# Text classification using CNN

vijay choubey

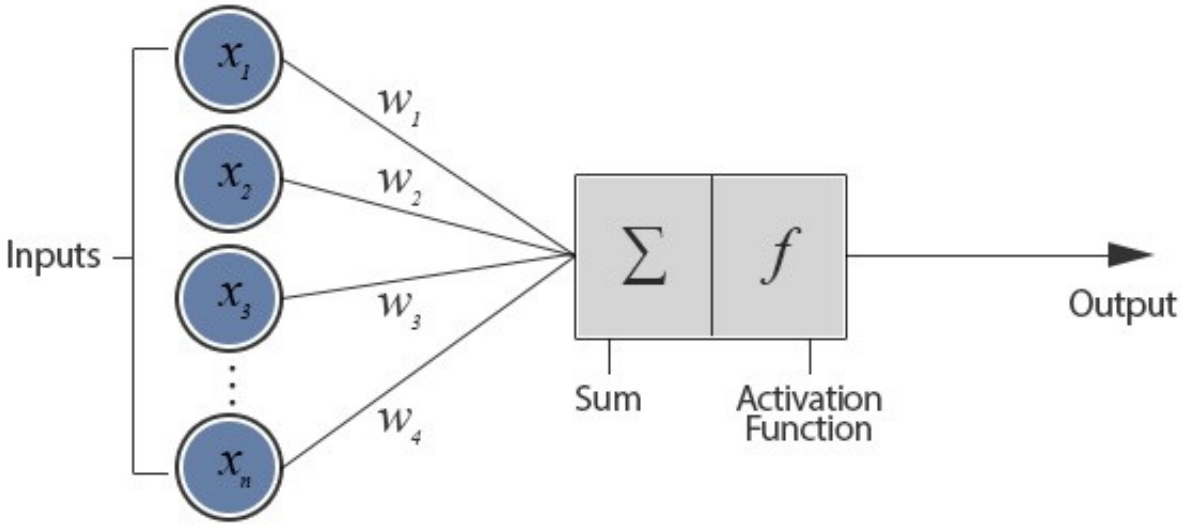
In this article, we are going to do text classification on IMDB data-set using Convolutional Neural Networks(CNN). We will go through the basics of Convolutional Neural Networks and how it can be used with text for classification.


- Basics of Convolutional Neural Networks
- How to use CNN for text classification?
- Code Demonstration

## Convolutional Neural Networks

Let’s first understand the term neural networks. In a neural network, where neurons are fed inputs which then neurons consider the weighted sum over

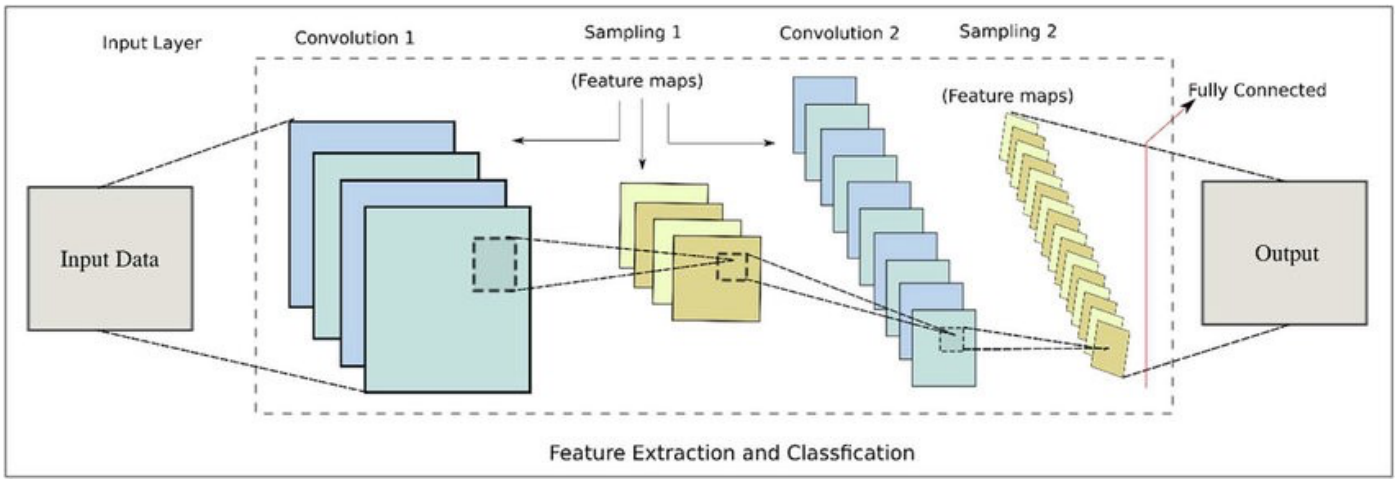
them and pass it by an activation function and passes out the output to next neuron.




