LANGUAGE MODELLING IN *ML*

A PREPRINT

Nitin Garg
Department of Mathematics and Statistics
IIT Kanpur
nitingrg@iitk.ac.in

Pulkit Gopalani
Department of Electrical Engineering
IIT Kanpur
gpulkit@iitk.ac.in

Tanvi Ajay Nerkar
Department of Electrical Engineering
IIT Kanpur
tanvin@iitk.ac.in

Chauhan Dev
Department of Computer Science and Engineering
IIT Kanpur
devgiri@iitk.ac.in

Aviral Aggarwal
Department of Chemical Engineering
IIT Kanpur
aviralag@iitk.ac.in

Shivam Gupta
Department of Economics
IIT Kanpur
guptash@iitk.ac.in

Aman Mittal
Department of Mathematics and Statistics
IIT Kanpur
amanmitl@iitk.ac.in

Garvit Bhardwaj
Department of Electrical Engineering
IIT Kanpur
garvitbh@iitk.ac.in

Abhishek Mittal
Department of Computer Science and Engineering
IIT Kanpur
amittal@iitk.ac.in

Abhay
Department of Computer Science and Engineering
IIT Kanpur
kabhay@iitk.ac.in

Raghav Kabra
Department of Mathematics and Statistics
IIT Kanpur
raghavka@iitk.ac.in

Sahaj Agrawal
Department of Electrical Engineering
IIT Kanpur
sahajag@iitk.ac.in

Ankur Banga
Department of Electrical Engineering
IIT Kanpur
ankurb@iitk.ac.in

Darsh Sharma
Department of Biological Sciences and Bio-Engineering
IIT Kanpur
darsh@iitk.ac.in

Dhruvil Sangani
Department of Mathematics and Statistics
IIT Kanpur
dhruvil@iitk.ac.in

Nikhar Agrawal
Department of Material Science and Engineering
IIT Kanpur
nikhar@iitk.ac.in

Supreet Agrawal
Department of Chemical Engineering
IIT Kanpur
supreeta@iitk.ac.in

Japneet Singh
Department of Electrical Engineering
IIT Kanpur
japneet@iitk.ac.in

JS Ladhar
Department of Civil Engineering
IIT Kanpur
jsingh@iitk.ac.in

Nishtha Attri
Department of Computer Science and Engineering
IIT Kanpur
nishthaa@iitk.ac.in

1 Introduction to Language Modelling

The goal of language modelling is to estimate the probability distribution of various linguistic units, e.g., words, sentences etc. Language Modeling is an important idea behind many Natural Language Processing tasks such as Machine Translation, Spelling Correction, Speech Recognition, Summarization, Question-Answering and Text suggestion.

Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken.Human speech, however, is not always precise – it is often ambiguous and the linguistic structure can depend on many complex variables, including slang, regional dialects and social context. NLP uses syntax to assess meaning from a language based on grammatical rules which includes parsing, word segmentation, sentence breaking, morphological segmentation and stemming. Semantics involves the use and meaning behind words. NLP applies algorithms to understand the meaning and structure of sentences. Techniques that NLP uses with semantics include word sense disambiguation, named entity recognition and natural language recognition.

1.1 Problem Statement

Pre-trained word representations are a key component in many language understanding models. However, learning high quality representations can be challenging. They should ideally model both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts. The idea in the project is to develop a type of deep contextualized word representation, called ELMo (Embeddings from Language Models) that directly addresses both challenges, can be easily integrated into existing models, and improves the state of the art in every considered case across a range of challenging language understanding problems.

1.2 Method of Approach

Different from traditional word embeddings, ELMo produced multiple word embeddings per single word for different scenarios. Higher-level layers capture context-dependent aspects of word embeddings while lower-level layers capture model aspects of syntax.

Each token is assigned a representation that is a function of the entire input sentence. We use vectors derived from a bidirectional LSTM that is trained with a coupled language model (LM) objective on a large text corpus.

