Sentiment Analysis

Introduction

Sentiment analysis is contextual mining of text which identifies and extract subjective information in source material, and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations. However, analysis of social media streams is usually restricted to just basic sentiment analysis and count based metrics. This is akin to just scratching the surface and missing out on those high value insights that are waiting to be discovered. So what should a brand do to capture that low hanging fruit.

These days the data are growing in skyrocketing speed in the website and the trend of online shopping and the showing and the promoting of the products in the website has suddenly increased these days. The big companies like **Netflix, Amazon, Alibaba** has increased the selling of their products in large amount. They use the special type of Algorithm to attract the people so that they visit the Website more times, eventually the traffic of the website increased and finally the selling of the product increase. They collect the data and their sentiment by asking them to rate the certain products, which may be in the form of the Rating or any other text format, then they analyse those data and predict the sentiment of people and for the next time, when they visit the site the similar type of product are recommended to the people. This type of system is applied is applied in almost all kind of Website. The respective company or any other people who are providing the service are tracking us every second and trying to get the data from us and do a similar type of Analysis of our activity.

Let's take a real world example, we use google for solving most of our daily activities, the google is tracking us in every aspect. We notice the google seek for our location and ask for some kind of review or rating, and keeps a log of our activities and for the next time, the similar type of services can be seen on other social media like Facebook, this is the real world experience.

Sentiment Analysis also known as **Opinion Mining** is a field within **Natural Language Processing** (NLP) that builds systems that try to identify and extract opinions within the text. Usually, besides identifying the opinion, these systems extract attributes of the expression e.g.:

- *Polarity*: if the speaker expresses a *positive* or *negative* opinion,
- *Subject*: the thing that is being talked about,
- *Opinion holder*: the person, or entity that expresses the opinion.

Natural Language Processing (NLP) is a field at the intersection of computer science, artificial intelligence, and linguistics. The goal is for computers to process or "understand" natural language in order to perform various human like tasks like language translation or answering questions.

With the rise of voice interfaces and chatbots, NLP is one of the most important technologies of the 4th Industrial Revolution and become a popular area of AI. There's a fast-growing collection of useful applications derived from the NLP field. They range from simple to complex. Below are a few of them:

- Search, spell checking, keyword search, finding synonyms, complex questions answering
- Extracting information from websites such as: products, prices, dates, locations, people or names
- Machine translation (i.e. Google translate), speech recognition, personal assistants *(think about Amazon Alexa, Apple Siri, Facebook M, Google Assistant or Microsoft Cortana)*
- Chat bots/dialog agents for customer support, controlling devices, ordering goods
- Matching online advertisements, **sentiment analysis** for marketing or finance/trading
- Identifying financial risks or fraud

Background

For this project we have also used the similar kind of the approach for the Book Recommendation System. The recommendation system consists of mainly 4 parts, One of them is the sentiment Analysis for the Text review. The English text review from the customer/user is analysed so that the sentiment of the respective can be acquired for the respective books, we can infer, about the customer thoughts toward to book and the overall rating is done, by considering the number of times of clicking the book and the starring of the book by the user. When the end user gives the certain review about the book then the certain Deep Learning Algorithms are applied to the review so the corresponding rating is generated as the output of the model. This section is mainly for the different approaches used by us during the analysis of the user text review. This sentiment analysis part does not only rely on book, this can be used for every aspect of the product. This can be used for any other system,

like Restaurant, Hotels, Party Palaces, Tourist Destination or any other commercial product where there is a provision of giving review for that item. This project mainly focuses on the Book Recommendation System, so we are using the sentiment score of the text input to evaluate the overall sentiment of Any Book that the user gives/uses.

Data Sources

The data are the main part of any project. For the successful completion of the project the data, facts and other information bolster the outcome and the accuracy of the project. The whole project is of data driven approach in which we use data for discovering new knowledge from the existing one. If we have some data and information about something then we can infer or predict the outcome of the system for the new attributes. Similar kind of approach is applied in our project. The deep learning models for the generation of sentiment score of the text, we have used the Movie Datasets provided by the IMDd. The Deep Learning Framework PyTorch Provides the access of the IMDb datasets for the educational purpose, hence we have used for our training model

1. **Review Dataset v1.0**

Overview

This dataset contains movie reviews along with their associated binary sentiment polarity labels. It is intended to serve as a benchmark for sentiment classification. This document outlines how the dataset was gathered, and how to use the files provided

Dataset

The core dataset contains 50,000 reviews split evenly into 25k train and 25k test sets. The overall distribution of labels is balanced (25k pos and 25k neg). We also include an additional 50,000 unlabeled documents for unsupervised learning.

