The tweets, therefore, have to be preprocessed to standardize the dataset.

The provided training and test dataset have 21465 and 5398 tweets respectively. The shape of training dataset is (21465,3) and shape of testing dataset is (5398,2).

3 :Methodology and Implementation

Pre-processing

Raw tweets scraped from twitter generally result in a noisy dataset. This is due to the casual nature of people's usage of social media. Tweets have certain special characteristics such as retweets, emoticons, user mentions, etc. which have to be suitably extracted. Therefore, raw twitter data has to be normalized to create a dataset which can be easily learned by various classifiers. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. We first do some general pre-processing on tweets which is as follows.

A. Convert the tweet to lower case.

B .Replace 2 or more dots (.) with space.

C. Strip spaces and quotes (" and ') from the ends of tweet.

D. Replace 2 or more spaces with a single space.

E. Stemming or lemmatization

Libraries and Packages used:

1. numpy
2. pandas
3. nltk
4. os
5. gc
6. keras
7. sklearn

4 :Experiments

We perform experiments using various different classifiers. Unless otherwise specified, we use 20% of the training dataset for validation of our models to check against overfitting i.e. we use 17172 tweets for training and 4293 tweets for validation.

1.LSTM MODEL:

Model : "sequential"

Layer(type)	Output Shape	Param#
embedding(Embedding)	(None,None,100)	3321800
lstm(LSTM)	(None,None,64)	42240
lstm_1(LSTM)	(None,32)	12416
dense(Dense)	(None,3)	99

Total params:3,376,555

Trainable params:3,376,555

Non-trainable params:0

Long short-term memory (**LSTM**) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural network, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

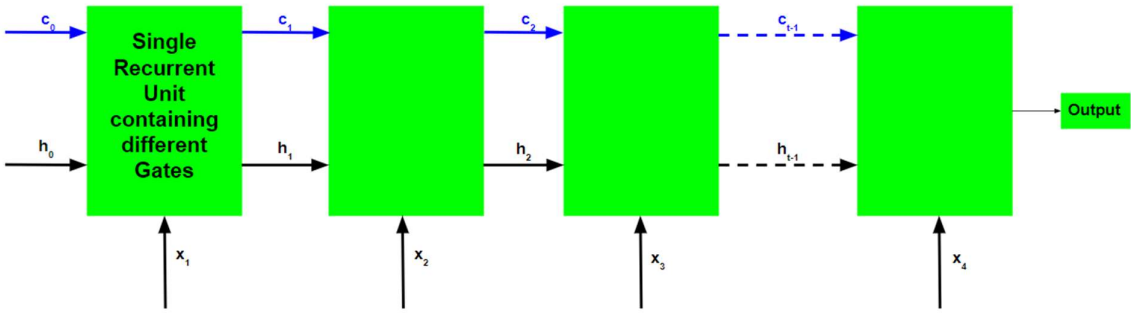
Long Short Term Memory Networks Explanation

Prerequisites: Recurrent Neural Networks

To solve the problem of Vanishing and Exploding Gradients in a deep Recurrent Neural Network, many variations were developed. One of the most famous of them is the **Long Short Term Memory Network**(LSTM). In concept, an LSTM recurrent unit tries to "remember" all the past knowledge that the network is seen so far and to "forget" irrelevant data. This is done by introducing different activation function layers called "gates" for different purposes. Each LSTM recurrent unit also maintains a vector called the **Internal Cell State** which conceptually describes the information that was chosen to be retained by the previous LSTM recurrent unit. A Long Short Term Memory Network consists of four different gates for different purposes as described below:-

1. **Forget Gate(f):** It determines to what extent to forget the previous data.
2. **Input Gate(i):** It determines the extent of information to be written onto the Internal Cell State.
3. **Input Modulation Gate(g):** It is often considered as a sub-part of the input gate and many literatures on LSTM's do not even mention it and assume it inside the Input gate. It is used to modulate the information that the Input gate will write onto the Internal State Cell by adding non-linearity to the information and making the information **Zero-mean**. This is done to reduce the learning time as Zero-mean input has faster convergence. Although this gate's actions are less important than the others and is often treated as a finesse-providing concept, it is good practice to include this gate into the structure of the LSTM unit.
4. **Output Gate(o):** It determines what output(next Hidden State) to generate from the current Internal Cell State.