Now, a convolutional neural network is different from that of a neural network because it operates over a volume of inputs.

Each layer tries to find a pattern or useful information of the data.

An example of multi-channel input is that of an image where the pixels are the input vector and RGB are the 3 input channels representing channel.




This is what the architecture of a CNN normally looks like. It will be different

depending on the task and data-set we work on. There are some terms in the architecutre of a convolutional neural networks that we need to understand before proceeding with our task of text classification.

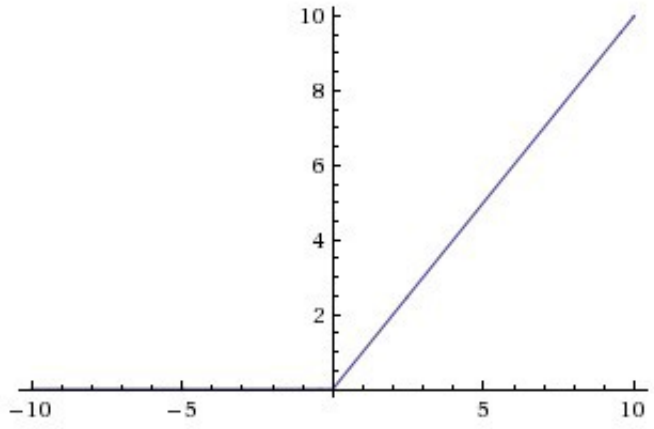
***Convolution***: It is a mathematical combination of two relationships to produce a third relationship. Joins two sets of information.

***Convolution over input***: We slide over input data the convolution to extract features by applying a filter/ kernel (both can be used interchangeably). This is important in feature extraction. There are some parameters associated with that sliding filter like how much input to take at once and by what extent should input be overlapped.

- Stride: Size of the step filter moves every instance of time.
- Filter count: Number of filters we want to use.

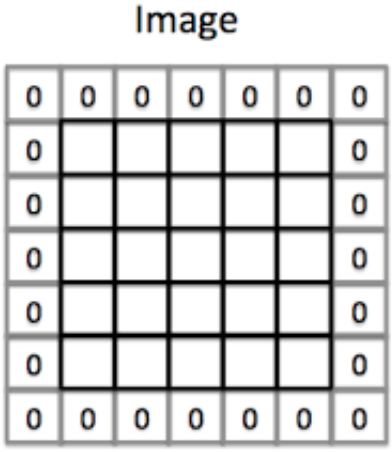
When we are done applying the filter over input and have generated multiple feature maps, an **activation function** is passed over the output to provide a non-linear relationship for our output.

An example of activation function can be ReLu.




Now, we generally add **padding** surrounding input so that feature map doesn’t shrink. If we don’t add padding then those feature maps which will

be over number of input elements will start shrinking and the useful information over the boundaries start getting lost.