This is one of the biggest labeled movie review datasets available and hundreds of researches on sentiment analysis and NLP have been carried out that use this datasets. But one of the main problems with this dataset is that it only contains highly polar reviews for training and testing, i.e the reviews in this datasets are either very positive, or very negative. There are no slightly positive, slightly negative, or neutral movie reviews. So this datasets, while being useful for the binary sentiment classification tasks, is skewed and not very effective for the fine-grained sentiment classification task like in this project where it is required to classify a review into a rating scale from 1 to 5.

2.GloVe: Global Vectors for Word Representation

GloVe, coined from Global Vectors, is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity.Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. It is developed as an open-source project at Stanford. As log-bilinear regression model for unsupervised learning of word representations, it combines the features of two model families, namely the global matrix factorization and local context window method. There are different types of the GloVe models for the word representations.

Here, we'll be using the "**glove.6B.100d**" vectors". glove is the algorithm used to calculate the vectors, go here for more. 6B indicates these vectors were trained on 6 billion tokens and 100d indicates these vectors are 100-dimensional

The theory is that these pre-trained vectors already have words with similar semantic meaning close together in vector space, e.g. "terrible", "awful", "dreadful" are nearby. This gives our embedding layer a good initialization as it does not have to learn these relations from scratch.

Data Preparation

One of the main concepts of TorchText is the Field. These define how your data should be processed. In our sentiment classification task the data consists of both the raw string of the review and the sentiment, either "pos" or "neg".
The parameters of a Field specify how the data should be processed.
We use the TEXT field to define how the review should be processed, and the LABEL field to process the sentiment.
Our TEXT field has tokenize='spacy' as an argument. This defines that the "tokenization" (the act of splitting the string into discrete "tokens") should be done using the spaCy tokenizer. If no tokenize argument is passed, the default is simply splitting the string on spaces.
LABEL is defined by a LabelField, a special subset of the Field class specifically used for handling labels. We will explain the *dtype* argument later.

The IMDb dataset only has train/test splits, so we need to create a validation set. We can do this with the .split() method.

By default this splits 70/30, however by passing a split_ratio argument, we can change the ratio of the split, i.e. a split_ratio of 0.8 would mean 80% of the examples make up the training set and 20% make up the validation set.

We also pass our random seed to the random_state argument, ensuring that we get the same train/validation split each time.

Data Preprocessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing. This is the basic after the collection of the data. The user may not get the desired datasets for his need, so the user needs to modify the datasets according the the need and demand of the project. During this course, the user may need to handle the missing values, outliers or any other mis-representation of the datasets.

We had also faced such disabilities after the collection of datasets from the amazon. We needed only the words that represent the real meaning, but the text review contains a large number of words including regular expression, punctuation, commas, full stops and other forms of the words like past or future form, which are not needed and they are redundant to our project. For this we used NLTK python library for Natural Language processing.

Data Cleaning

Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Since the data obtained from the IMDB movie datasets were already built, there were no any faults to be cleaned.

Data Reduction

Data reduction is the transformation of numerical or alphabetical digital information derived empirically or experimentally into a corrected, ordered, and simplified form. The basic concept is the reduction of multitudinous amounts of data down to the meaningful parts.

Data reduction can be achieved several ways. The main types are data deduplication, compression and single-instance storage. Data deduplication, also known as *data dedupe*, eliminates redundant segments of data on storage systems. It only stores redundant segments once and uses that one copy whenever a request is made to access that piece of data. Data dedupe is more granular than single-instance storage. Single-instance storage finds files such as email attachments sent to multiple people and only stores one copy of that file. As with dedupe, single-instance storage replaces duplicates with pointers to the one saved copy.