The basic work-flow of a Long Short Term Memory Network is similar to the work-flow of a Recurrent Neural Network with only difference being that the Internal Cell State is also passed forward along with the Hidden State.



Working of an LSTM recurrent unit:

1. Take input the current input, the previous hidden state and the previous internal cell state.
2. Calculate the values of the four different gates by following the below steps:-

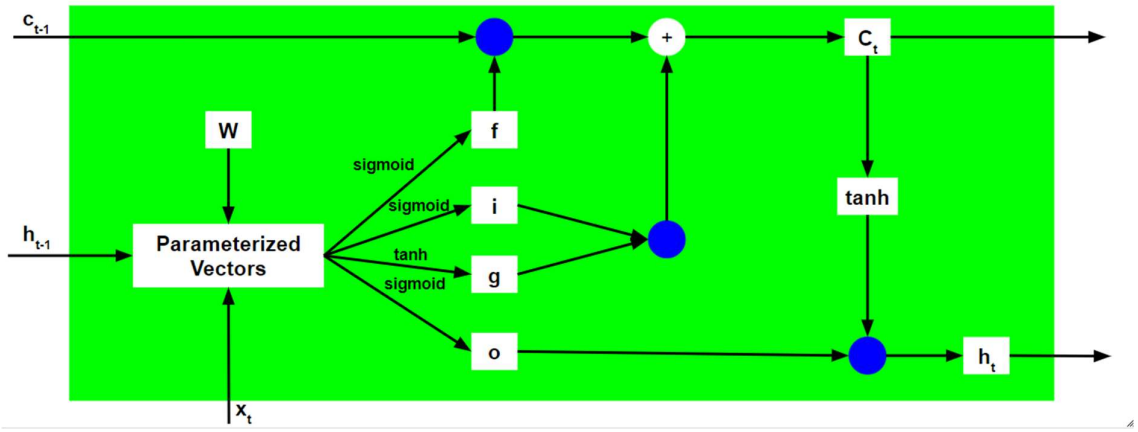
- For each gate, calculate the parameterized vectors for the current input and the previous hidden state by element-wise multiplication with the concerned vector with the respective weights for each gate.
- Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

Input Gate : Sigmoid Function
Forget Gate : Sigmoid Function
Output Gate : Sigmoid Function
Input Modulation Gate : Hyperbolic Tangent Function

3. Calculate the current internal cell state by first calculating the element-wise multiplication vector of the input gate and the input modulation gate, then calculate the element-wise multiplication vector of the forget gate and the previous internal cell state and then adding the two vectors.

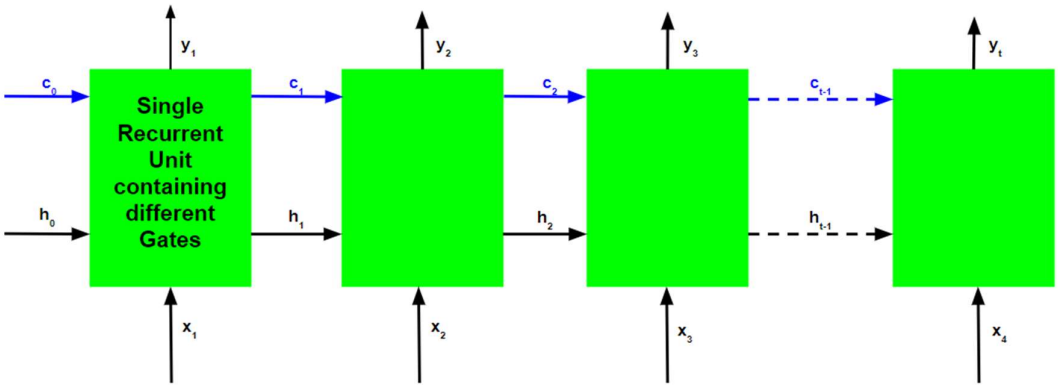
4. Calculate the current hidden state by first taking the element-wise hyperbolic tangent of the current internal cell state vector and then performing element wise multiplication with the output gate.

The above stated working is illustrated as below:-



Note that the blue circles denote element-wise multiplication. The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

Just like Recurrent Neural Networks, an LSTM network also generates an output at each time step and this output is used to train the network using gradient descent.



