Language Translation using Deep Learning

Project Submitted to the

INSTITUTE OF CHEMICAL TECHNOLOGY, MUMBAI

for the award of the degree of

MASTER OF SCIENCE

In

ENGINEERING MATHEMATICS

By

Rahul Chandraprakash Pal

Under the supervision of

Dr. Amiya Ranjan Bhowmick

Department of Mathematics

Institute of Chemical Technology, Mumbai

(University under Section 3 of UGC Act 1956;

Elite Status and Centre of Excellence, Government of Maharashtra)

Maharashtra, India

April 2019

INSTITUTE OF CHEMICAL TECHNOLOGY

(university under section-3 of UGC Act 1956)

Elite Status & Center of Excellence - Government of Maharashtra Matunga,

Mumbai - 400019, India.

DECLARATION

I, Rahul Chandraprakash Pal (Roll no:17MAT210), the author of the project report entitled **"Language Translation using Deep Learning"**, hereby declares that this is an independent work of mine to the Institute of Chemical Technology, Mumbai carried out towards partial requirement of Master of Science in Engineering Mathematics. This work is not submitted to any other institution for award or degree.

Mr. Rahul Chandraprakash Pal

INSTITUTE OF CHEMICAL TECHNOLOGY

(university under section-3 of UGC Act 1956)

Elite Status & Center of Excellence - Government of Maharashtra

Matunga, Mumbai - 400019, India.

CERTIFICATE

This is to certify that the work contained in this project report entitled **"Language Translation using Deep Learning** submitted by **Mr. Rahul Chandraprakash (Roll No.: 17MAT210)** to the Institute of Chemical Technology, Matunga towards partial requirement of Master of Science in Engineering Mathematics has been carried by him under my supervision.

Dr. Amiya Ranjan Bhowmick

Project Guide

INSTITUTE OF CHEMICAL TECHNOLOGY

(university under section-3 of UGC Act 1956)

Elite Status & Center of Excellence - Government of Maharashtra

Matunga, Mumbai - 400019, India.

APPROVAL OF PROJECT

This is to certify that the work contained in this project report entitled **"Language Translation using Deep Learning** submitted by **Mr. Rahul Chandraprakash (Roll No.:17MAT210")** submitted by to the Institute of Chemical Technology, Matunga towards partial requirement of Master of Science in Engineering Mathematics has been carried by him under my supervision.

Guide: **Head of the Department :**

Dr. Amiya Ranjan Bhowmick **Dr. Ajit Kumar**

External Examiner :

Date:

INSTITUTE OF CHEMICAL TECHNOLOGY

(university under section-3 of UGC Act 1956)

Elite Status & Center of Excellence - Government of Maharashtra

Matunga, Mumbai - 400019, India.

ACKNOWLEDGEMENT

I am thankful to acknowledge the management of Institute of Chemical Technology and Department of Mathematics for arranging the project as a part of M.Sc.(Engineering Mathematics) curriculum. I would like to express my special thanks to my guide Dr. Amiya R. Bhowmick and Mr. Venkata Reddy Konasani, Statinfer, for giving me the opportunity to do this project related to Language Translation using Deep Learning, which also helps me to know about so many new things. I am very much thankful for inspiration and guidance, which I have received from them. I would also like to thank my classmates who rendered their help during the course of this seminar.

Mr. Rahul Chandraprakash Pal

Contents

1 Background Research

1. **General Observation :** I realised that the project we are doing the background research for is part of "Machine Translation". It is generally known as MT (abbreviation), and it deals with the investigation of the use of software to translate text or speech from one language to another.
    i. Machine Translation is not a new idea, it was proposed few decades before, more precisely around 1949 in Warren weaver's Memorandum on Translation.
    ii. It is also said that this idea of universal language with equivalent idea in different tongues sharing one symbol is proposed by Rene Descartes in 1629.
    iii. From 1949 on wards there is a long list of researchers contributed to this idea in different ways and it is materialized in 21st century through the advancement of modern scientific computation.

2. **Basic Translation Idea :**
    i. Human translation can be explained as decoding the meaning of the text from the source language and re-encoding the meaning into the target language.
    ii. As this process is concerned, one should know the basic rules of both the languages (source language and target language), precisely the *Grammar*. This suggests that it involves a lot of human intuitions.
    iii. To carry out this using machine (computationally) is a big challenge. So, for making this possible, it cannot be done automatically, a kind of human aid is needed.
    iv. Adding to this, computational translation may not be exact but this might give a good idea of the scenario which is enough for human intelligence to interpret. But this kind of giving only idea may satisfy a small fraction of the appetite but what about the fields where exact interpretation cannot be compromised.
    v. So, the different approaches have evolved with the time to attain the needed accuracy. (work is still on)

3. **Approaches :** There are some of the approaches are discussed below
    i. **Rule-Based machine translation :** This idea basically includes basically three types of approaches.
        A. **Transfer-based machine translation :** Basically, this approach depends on the language pairing. To explain in detail, in this method text from the source language is first analysed morphologically and syntactically to get the syntactic representation.

This representation is then taken to the next level where more emphasis is given on the parts relevant for the translation. In this process some extra information such as, the use of prepositions, are ignored. Then with the help of transfer process (there are different types of transfer process are involved depending on the linguistic structure) the same level of significance is attained in the target language. The final text is represented in target language. These intermediate steps are also carried out in reverse order may be to check the level of accuracy.

B. **Interlingual machine translation :** This approach involves an intermediate step called interlingua, this helps in translating the source text into more than one target language simultaneously. This method is considered to be economical.

C. **Dictionary based machine translation :** In this approach, the translation carried out as per the dictionary. Speaking precisely it translates as per the dictionary translation.

ii. **Statistical machine translation :** This approach basically depends on the bilingual corpus, which means it should be fed with huge amount of data of the human translated documents form different sources available concerning the source and target language. More the amount of such data available to feed the model better will be the accuracy. Such documents are generally available from European parliament for the language translation like English to french. This method requires huge database, and it struggles with the translation of morphologically rich languages.

New approaches into statistical learning are METIS II and PRESEMT, they require less corpus and mainly focus on syntactic representation of the translated text.

iii. **Example based machine translation :** This approach involves the corpus, which have the text already translated, the text from the source language matched with text from the corpus and this sub-sentential parts are translated with the help of the corpus and then the parts of the text is incorporated to form the final translation.

iv. **Hybrid machine translation :** This approach is the combination of rule based and the statistical machine translation, this is believed to more effective than the above mentioned approaches. This is carried in different ways :

A. **Rules post-processed by statistics :** The translation is carried out using the rules and then statistical method is used to adjust the translation.

B. **Statistics guided by rules :** The text from the source language is processed with the rules (source language) and makes it available for statistical method to translate and then the resulting text goes through the rules (target language) again. This method is

believed to be powerful since it incorporated both the method.

v. **Neural machine translation :** This approach is deep learning based , Neural machine translation is been in trend in recent time. Google, which was using the statistical machine translation is also announced to use this approach for translation, there are many giants out there are using NMT since 2017.

4. **Major issues :**

i. **Disambiguation :** In any language, when the same word is used in different situation to explain the scenario, it becomes a challenge for the machine translation to pick it up.

ii. **Non-standard speech :** Since, the machine translation models are trained with the standard language text data, it becomes difficult to translate the non-standard languages. Non-standard languages like vernacular source, as a result, erroneous translation is acquired.

iii. **Name-entity :** This is being one of the major issues, since name should not be translated, it has to be same in any language. So, it requires transliteration not the translation. Now, there are many instances where it was challenging situation for the model to differentiate between the text to transliterate and the text to translate. So, there different ideas proposed for negating this issues, like assigning the class to the nouns.

Translation from multi parallel sources : There is an approach where the text is translated to the target language using the combination of more than one source language, this would definitely produce more accurate results than it would with the single source language.

Evaluation of the machine translation : Before evaluating any machine translation approach, one thing need to be understood that different approaches performs well in particular fields.

Example Statistical machine translation out performs the example based machine translation in a given situation but when comes to translation of the english to french texts, example based machine translation easily out performs the statistical machine translation.

So, the best machine translation models need to be selected based on the situation given. There are some automated evaluating systems like BLEU, NIST, METEOR, LEPOR etc. which gives rating to the translation, based on how close is the machine translation to the human translation. Closer to the human translation higher is the rating.

Some of the links explored are as follows:	
Source	Content
https://github.com/OValery16/Language-Translation-with-deep-learning-	It's a project in GitHub, related to language translation with deep learning using keras.
https://machinelearningmastery.com/prepare-french-english-dataset-machine-translation/	This is a project, readily available, where the dataset is prepared for french to english translation, with the detailed steps of machine translation.
http://www.nltk.org/	Documentation: NLTK (natural language toolkit): This includes API, and other necessary documentation related to NLTK.
http://opennmt.net/OpenNMT-tf/	Documentation: Open NMT(Neural Machine Translation)-TensorFlow
https://github.com/OpenNMT/OpenNMT-tf	GitHub: Open NMT(Neural Machine Translation)-TensorFlow
https://github.com/OpenNMT/OpenNMT-py	GitHub: Open NMT(Neural Machine Translation)-PyTorch Here, all the codes related to PyTorch is provided as a step by step process.
http://opennmt.net/OpenNMT-py/	Documentation: Open NMT(Neural Machine Translation)-PyTorch All the required documentation is provided under this link.

http://opennmt.net/Models-py/	Pretrained models:Open NMT(Neural Machine Translation) PyTorch Available models trained using OpenNMT. German to English, English Summarization, Chinese Summarization, Dialog System.
https://cloud.ibm.com/apidocs/language-translator?language=python#get-model-details	API docs: IBM watson language translator, which translates text from one language to another, it is claiming to offer multiple translation models which can be customised as per the need, with regards to terminologies and the language.
https://github.com/IBM-Bluemix-Docs/language-translator/blob/master/customizing.md	GitHub: Here, the code is shared, also provided the step by step process of the model. It's seen that "Parallel corpus customization" is used here.
https://github.com/shvmshukla/Machine-Translation-Hindi-to-english-/blob/master/Machine_translation_code.ipynbhttps://github.com/shvmshukla/Machine-Translation-Hindi-to-english-/blob/master/Machine_translation_code.ipynb	GitHub: It is a project for translation from hindi to english with the python code, done by IITB students.

2 Introduction

"If you talk to a man in a language he understands, that goes to his head. If you talk to him in his own language, that goes to his heart." – Nelson Mandela.

The skills of translation are becoming ever more important and desirable. Today's multicultural and multilingual society demands effective, efficient, and empathetic communication between languages and cultures. That's important for various reasons. i) Not Everybody Speaks English(Global language). ii) It Enables A Global Economy. iii) The Spread of Information and Ideas.

Problem statement: The objective is to convert a Engish sentence to its Hindi counterpart using a Neural Machine Translation (NMT) system. The goal is to build a model with the help of neural networks, using python (precisely Tensorflow, keras), which translates the text into desired language. But, for this model to build, we need good size of the pre-translated data and then data should be prepared for the further process of building the model.

3 Logistic Regression

3.1 What is the need of logistic regression?

We know linear regression is useful for finding the relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. In linear regression, the dependent variable is predicted using independent variables. But what is need of non-linear regression if we have already linear regression that means we have to some other requirements or some other need to go for non-linear regression. let's understand with simple example data set provided by Venkata Reddy Konasani. We have **Product Sales Data** which contains two variable **Bought** and **Age**. We build a predictive model for **Bought** vs. **Age** which is linear regression model. Age is predictor variable and bought is predictive variable. Our goal is to predict whether the person is going to buy a product or not corresponding their age. Now our linear regression model is,

$$\text{Bought} = \beta_0 + \beta_1 \times \text{Age}$$

R-squared

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. R-squared is percentage of the response variable variation that is explained by a linear model.
R-squared = $\frac{\text{Explained Variation}}{\text{Total Variation}}$.

In general, the higher the R-squared, the better the model fits your data.

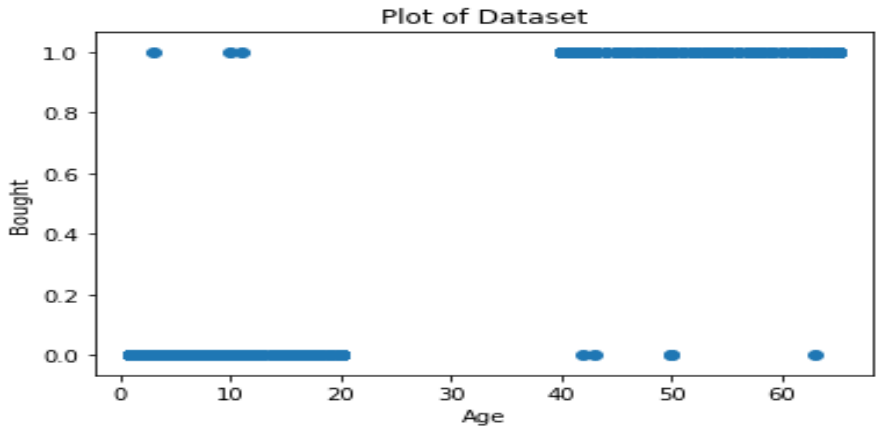
For fitting a linear regression model on **Product Sales Data** set in `Python` environment, we create linear regression between **Bought** and **Age** and then we find **R-Squared**. Our goal is to find if the customer's age is 4, then that customer will buy the product or not. And if the customer's age is 105, then the customer will buy the product or not.

```
##### LAB-What is the need of logistic regression #####
#Import the Dataset: Product Sales Data/Product_sales.csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
sales=pd.read_csv("/home/msc1/Desktop/Rahul/Product_sales.csv")
#What are the variables in the dataset?
sales.columns.values
#Build a predictive model for Bought vs Age
#If Age is 4 then will that customer buy the product?
import sklearn as sk
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(sales[["Age"]], sales[["Bought"]])
age1=4
predict1=lr.predict(age1)
print(predict1)
### for age=4 value is less than zero
#If Age is 105 then will that customer buy the product?
age2=105
predict2=lr.predict(age2)
print(predict2)
### for age=105 value is greater than one.
x = sales.Age
y = sales.Bought
# plot of dataset
```

```
plt.scatter(x,y)
plt.title('Plot of Dataset')
plt.ylabel('Bought')
plt.xlabel('Age')
```

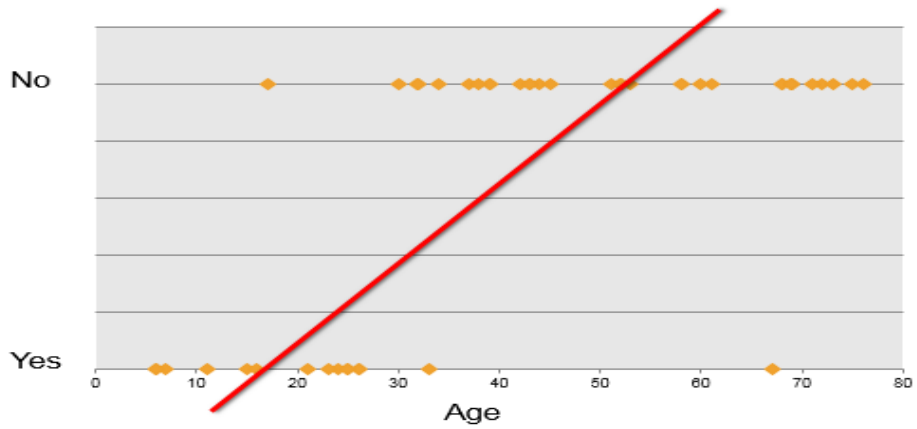
Result : For customer's age is 4, we get a less than zero value and if the customer's age is 105, then the value is greater than one. So, from this linear regression we can not interpret whether a person buys the product or not.

As we see, **Age** is continuous data and it takes value according to the given range (0 to 60). But, on other hand **Buy** is not continuous data, it is binary and it has only 0 and 1 data.



Conclusion :

1. We observe that the model which we build above is not right.
2. There is certain issue with the type of dependent variable.
3. Here dependent variable is not continuous, it is binary.
4. So, here we can't fit linear.