1.3 Reasons behind Approach

Unlike previous approaches for learning contextualized word vectors, ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM. More specifically, we learn a linear combination of the vectors stacked above each input word for each end task, which markedly improves performance over just using the top LSTM layer. Combining the internal states in this manner allows for very rich word representations. Using intrinsic evaluations, we show that the higher-level LSTM states capture context-dependent aspects of word meaning (e.g., they can be used without modification to perform well on supervised word sense disambiguation tasks) while lower level states model aspects of syntax (e.g., they can be used to do part-of-speech tagging). Simultaneously exposing all of these signals is highly beneficial, allowing the learned models select the types of semi-supervision that are most useful for each end task.

2 Introduction To Deep Learning

In Deep Learning we talk about neural networks having very deep structure. Neural networks are inspired by the neurones in our brains and were made to learn various parameters on their own based on experience just like the human brain but one shouldn't really compare the two as no one really understands how the human brain works. Deep learning started growing these days very fast than some decades ago because of increase in huge amount of digital data, very fast GPUs, tremendous algorithmic innovations etc. Due to these developments now we can try new ideas and test it on deep structures with very less time than before which gave Deep Learning an acceleration towards development. Here are some basic and important terminologies explained:

2.1 Neural Network Units

Neural networks consist of lots of units whose function is to take input, do some process and give output. It is a fundamental structural unit of neural network which are connected to each other to take input and finally outputs the prediction. Our final goal is to minimize the cost.

2.2 Parameters of Neural Network

Neural Network are used for supervised learning and take input and process them using learned parameters to predict the result. These parameters or the weights are the result of the training of neural network which are used to predict the result of new unknown inputs.

2.3 Activation Function

A neural network unit takes input from other units and activates that input by applying activation function and outputs the activated value. There are so many activation functions like, ReLU, sigmoid, tanh, etc. We can choose it to be different for different units also. E.g. we can use tanh function for units except in the final unit and we can use sigmoid there because it gives answer between 0 to 1.

2.4 Loss Function

During training of a neural network first we implement forward propagation which calculates all the activations and the predicted value according to our initial parameters then we take the real value and the prediction and use the loss function to determine the error in the prediction. During the training we will try to minimize the loss function so that prediction will be as accurate as possible. For binary classification we use the loss function:

$$J(\hat{y}|y) = -y \cdot log(\hat{y}) - (1-y) \cdot log(1-\hat{y})$$

where,

y = expected result
$\hat{y}$ = predicted result

Here in binary classification we represent one class as 1 and another as 0 as the answer of inputs. For models where the output is a real number rather than a discrete value we use:

$$J(\hat{y}|y) = (y-\hat{y})^2$$

2.5 Layers

Neural networks generally consist of many layers where the first layer through which it takes an input is called the input layer and the last layer where it predicts the result is called the output layer. There can exist one or more layers of units between the input and output layers. If there are no layers between them then it can be called a simple 1-Layered neural network that has no hidden layers. Data passes through all the layers getting manipulated using the obtained parameters and the activation functions to finally give an output.

2.6 Gradient Descent

It is an algorithm which can be used to train a machine learning model. In this algorithm we minimize our cost function for a given training dataset by updating our parameters upon each iteration such that we move opposite the gradient of the cost function. Upon each iteration we calculate the loss function using our current parameters and calculate gradients of loss function with respect to each parameter. Gradient with respect to a parameter gives a direction in which loss function increases maximum so we update that parameter by moving in the opposite direction w.r.t. its gradient using a learning rate.. This update will change our parameter such that it decreases the loss. Let J be the cost function, α be the learning rate and the parameters be W and b then equations for each iteration to update parameters are as below:

$$W = W - \alpha \frac{\partial J}{\partial W}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