The only main difference between the Back-Propagation algorithms of Recurrent Neural Networks and Long Short Term Memory Networks is related to the mathematics of the algorithm

2.CNN(Convolutional Neural Network):

Model:"sequential_1"

Layer type	Output Shape	Param#
embedding_1(Embedding)	(None,1133,100)	3321800
dropout(Dropout)	(None,1133,100)	0
conv1d (Conv1D)	(None,1133,64)	19264
gloabal_max_pooling1d (Global	(None,64))	
dense_1(Dense)	(None,128)	8320
dropout_1(Dropout)	(None,128)	0
dense_2(Dense)	(None,3)	387

Total params:3,349,771

Trainable params:3,349,771

Non-trainable params:0

In deep learning, a **convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep neural most commonly applied to analysing visual imagery.[1] They are also known as **shift invariant** or **space invariant artificial neural networks** (**SIANN**), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing and financial time series

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

It is assumed that reader knows the concept of Neural Network.

When it comes to Machine Learning, Artificial Neural Networks perform really well. Artificial Neural Networks are used in various classification task like image, audio, words. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural Network. In this blog, we are going to build basic building block for CNN. Before diving into the Convolution Neural Network, let us first revisit some concepts of Neural Network. In a regular Neural Network there are three types of layers:

1. **Input Layers:** It's the layer in which we give input to our model. The number of neurons in this layer is equal to total number of features in our data (number of pixels incase of an image).
2. **Hidden Layer:** The input from Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layers can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by addition of learnable biases followed by activation function which makes the network nonlinear.
3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into probability score of each class.

The data is then fed into the model and output from each layer is obtained this step is called feedforward, we then calculate the error using an error function, some common error functions are cross entropy, square loss error etc. After that, we backpropagate into the model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.

Here's the basic python code for a neural network with random inputs and two hidden layers.

filter_none

brightness_4

```
activation = lambda x: 1.0/(1.0 + np.exp(-x)) # sigmoid function
```

input = np.random.randn(3, 1)

hidden_1 = activation(np.dot(W1, input) + b1)

hidden_2 = activation(np.dot(W2, hidden_1) + b2)

output = np.dot(W3, hidden_2) + b3

W1,W2,W3,b1,b2,b3 are learnable parameter of the model.

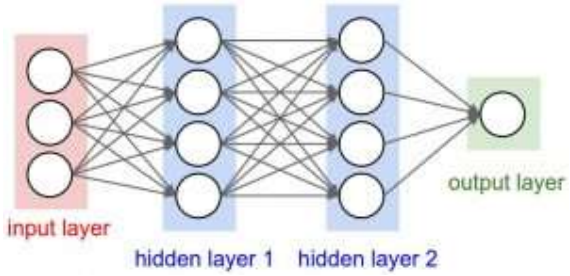
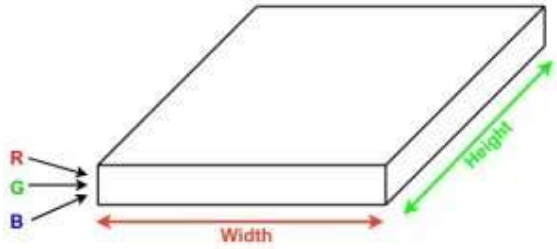


Image source: cs231n.stanford.edu

Convolution Neural Network

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image) and height (as image generally have red, green, and blue channels).



Now imagine taking a small patch of this image and running a small neural network on it, with say, k outputs and represent them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different width, height, and depth. Instead of just R, G and B channels now we have more channels but lesser width and height. his operation is called Convolution. If patch size is same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

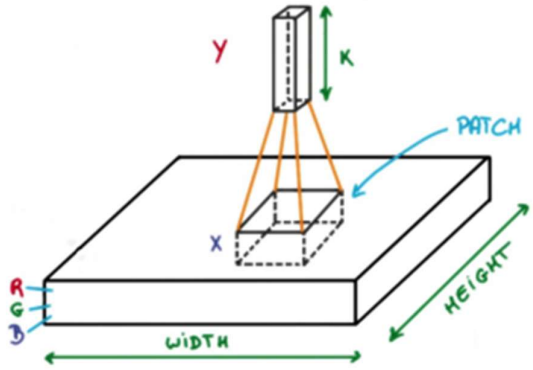


Image source: Deep Learning Udacity

Now let's talk about a bit of mathematics which is involved in the whole convolution process.