Normalization

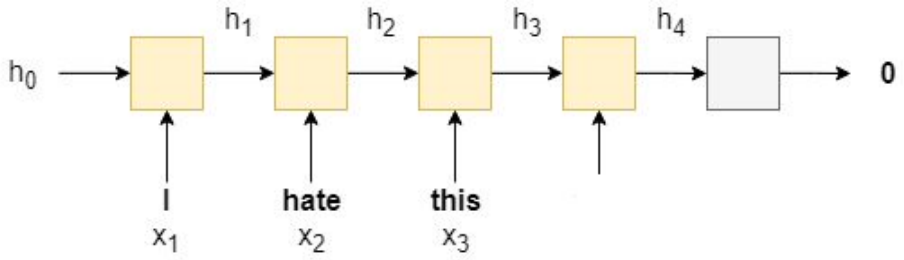
Data normalization is the process of converting the data to any kind of canonical forms. In terms of the machine learning or signal processing, data normalization refers to transforming the values to al limited range or scale. For example, if some movie review data de[rovides sentiment labels as rating scales from 1 to 10, it is required to normalize that to convert the labels to the required value that is 1 to 5 . In this case, a simple division by 2 seems to be sufficient but on other cases, it might take more such operations such as subtracting mean value and then dividing by the standard deviation.

Tools and Technologies

Extracting the sentiment of the text is really tough job, but the availability of the different kinds of modern hardware and software and Artificial Intelligence technologies we have successfully overcome that barrier and able to extract the sentiment of the text review which we give as use input for the product, being specific for this project:BOOK

Simple Sentiment Analysis

They are the form of the Artificial Neural Network, which are mainly used in sequence modelling for in Natural Language Processing and other different kinds of speech and text processing methodologies. The recurrent neural networks have to property of saving the previous information of the sequences so it has made possible to do the Speech Recognition and Understanding job. These frameworks can be used for other various types of application like getting the polarity of a sentence and finding the sentiment score of a text.



We'll be using a recurrent neural network (RNN) as they are commonly used in analysing sequences. An RNN takes in sequence of words, $X = \{x_1,...,x_T\}$, one at a time, and produces a hidden state, h , for each word. We use the RNN recurrently by feeding in the current word x_t as well as the hidden state from the previous word, h_{t-1} , to produce the next hidden state, ht .

$ht=RNN(x_t,h_{t-1})$

Once we have our final hidden state, h_T , (from feeding in the last word in the sequence, x_T) we feed it through a linear layer, f , (also known as a fully connected layer), to receive our predicted sentiment, $y\hat{} = f(h_T)$.

Updated Sentiment Analysis

The output of the simple sentiment analysis was very poor, it could classify the sentences with less than 50% with was very poor. As a solution we made the RNN architecture a little bit complex by adding following parameters.

1.Packed padded sequences

We'll be using *packed padded sequences*, which will make our RNN only process the non-padded elements of our sequence, and for any padded element the output will be a zero tensor. To use packed padded sequences, we have to tell the RNN how long the actual sequences are.

2.Pre-trained word embeddings: GloVe

Pre-trained word embedding help to align the input word vector in the vector of words space, in which the similar kinds of words are aligned in the similar dimension of word space. For this we have used. The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations, and although many such methods now exist, the question still remains as to how meaning is generated from these statistics, and how the resulting word vectors might represent that meaning.

The motivation for GloVe starts with that *ratios* between the probabilities of words appearing next to each other carry more information than considering those probabilities individually. Don't worry if this doesn't make sense. I'll break it down. Let us assume (i, j) element of the co-occurrence matrix X, $X_{i,j}$ denotes the number of times words j occur near i. Next let us define $P_{ij} = P(j|i) = X_{ij}/\sum_{\forall k} X_{ik}$. These are pretty basic things.

With this notation we can find why the ratio between probabilities work. Consider 4 words; *solid, gas, water* and *fashion*. If you calculate the ratio $P_{ice,k}/P_{steam,k}$ where $k \in \{solid, gas, water, fashion\}$ from X, you will make following observations.