2.7 Forward Propagation

Let's consider an L-layered neural network with parameters $W^{[1]}, W^{[2]}, W^{[3]}, \ldots, W^{[L]}$ and $b^{[1]}, b^{[2]}, b^{[3]}, \ldots, b^{[L]}$. Where $W^{[L]}$ will be a matrix of dimension (number of units in layer L, number of units in layer (L-1)) and $b^{[L]}$ will be a matrix of dimension (number of units in layer L, 1). Let activation of layer L will be $A^{[L]}$ and input be represented as $A^{[0]}(or X)$. Here input layer is taken as 0th layer so output layer will be Lth layer. So for the L-th layer we will have input $A^{[L-1]}$ and parameters $W^{[L]}$ and $b^{[L]}$. So we will have to calculate activation of L-th layer. Equations to calculate it are given below:

$$Z^{[L]} = W^{[L]} \cdot A^{[L-1]} + b^{[L]}$$

$$A^{[L]} = g(Z^{[L]})$$

Here g is the activation function , for example let for l = L , g = sigmoid and for the other layers $g = tanh$. For multiclass classification we can also use g = softmax function for l = L. Commonly used activation functions are Sigmoid, Hyperbolic Tangent, ReLu etc. There exists a possibility that a node might die out if ReLu is used and therefore a better version called Leaky ReLu is used. ReLu: $g = max(0, z)$ Leaky ReLu: $g = max(0.01z, z)$ This factor of 0.01 might vary.

2.8 Backword Propagation

Above we calculated the activations of all the layers through forward propagation and finally predicted the result but during training we have to calculate their gradients too. To calculate the partial derivative of the cost function w.r.t. each parameter we will have to back propagate through all the layers. Back propagation uses chain rule of derivatives to calculate gradients. Equations for back propagation are given below:

Here for layer l we have input $\frac{\partial J}{\partial A^{[l]}}$ and output will be $\frac{\partial J}{\partial A^{[l-1]}}$, $\frac{\partial J}{\partial W^{[l]}}$ and $\frac{\partial J}{\partial b^{[l]}}$.

$$\frac{\partial J}{\partial Z^{[l]}} = \frac{\partial J}{\partial A^{[l]}} \cdot g'(Z^{[l]})$$

$$\frac{\partial J}{\partial W^{[l]}} = \frac{\partial J}{\partial Z^{[l]}} \cdot A^{[l-1]T}$$

$$\frac{\partial J}{\partial b^{[l]}} = \frac{\partial J}{\partial Z^{[l]}}$$

$$\frac{\partial J}{\partial A^{[l-1]}} = W^{[l]T} \cdot \frac{\partial J}{\partial Z^{[l]}}$$

These equations are for input of a single data, if we make a matrix having m columns and each column is a different input then equations will be a little different. Here, l-th activation of i-th input will be $A^{[l](i)}$ then $A^{[l]} = [A^{[l](1)}, A^{[l](2)}, ..., A^{[l](m)}]$ then $\frac{\partial J}{\partial Z^{[l]}} = [\frac{\partial J}{\partial Z^{[l](1)}}, \frac{\partial J}{\partial Z^{[l](2)}}, ..., \frac{\partial J}{\partial Z^{[l](m)}}]$.

$$\frac{\partial J}{\partial Z^{[l]}} = \frac{\partial J}{\partial A^{[l]}} \cdot g'(Z^{[l]})$$

$$\frac{\partial J}{\partial W^{[l]}} = \frac{\frac{\partial J}{\partial Z^{[l]}} \cdot A^{[l-1]T}}{m}$$

$$\frac{\partial J}{\partial b^{[l]}} = \frac{\sum_{i=1}^{m} \frac{\partial J}{\partial Z^{[l](i)}}}{m}$$

$$\frac{\partial J}{\partial A^{[l-1]}} = W^{[l]T} \cdot \frac{\partial J}{\partial Z^{[l]}}$$