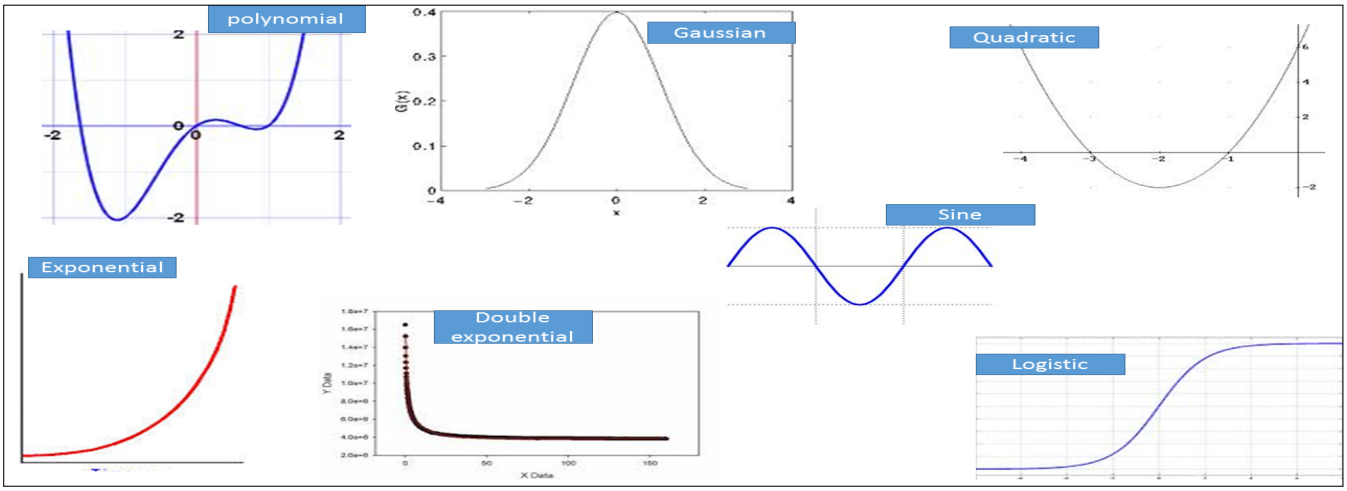


Hence, at the end no matter what we do or no matter what optimization we use, we are not able to fit the linear regression line to this data.

So, there are many real-life example where we can't fit the linear regression line they are as follow,

1. **Gaming - Win vs. Loss**
2. **Sales - Buying vs. Not Buying**
3. **Marketing - Response vs. No Response**
4. **Credit Card Loan - Default vs. Non Default**
5. **Operation - Attrition vs. Retention**
6. **Websites - Click vs. No Click**
7. **Fraud Identification - Fraud vs. Non Fraud**
8. **Healthcare - Cure vs. No Cure**

Some Non-linear Function :



As we can see in the functions given, there is a polynomial function; will that be able to fit our data? The answer is that it won't fit our data. Again, there are a Gaussian function, quadratic equation, exponential function, double exponential and sine function won't be fitting to our data. Now, there is logistic function; which we can fit to our data.

3.2 Building logistic Regression line

A Logistic Function :

As it looks like "**S**" shape and may be if we adjust some parameters, then we can fit the our data, because it looks like it's tails are longer and the mid portion are shorter and it will be a good fit to our dataset. So we will use this logistic function to fit to our data rather than linear function. Earlier the equation of

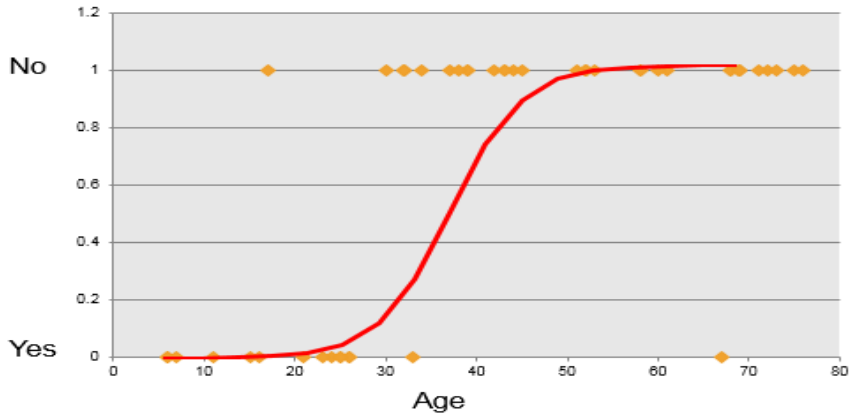
linear regression was

$$y = \beta_0 + \beta_1 x.$$

Now, for logistic regression we need the model that predicts the probabilities between 0 and 1. Some people of age less than 0s portion are not buying and some people of age more than 1s are buying. Now, what is the logistic regression equation? Earlier in the linear regression line, the value of equation will be simple equation but for the logistic regression line equation we have this,

$$\text{Probability} = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}.$$

So if we want a fit a logistic function on this data set it is descent function atleast better than linear line.



For building a logistic regression model on **Product Sales Data** set in `Python` environment. Thus, for that we have to create logistuc regression line between **Bought** and **Age**.

```
# Importing Dataset: Product_sales.csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
sales=pd.read_csv("/home/msc1/Desktop/Rahul/Product_sales.csv")
sales
# Build a logistic Regression line between Age and Buying
import statsmodels.formula.api as sm
logit=sm.Logit(sales['Bought'],sales['Age'])
logit
result = logit.fit()
```

```
result
result.summary()
# One more way of fitting the model.
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression()
logistic.fit(sales[["Age"]],sales["Bought"])

# A 4 years old customer, will he buy the product?
age1=4
predict_age1=logistic.predict(age1)
print(predict_age1)

# If Age is 105 then will that customer buy the product?
age2 = 60
predict_age2 = logistic.predict(age2)
print(predict_age2)
```

Result : Now after using logistic regression model on **Product Sales Data** for customer age is 4 we get 0 values and for the customer age is 105 we get the 1 value. So from this logistic regression we can easily interpret if customer age is 4, then they will not buy the product and if customer age is 105, then they will buy the product.

3.3 Multiple Logistic Regression

In the previous section we build a logistic regression model using single independent variable was **Age** using this independent variable we predicted the value if customer going to buy a product or not. But most of the time not always there is only one independent variable there would be more than one variable and using those variable we have to find out our outcome. Like Buying/Non-Buying depends on customer attributes like age, gender, place, income, etc. There is a data set called **Fiberbits** data provided by statinfer(Venkat Sir) and this is an internet service provider data set. Since last few years, there are some customers who stick with this service provider and there are some customers who left this service provider. Now Our goal is to build an attrition kind of model that will check whether the customer will be there with service provider or he will left. We are trying to do these things, because if we get to know who are going to leave and who are going to stick with service provider, then we can send them promotional code and offers to retain to this service provider. Thus, **Active cust** is the variable from which we can be

able to retrieve whether the customer is active or already left the network. There might be other variables that will be used as predictor variables. There are many reasons for the customers to stay back or leave the network. Here, the **Fiberbits** data is having 100000 rows and 9 columns. Columns like **active cust** which is having the value 0 or 1 and is the target variable. Data set having being in the network and leaving the network depends on the income, months on network, number of complaints they gave, number of plan changes they made, relocation status is like whether they relocated or not, the monthly bill that they are getting, technical issues per month and speed test results like whatever the speed they promised whether they gave it or not.

We will trying to build the model that will predict whether the customer will stay back or the customer will leave. And the model that we have to build is on the **Fiberbits** data.

Now here the **active customer** is having the values 0 or 1. Therefore we will directly build the model using rest of the variables. Here, the model will be depending on the active customer versus all the other variables.

Our Goal is to build a model to predict the chance of attrition for a given customer using all the features and what are the most imapcting variables?

```
# Lab code for multiple logistic regression
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
Fiber=pd.read_csv("E:\Rahul\Machine Learning Project\Logistic Regression/Fiberbits.csv")
list(Fiber.columns.values)
# Build a model to predict the chance of attrition for a given customer using all the
features.
from sklearn.linear_model import LogisticRegression
logistic= LogisticRegression()
logistic.fit(Fiber[["income"]+['months_on_network']+['Num_complaints']+
['number_plan_changes']+['relocated']+['monthly_bill']+['technical_issues_per_month']+
['Speed_test_result']],Fiber[['active_cust']])
predict1=logistic.predict(Fiber[["income"]+['months_on_network']+['Num_complaints']+
['number_plan_changes']+['relocated']+['monthly_bill']+['technical_issues_per_month']+
['Speed_test_result']])
predict1
```

Result : Here the model building is been done. Looking at the coefficients values of all the variables, we

can say if the value of probability by substituting all the variables is 1 or near to 1, then the customer might leave the network and if the probability is 0 or near to 0, then the customer will not leave the network. According to value of probability we start building the strategies thus somehow retain them.

3.4 Goodness of fit measures

After building the logistic regression line, the question arises that how good the model is? what is the confidence in your prediction or how good is your model or what is the goodness of fit of the particular model. What is the goodness of the fit measures in a logistic regression? In linear regression **R Squared** and **Adjusted R-Squared** are measuring the goodness of fit of model but in logistic regression it won't be work. Because **R Squared** and **Adjusted R-Squared** work explaining the variance by the regression line. The variance of y which was explained by regression is denoted by **R Squared** and **Adjusted R-Squared**. In logistic regression we don't have concept of the variance so we need to find some other alternative to find our goodness of fit for the this model.

There are other ways of finding out the way of goodness of fit for logistic regression as follow :

1. Classification matrix
2. AIC and BIC
3. ROC and AUC

We will only see Classification matrix, for building any model we have few actual values which are denoted by 0 and 1 and against those we build model and make prediction and our predicted value could be 0 and 1. It can sometime classify 0 as 0 and sometime classify 0 as 1.

Actual/Predicted	0	1
0	True Positive(TP)	False Negative(FN)
1	False Postive(FP)	True Negative(TN)

If the **"True Positives"** and **"True Negatives"** will be lesser and **"False Positives"** and **"False Negatives"** will be higher, then there will be something wrong in the prediction. For a good model, all 0 should be predicted as 0 and all 1 should be predicted as 1. The given table is called the confusion matrix or classification table. Accuracy is predicting the 0 as 0 and 1 as 1. If 0 is predicted as 1 and 1 is predicted as 0, then that is wrong prediction or called as miss-classification.

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP+TN+FP+FN})}$$

$$\text{Miss-classification} = \frac{(\text{FP+FN})}{(\text{TP+TN+FP+FN})} \quad \text{or} \quad 1 - \text{Accuracy}$$

Now our goal is to find accuracy value for fiber-bits model using spyder environment.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.cross_validation import train_test_split
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(Fiber[['active_cust']],predict1)
print(cm1)
# accuracy for the fiber bits model
total1=sum(sum(cm1))
accuracy1=(cm1[0,0]+cm1[1,1])/total1
accuracy1
```

Result : We get 76 % accuracy

3.5 Multicollinearity

Multicollinearity is nothing but the inter-dependency of the predictor variables. Within predictor variables, some variables are interdependent. The relation between X and Y is non linear, as we used logistic regression. The multicollinearity is an issue related to predictors variable. Multicollinearity need to be fixed in logistic regression also otherwise individual coefficients of the predictor will be effected by inter-dependency. The process of identification of multicollinearity is same as linear regression i.e. calculating VIF(Variance Inflation Factors) values beacuse multicollinearity is all about the relation within X variables i.e. $X1$ versus remaining variables that is how we find the VIF values. Hence we have to use VIF values to identify multicollinerity

Multicollinearity in Python

We will take **Fiber-bits** data set and for multicollinearity we need to find VIF. Any VIF value more that 5, which is an indication of multiollinearity. If we get for all the variable have VIF values is more than 5 that does not mean we have to drop all the varibles. We will drop one by one variable. First find out the VIF, go for the highest VIF, then observe is there ant VIF that is more than 5 then we can drop the highest variablr of VIF.

```
def vif_cal(input_data, dependent_col):
    x_vars=input_data.drop([dependent_col], axis=1)
```

```
    xvar_names=x_vars.columns
    for i in range(0,xvar_names.shape[0]):
        y=x_vars[xvar_names[i]]
        x=x_vars[xvar_names.drop(xvar_names[i])]
        rsq=sm.ols(formula="y~x", data=x_vars).fit().rsquared
        vif=round(1/(1-rsq),2)
        print (xvar_names[i], " VIF = " , vif)
vif_cal(input_data=Fiber, dependent_col="active_cust")
```

Result :

1. income VIF = 1.02
2. month on network VIF = 1.03
3. Num complaints VIF = 1.01
4. number plan changes VIF = 1.59
5. relocated VIF = 1.56
6. monthly bill VIF = 1.02
7. technical issues per month VIF = 1.06
8. Speed testresult VIF = 1.0

As we see no variable is having more than 5 VIF value, then it means that all the variables will be having independent information. We cannot drop the variables that are nearer values to 5, because if we drop it, then we may lose out on some information that might impact the overall accuracy of our model. Hence, that is how we find out multicollinearity.

3.6 Individual Impact of variables

Out of all the predictor variables, that we have used for prediction of the Buying or Non-Buying or that we have used for attrition versus non-attrition, hence the question arises is what are the important variables? If we have to choose top 5 variables, then what will be those variables? While selecting any model, is there any way that we can drop the variables that are not impacting or that are less impacting and keep only the important variables thus, we don't have to collect or maintain the data for those less impacting variables. Why should we keep the variables those who are not impacting? How to rank the predictor variables in order of their importance?

We have to consider the **Wald Chi-square** value or **Chi-square** value or the **z-square** value. Let's observe what are the top impacting variable or how do we rank the variable in our **fiberbits** data.

```
result1=sm.Logit(Fiber['active_cust'],Fiber[["income"]+['months_on_network']+
['Num_complaints']+['number_plan_changes']+['relocated']+['monthly_bill']+
['technical_issues_per_month']+['Speed_test_result']])
result1
result1.fit()
result1.fit().summary()
result1.fit().summary2()
```

Result: Top impacting variables are – relocated and Speed-test-result Least impacting variables are – monthly-bill and income.

3.7 Model selection

While trying to build the best model, we tend to add many variables, we tend to derive many variables, we tend to add many data and we sometimes try to build an optimal model by dropping the variables. For improving the model, there are many possibilities which we can implement it.
May be instead of building a model with 200 or 300 variables, we want to build a model with just 50 or 60 variables or may be 20 variables and have the best accuracy possible. We can have the most top impacting variable and have an accuracy of 80 % , rather than having 200-300 variables and accuracy of 85 %. Thus, how do we build if we have several models with same accuracy level or how do we choose best models. Considering that there are different models called M1, M2 or M3, then how do we get to know that M1 or M2 or M3 is optimal model for the data? Hence, that question can be answered by observing the **AIC**(Akaike Information Criterion) and **BIC**(Bayesian Information Criterion) values.
AIC is the information loss while building the model. Hence, we want to lose the least amount of information while building the model.