1. $P_{ice,solid}/P_{steam,solid}$ will be very high
2. $P_{ice,gas}/P_{steam,gas}$ is very low
3. $P_{ice,water}/P_{steam,water}$ will be higher than $P_{ice,fashion}/P_{steam,fashion}$

Deriving the Cost Function

So it can be seen that this ratio is quite expressive. Now we say that whatever we are to derive should start with this ratio, leading to the function below.

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Then after some serious mathematical crunching the authors arrive at the following cost function.

$$J = \sum_{i,j=1}^{V} (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - log(1 + X_{ij}))^2$$

where V is the vocabulary size, w_i is the target word, $\tilde{w}_j$ is the context word and b_i is the bias for word w_i. But this cost function treat words far-apart and close together equally. To solve this, we introduce a weighing function, turning the equation to,

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - log(1 + X_{ij}))^2$$

You can choose any function for f that satisfy a set of properties specified in the paper. The recommended function for f is as below.

$$f(x) = \begin{cases} x/x_{max} \text{ if } x < x_{max} \\ 1 \text{ else} \end{cases}$$

This is it. The main contribution of the paper is this new cost function we just saw. Then you train the model as you would do with skip-gram but with the new cost function.

Big Picture

This is how it looks like when everything comes together. Inputs are w_is, Outputs are $\tilde{w}_j$s, bias embeddings are b_i and $\tilde{b}_j$s. Note that the cost function is defined only for a single input,output set. But the picture depicts the algorithm for a batch of data of size b.

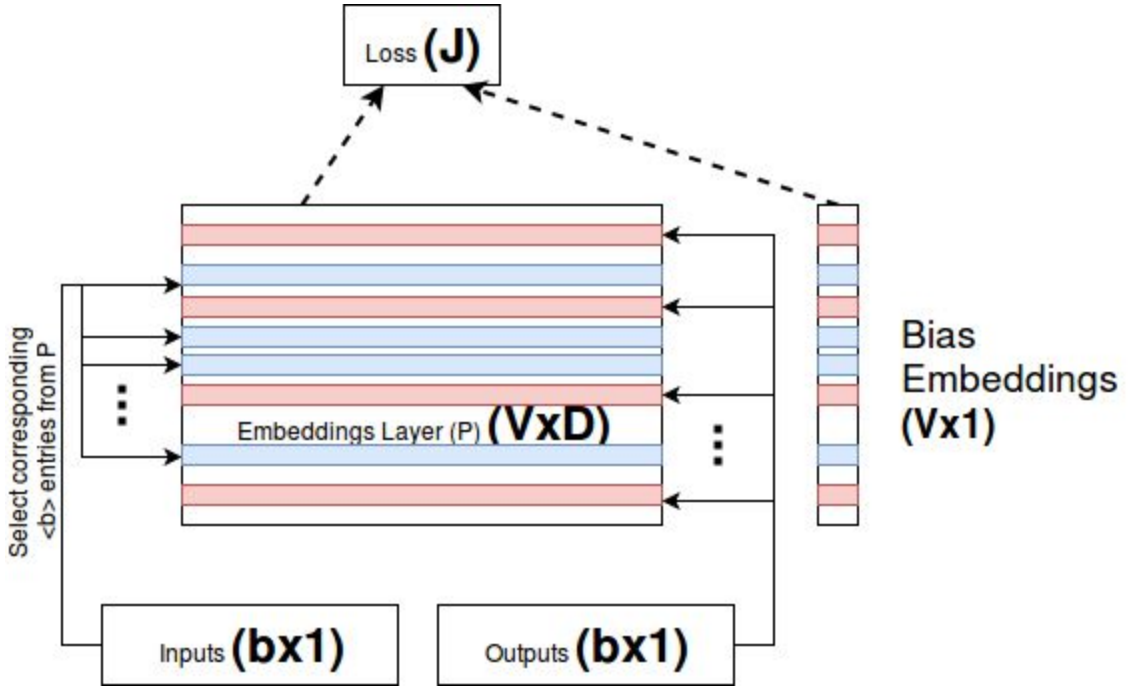
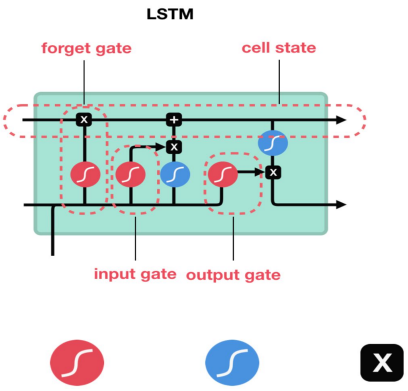