3 Backpropagation

Back-propagation is by far one of the most mathematical, and at the same time the most logical and interesting part of neural nets. It uses basic calculus principles and Gradient descent to efficiently tune parameters, reducing the error on training set. Basically we calculate how much of the total error in the final output is caused due to the individual weights, and then change the weights accordingly to reduce error in prediction. We will henceforth use the logistic error function as our error, given by

$$L(Y, \hat{Y}) = -(Y \log(\hat{Y}) + (1 - Y) \log(1 - \hat{Y}))$$

Where Y is the ground truth value, and $\hat{Y}$ is the predicted value. It is summed over all training examples to give the total error. The parameters are tuned using gradient descent, i.e. moving towards the direction of steepest decrease of the Loss function. Something like :

$$W_{ij} = W_{ij} - \alpha \cdot \frac{\partial L(Y, \hat{Y})}{W_{ij}}$$

In backprop, there are various layers to be taken into account for the total error, as the error propagates through all layers till the final output. Suppose we need to find the gradient of loss wrt. a weight W_{ij} (j^{th} weight of the i^{th} layer). One could write that as

$$\frac{\partial L(Y, \hat{Y})}{\partial W_{ij}}$$

But this in turn depends on the previous layer $\hat{Y}$. Therefore we first need to differentiate wrt $\hat{Y}$

$$\frac{\partial L(Y, \hat{Y})}{\partial \hat{Y}}$$

and so on for subsequent layers until we reach the required layer / weight.

Using chain rule the derivative can be written as

$$\frac{\partial L(Y, \hat{Y})}{\partial W_{ij}} = \frac{\partial L(Y, \hat{Y})}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial W_{ij}}$$

Similarly, we will back-trace our path to the W_{ij} neuron, computing derivatives on our way, just as we did above. The chain rule makes life simpler, we just need to multiply to change the variable of differentiation. The logistic loss function derivative wrt $\hat{Y}$ in (2) is

$$\frac{\partial L(Y, \hat{Y})}{\partial \hat{Y}} = \frac{(\hat{Y} - Y)}{(\hat{Y}(1 - \hat{Y}))}$$

Further differentiating wrt the output of the last layer, Z, we get

$$\frac{\partial \hat{Y}}{\partial W_{ij}} = \frac{\partial \hat{Y}}{\partial Z} \frac{\partial Z}{\partial W_{ij}}$$

Assuming we used the sigmoid function to calculate

$$\hat{Y} = \sigma(Z)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

the required derivative at this stage is

$$\frac{\partial \hat{Y}}{\partial Z} = \sigma(1-\sigma) = \hat{Y}(1-\hat{Y})$$

Similarly differentiating wrt different layers and multiplying, we get a chain of derivatives which eventually give the gradient wrt the required weight.

The values required in the gradient are stored as cache with each layer, so that during back-propagation pass the values do not have to be calculated again, and simply plugging in the values in the above equations we can get the gradient w.r.t. any weight / bias / input. Now to implement gradient descent we just have to repeat the following process :

$$W_{ij} = W_{ij} - \alpha \cdot dW_{ij}; \qquad dW_{ij} = \frac{\partial L(Y, \hat{Y})}{\partial W_{ij}}$$

α is the learning rate, i.e. how long jumps we take in the direction of gradient.