```
# logistic regression model selction
#Find AIC and BIC values for the first fiberbits model (M1) including all the variables.
m1=sm.Logit(Fiber['active_cust'],Fiber[["income"]+['months_on_network']+
['Num_complaints']+['number_plan_changes']+['relocated']+['monthly_bill']+
['technical_issues_per_month']+['Speed_test_result']])
m1
m1.fit()
m1.fit().summary2()
# What are the top-2 impacting variables in fiber bits model?
```

```
#What are the least impacting variables in fiber bits model?
m1.fit().summary()
# Income and Monthly Bill Dropped because those are the least impacting variables.
m2=sm.Logit(Fiber['active_cust'],Fiber[['months_on_network']+['Num_complaints']+
['number_plan_changes']+['relocated']+['technical_issues_per_month']+
['Speed_test_result']])
m2
m2.fit()
m2.fit().summary()
m2.fit().summary2()
# Dropping high impacting variables relocated and Speed_test_result.
m3=sm.Logit(Fiber['active_cust'],Fiber[["income"]+['months_on_network']+
['Num_complaints']+['number_plan_changes']+['monthly_bill']+
['technical_issues_per_month']])
m3
m3.fit()
m3.fit().summary()
m3.fit().summary2()
```

Results : Logit			
Model:	Logit	Pseudo R-squared:	0.143
Dependent Variable:	active cust	AIC:	116629.0918
Date:	2019-04-13 15.37	BIC:	116686.1694
No. Observation:	100000	Log-Likelihood:	-58309.
Df Model:	5	LL-Null:	-68704.
Df Residuals:	99994	LLR p-value:	0.0000
Converged:	1.0000	Scale:	1.0000
No. Iteration	7.0000		

	Coef.	Std.Err	z	$P > \|z\|$	[0.025	0.0975]
income	0.0012	0.0000	33.1411	0.0000	0.0011	0.0013
months on network	0.0424	0.0006	75.9838	0.0000	0.0413	0.0434
Num complaints	-0.5252	0.0216	-24.2799	0.0000	-0.5676	-0.4828
number plan changes	-0.3992	0.0056	-71.4290	0.0000	-0.4101	-0.3882
monthly bill	-0.0015	0.0001	-10.7074	0.0000	-0.0018	-0.0012
technical issues per month	-0.4333	0.0066	-65.6161	0.0000	-0.4462	-0.4203

4 Activation Function

4.1 Introduction

Activation Function are maps a particular output to particular set of inputs, so they are used for the containing the outputs in between 0 to 1 or any given values. Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Famous Activation Function

They are some famous activation function used which are used in neural network.

1. **Identity**
2. **Binary step**
3. **Logistic or Sigmoid**
4. **Tanh**
5. **Arctan**
6. **Rectified Linear Unit(ReLU)**
7. **Leaky ReLU**
8. **Softmax**

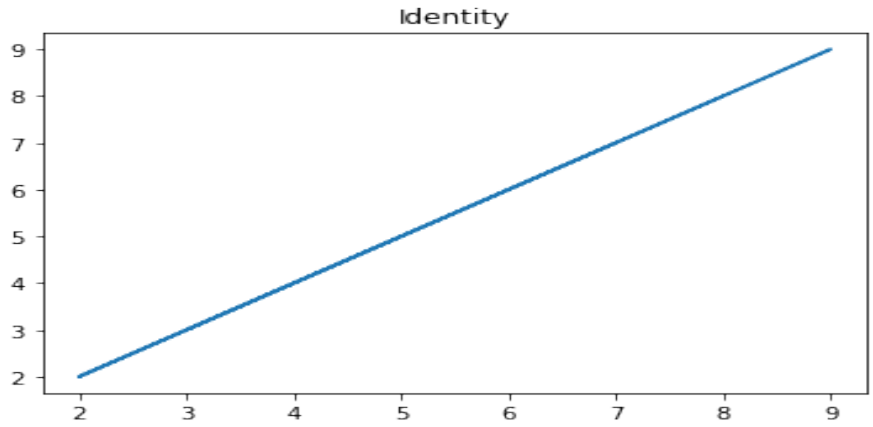
4.1.0.1 Identity

Identity function is,

$$f(x) = x,$$

Identity function is simple as here x is input and it gives x.

The graph of **Indetity :**



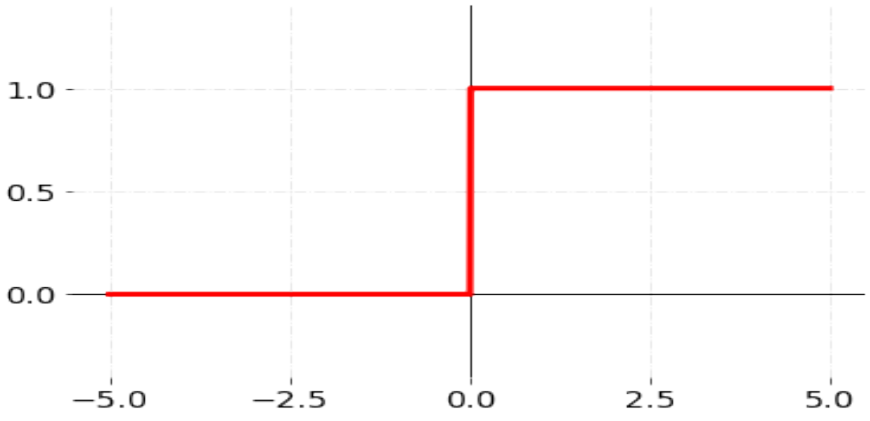
So this is the graoh of it and whatever curve is will will get the exact same curve.

4.1.0.2 Binary Step

Here,

$$f(x) = \begin{cases} 0 & \text{for} \quad x<0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

In Binary step function if input is greater that or equal to zero then it gives 1 and if input is less than zero is it gives 0. So it take only positive input and makes it 1 and all the negative inputs makes as 0. This is very useful in classify. The graph of **Binary Step :**

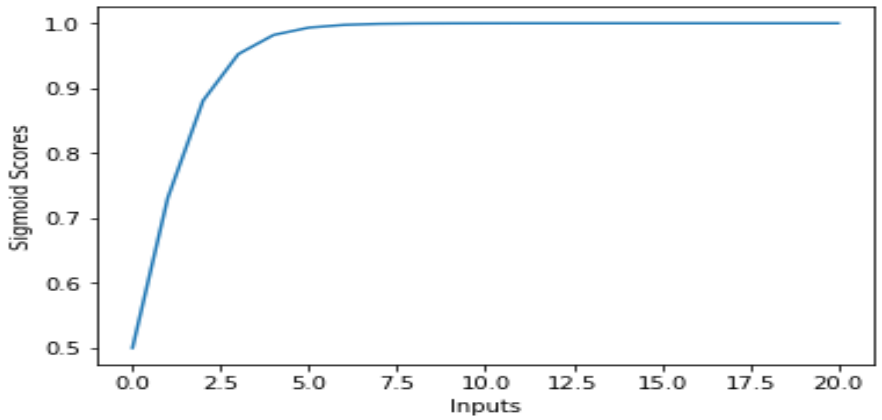


4.1.0.3 Logistic or Sigmoid

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

it look as "S" shape, Logistic or Sigmoid takes all inputs and it maps between 0 to 1. So even our input is large as thousand or lakhs it is map it between 0 to 1. The sigmoid function used for **binary classification** in logistic regression model even creating artificial neurons sigmoid function used as the **activation function**.

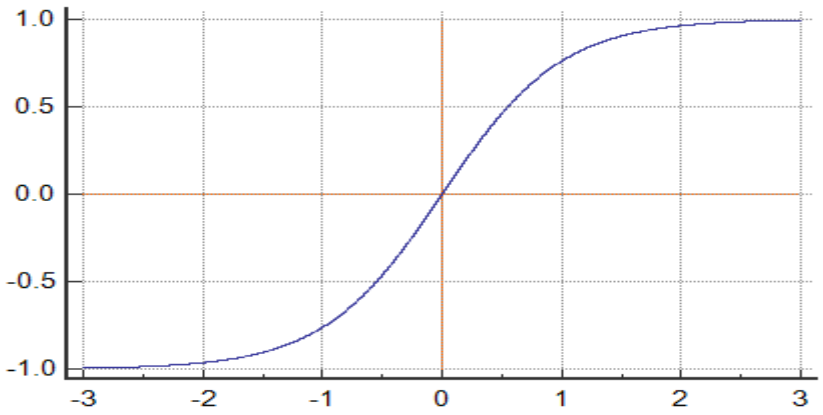
```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(inputs):
    sigmoid_scores = [1 / float(1 + np.exp(- x)) for x in inputs]
    return sigmoid_scores
def line_graph(x, y, x_title, y_title):
    plt.plot(x, y)
    plt.xlabel(x_title)
    plt.ylabel(y_title)
    plt.show()
graph_x = range(0, 21)
graph_y = sigmoid(graph_x)
print("Graph X readings: {}".format(graph_x)))
print("Graph Y readings: {}".format(graph_y))
line_graph(graph_x, graph_y, "Inputs", "Sigmoid Scores"))
```



4.1.0.4 TanH

$$\begin{aligned} f(x) &= \frac{2}{1+e^{-2x}} - 1 \\ &= 2 * \text{Sigmoid}(2x) - 1 \end{aligned}$$

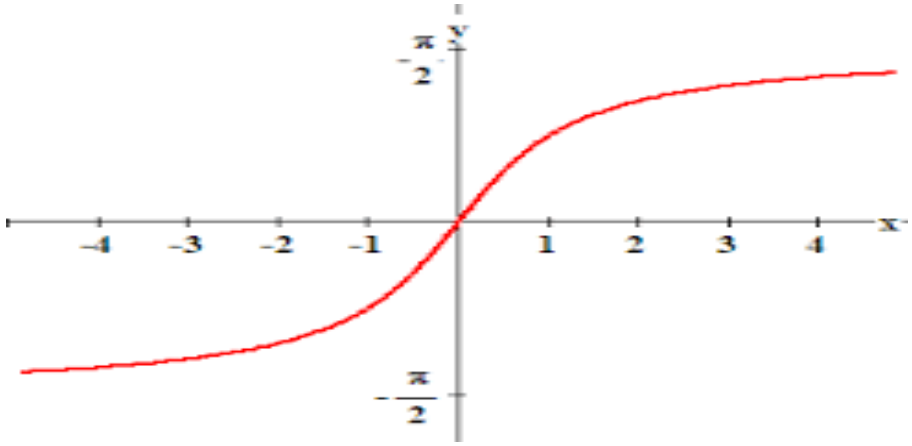
The activation that works almost always better than sigmoid function is Tanh function also knows as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other. TanH used in hidden layers of a neural network as it's values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.
The graph of **TanH :**



4.1.0.5 Arctan

$$f(x) = \tan^{-1}(x)$$

The most common activation function are the sigmoid and tanh functions. The sigmoid and tanh functions accept as input any value from negative infinity to positive infinity. The graphs of both functions resemble an S shape. The sigmoid returns a value between 0 and 1. The tanh function returns a value between -1 and +1. But arctan returns a value between -$\frac{\pi}{2}$ and +$\frac{\pi}{2}$ and is flatter than sigmoid or tanh. It worked well in some situations, not so well in others.
The graph of **Arctan :**

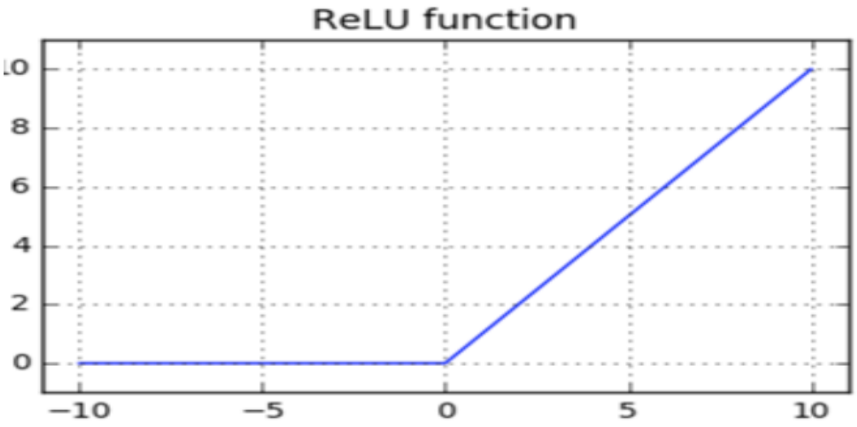


4.1.0.6 Rectified Linear Unit (ReLU)

$$f(x) = \begin{cases} 0 & \text{for} \quad x<0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

It is a very popular in deep learning and even in normal neural networks. So what it does , it gives 0 if our function is less than 0 and whenever function is greater than 0 it remains as it is. It remove the negative part of our function. ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

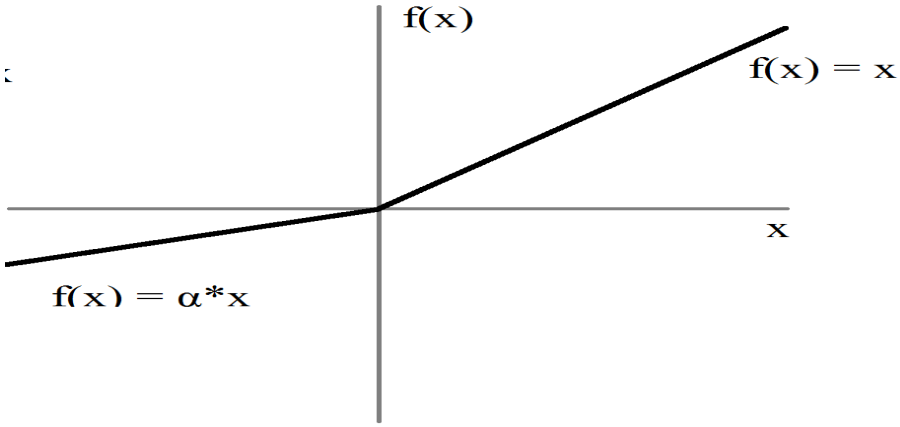
The graph of **Rectified Linear Unit :**



4.1.0.7 Leaky ReLU

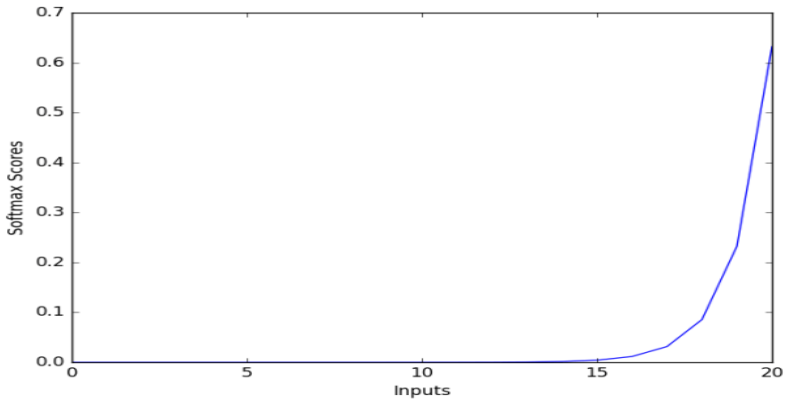
$$f(x) = \begin{cases} 0.01x & \text{for} \quad x<0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

Leaky ReLU does similar job as ReLU only difference is it does not make 0 for negative values just reduce the magintute of it. Therefore the range of the Leaky ReLU is $(-\infty, \infty)$. It fixes the "dying ReLU" problem, as it doesn't have zero-slope parts. The graph of **Leaky ReLU :**



4.1.0.8 Softmax

The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems. Usually used when trying to handle multiple classes.
The graph of **Softmax :**



4.2 Conclusion

1. The basic rule of thumb is if you really don't know what activation function to use, then simply use ReLU as it is a general activation function and is used in most cases these days.
2. If your output is for binary classification then, sigmoid function is very natural choice for output layer.

3. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

5 Artificial Neural Network (ANN)

When there is a categorical output(Yes/No), binary output (1/0) and predictor variable is continuous then we were used logistic regression. Directly moving to neural network let us do a quick recap of how did we build the logistic regression and building a neural network based on combination of several logistic regression.

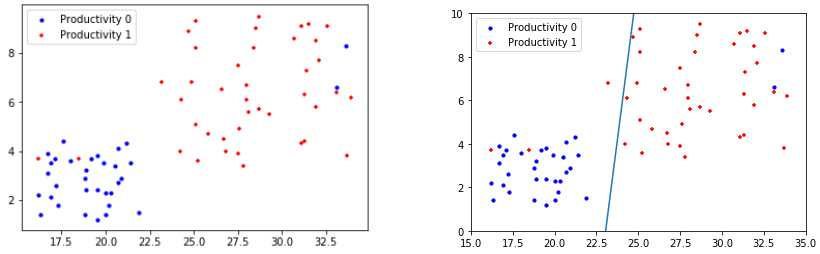
We have taken Dataset:Emp_Productivity/Emp_Productivity.csv provided by statinfer(Venkat Sir). First we filter the data and taken a subset of above dataset. Filter condition is Sample Set<3 and also draw a scatter plot that shows Age on X-axis and Experience on Y-axis and trying to distinguish the two classes with colors or shapes(visualizing the classes).

Our aim is to building a logistic regression model to predict productivity using these two classes. Finally draw the decision boundary for the model and also calculated the confusion matrix.