Figure :GloVe, High-level Architecture

3.Different RNN architecture

We'll be using a different RNN architecture called a Long Short-Term Memory (LSTM). Standard RNNs suffer from the vanishing gradient problem. LSTMs overcome this by having an extra recurrent state called a cell, c - which can be thought of as the "memory" of the LSTM - and the use use multiple gates which control the flow of information into and out of the memory. We can simply think of the LSTM as a function of $\mathbf{x}_t$, $\mathbf{h}_t$ and $\mathbf{c}_t$, instead of just $\mathbf{x}_t$ and $\mathbf{h}_t$.



$$(h_t,c_t) = \text{LSTM}(x_t,h_t,c_t)$$

The initial cell state, **c0**, like the initial hidden state is initialized to a tensor of all zeros. The sentiment prediction is still, however, only made using the final hidden state, not the final cell state, i.e. $y\hat{}=f(h_T)$.

Figure: Basic LSTM cell

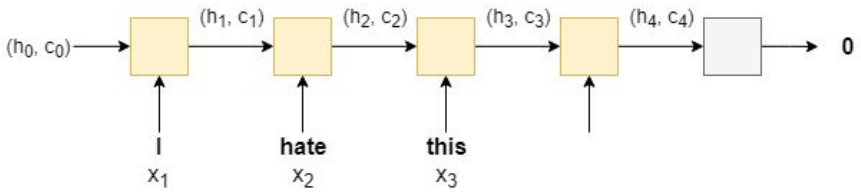


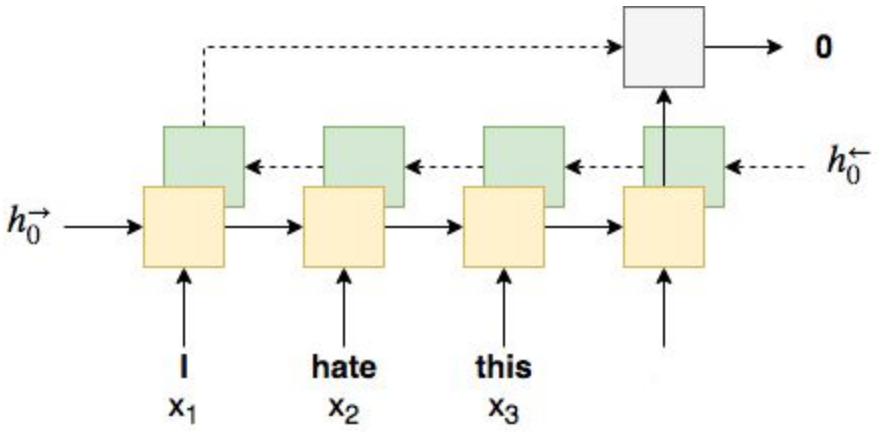
Figure: Cascaded LSTM network

Bidirectional RNN

The concept behind a bidirectional RNN is simple. As well as having an RNN processing the words in the sentence from the first to the last (a forward RNN), we have a second RNN processing the words in the sentence from the last to the first (a backward RNN). At time step t, the forward RNN is processing word x_t, and the backward RNN is processing word x_{T-t+1}.

We make our sentiment prediction using a concatenation of the last hidden state from the forward RNN (obtained from final word of the sentence), $h\rightarrow T$, and the last hidden state from the backward RNN (obtained from the first word of the sentence), $h\leftarrow T$, i.e. $y\hat{}=f(h\rightarrow T, h\leftarrow T)$

The image below shows a bi-directional RNN, with the forward RNN in orange, the backward RNN in green and the linear layer in silver.multi-layer RNN



Regularization

Although we've added improvements to our model, each one adds additional parameters. Without going into overfitting into too much detail, the more parameters we have in our model, the higher the probability that your model will overfit (memorize the training data, causing a low training error but high validation/testing error, i.e. poor generalization to new, unseen examples). To combat this, we use regularization. More specifically, we use a method of regularization called dropout. Dropout works by randomly dropping out (setting to 0) neurons in a layer during a forward pass. The probability that each neuron is dropped out is set by a hyperparameter and each neuron with dropout applied is considered independently. One theory about why dropout works is that a model with parameters dropped out can be seen as a "weaker" (less parameters) model. The predictions from all these "weaker" models (one for each forward pass) get averaged together within the parameters of the model. Thus, your one model can be thought of as an ensemble of weaker models, none of which are over-parameterized and thus should not overfit.