- Convolution layers consist of a set of learnable filters (patch in the above image). Every filter has small width and height and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimension 34x34x3. Possible size of filters can be axax3, where 'a' can be 3, 5, 7, etc but small as compared to image dimension.
- During forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have value 2 or 3 or even 4 for high dimensional images) and compute the dot product between the weights of filters and patch from input volume.
- As we slide our filters we'll get a 2-D output for each filter and we'll stack them together and as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

Layers used to build ConvNets

A covnets is a sequence of layers, and every layer transforms one volume to another through differentiable function.

Types of layers:

Let's take an example by running a covnets on of image of dimension 32 x 32 x 3.

1. **Input Layer:** This layer holds the raw input of image with width 32, height 32 and depth 3.
2. **Convolution Layer:** This layer computes the output volume by computing dot product between all filters and image patch. Suppose we use total 12 filters for this layer we'll get output volume of dimension 32 x 32 x 12.
3. **Activation Function Layer:** This layer will apply element wise activation function to the output of convolution layer. Some common activation functions are RELU: max(0, x), Sigmoid: 1/(1+e^-x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimension 32 x 32 x 12.
4. **Pool Layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents from overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

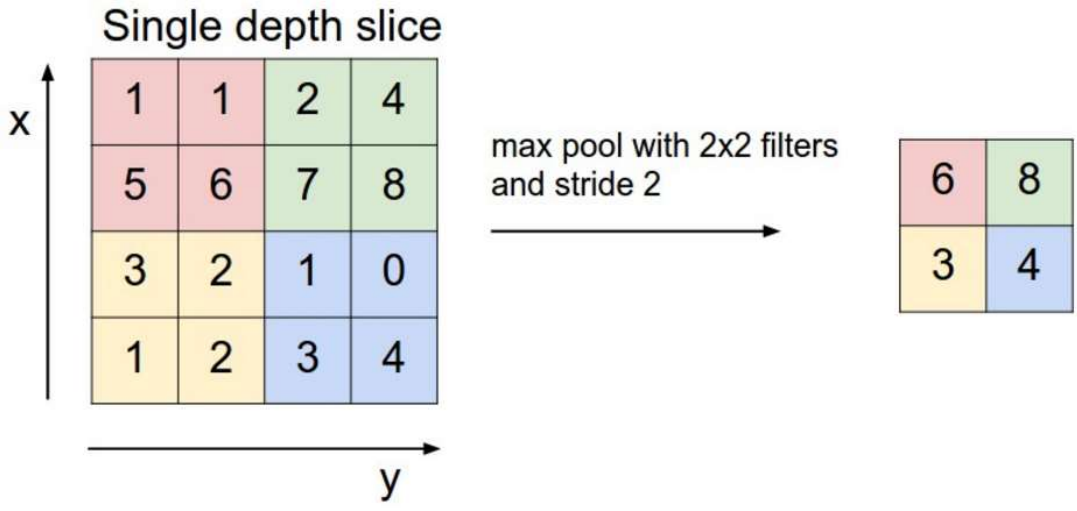
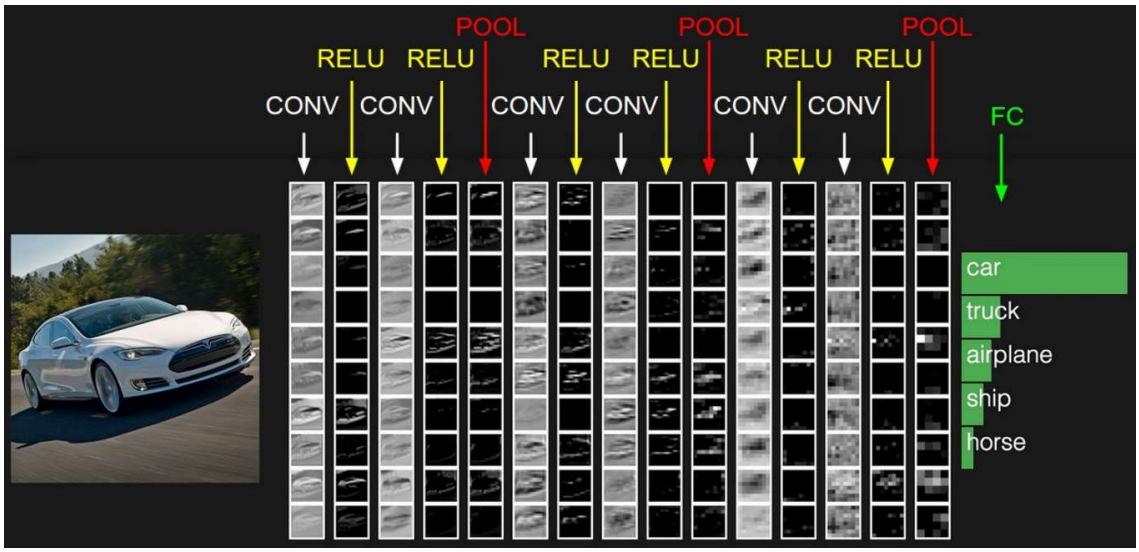