```
import pandas as pd
Emp_Productivity_raw=pd.read_csv("datasetsEmp_ProductivityEmp_Productivity.csv")
Emp_Productivity_raw.head(10)
#Filter the data and take a subset from above dataset
# Filter condition is Sample_Set<3
Emp_Productivity1=Emp_Productivity_raw[Emp_Productivity_raw.Sample_Set<3]
Emp_Productivity1.shape
#frequency table of Productivity variable
Emp_Productivity1.Productivity.value_counts()


##The clasification graph
#Draw a scatter plot that shows Age on X axis and Experience on Y-axis.
# Try to distinguish the two classes #with colors or shapes.
import matplotlib.pyplot as plt
%matplotlib inline


fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.scatter(Emp_Productivity1.Age[Emp_Productivity1.Productivity==0],
```

```
Emp_Productivity1.Experience[Emp_Productivity1.Productivity==0], s=10, c='b',
marker="o", label='Productivity 0')
ax1.scatter(Emp_Productivity1.Age[Emp_Productivity1.Productivity==1],
Emp_Productivity1.Experience[Emp_Productivity1.Productivity==1], s=10, c='r',
marker="+", label='Productivity 1')
plt.legend(loc='upper left');
plt.show()

#predict Productivity using age and experience Logistic Regerssion model1
import statsmodels.formula.api as sm
model1 = sm.logit(formula='Productivity ~ Age+Experience', data=Emp_Productivity1)
fitted1 = model1.fit()
fitted1.summary()
#coefficients
coef=fitted1.normalized_cov_params
print(coef)
# getting slope and intercept of the line
slope1=coef.Intercept[1]/(-coef.Intercept[2])
intercept1=coef.Intercept[0]/(-coef.Intercept[2])
print(slope1)
print(intercept1)

#Finally draw the decision boundary for this logistic regression model
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(Emp_Productivity1.Age[Emp_Productivity1.Productivity==0],
Emp_Productivity1.Experience[Emp_Productivity1.Productivity==0], s=10, c='b',
marker="o", label='Productivity 0')
ax1.scatter(Emp_Productivity1.Age[Emp_Productivity1.Productivity==1],
Emp_Productivity1.Experience[Emp_Productivity1.Productivity==1], s=10, c='r',
```

```
marker="+", label='Productivity 1')
plt.legend(loc='upper left');

x_min, x_max = ax1.get_xlim()
ax1.plot([0, x_max], [intercept1, x_max*slope1+intercept1])
ax1.set_xlim([15,35])
ax1.set_ylim([0,10])
plt.show()

#Confusion Matrix, Accuracy and Error
from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(Emp_Productivity1[['Productivity']],predicted_class)
print('Confusion Matrix :', ConfusionMatrix)
accuracy=(ConfusionMatrix[0,0]+ConfusionMatrix[1,1])/sum(sum(ConfusionMatrix))
print('Accuracy : ',accuracy)
error=1-accuracy
print('Error: ',error)
```



(a) (b)

Figure 1: figure (a) depicted a scatter plot of Age on X axis and Experience on Y-axis and figure (b) depicted decision boundary on logistic regression model

Confusion Matrix : $\begin{bmatrix} 31 & 2 \\ 2 & 39 \end{bmatrix}$. Accuracy : 0.9459459459459459 Error: 0.05405405405405406

Decision Boundary :

I The line or margin that separates the classes.

II Classification algorithm are all about finding the decision boundaries.

III It need not be straight line always.

New Representation for Logistic Regression : Our aim is trying to understand neural network through logistic regression. Logistic regression line that we have built for n = 2,

$$\begin{aligned} y &= \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2}}{1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2}} \\ &= \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2)}} \end{aligned}$$

Thus, this particular line that can be taken as one equation and one can write it as,

$$y = g(w_0 + w_1 x_1 + w_2 x_2) \quad \text{Where,} \quad g(x) = \frac{1}{1+e^{-x}}$$

$$y = g\left(w_0 + \sum w_k x_k\right).$$

X_1 and X_2 whose weights are w_1 and w_2 or we can simply say $w_0 + w_1 x_1 + w_2 x_2$ is the line equation , that is going through this logistic equation,

$$y = g\left(\sum w_k x_k\right).$$

Finding the weights in logistic regression that is we have to minimize,

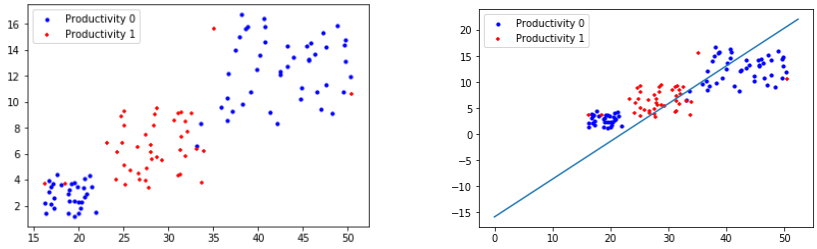
$$\sum_{i=1}^{n} \left[y_i - g\left(\sum w_k x_k\right)\right]^2$$

Now We have taken Dataset:Emp_Productivity/Emp_Productivity.csv provided by Mr. Venkata Reddy. Here we taken full dataset without any filtration also draw a scatter plot that shows Age on X-axis and Experience on Y-axis and trying to distinguish the two classes with colors or shapes (visualizing the classes).

Our aim is to building a logistic regression model to predict productivity using these two classes. Finally draw the decision boundary for the model and also calculated the confusion matrix.

```
import pandas as pd
Emp_Productivity_raw = pd.read_csv("/home/msc1/Desktop/Rahul/Emp_Purchase.csv")
#plotting the overall data
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(Emp_Productivity_raw.Age[Emp_Productivity_raw.Purchase==0],
Emp_Productivity_raw.Experience[Emp_Productivity_raw.Purchase==0], s=10, c='b',
marker="o", label='Productivity 0')
ax.scatter(Emp_Productivity_raw.Age[Emp_Productivity_raw.Purchase==1],
Emp_Productivity_raw.Experience[Emp_Productivity_raw.Purchase==1], s=10, c='r',
```

```
marker="+", label='Productivity 1')
plt.legend(loc='upper left');
plt.show()
#Logistic Regerssion model1
import statsmodels.formula.api as sm
model = sm.logit(formula='Purchase ~ Age+Experience', data=Emp_Productivity_raw)
fitted = model.fit()
fitted.summary()
#Logistic Regerssion model1
import statsmodels.formula.api as sm
model = sm.logit(formula='Productivity ~ Age+Experience', data=Emp_Productivity_raw)
fitted = model.fit()
fitted.summary()
#coefficients
coef=fitted.normalized_cov_params #coef
# getting slope and intercept of the line
slope=coef.Intercept[1]/(-coef.Intercept[2])
intercept=coef.Intercept[0]/(-coef.Intercept[2])
print('Slope :', slope)
print('Intercept :', intercept)
#Finally draw the decision boundary for this logistic regression model
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(Emp_Productivity_raw.Age[Emp_Productivity_raw.Purchase==0],
Emp_Productivity_raw.Experience[Emp_Productivity_raw.Purchase==0], s=10, c='b',
marker="o", label='Productivity 0')
ax.scatter(Emp_Productivity_raw.Age[Emp_Productivity_raw.Purchase==1],
Emp_Productivity_raw.Experience[Emp_Productivity_raw.Purchase==1], s=10, c='r',
marker="+", label='Productivity 1')
plt.legend(loc='upper left');
x_min, x_max = ax.get_xlim()
ax.plot([0, x_max], [intercept, x_max*slope+intercept])
plt.show()
```



(a) (b)

Figure 2: figure (a) depicted a scatter plot of Age on X axis and Experience on Y-axis and figure (b) depicted decision boundary on logistic regression model

Result : As We can see above that the linear boundary layer is so bad in distinguishing the classes. Now we can see what is accuarcy and error of this model :

```
#Creating the confusion matrix and predicting values
predicted_values=fitted.predict(Emp_Productivity_raw[["Age"]+["Experience"]])
predicted_values[1:10]
#Lets convert them to classes using a threshold
threshold=0.5
threshold
import numpy as np
predicted_class=np.zeros(predicted_values.shape)
predicted_class[predicted_values>threshold]=1
#Predcited Classes
predicted_class[1:10]
from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(Emp_Productivity_raw[['Purchase']],predicted_class)
ConfusionMatrix
#Accuracy and Error
accuracy=(ConfusionMatrix[0,0]+ConfusionMatrix[1,1])/sum(sum(ConfusionMatrix))
print('Accuracy : ', accuracy)
error=1-accuracy
print('Error: ',error).
```

Result : We got the accuracy is 0.57, which is not at all considerable. That means this is not single decision boundary we have to think of Non-Linear Decision Boundaries.

Non-Linear Decision Boundaries :

I When there is no linear separation between two classes or when single straight line can not help us to divide the two classes then one might have to go for a non-linear decision boundary.

II So, here one line not be sufficient.

Non-Linear Decision Boundaries-Solution : As now we see a single logistic regression line can not work for non-decision boundary. We have the classes than can not be separated by using one linear or a classifier. So how we can think of can we fit fit two models for that ? Model-1 that will separates first portion and model-2 takes care of another portion of the dataset. Instead of finding the final output, which is non-linear we will get intermediate output say $h1$ which is coming out of model-1 and there is an intermediate output $h2$ which is coming out of model-2. Then we go for final output using this $h1$ and $h2$.

So, again we have taken Dataset:Emp_Productivity/Emp_Productivity.csv provided by statinfer(Venkat Sir). Here we have not taken full dataset we taken first 74 observation from above dataset.

Our aim is to building a logistic regression model to predict productivity using age and experience. Finally draw the decision boundary for the model and also calculated the confusion matrix. Our sampled data Emp_Productivity1 has first 74 observations. Let's build the model on this sample data (sample-1).

```
import pandas as pd
Emp_Productivity_raw=pd.read_csv("/home/msc1/Desktop/Rahul/Emp_Purchase.csv")
Emp_Productivity_raw.head(10)

#Filter the data and take a subset from whole dataset. Filter condition is

Sample_Set<3
Emp_Productivity1=Emp_Productivity_raw[Emp_Productivity_raw.Sample_Set<3]
Emp_Productivity1.shape
#Building a Logistic regression model1 to predict Productivity using age and

experience
import statsmodels.formula.api as sm
model1 = sm.logit(formula='Purchase ~ Age+Experience', data=Emp_Productivity1)
fitted1 = model1.fit()
fitted1.summary()
```

```
#coefficients
coef=fitted1.normalized_cov_params
print(coef)
#getting slope and intercept of the line
slope1=coef.Intercept[1]/(-coef.Intercept[2])
intercept1=coef.Intercept[0]/(-coef.Intercept[2])
slope1
intercept1
#Drawing the decision boundary for this logistic regression model
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(Emp_Productivity1.Age[Emp_Productivity1.Purchase==0],
Emp_Productivity1.Experience[Emp_Productivity1.Purchase==0], s=10, c='b',
marker="o", label='Productivity 0')
ax1.scatter(Emp_Productivity1.Age[Emp_Productivity1.Purchase==1],
Emp_Productivity1.Experience[Emp_Productivity1.Purchase==1], s=10, c='r',
marker="+", label='Productivity 1')
plt.xlim(min(Emp_Productivity1.Age), max(Emp_Productivity1.Age))
plt.ylim(min(Emp_Productivity1.Experience), max(Emp_Productivity1.Experience))
plt.legend(loc='upper left');

x_min, x_max = ax1.get_xlim()
ax1.plot([0, x_max], [intercept1, x_max*slope1+intercept1])
plt.show()
#Calculating and Storing prediction probabilities in inter1 variable for data
#Emp_Productivity1
Emp_Productivity_raw['inter1'] = fitted1.predict(Emp_Productivity_raw[["Age"]+
["Experience"]])
```

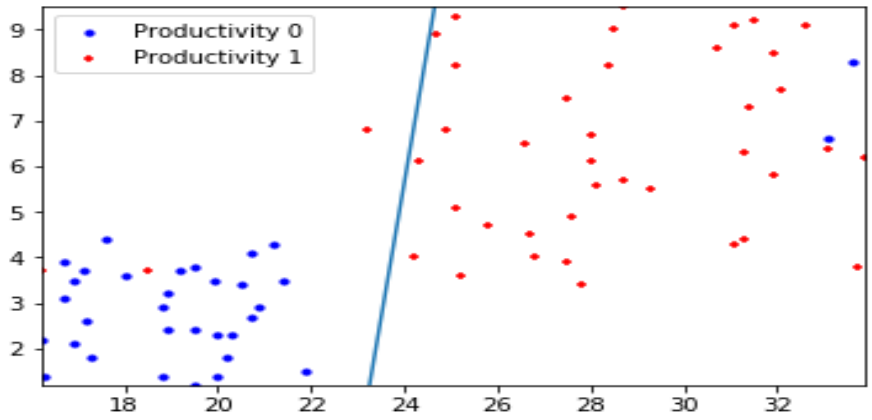


Figure 3: decision boundary for model-1 using logistic regression model

First, we take first 74 observation from datastet and we build a logistic model on this datatset and draw the decision boundary and also calculated the prediction probabnility for all inputs we store the probabilities in inter1 variable. Now we filter the data and taken observation from row 34 onwards and filter condition is sample_set<1 and agian we build logistic regression model on this dataset and draw the decision boundary and calculated the prediction probability for all inputs we store in inter2 variable.

```
Filter the data and take observations from row 34 onwards. Filter condition is
Sample_Set<1
Emp_Productivity2=Emp_Productivity_raw[Emp_Productivity_raw.Sample_Set>1]
Emp_Productivity2.shape

#Build a logistic regression model to predict Productivity using age and

experience of
data Emp_Productivity2 and Logistic Regerssion model1
import statsmodels.formula.api as sm
model2 = sm.logit(formula='Purchase ~ Age+Experience', data=Emp_Productivity2)
fitted2 = model2.fit(method="bfgs")
fitted2.summary()

#coefficients
coef=fitted2.normalized_cov_params
print(coef)
```

```
#getting slope and intercept of the line
slope2=fitted2.params[1]/(-fitted2.params[2])
intercept2=fitted2.params[0]/(-fitted2.params[2])

#Finally draw the decision boundary for this logistic regression model
import matplotlib.pyplot as plt

fig = plt.figure()
ax2 = fig.add_subplot(111)
ax2.scatter(Emp_Productivity2.Age[Emp_Productivity2.Purchase==0],
Emp_Productivity2.Experience[Emp_Productivity2.Purchase==0], s=10,
c='b', marker="o", label='Productivity 0')
ax2.scatter(Emp_Productivity2.Age[Emp_Productivity2.Purchase==1],
Emp_Productivity2.Experience[Emp_Productivity2.Purchase==1], s=10,
c='r', marker="+", label='Productivity 1')
plt.xlim(min(Emp_Productivity2.Age), max(Emp_Productivity2.Age))
plt.ylim(min(Emp_Productivity2.Experience), max(Emp_Productivity2.Experience))
plt.legend(loc='upper left');

x_min, x_max = ax2.get_xlim()
y_min,y_max=ax2.get_ylim()
ax2.plot([x_min, x_max], [x_min*slope2+intercept2, x_max*slope2+intercept2])
plt.show()

#Calculate the prediction probabilities for all the inputs. Store the probabilities
in
# inter2 variable for data Emp_Productivity2
Emp_Productivity_raw['inter2']=fitted2.predict(Emp_Productivity_raw[["Age"]+
["Experience"]])
```

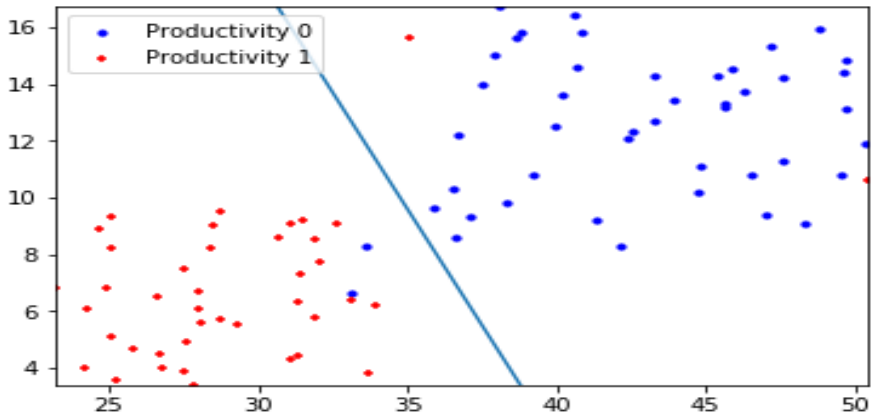


Figure 4: decision boundary for model-2 using logistic regression model

Now, we have been created model-1 and model-2. Now our next task is to combine this two models and draw the decision boundary for the final output and calculate the accuracy and error.