A different Optimizer

The only change we'll make here is changing the optimizer from SGD to Adam. SGD updates all parameters with the same learning rate and choosing this learning rate can be tricky. Adam adapts the learning rate for each parameter, giving parameters that are updated more frequently lower learning rates and parameters that are updated infrequently higher learning rates.

Faster Sentiment Analysis

In the updated sentiment analysis we were able to have pretty good accuracy of about 84%. After further study, came to know that it can be implemented and accuracy can be increased with FastText model from the paper "Bag of Tricks for Efficient Text Classification". The model is much more less and with fewer parameters, but having greater accuracy which is good for both time and space complexity. The details of the hyperparameter used will be showed in the Experiment and the Outcome section.

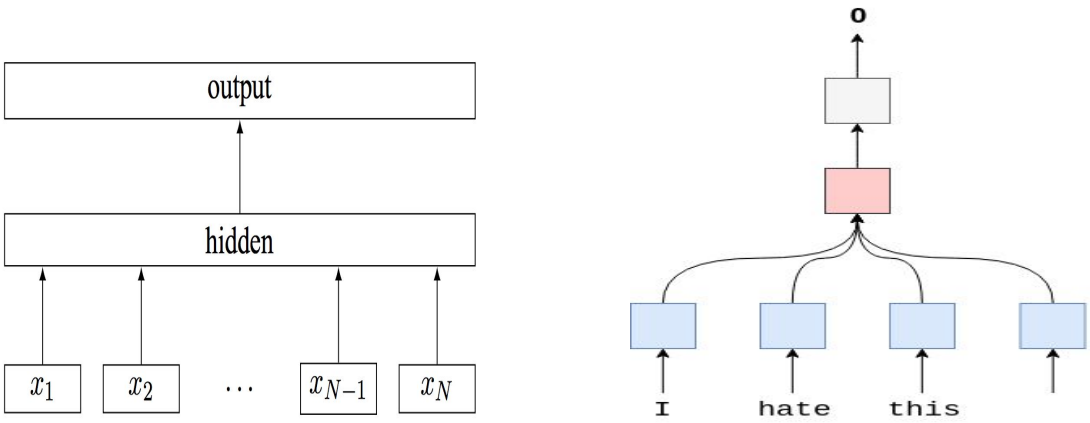


Figure : Model Architecture of *FastText*

The negative log likelihood is minimized with the following function

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)),$$

Convolutional Sentiment Analysis

Traditionally, CNNs are used to analyse images and are made up of one or more *convolutional* layers, followed by one or more linear layers. The convolutional layers use filters (also called *kernels* or *receptive fields*) which scan across an image and produce a processed version of the image. This processed version of the image can be fed into another convolutional layer or a linear layer. Each filter has a shape, e.g. a 3x3 filter covers a 3 pixel wide and 3 pixel high area of the image, and each element of the filter has a weight associated with it, the 3x3 filter would have 9 weights. In traditional image processing these weights were specified by hand by engineers, however the main advantage of the convolutional layers in neural networks is that these weights are learned via backpropagation.

The intuitive idea behind learning the weights is that our convolutional layers act like *feature extractors*, extracting parts of the image that are most important for your CNN's goal, e.g. if using a CNN to detect faces in an image, CNN may be looking for features such as the existence of a nose, mouth or a pair of eyes in the image.

So why use CNNs on text? In the same way that a 3x3 filter can look over a patch of an image, a 1x2 filter can look over a 2 sequential words in a piece of text, i.e. a bi-gram. In this CNN model we will instead use multiple filters of different sizes which will look at the bi-grams (a 1x2 filter), tri-grams (a 1x3 filter) and/or n-grams (a 1xn filter) within the text.