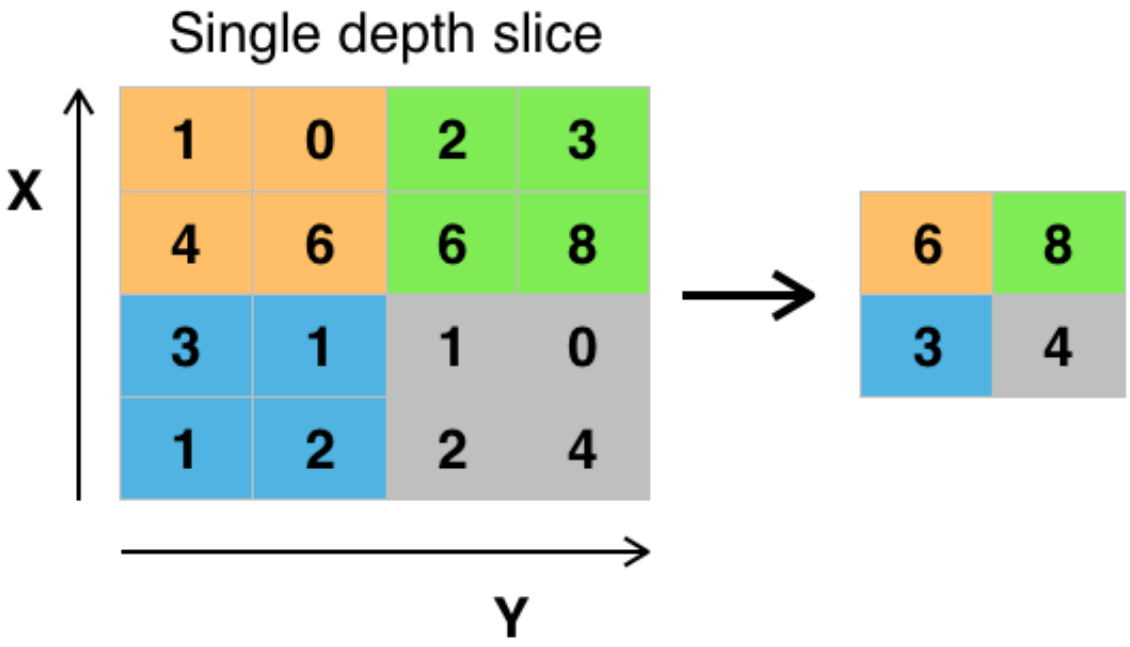



It also improves the performance by making sure that filter size and stride fits in the input well.

We are not done yet. We need something that helps us to reduce this high computation in the CNN and not overfit the data. Overfitting will lead the model to **memorize** the training data rather than **learning** from it.

We use a **pooling layer** in between the convolutional layers that reduces the dimensional complexity and stil keeps the significant information of the convolutions.

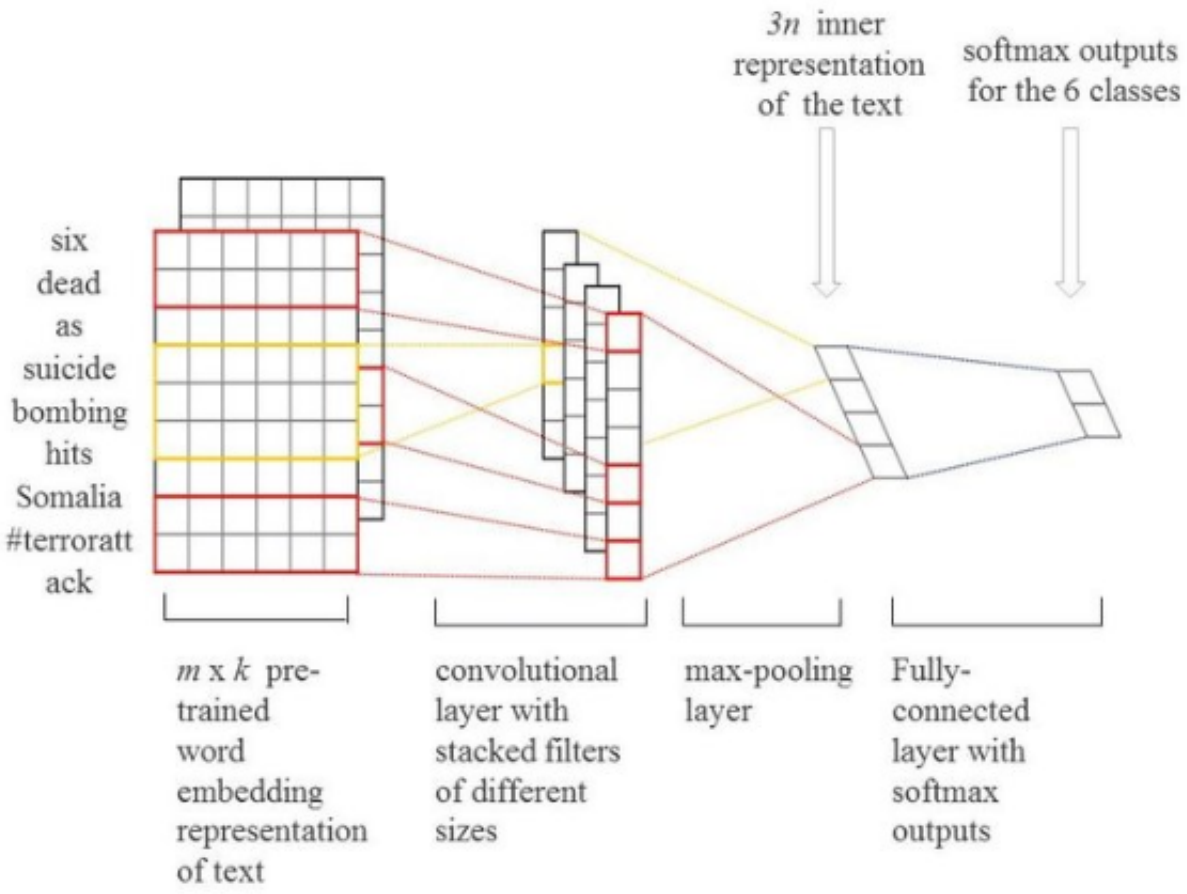
One example is of **max pooling** layer. It finds the maximum of the pool and sends it to the next layer as we can see in the figure below.




Sometimes a **Flatten** layer is used to convert 3-D data into 1-D vector.

In a CNN, the last layers are fully connected layers i.e. each node of one layer is connected to each node of the other layer.