```
#plotting the new columns
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(Emp_Productivity_raw.inter1[Emp_Productivity_raw.Purchase==0],
Emp_Productivity_raw.inter2[Emp_Productivity_raw.Purchase==0], s=50, c='b',
marker="o", label='Productivity 0')
ax.scatter(Emp_Productivity_raw.inter1[Emp_Productivity_raw.Purchase==1],
Emp_Productivity_raw.inter2[Emp_Productivity_raw.Purchase==1], s=50, c='r',
marker="+", label='Productivity 1')
plt.xlim(min(Emp_Productivity_raw.inter1), max(Emp_Productivity_raw.inter1)+0.2)
plt.ylim(min(Emp_Productivity_raw.inter2), max(Emp_Productivity_raw.inter2)+0.2)

plt.legend(loc='lower left');
plt.show()
#Logistic Regerssion model with Intermediate outputs as input
import statsmodels.formula.api as sm
model_combined = sm.logit(formula='Purchase ~ inter1+inter2',
data=Emp_Productivity_raw)
fitted_combined = model_combined.fit(method="bfgs")
fitted_combined.summary()
```

```
#coefficients
coef=fitted_combined.normalized_cov_params
print(coef)
#getting slope and intercept of the line
slope_combined=fitted_combined.params[1]/(-fitted_combined.params[2])
intercept_combined=fitted_combined.params[0]/(-fitted_combined.params[2])
#Finally draw the decision boundary for this logistic regression model
import matplotlib.pyplot as plt
fig = plt.figure()
ax2 = fig.add_subplot(111)
ax2.scatter(Emp_Productivity_raw.inter1[Emp_Productivity_raw.Purchase==0],
Emp_Productivity_raw.inter2[Emp_Productivity_raw.Purchase==0], s=10, c='b',
marker="o", label='Productivity 0')
ax2.scatter(Emp_Productivity_raw.inter1[Emp_Productivity_raw.Purchase==1],
Emp_Productivity_raw.inter2[Emp_Productivity_raw.Purchase==1], s=10, c='r',
marker="+", label='Productivity 1')
plt.xlim(min(Emp_Productivity_raw.inter1), max(Emp_Productivity_raw.inter1)+0.2)
plt.ylim(min(Emp_Productivity_raw.inter2), max(Emp_Productivity_raw.inter2)+0.2)
plt.legend(loc='lower left');
x_min, x_max = ax2.get_xlim()
y_min,y_max=ax2.get_ylim()
ax2.plot([x_min, x_max], [x_min*slope_combined+intercept_combined,
x_max*slope_combined+intercept_combined])
plt.show()
# Confusion Matrix, Accuracy and Error of the Intermediate
#Predciting Values
predicted_values=fitted_combined.predict(Emp_Productivity_raw[["inter1"]
["inter2"]])
predicted_values[1:10]
#Lets convert them to classes using a threshold
threshold=0.5
threshold
import numpy as np
```