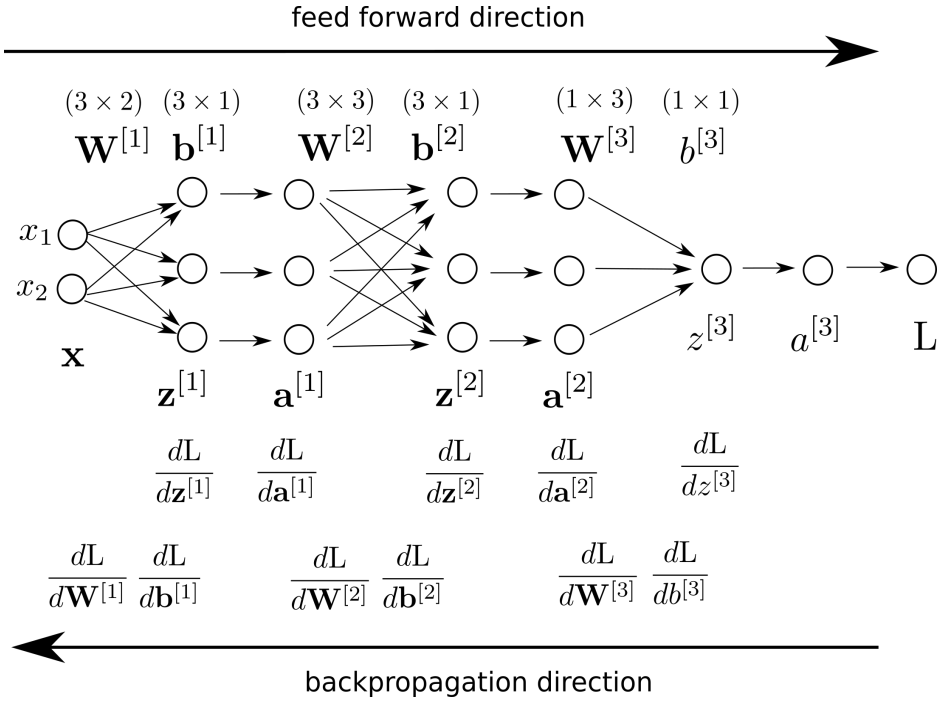

Figure 1: Forward and Backpropagation

There are some problems with training neural networks having many hidden layers. It so happens that due to very deep networks, the initial layers of the network are updates very slowly, due to very less gradient corresponding to those layers. The gradients are low because of the "Vanishing Gradients" problem, where a chain of very low gradients when multiplied sequentially from the end of the network result in negligible resultant gradient after say, around 10-20 layers into the network. Learning rates as seen are proportional to the gradient of the neuron. The above problem is demonstrated from the following graph (Figure 2) :

Activation functions like tanh, sigmoid are generally the cause of vanishing gradients, because of the nature of the functions (gradient tends to 0 after some range of input). Therefore to avoid the vanishing gradient problem, activations like ReLU are used, whose gradient does not have the above problem.

To speed up the computation we use a technique called vectorization in which we combine all individual vectors of layers into one single matrix and then apply the relevant operations. The back-propagation equations in vectorized method can be summarized as below:

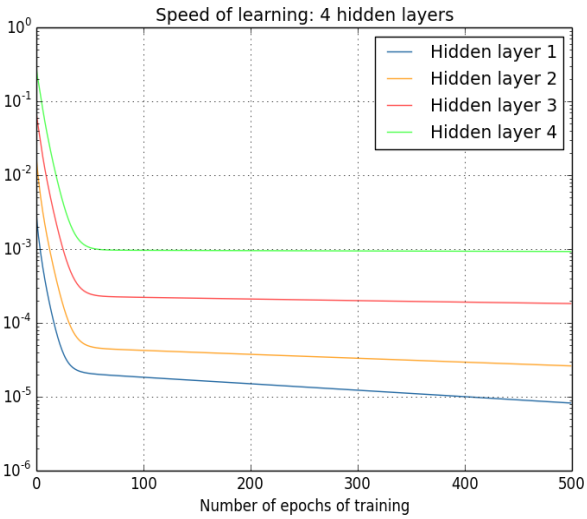

Figure 2: Learning rates for various layers

For the last layer(L) the equations are:

$$dZ^{[L]} = A^{[L]} - Y$$
$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{[L]^T}$$
$$db^{[L]} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[L](i)}$$

For any other intermediate layer(l):

$$dZ^{[l]} = dW^{[L]^T} dZ^{[l+1]} g'^{[l]}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l]^T}$$
$$db^{[l]} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$$

where W is the matrix formed by concatenating all the W_{ij}s vertically
Z is the matrix formed by concatenating all the Z_{ij}s vertically
b is the matrix formed by concatenating all the b_{ij}s vertically
A is the matrix formed by concatenating all the A_{ij}s vertically

$$A_{il} = g(Z_{il})$$

where $g^{[l]}$ is the activation function of l^{th} layer.

References

[1] Andrew Ng. Neural networks and Deep learning, Coursera

[2] Michael Nielsen. Neural Networks and Deep learning (Webpage)

4 Sequence Models

Sequence Models are models in which the data is in the form of a sequence, for example words in a sentence or notes in a song.