Image source: cs231n.stanford.edu

5. **Fully-Connected Layer:** This layer is regular neural network layer which takes input from the previous layer and computes the class scores and outputs the 1-D array of size equal to the number of classes.



3.CNN+GRU:

Model:"sequential_2"

Layer type	Output Shape	Param#
embedding_2(Embedding)	(None,1133,100)	3321800
conv1d_1 (Conv1D)	(None,1133,64)	19264
max_pooling1d (MaxPooling1D)	(None,566,64)	0
dropout_2(Dropout)	(None,566,64)	0
gru(GRU)	(None,566,128)	74496
dropout_3(Dropout)	(None,566,128)	0
flatten(Flatten)	(None,72448)	0
dense_3(Dense)	(None,128)	9273472
dropout_4(Dropout)	(None,128)	0
dense_4(Dense)	(None,3)	387

Total params: 12,689,419

Trainable params:12,689,419

Non-trainable params: 0

In deep learning, a **convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep learning, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks(SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

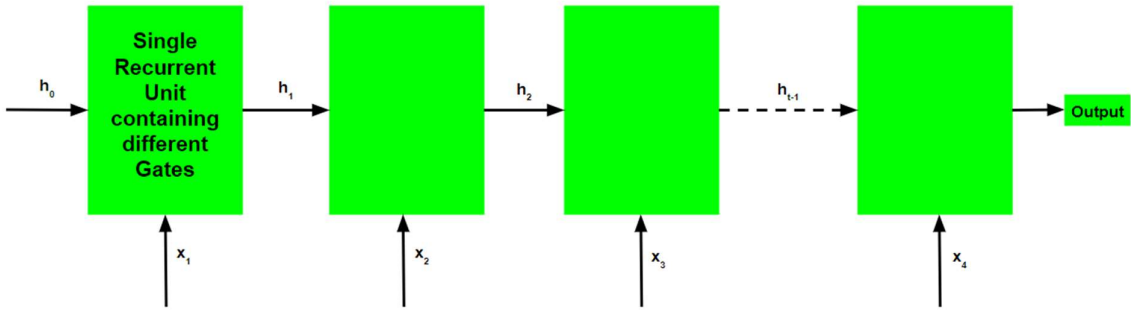
Each image the CNN processes results in a vote. ... After doing this for every feature pixel in every convolutional layer and every weight in every fully connected layer, the new weights give an answer that works slightly better for that image. This is then repeated with each subsequent image in the set of labelled images.

Gated Recurrent Unit Networks

Prerequisites: Recurrent Neural Networks, Long Short Term Memory Networks
To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed. One of the most famous variations is the **Long Short Term Memory Network(LSTM)**. One of the lesser known but equally effective variations is the **Gated Recurrent Unit (GRU)**.
Unlike LSTM, it consists of only three gates and does not maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM recurrent unit is incorporated into the hidden state of the Gated Recurrent Unit. This collective information is passed onto the next Gated Recurrent Unit. The different gates of a GRU are as described below:-

1. **Update Gate:** It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.
2. **Reset Gate:** It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.
3. **Current Memory Gate:** It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future.

The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.