```
predicted_class=np.zeros(predicted_values.shape)
predicted_class[predicted_values>threshold]=1
#Predcited Classes
predicted_class[1:10]
from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(Emp_Productivity_raw[['Purchase']],predicted_class)
print('Confusion Matrix : ',ConfusionMatrix)
accuracy=(ConfusionMatrix[0,0]+ConfusionMatrix[1,1])/sum(sum(ConfusionMatrix))
print('Accuracy : ', accuracy)
error=1-accuracy
print('Error : ', error)
```

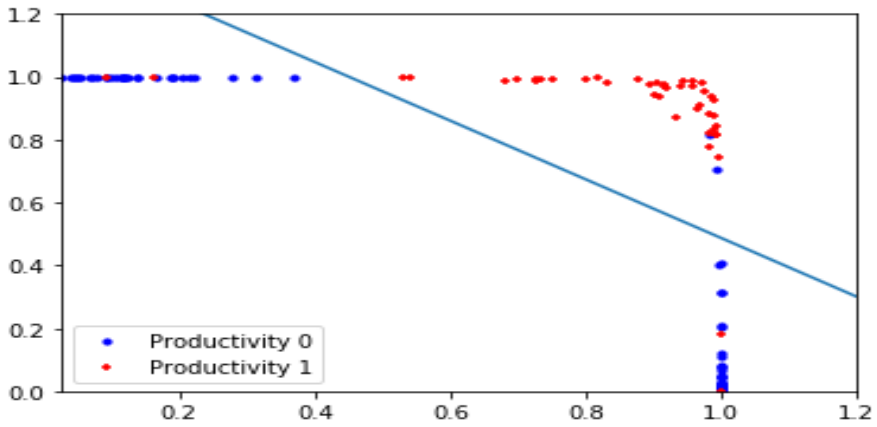


Figure 5: decision boundary of combination of model-1 and model-2 using logistic regression model

Result : We got an accuracy of 94.95% with an Intermediate model i.e model-1 and model-2.

So, what we did is nothing but intuition of neural network, let's see

5.1 Neural Network Intuition

$$y = g\left(\sum w_j h_j\right)$$

$$h_j = g\left(\sum w_{jk} x_k\right)$$

Here, h_j is a non-linear function of linear combination of inputs and y is non-linear function of linear combination of outputs of logistic regressions(h_j). y is a non linear function of linear combination of non linear functions of linear combination of inputs. For finding w we are minimize,

$$\sum_{i=1}^{n} \left[y_i - g\left(\sum w_j h_J\right)\right]^2$$

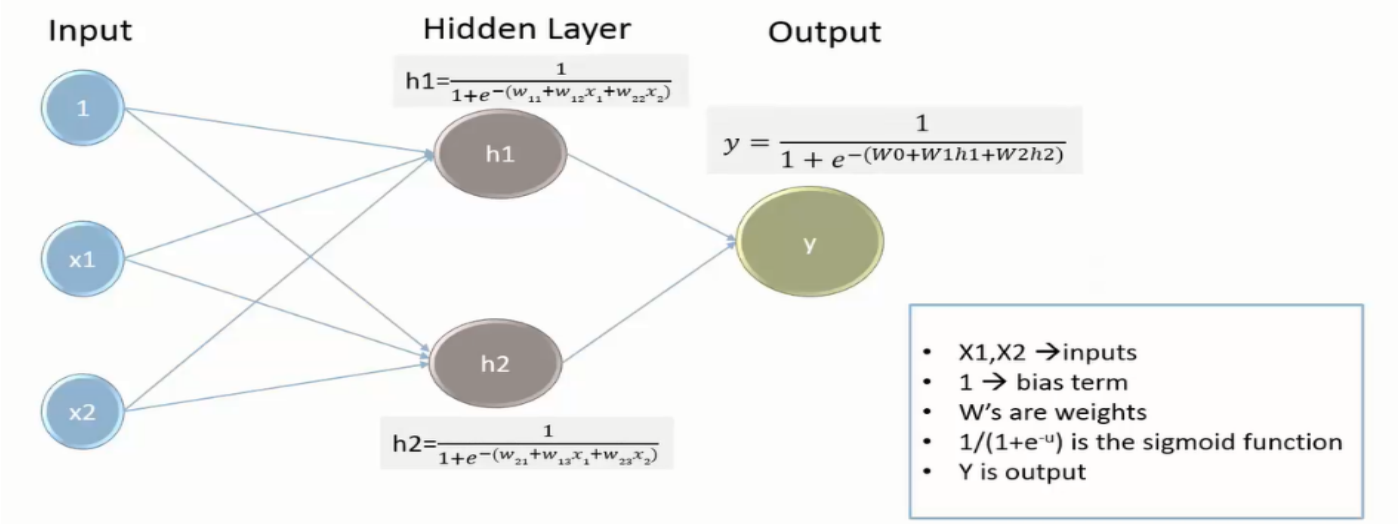
$$\sum_{i=1}^{n} \left[y_i - g\left(\sum w_j g\left(w_{jk} x_k \right) \right) \right]^2 .$$

Neural networks is all about finding the sets of weights w_j and w_{jk} using Gradient Descent Method.

5.2 Neural Network and Vocabulary

The neural networks methodology is similar to the intermediate output method explained above. But we will not manually subset the data to crate the different models. The neural network technique automatically takes care of all the intermediate outputs using hidden layers. It works very well for the data with non-linear decision boundaries. The intermediate output layer in the network is known as hidden layer. In Simple terms, neural networks are multi-layer nonlinear regression model. If we have sufficient number of hidden layers, then we can estimate any complex non-linear function.

Suppose, we have only two inputs then this will be structure of neural network,



Hidden Layer :

I A hidden layer "hides" the desired output.

II Instead of predicting the actual output using a single model, we build multiple models to predict intermediate output.

How many hidden layer?? There is no standard way of deciding the number of hidden layers but with experience and looking at the complexity or looking at the final accuracy of the model, we can experiment with the number of hidden layers.

So, this is the overall intuition of the neural network.

5.3 Neural Network Algorithm

I **Step 1**: **Initialization of weights**: Randomly select some weights.

II **Step 2**: **Training Activation**: Input the training values and perform the calculations forward.

i. We have the dataset with us, so we put the values of x, then we find the values of y with which we can calculate forward.

ii. Once we calculate the value of y, then those are predicted values of y.

iii. In training itself i.e., in dataset, we will have the actual values of y.

III **Step 3**: **Error Calculation**: Calculate the error at the outputs. Use the output error to calculate error fractions at each hidden layer.

IV **Step 4**: **Weight training**: Update the weights to reduce the error, recalculate and repeat the process of training updating the weights for all the examples.

V **Step 5**: **Stopping criteria**: Stop the training and weights updating process when the minimum error criteria is met.

i. So we start with some weights and then calculate the errors and then adjust the weights to reduce the error, once there is very minimum error we stop it at that point.

5.4 Building the Neural Network in Python

I We have a couple of packages available in Python.

II We need to mention the dataset, input, output number of hidden layers as input.

III Neural network calculations are very complex.

Python Code Options

I **Step 1**: **Import the Neurolab**.

II **Step 2**: We need a epouch(error pouch) error = [],In this we store error value of each iteration or epoach.

III **Step 3**: **Create a network**:

i. Here nl.net.newff([[0, 1],[0,1]],[4,1],transf=[nl.trans.LogSig()] * 2) is feed forward neural network.

ii. 1st argument is min max values of predictor variables; can be a list of list.

iii. 2nd argument is no. of nodes in each layer i.e., 4 in hidden and 1 in output layer.

iv. transf is transfer function applied in each layer

IV **Step 4**: **Train Network**:

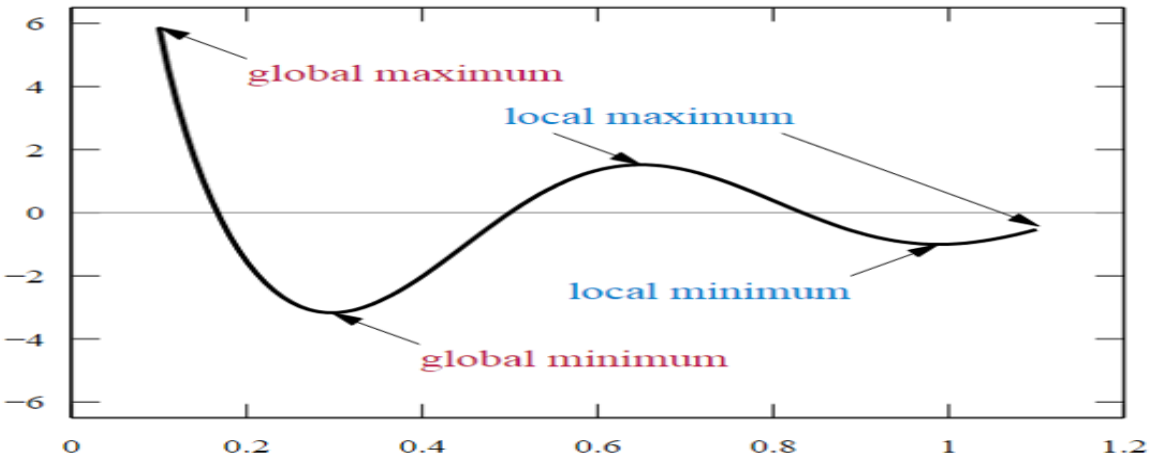
i. net.train() outputs error which is appended to error variable and has a few parameters.

V **Step 5**: **Simulate Network**:

i. net.sim(input) gives the output for the network.

5.5 Math Behind Neural Network Algorithm

In neural network we were finding all wights using gradient descent method. Thus, gradient decent method is which we use in finding the weights in neural network is not finding the final global minima but it is finding the nearest local minima and most of the times local minima. So what global minimum in this particular graph is.



Algorithms will try to find the local minima rather than global minima, because you might see multiple solutions for a given neural network problem. That is a kind of uncomfortable situation, but we can perform some cross validation checks to find out the real final optimize solution. So there can be multiple optimal solutions of neural network.

5.6 Neural Network Application

I Self driving car by taking the video as input.
II Speech recognition
III Face recognition
IV Cancer cell analysis
V Heart attack predictions
VI Currency predictions and stock price predictions
VII Credit card default and loan predictions
VIII Marketing and advertising by predicting the response probability
IX Weather forecasting and rainfall prediction

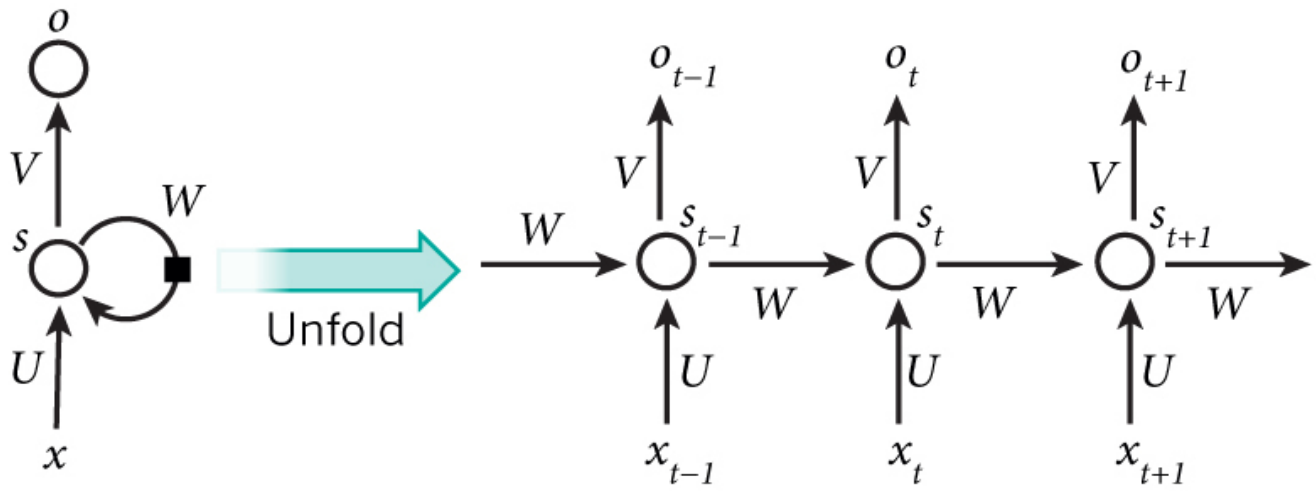
6 Recurrent Neural Network (RNN)

6.1 Introduction

Humans don't start their thinking from scratch every second. When we read some article, we understand each word based on our understanding of previous words. We are not throwing everything away and start thinking from scratch again our thoughts have persistence. Traditional neural network can't do this and it seems like a major shortcoming. Recurrent neural network address this issue. They are networks with loop in them, allowing information to persist.

The idea behind **RNNs** is to make use of sequential information. In a traditional neural netwrok we assume that all inputs and outputs are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. **RNNs** are called recurrent because they perform the same task for every element of sequence, with the output being depended on the previous computations.

Another way to think about **RNNs** is that they have a **"memory"** which captures information about what has been calculated so far. Here is what a typical **RNNs** looks like :



The above diagram shows a **RNN** being unfolded into a full network. By unfolding we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unfolded into a 5-layer neural network, one layer for each word. The formulas that govern the computation happening in a **RNN** are as follows:

I x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.

II s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step :

$$s_t = f\left(Ux_t + Ws_{t-1}\right)$$

III The function f usually is non-linearity such as **TanH** or **ReLU**

IV s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.

V o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary,

$$o_t = \text{Softmax}(Vs_t)$$

Some Important thing to note here : We can think of the hidden state s_t as the memory of the network. s_t captures information about what happened in all the previous time steps. The output at step o_t is calculated solely based on the memory at time t. As briefly mentioned above, it's a bit more complicated in practice because s_t typically can't capture information from too many time steps ago.

Unlike a traditional deep neural network, which uses different parameters at each layer, a **RNN** shares the same parameters $(U, V, W\text{above})$ across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn. The main feature of an RNN is its hidden state, which captures some information about a sequence.

What can RNNs do?

I **RNNs** have shown great succes in many **NLP** tasks.

II The most commonly used type of **RNNs** are **LSTM**

III **LSTM** are essentially the same thing as the **RNN**, they just have a different way of computing the hidden satte.

Here are some example applications of **RNNs**.

I **Language Modeling and Generating Text**

II **Speech Recognition**

III **Generating Image Descriptions**

IV **Machine Translation**

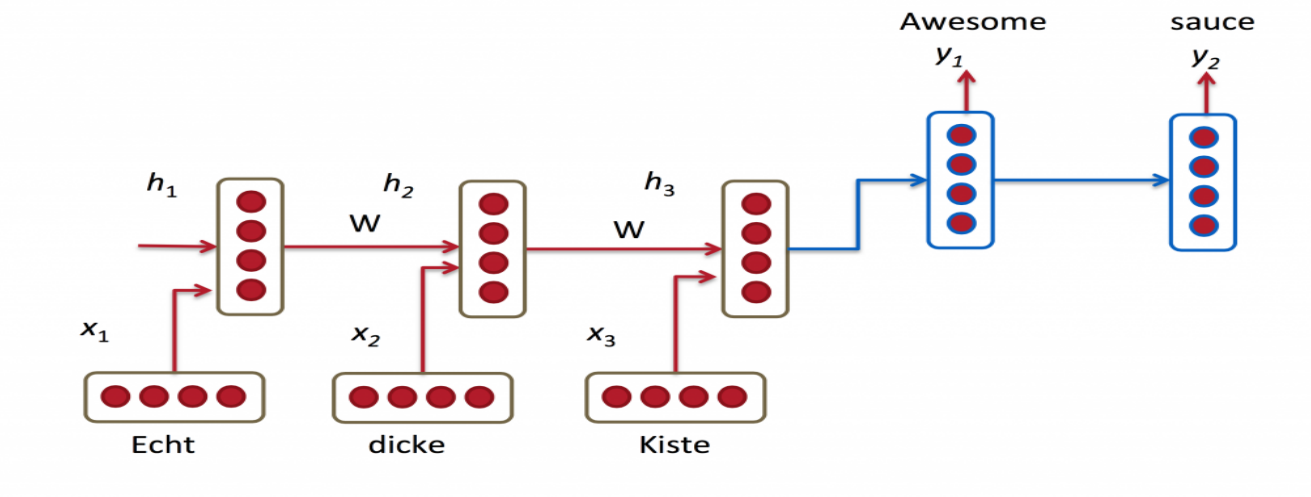
6.2 Machine Translation

I Machine Translation is similar to language modeling.

II Input is a sequence of words in our source language.

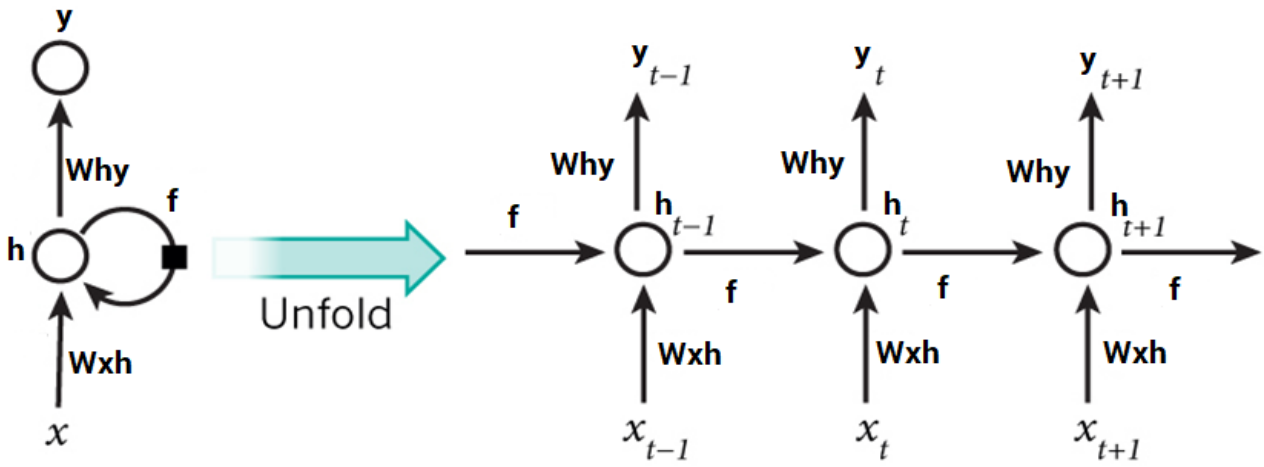
III Output a sequence of words in our target language

IV A key difference is that our output only starts after we have seen the complete input sequence.



6.3 Back propagation in a Recurrent Neural Network(BPTT)

To imagine how weights would be updated in case of a recurrent neural network, might be a bit of a challenge. So to understand and visualize the back propagation, let's unroll the network at all the time steps. In an **RNNs** we may or may not have outputs at each time step. In case of a forward propagation, the inputs enter and move forward at each time step. In case of a backward propagation in this case, we are figuratively going back in time to change the weights, hence we call it the Back propagation through time(BPTT).



y_t is the predicted value, $\hat{y}_t$ is the actual value, the error is calculated as a cross entropy loss –

$$E_t(\hat{y}_t, y_t) = \breve{}\hat{y}_t log(y_t)$$

$$E(\hat{y}, y) = \breve{} \sum \hat{y}_t log(y_t)$$

We typically treat the full sequence (word) as one training example, so the total error is just the sum of the errors at each time step (character). The weights as we can see are the same at each time step. Let's summarize the steps for back-propagation

I The cross entropy error is first computed using the current output and the actual output.

II Remember that the network is unfolded for all the time steps.

III For the unfolded network, the gradient is calculated for each time step with respect to the weight parameter.

IV Now that the weight is the same for all the time steps the gradients can be combined together for all time steps.

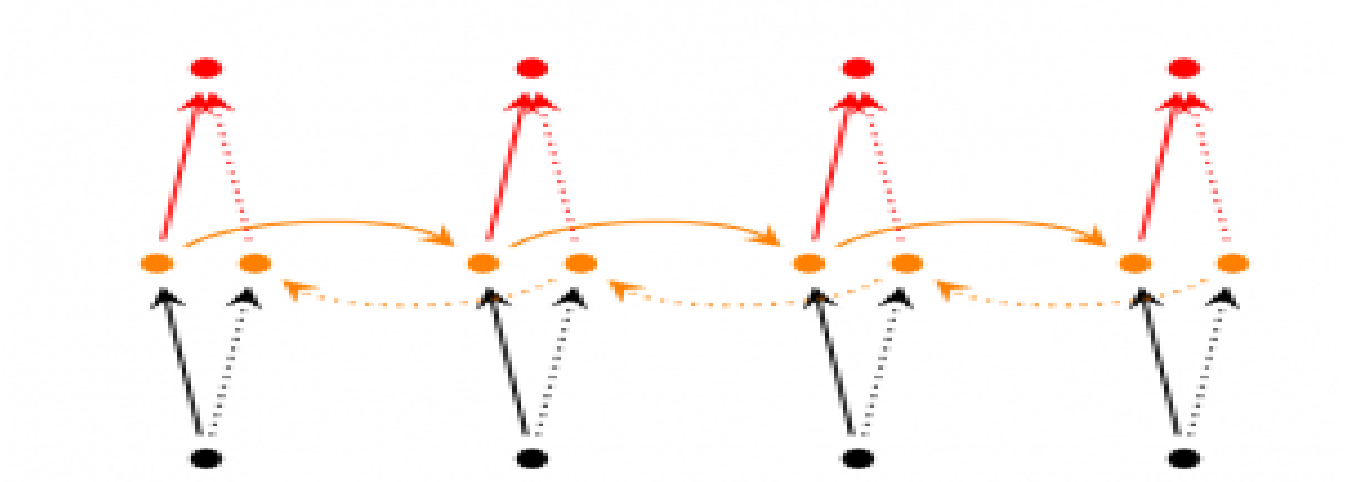
V The weights are then updated for both recurrent neuron and the dense layers.

6.4 RNN Extensions

Over the years researchers have developed more sophisticated types of **RNNs** to deal with some of the shortcomings of the vanilla RNN model. We will cover them in more detail in a later post, but I want this section to serve as a brief overview so that you are familiar with the taxonomy of models.

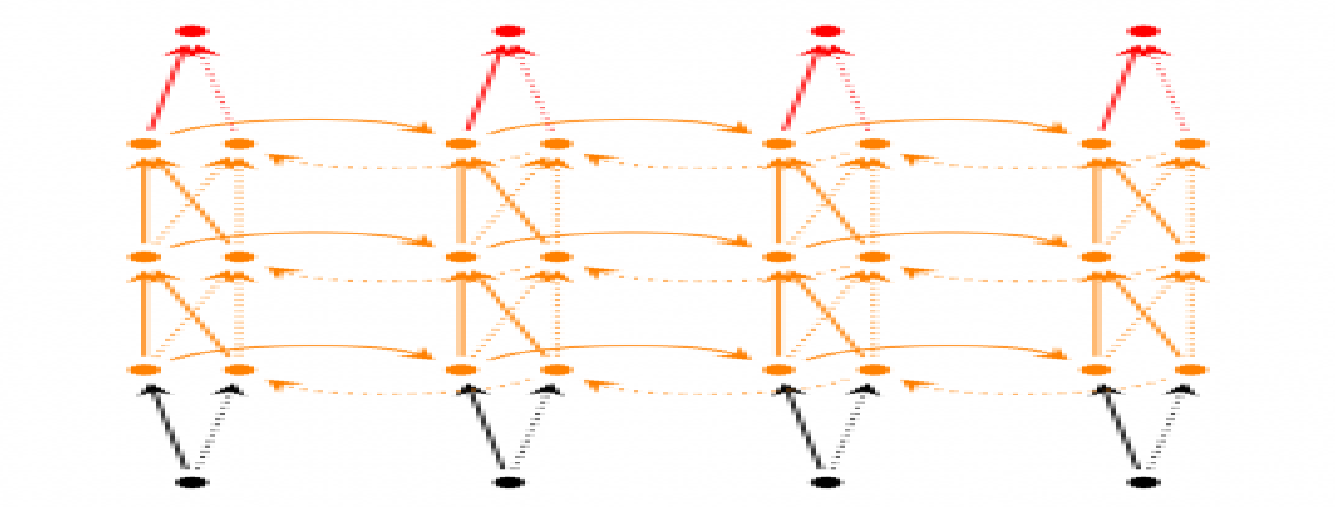
Bidirectional RNNs

Bidirectional **RNNs** are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements. For example, to predict a missing word in a sequence you want to look at both the left and the right context. Bidirectional **RNNs** are quite simple. They are just two **RNNs** stacked on top of each other. The output is then computed based on the hidden state of both **RNNs**.



Deep (Bidirectional) RNNs

Deep (Bidirectional) **RNNs** are similar to Bidirectional **RNNs,** only that we now have multiple layers per time step. In practice this gives us a higher learning capacity (but we also need a lot of training data.



6.5 Vanishing Gradient problem

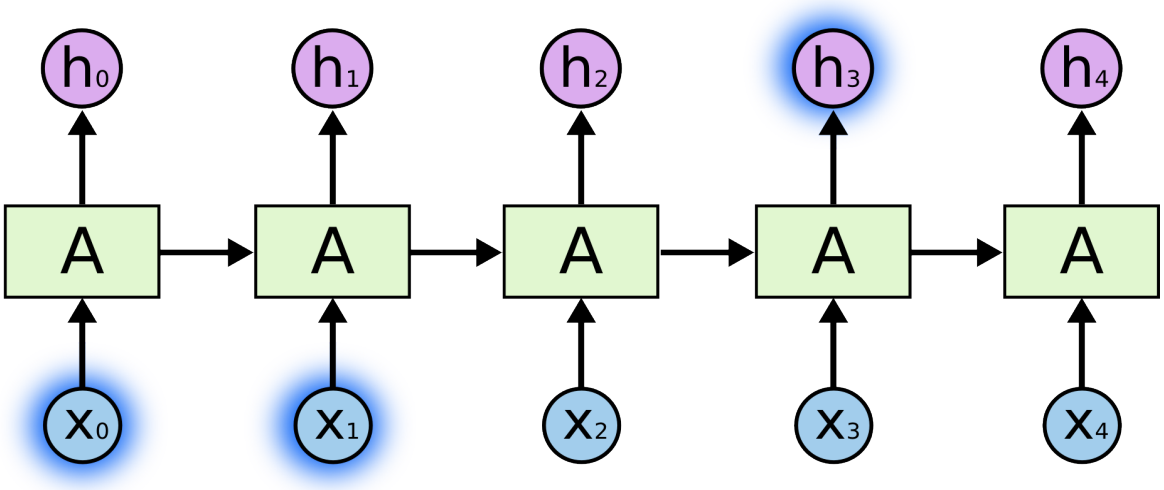
Vanishing Gradient problem mostly occurs when we are dealing with large time series data sets. Error propagate backwards from output to input layer propagating the input error gradient. With deeper neural networks issues can arise from back propagation like vanishing and exploding gradients.

Vanishing Gradients : As we go back to the lower layers gradient often get smaller. Eventually causing weights to never change at lower layers.

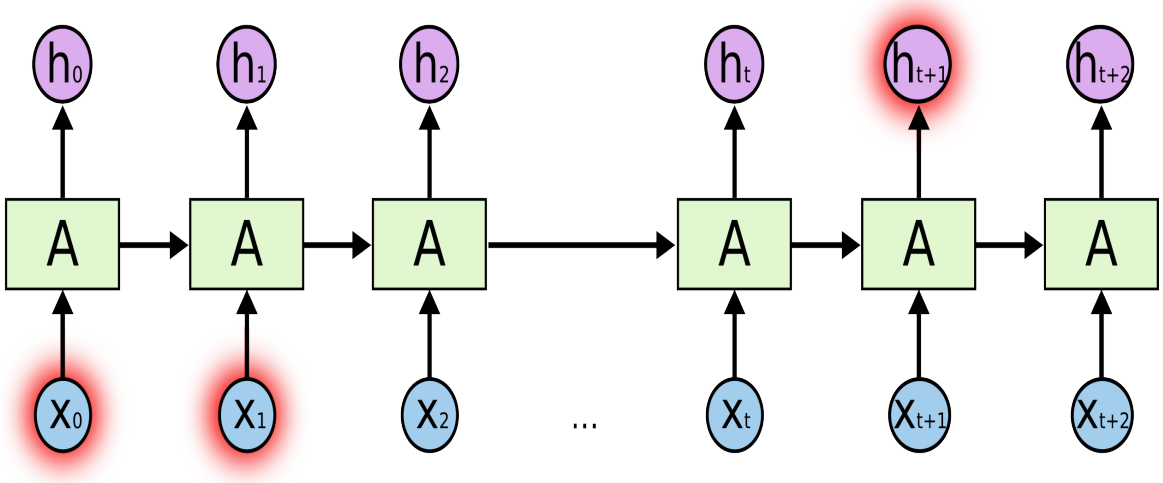
Exploding gradients : Opposite of Vanishing gradients, gradient explode on the way back.

Both Vanishing and Exploding Gradients is an issue in Neural Networks but mostly **RNNs** suffer with Vanishing Gradients due to their large and complex structures.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, **RNNs** can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in India ... I speak fluent Hindi." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of India, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.



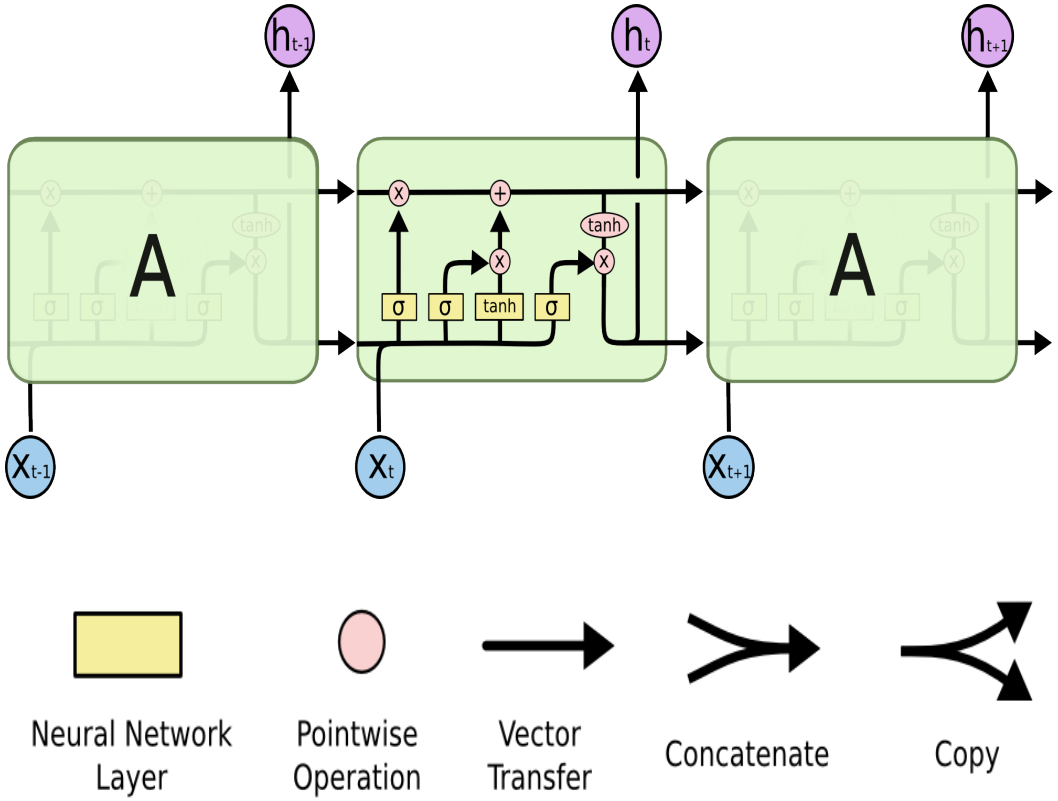
Unfortunately, as that gap grows, **RNNs** become unable to learn to connect the information.

6.6 Long-Short Term Memory(LSTM)

Long Short Term Memory networks – usually just called "**LSTMs**" – are a special kind of **RNN**, capable of learning long-term dependencies. **LSTMs** are explicitly designed to avoid the long-term dependency problem. Remembering information for long period of time is practically their default behavior, not something they struggle to learn. Almost all exciting results based on recurrent neural

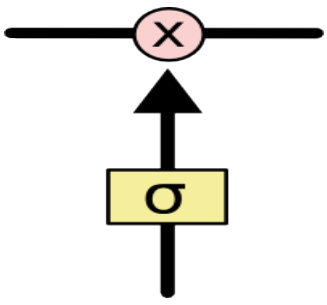
network are achieved with **LSTM**.

Structure of LSTM



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent point-wise operations, like vector addition and vector multiplication, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

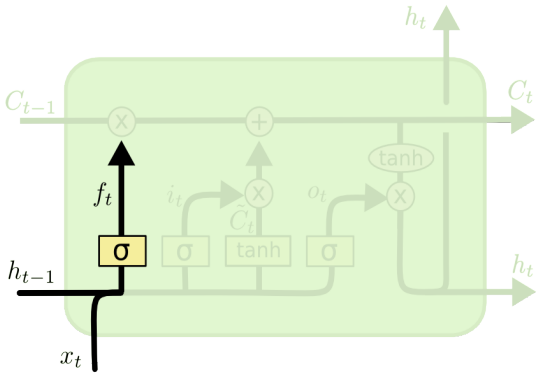
Cell State : The key to **LSTMs** is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The **LSTM** does have the ability to remove or add information to cell state, carefully regulated by structure called gates. **Gates** are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a point-wise multiplication operation.



The sigmoid layer gives the outputs between 0 and 1. That is describing how much of each component should be let through. A value of 0 means "let nothing through," while a value of 1 means "let everything through."

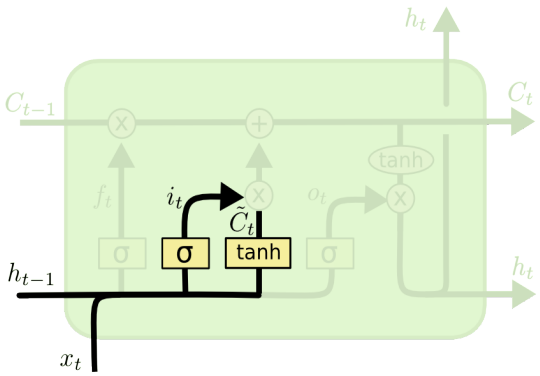
An **LSTM** has three of these gates, to protect and control the cell state.

Forget Gate Layer : The first step in **LSTM** is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and gives the outputs a number between 0 and 1 for each number in the cell state C_{t-1}. Where 1 represents "completely keep this" while a 0 represents "completely get rid of this." Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

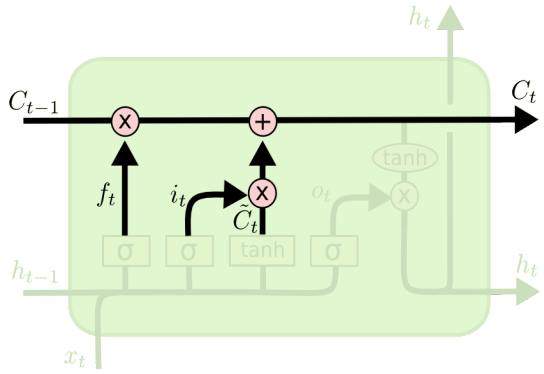
Input Gate Layer : The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a **tanh** layer creates a vector of new candidate values, , that could be added to the state. In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

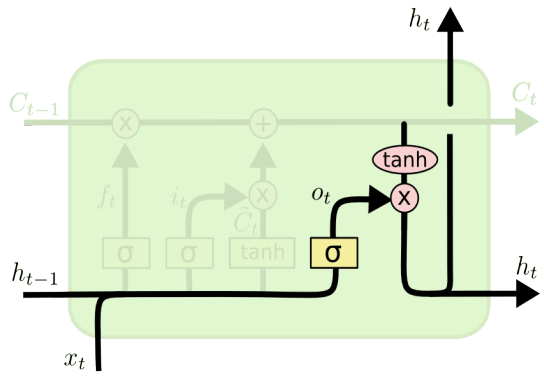
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

The Current State : It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add with $i_t\hat{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value. In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate Layer : Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through **tanh** and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to. For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

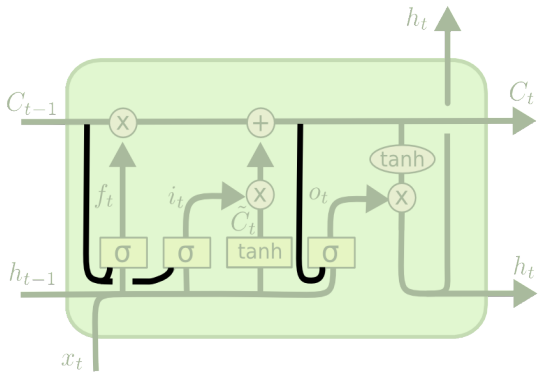


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

This is a pretty normal **LSTM**. But not all **LSTMs** are the same as the above. In fact, it seems like almost every paper involving **LSTMs** uses a slightly different version. The differences are minor, but it's worth mentioning some of them.

One popular LSTM variant, introduced by Gers Schmidhuber (2000), is adding "peephole connec-

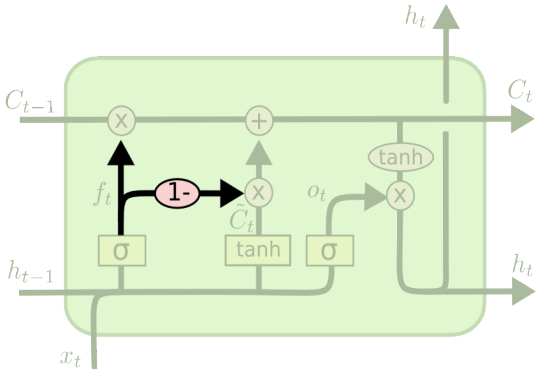
tions." This means that we let the gate layers look at the cell state.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

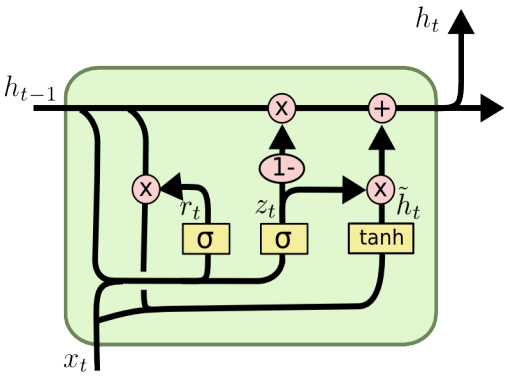
Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Gated Recurrent Unit

A slightly more dramatic variation on the **LSTM** is the Gated Recurrent Unit(GRU), introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular. A gated recurrent unit neural network.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

These are only a few of the most notable **LSTM** variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015).

There's also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014).

7 Lab : Translation

English-Hindi Translation (Python Code)

The dependent packages: 1) Tensorflow 2) keras 3) string 4) re 5) numpy 6) pandas 7) matplotlib

In [31]:

```
import tensorflow
import keras
```

In [32]:

```
import string
import re
from numpy import array, argmax, random, take
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding, RepeatVector
from keras.preprocessing.text import Tokenizer
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from keras import optimizers
import matplotlib.pyplot as plt
%matplotlib inline
pd.set_option('display.max_colwidth', 200)
```

Our data is a text file (.txt) of English-Hindi sentence pairs. First, we will read the file using the function defined below.

In [33]:

```
# function to read raw text file
def read_text(filename):
        # open the file
        file = open(filename, mode='rt', encoding='utf-8')

        # read all text
        text = file.read()
        file.close()
        return text
```

Let's define another function to split the text into English-Hindi pairs separated by '\n'. We'll then split these pairs into English sentences and Hindi sentences respectively.

In [34]:

```
# split a text into sentences
def to_lines(text):
      sents = text.strip().split('\n')
      sents = [i.split('\t') for i in sents]
      return sents
```

We can now use these functions to read the text into an array in our desired format.

In [35]:

```
data = read_text("hin-eng.txt")
hin_eng = to_lines(data)
hin_eng = array(hin_eng)
```

Let's first take a look at our data. This will help us decide which pre-processing steps to adopt.

In [36]:

```
hin_eng
```

```
hin_eng
```

Out[36]:

```
array([['\ufeffa babe in the woods ', 'जंगल में एक बच्चा '],
       ['a baby at her breast ', 'उसके स्तन में एक बच्चा '],
       ['a baby brother or sister ', 'एक बच्चा भाई या बहन '],
       ...,
       ['zoos and aquariums in silver ',
        'चिड़ियाघर और एक्वैरियम चांदी में '],
       ['zukerman joins us now to ',
        'जुकर्मन अब हमें इसमें शामिल हो जाता है '],
       ['zulu nationalist inkatha freedom party ',
        'जुलू राष्ट्रवादी इनकाथा स्वतंत्रता पार्टी']], dtype='<U174')
```

In [37]:

```
len(hin_eng)
```

Out[37]:

```
1044268
```

The actual data contains over 1044268 sentence-pairs. However, we will use full data. You can change this number as per your system's computation power.

In [38]:

```
hin_eng =hin_eng[:1044268,:]
```

We will get rid of the punctuation marks and then convert all the text to lower case.

In [39]:

```
# Remove punctuation
hin_eng[:,0] = [s.translate(str.maketrans('', '', string.punctuation)) for s in hin_eng[:,0]]
hin_eng[:,1] = [s.translate(str.maketrans('', '', string.punctuation)) for s in hin_eng[:,1]]

hin_eng
```

Out[39]:

```
array([['\ufeffa babe in the woods ', 'जंगल में एक बच्चा '],
       ['a baby at her breast ', 'उसके स्तन में एक बच्चा '],
       ['a baby brother or sister ', 'एक बच्चा भाई या बहन '],
       ...,
       ['zoos and aquariums in silver ',
        'चिड़ियाघर और एक्वैरियम चांदी में '],
       ['zukerman joins us now to ',
        'जुकर्मन अब हमें इसमें शामिल हो जाता है '],
       ['zulu nationalist inkatha freedom party ',
        'जुलू राष्ट्रवादी इनकाथा स्वतंत्रता पार्टी']], dtype='<U174')
```

In [40]:

```
# convert text to lowercase
for i in range(len(hin_eng)):
    hin_eng[i,0] = hin_eng[i,0].lower()
    hin_eng[i,1] = hin_eng[i,1].lower()

hin_eng
```

Out[40]:

```
array([['\ufeffa babe in the woods ', 'जंगल में एक बच्चा '],
       ['a baby at her breast ', 'उसके स्तन में एक बच्चा '],
       ['a baby brother or sister ', 'एक बच्चा भाई या बहन '],
       ...,
       ['zoos and aquariums in silver ',
        'चिड़ियाघर और एक्वैरियम चांदी में '],
       ['zukerman joins us now to ',
```

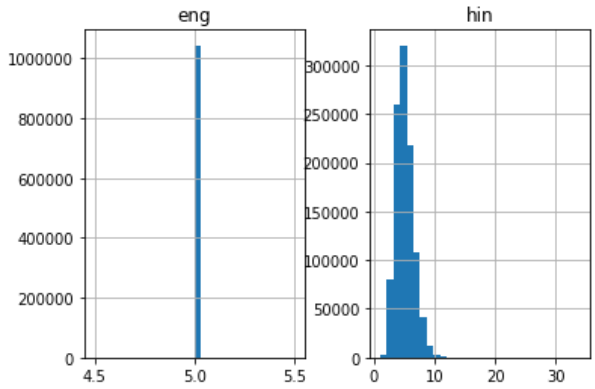
```
 'जुकमन अब हम इसमें शामिल हो जाता है '],
['zulu nationalist inkatha freedom party ',
 'जुलू राष्ट्रवादी इनकाथा स्वतंत्रता पार्टी']], dtype='<U174')
```

A Seq2Seq model requires that we convert both the input and the output sentences into integer sequences of fixed length.

But before we do that, let's visualise the length of the sentences. We will capture the lengths of all the sentences in two separate lists for English and Hindi, respectively.

In [41]:

```
# empty lists
eng_l = []
hin_l = []

# populate the lists with sentence lengths
for i in hin_eng[:,0]:
      eng_l.append(len(i.split()))

for i in hin_eng[:,1]:
      hin_l.append(len(i.split()))

length_df = pd.DataFrame({'eng':eng_l, 'hin':hin_l})

length_df.hist(bins = 30)
plt.show()
```



In [42]:

```
hin_eng[115,0].split()
```

Out[42]:

```
['a', 'balance', 'between', 'the', 'two']
```

In [43]:

```
len(hin_eng[115,0].split())
```

Out[43]:

```
5
```

Quite intuitive – the maximum length of the Hindi sentences is 11 and that of the English phrases is 5.

Next, vectorize our text data by using Keras's Tokenizer() class. It will turn our sentences into sequences of integers. We can then pad those sequences with zeros to make all the sequences of the same length.

Note that we will prepare tokenizers for both the Hindi and English sentences:

In [16]:

```
# function to build a tokenizer
def tokenization(lines):
      tokenizer = Tokenizer()
```

```
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

In [44]:

```
# prepare english tokenizer
eng_tokenizer = tokenization(hin_eng[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1

eng_length = 11
print('English Vocabulary Size: %d' % eng_vocab_size)
```

```
English Vocabulary Size: 24864
```

In [45]:

```
# prepare Deutch tokenizer
hin_tokenizer = tokenization(hin_eng[:, 1])
hin_vocab_size = len(hin_tokenizer.word_index) + 1

hin_length =11
print('hindi Vocabulary Size: %d' % hin_vocab_size)
```

```
hindi Vocabulary Size: 26116
```

The below code block contains a function to prepare the sequences. It will also perform sequence padding to a maximum sentence length as mentioned above.