4.1 Recurrent Neural Networls (RNNs)

Deep learning is implemented in sequence models through recurrent neural networks or RNNs. Conventional deep neural nets aren't used to train sequence models for two main reasons:

- They won't be generalized as it won't share features learned across different positions in the sequence
- Input and output are not necessarily of the the same or known length for a sequence model

RNNs avoid these two issues by implementing a network in which the data (activation output) from the previous timestamp in the sequence is passed onto the next and used as a parameter when evaluating the output of that time stamp. There are different types of RNNs for different types of sequence data for example:

- Speech recognition has a continuous input given for a continuous sequential output
- For generation of audio, it could just be a number going to a continuous sequential output
- Sentiment analysis is an example of classification where input can be variable

Another issue in this model is that it's generalizing only in one direction. Bidirectional RNNs are used heavily in language models to get rid of this issue.
There are multiple cell structures in RNNs eg. GRU, LSTMs,basic RNN cells etc. LSTMs are most widely used so that information from previous data in a sentence can be carried through the sentence if needed. The basic layout of an RNN and the LSTM(Long Short Term Memory) cell structure is shown below.

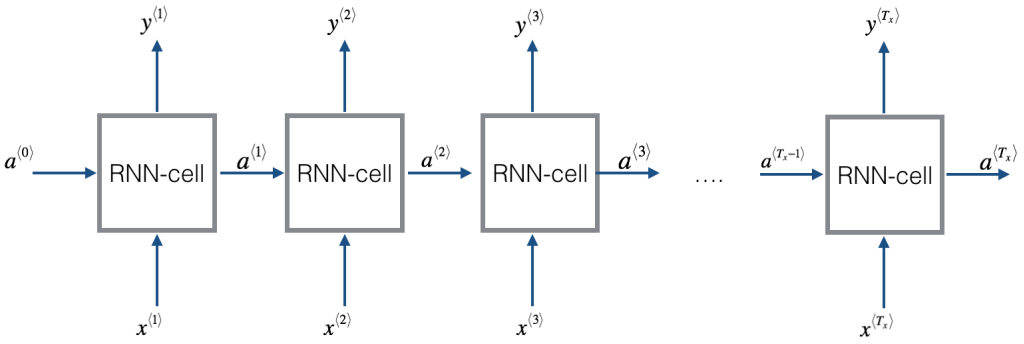

Figure 3: Forward Prop in Basic RNN Structure

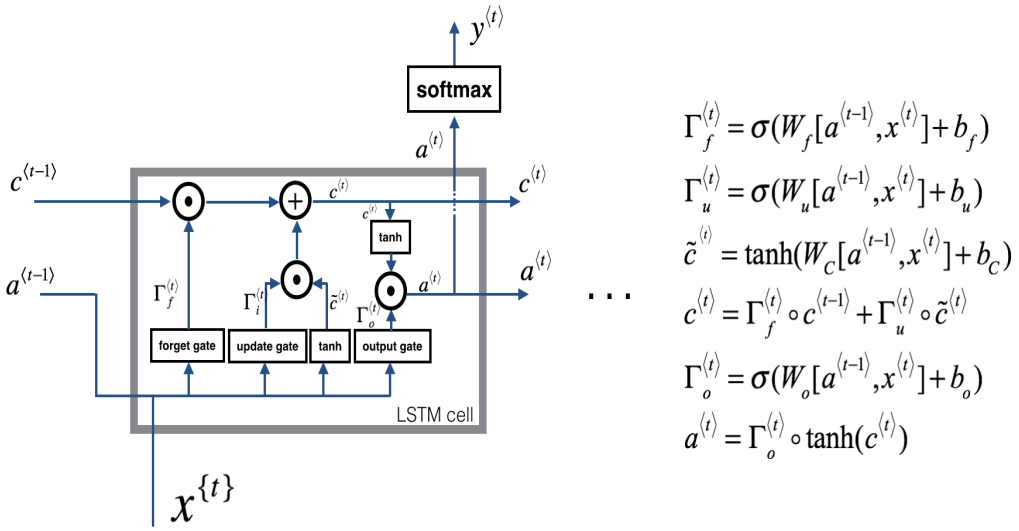

Figure 4: LSTM Cell and Equations Involved

5 Introduction To Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platform.

5.1 What Can Python Do?

1. Used to create Web Applications.
2. Used alongside software to create workflows.
3. Can be used to read and modify files.
4. Can be used to handle big data and further data analysis.
5. Can be used for rapid prototyping, or for production-ready software development.
6. Can be used to train and build machine learning models using various in-built libraries.