Working of a Gated Recurrent Unit:

Take input the current input and the previous hidden state as vectors.
Calculate the values of the three different gates by following steps given below:-

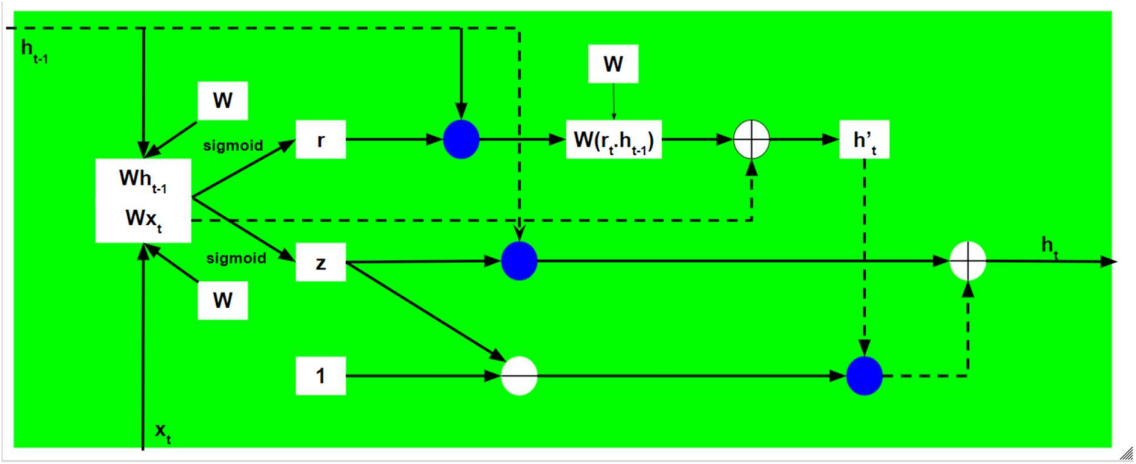
1. For each gate, calculate the parameterized currrent input and previous hidden state vectors by performing element-wise multiplication (hadmard product) between the concerned vector and the respective weights for each gate.
2. Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

Update Gate : Sigmoid Function
Reset Gate : Sigmoid Function

The process of calculating the Current Memory Gate is a little different. First, the Hadmard product of the Reset Gate and the previous hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.

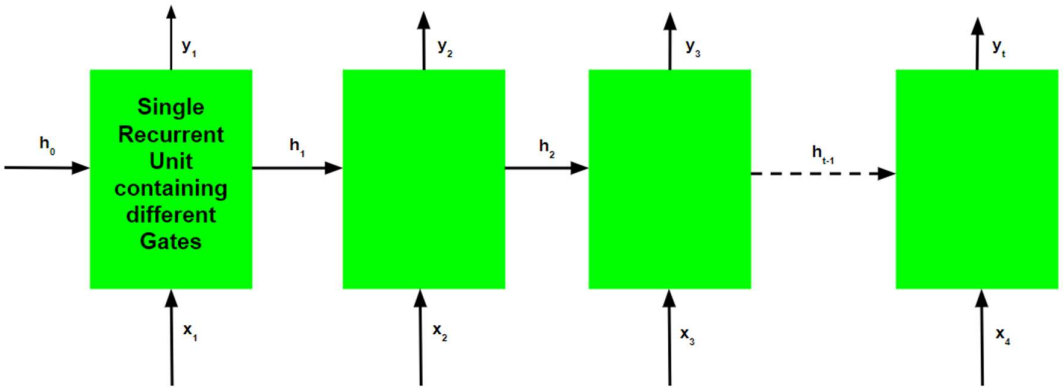
To calculate the current hidden state, first a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First calculate the hadmard product of the update gate and the previous hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the hadmard product of the newly generated vector with the current memory gate. Finally add the two vectors to get the current hidden state vector.

The above stated working is stated as below:



Note that the blue circles denote element-wise multiplication. The positive sign in the circle denotes vector addition while the negative sign denotes vector subtraction(vector addition with negative value).The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.



Note that just like the workflow, the training process for a GRU network is also diagramatically similar to that of a basic Recurrent Neural Network and differs only in the internal working of each recurrent unit.

The Back-Propagation Through Time Algorithm for a Gated Recurrent Unit Network is similar to that of a Long Memory Network and differs only in the differential chain formation.