## How to use for text classification?




***A simple CNN architecture for classifying texts***

Let's first talk about the word embeddings. When using Naive Bayes and KNN we used to represent our text as a vector and ran the algorithm on that vector but we need to consider similarity of words in different reviews because that will help us to look at the review as a whole and instead of focusing on impact of every single word.

We use a pre-defined word embedding available from the library. Generally, if the data is not embedded then there are many various embeddings available open-source like **Glove** and **Word2Vec**.

When we do dot product of vectors representing text, they might turn out zero even when they belong to same class but if you do dot product of

those embedded word vectors to find similarity between them then you will be able to find the interrelation of words for a specific class.

Then, we slide the filter/ kernel over these embeddings to find convolutions and these are further dimensionally reduced in order to reduce complexity and computation by the Max Pooling layer.

Lastly, we have the fully connected layers and the activation function on the outputs that will give values for each class.

# Making it work

# Setup

For this task, we'll need:

- Python: To run our script
- Pip: Necessary to install Python packages

Now we can install some packages using pip, open your terminal and type these out.

```
pip install tensorflow
pip install keras
```

Tensorflow: open-source software library for dataflow and differentiable programming across a range of tasks.
Keras: open-source neural-network library

# Code

Let’s first start by importing the necessary libraries and the Reuters data-set which is availabe in data-sets provided by keras,

```
# Importing libraries
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence# Our dictionary will co
top_words = 7000# Now we split our data-set into training and te
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=
print(X_train[0])
print(y_train[0])print('Shape of training data: ')
print(X_train.shape)
print(y_train.shape)print('Shape of test data: ')
print(X_test.shape)
print(y_test.shape)
```

Output

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 394
1
Shape of training data:
(25000,)
(25000,)
Shape of test data:
(25000,)
(25000,)
```

- As we see, our dataset consists of 25,000 training samples and 25,000

test samples. Every data is a vector of text indexed within the limit of top words which we defined as 7000 above.

- Now, we pad our input data so the kernel filter and stride can fit in input well. We limit the padding of each review input to 450 words. Keras provides us with function to pad sequences. So, we use it on our reviews.

```
# Padding the data samples to a maximum review length in words
max_words = 450X_train = sequence.pad_sequences(X_train, maxlen=
X_test = sequence.pad_sequences(X_test, maxlen=max_words)# Build
model = Sequential()      # initilaizing the Sequential nature f
model.add(Embedding(top_words, 32, input_length=max_words))model
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- The model first consists of embedding layer in which we will find the embeddings of the top 7000 words into a 32 dimensional embedding and the input we can take in is defined as the maximum length of a review allowed.
- Then, we add the convolutional layer and max-pooling layer. Finally, we flatten those matrices into vectors and add **dense** layers(basically scale,rotating and transform the vector by multiplying Matrix and vector).
- The last Dense layer is having one as parameter because we are doing a binary classification and so we need only one output node in our vector.

```
model.compile(loss='binary_crossentropy', optimizer='adam', metr
model.summary()
```

# Output

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 450, 32)           224000
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 450, 32)           3104
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 225, 32)           0
_________________________________________________________________
flatten_3 (Flatten)          (None, 7200)              0
_________________________________________________________________
dense_15 (Dense)             (None, 250)               1800250
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 251
=================================================================
Total params: 2,027,605
Trainable params: 2,027,605
Non-trainable params: 0
_________________________________________________________________
```

Now, we will fit our training data and define the the **epochs**(number of passes through dataset) and batch size(nunmber of samples processed before updating the model) for our learning model.

- Batch size is kept greater than or equal to 1 and less than the number of samples in training data.

```
# Fitting the data onto model
model.fit(X_train, y_train, validation_data=(X_test, y_test), ep
```

```
scores = model.evaluate(X_test, y_test, verbose=0)# Displays the
print("Accuracy: %.2f%%" % (scores[1]*100))
```

# Output

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/2
 - 28s - loss: 0.4582 - acc: 0.7563 - val_loss: 0.2889 - val_acc
Epoch 2/2
 - 27s - loss: 0.2095 - acc: 0.9188 - val_loss: 0.2725 - val_acc
Accuracy: 88.64%
```

**We were able to achieve an accuracy of 88.6% over IMDB movie reviews' test data.**

# Tips

- We can improve our CNN model by adding more layers. It is always preferred to have more(dense) layers than to have wide layers of less number.
- But, we must take care to not overfit the data and for that we can try using various regularization methods.

Thanks for reading! I am going to be writing more **NLP articles** in the future too. **Follow** me up to be informed about them. And I am also a freelancer,If there is some freelancing work on data-related projects feel free to reach out over **Linkedin**.Nothing beats working on real projects!