5.2 Why Python?

Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc). It has a simple syntax similar to the English language. It runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

5.3 Python Compared To Other Programming Languages

Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

5.4 Basic Implementation Of Python In This Project

- Since computer doesn't understands raw text(human language), first step while developing any deep learning algorithm(where we have raw text as input) is to clean the text, remove useless unigrams and bigrams and other common words that don't affect the meaning of the text and convert the given text into computer readable feature vectors(multi-dimensional arrays).

- After conversion to computer readable form the feature vectors the dataset is split into train and test set.(and cross validation set).
- The training set of feature vectors is then fed to a machine learning algorithm and the trained model is then used to predict the outcomes for the test set.
- Comparing the predicted results(on cross validation set)and actual results we can get an idea of the accuracy and efficiency of our model and we can then change various parameters to increase the accuracy of our model.

Since Python is an Object Oriented Programming Language, it has various in-built libraries with different classes, modules and functions that can help in making such models.

5.5 Python Libraries Used

The different libraries that will be used in this project include:

1. **NUMPY:** NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.It contains various features including these important ones:
   - A powerful N-dimensional array object
   - sophisticated (broadcasting) functions
   - Useful linear algebra, Fourier transform, and random number capabilities
2. **MATPLOTLIB:** Matplotlib is a python-package that can be used to generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. These figures help us to get a better understanding of the working of our models.
3. **PANDAS:** PANDAS is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
4. **SCI-KIT LEARN:** Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
5. **NLTK:** This library is used for text preprocessing. NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

6 N-grams

The simplest model that assigns probabilities to sentences and sequences of words, the n-gram. An n-gram is a sequence of N words: a 2-gram (or bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”, and a 3-gram (or trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”. In a bit of terminological ambiguity, we usually drop the word “model”, and thus the term n- gram is used to mean either the word sequence itself or the predictive model that assigns it a probability.

P(w|h), the probability of a word w given some history h. Suppose the history h is *“its water is so transparent that”* and we want to know the probability that the next word is *the*:

$$P(the \mid its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

We need to introduce clever ways of estimating the probability of a word w given a history h, or the probability of an entire word sequence W . To represent the probability of a particular random variable X_i taking on the value *“the”*, or P(X_i = *“the”*), we will use the simplification P(*the*). We’ll represent a sequence of N words either as $w_1 \ldots w_n$ or w_n (so the expression w_{n-1} means the string $w_1, w_2, \ldots, w_{n-1}$). For the joint probability of each word in a sequence having a particular value P($X = w_1, Y = w_2, Z = w_3, \ldots, W = w_n$) we’ll use P($w_1, w_2, \ldots, w_n$).

We can decompose this probability using the chain rule of probability:

$$\begin{aligned} P(X_1 \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1^2)\ldots P(X_n|X_1^{n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_1^{k-1}) \end{aligned}$$

Applying the chain rule to words, we get

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned}$$

We can't just estimate by counting the number of times every word occurs following every long string, because language is creative and any particular context might have never occurred before! The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

When we use a bi-gram model to predict the conditional probability of the next word, we are thus making the following approximation: The assumption that the probability of a word depends only on the previous word is called a Markov assumption. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past.

The general equation for this n-gram approximation to the conditional probability of the next word in a sequence is

$$P(w_i|w_1^{n-1}) \approx P(w_i|w_{n-N+1}^{n-1}) \tag{1}$$

An intuitive way to estimate probabilities is called maximum likelihood estimation or MLE. We get the MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1.

To compute a particular bigram probability of a word y given a previous word x, we'll compute the count of the bigram C(*xy*) and normalize by the sum of all the bigrams that share the same first word x:

$$P(w_i|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{2}$$