4.Bidirectional GRU

Model:"sequential_3"

Layer type	Output Shape	Param#
embedding_3(Embedding)	(None,1133,100)	3321800

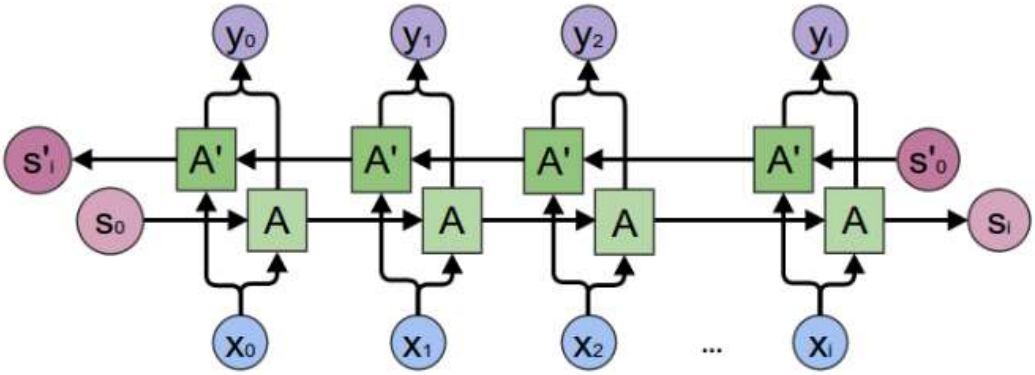
spatial_dropout1d (SpatialDr)	(None,1133,100)	0
bidirectional (Bidirectional)	(None,256)	176640
dropout_5(Dropout)	(None,256)	0
dense_5(Dense)	(None,3)	771

Total params:3,499,211

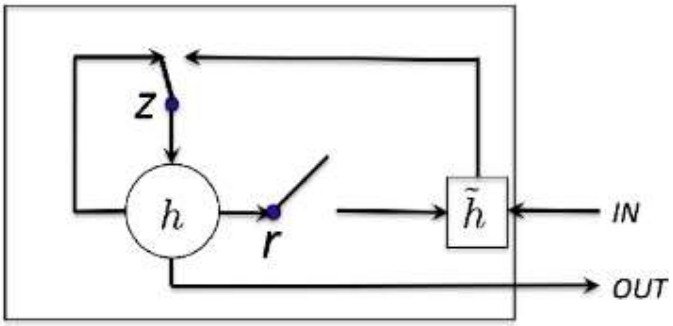
Trainable params:3,499,211

Non-trainable params:0

Bidirectional GRU's are a type of bidirectional recurrent neural networks with only the input and forget gates. It allows for the use of information from both previous time steps and later time steps to make predictions about the current state.



Replacing every A and A' in the diagram with a gated recurrent unit yields the bidirectional GRU. Figure 4 indicates the typical structure of a gated recurrent unit, where, Z is the update gate and r is the reset gate.



Using the Keras implementation the model was developed with pre-trained Fasttext embeddings in the embedding layer followed by a 1D dropout layer of 0.2 then a bidirectional GRU layer and then

both average and max-pooling where conducted and then later concatenated. Furthermore, class balancing was considered when fitting the model to the training dataset. Model performance was then evaluated by the F1 score metric as per rules of the competition.

CONCLUSION:

Although the model performs fairly well and can be useful for the task of classifying text based on its relevance to SDG#3 indicators it is not enough to win the competition. As the competition remains open, further adjustments such as parameter tuning and assembling with other models shall be considered to help improve the leader-board position.

RESULTS:

In this Kaggle competition average F1 score over positive , negative and neutral tweets as the scoring metric. When I submitted my predictions for each model in the form of csv(Comma separated values) file I got the following results:

1:LSTM Model: With the help of first model that is LSTM model I got a F1 score of 0.64126

2.CNN(Convolutional Neural Networks): With the help of second model that is CNN(Convolutional Neural Network) I got a F1 score of 0.65446

3:CNN + GRU(Convolutional Neural Networks + Gated Recurrent Unit): With the help of third model that is CNN + GRU(Convolutional Neural Networks + Gated Recurrent Unit) I got a F1 score of 0.64451

4.Bidirectional GRU: With the help of fourth model that is bidirectional GRU I got a F1 score of 0.62343.

From above observations we can see that CNN model has performed the best among the four models which I have used for calculation.

FUTURE DIRECTIONS:

• **Handling emotion ranges**: We can improve and train our models to handle a range of sentiments. Sentiment can also have gradations like the sentence, This is good, is positive but the sentence, This is extraordinary. is somewhat more positive than the first. We can therefore classify the sentiment in ranges, say from -2 to +2.

• **Using symbols**: During our pre-processing, we discard most of the symbols like commas, full-stops, and exclamation mark. These symbols may be helpful in assigning sentiment to a sentence.

REFERENCES:

1. Machine learning book by Bishop
2. Geeks for Geeks
3. Research Papers on Twitter Analysis