In [46]:

```
# encode and pad sequences
def encode_sequences(tokenizer, length, lines):
        # integer encode sequences
        seq = tokenizer.texts_to_sequences(lines)
        # pad sequences with 0 values
        seq = pad_sequences(seq, maxlen=length, padding='post')
        return seq
```

ModelBuilding

We will now split the data into train and test set for model training and evaluation, respectively.

In [47]:

```
import sklearn
```

In [48]:

```
from sklearn.model_selection import train_test_split

# split data into train and test set
train, test = train_test_split(hin_eng, test_size=0.1, random_state = 12)
```

It's time to encode the sentences. We will encode English sentences as the input sequences and Hindi sentences as the target sequences. This has to be done for both the train and test datasets.

In [49]:

```
# prepare training data
trainX = encode_sequences(hin_tokenizer, hin_length, train[:, 1])
trainY = encode_sequences(eng_tokenizer, eng_length, train[:, 0])

# prepare validation data
testX = encode_sequences(hin_tokenizer, hin_length, test[:, 1])
testY = encode_sequences(eng_tokenizer, eng_length, test[:, 0])
```

We'll start off by defining our Seq2Seq model architecture:

For the encoder, we will use an embedding layer and an LSTM layer For the decoder, we will use another LSTM layer followed by a dense layer

In [50]:

```
# build NMT model
def define_model(in_vocab,out_vocab, in_timesteps,out_timesteps,units):
      model = Sequential()
      model.add(Embedding(in_vocab, units, input_length=in_timesteps, mask_zero=True))
      model.add(LSTM(units))
      model.add(RepeatVector(out_timesteps))
      model.add(LSTM(units, return_sequences=True))
      model.add(Dense(out_vocab, activation='softmax'))
      return model
```

We are using the RMSprop optimizer in this model as it's usually a good choice when working with recurrent neural networks.

In [51]:

```
# model compilation
model = define_model(eng_vocab_size, hin_vocab_size, eng_length, hin_length, 512)
```

In [52]:

```
rms = optimizers.RMSprop(lr=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')
```

Please note that we have used 'sparse_categorical_crossentropy' as the loss function. This is because the function allows us to use the target sequence as is, instead of the one-hot encoded format. One-hot encoding the target sequences using such a huge vocabulary might consume our system's entire memory.

We are all set to start training our model!

We will train it for 12 epochs and with a batch size of 512 with a validation split of 10%. 90% of the data will be used for training the model and the rest for evaluating it. You may change and play around with these hyperparameters.

We will also use the ModelCheckpoint() function to save the model with the lowest validation loss.

In [58]:

```
filename = 'model.h1.13_apr_19'
checkpoint = ModelCheckpoint(filename, monitor='val_loss', verbose=1, save_best_only=True, mode='mi
n')

# train model
history = model.fit(trainY, trainX.reshape(trainX.shape[0], trainX.shape[1], 1),
                    epochs=12, batch_size=512, validation_split = 0.1,callbacks=[checkpoint],
                    verbose=1)
```

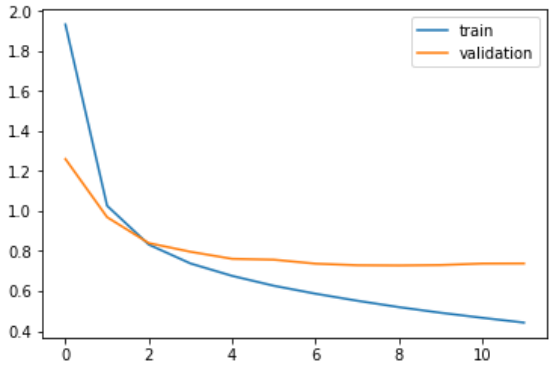
```
Train on 845856 samples, validate on 93985 samples
Epoch 1/12
845856/845856 [==============================] - 11780s 14ms/step - loss: 1.9319 - val_loss: 1.259
9

Epoch 00001: val_loss improved from inf to 1.25990, saving model to model.h1.13_apr_19
Epoch 2/12
845856/845856 [==============================] - 11701s 14ms/step - loss: 1.0260 - val_loss: 0.969
1

Epoch 00002: val_loss improved from 1.25990 to 0.96911, saving model to model.h1.13_apr_19
Epoch 3/12
845856/845856 [==============================] - 11722s 14ms/step - loss: 0.8320 - val_loss: 0.839
6

Epoch 00003: val_loss improved from 0.96911 to 0.83963, saving model to model.h1.13_apr_19
Epoch 4/12
845856/845856 [==============================] - 11747s 14ms/step - loss: 0.7378 - val_loss: 0.796
```

```
2

Epoch 00004: val_loss improved from 0.83963 to 0.79620, saving model to model.h1.13_apr_19
Epoch 5/12
845856/845856 [==============================] - 11757s 14ms/step - loss: 0.6757 - val_loss: 0.760
9

Epoch 00005: val_loss improved from 0.79620 to 0.76093, saving model to model.h1.13_apr_19
Epoch 6/12
845856/845856 [==============================] - 11742s 14ms/step - loss: 0.6263 - val_loss: 0.756
6

Epoch 00006: val_loss improved from 0.76093 to 0.75657, saving model to model.h1.13_apr_19
Epoch 7/12
845856/845856 [==============================] - 11750s 14ms/step - loss: 0.5867 - val_loss: 0.736
8

Epoch 00007: val_loss improved from 0.75657 to 0.73677, saving model to model.h1.13_apr_19
Epoch 8/12
845856/845856 [==============================] - 11765s 14ms/step - loss: 0.5515 - val_loss: 0.729
0

Epoch 00008: val_loss improved from 0.73677 to 0.72899, saving model to model.h1.13_apr_19
Epoch 9/12
845856/845856 [==============================] - 11756s 14ms/step - loss: 0.5196 - val_loss: 0.727
9

Epoch 00009: val_loss improved from 0.72899 to 0.72791, saving model to model.h1.13_apr_19
Epoch 10/12
845856/845856 [==============================] - 11748s 14ms/step - loss: 0.4915 - val_loss: 0.729
9

Epoch 00010: val_loss did not improve from 0.72791
Epoch 11/12
845856/845856 [==============================] - 11732s 14ms/step - loss: 0.4662 - val_loss: 0.737
4

Epoch 00011: val_loss did not improve from 0.72791
Epoch 12/12
845856/845856 [==============================] - 11730s 14ms/step - loss: 0.4423 - val_loss: 0.737
8

Epoch 00012: val_loss did not improve from 0.72791
```

Let's compare the training loss and the validation loss.

In [59]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train','validation'])
plt.show()
```



As you can see in the above plot, the validation loss stopped decreasing after 10 epochs.

Finally, we can load the saved model and make predictions on the unseen data – testY.

In [67]:

```
#Load Weights file
weightsfile= "model.h1.13_apr_19"

#FInd the number of the epoch from the file name
model.load_weights(weightsfile)

# compile network
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')


model.fit(trainY[0:10000], trainX[0:10000].reshape(trainX[0:10000].shape[0], trainX[0:10000].shape[
1], 1),epochs=1,
          batch_size=512, validation_split = 0.1, verbose=1)
# fit network
# model.fit(trainY, trainX.reshape(trainX.shape[0], trainX.shape[1], 1),epochs=1,
 #         batch_size=512, validation_split = 0.1,callbacks=[checkpoint], verbose=1)
```

```
Train on 9000 samples, validate on 1000 samples
Epoch 1/1
9000/9000 [==============================] - 128s 14ms/step - loss: 0.4790 - val_loss: 0.4844
```

Out[67]:

```
<keras.callbacks.History at 0x497d39b0>
```

In [70]:

```
model = load_model('model.h1.13_apr_19')
preds = model.predict_classes(testY[0:1000].reshape((testY[0:1000].shape[0],testY[0:1000].shape[1])
))
```

These predictions are sequences of integers. We need to convert these integers to their corresponding words. Let's define a function to do this:

In [71]:

```
def get_word(n, tokenizer):
      for word, index in tokenizer.word_index.items():
          if index == n:
              return word
      return None
```

Convert predictions into text (Hindi)

In [72]:

```
preds_text = []
for i in preds:
       temp = []
       for j in range(len(i)):
            t = get_word(i[j], hin_tokenizer)
            if j > 0:
                if (t == get_word(i[j-1], hin_tokenizer)) or (t == None):
                     temp.append('')
                else:
                     temp.append(t)
            else:
                  if(t == None):
                         temp.append('')
                  else:
                         temp.append(t)

       preds_text.append(' '.join(temp))
```

In [92]:

```
pred_df = pd.DataFrame({'actual' : test[0:1000][:,1], 'predicted' : preds_text})
```

We can randomly print some actual vs predicted instances to see how our model performs:

We can randomly print some actual Vs predicted instances to see how our model performs.

In [93]:

```
# print 15 rows randomly
pred_df.sample(10)
```

Out[93]:

	actual	predicted
357	क्या मुझे सौदा मिला है	मुझे एक मिल है
891	इस लेख के लिए एक साक्षात्कार	इस लेख के लिए एक साक्षात्कार
822	विषय के साथ रखने में	विषय के साथ रखते में
895	फिल्म की है	फिल्म की है
838	सेक्स करने के बारे में सामाजिक मानदंड	सेक्स बनाने के बारे में सेक्स
859	आप एक के साथ समाप्त हो गया	आप एक के साथ समाप्त हो गया
964	लेकिन ऐसा नहीं लगता	लेकिन ऐसा नहीं लगता
636	के साथ गति नहीं रखा है	के साथ गति नहीं है
861	कोई भी नहीं जानता कि क्या हुआ	कोई भी लिए नहीं क्या जानता
123	हालांकि यह नहीं था	हालांकि यह नहीं था

In [ ]:

```
Our Seq2Seq model does a decent job. But there are several instances where it misses out on underst
anding the key words.
```

French-English Translation (Python Code)

The dependent packages: 1) Tensorflow 2) keras 3) string 4) re 5) numpy 6) pandas 7) matplotlib

In [3]:

```
import tensorflow
import keras
```

In [4]:

```
import string
import re
from numpy import array, argmax, random, take
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding, RepeatVector
from keras.preprocessing.text import Tokenizer
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from keras import optimizers
import matplotlib.pyplot as plt
%matplotlib inline
pd.set_option('display.max_colwidth', 200)
```

Our data is a text file (.txt) of English-French sentence pairs. First, we will read the file using the function defined below.

In [5]:

```
# function to read raw text file
def read_text(filename):
        # open the file
        file = open(filename, mode='rt', encoding='utf-8')

        # read all text
        text = file.read()
        file.close()
        return text
```

We'll then split these pairs into English sentences and French sentences respectively.

In [6]:

```
# split a text into sentences
def to_lines(text):
      sents = text.strip().split('\n')
      sents = [i.split('\t') for i in sents]
      return sents
```

We can now use these functions to read the text into an array in our desired format.

In [7]:

```
data = read_text("fra.txt")
fra_eng = to_lines(data)
fra_eng = array(fra_eng)
```

Let's first take a look at our data. This will help us decide which pre-processing steps to adopt.

In [8]:

```
fra_eng
```

Out[8]:

```
array([['Go.', 'Va !'],
       ['Hi.', 'Salut !'],
       ['Run!', 'Cours\u202f!'],
       ...,
       ['Since there are usually multiple websites on any given topic, I usually just click the
back button when I arrive on any webpage that has pop-up advertising. I just go to the next page f
ound by Google and hope for something less irritating.',
        "Puisqu'il y a de multiples sites web sur chaque sujet, je clique d'habitude sur le bouton
retour arrière lorsque j'atterris sur n'importe quelle page qui contient des publicités
surgissantes. Je me rends juste sur la prochaine page proposée par Google et espère tomber sur que
lque chose de moins irritant."],
       ["If someone who doesn't know your background says that you sound like a native speaker, it
means they probably noticed something about your speaking that made them realize you weren't a nat
ive speaker. In other words, you don't really sound like a native speaker.",
        "Si quelqu'un qui ne connaît pas vos antécédents dit que vous parlez comme un locuteur nat
if, cela veut dire qu'il a probablement remarqué quelque chose à propos de votre élocution qui l'a
fait prendre conscience que vous n'êtes pas un locuteur natif. En d'autres termes, vous ne parlez
pas vraiment comme un locuteur natif."],
       ['It may be impossible to get a completely error-free corpus due to the nature of this kind
of collaborative effort. However, if we encourage members to contribute sentences in their own
languages rather than experiment in languages they are learning, we might be able to minimize erro
rs.',
        "Il est peut-être impossible d'obtenir un Corpus complètement dénué de fautes, étant donnée
la nature de ce type d'entreprise collaborative. Cependant, si nous encourageons les membres à pro
duire des phrases dans leurs propres langues plutôt que d'expérimenter dans les langues qu'ils app
rennent, nous pourrions être en mesure de réduire les erreurs."]],
      dtype='<U349')
```

In [9]:

```
len(fra_eng)
```

Out[9]:

```
160872
```

The actual data contains over 160872 sentence-pairs. However, we will use full data. You can change this number as per your system's computation power.

In [10]:

```
fra_eng = fra_eng[:160872,:]
```

We will get rid of the punctuation marks and then convert all the text to lower case.

In [11]:

```
# Remove punctuation
fra_eng[:,0] = [s.translate(str.maketrans('', '', string.punctuation)) for s in fra_eng[:,0]]
fra_eng[:,1] = [s.translate(str.maketrans('', '', string.punctuation)) for s in fra_eng[:,1]]

fra_eng
```

Out[11]:

```
array([['Go', 'Va '],
       ['Hi', 'Salut '],
       ['Run', 'Cours\u202f'],
       ...,
       ['Since there are usually multiple websites on any given topic I usually just click the
back button when I arrive on any webpage that has popup advertising I just go to the next page fou
nd by Google and hope for something less irritating',
        'Puisquil y a de multiples sites web sur chaque sujet je clique dhabitude sur le bouton
retour arrière lorsque jatterris sur nimporte quelle page qui contient des publicités surgissantes
Je me rends juste sur la prochaine page proposée par Google et espère tomber sur quelque chose de
moins irritant'],
       ['If someone who doesnt know your background says that you sound like a native speaker it
means they probably noticed something about your speaking that made them realize you werent a nati
ve speaker In other words you dont really sound like a native speaker',
```

```
       'si quelquun qui ne connait pas vos antecedents dit que vous parlez comme un locuteur nati
f cela veut dire quil a probablement remarqué quelque chose à propos de votre élocution qui la fai
t prendre conscience que vous nêtes pas un locuteur natif En dautres termes vous ne parlez pas vra
iment comme un locuteur natif'],
       ['It may be impossible to get a completely errorfree corpus due to the nature of this kind
of collaborative effort However if we encourage members to contribute sentences in their own
languages rather than experiment in languages they are learning we might be able to minimize error
s',
        'Il est peutêtre impossible dobtenir un Corpus complètement dénué de fautes étant donnée
la nature de ce type dentreprise collaborative Cependant si nous encourageons les membres à
produire des phrases dans leurs propres langues plutôt que dexpérimenter dans les langues quils ap
prennent nous pourrions être en mesure de réduire les erreurs']],
      dtype='<U349')
```

In [12]:

```
# convert text to lowercase
for i in range(len(fra_eng)):
    fra_eng[i,0] = fra_eng[i,0].lower()
    fra_eng[i,1] = fra_eng[i,1].lower()

fra_eng
```

Out[12]:

```
array([['go', 'va '],
       ['hi', 'salut '],
       ['run', 'cours\u202f'],
       ...,
       ['since there are usually multiple websites on any given topic i usually just click the
back button when i arrive on any webpage that has popup advertising i just go to the next page fou
nd by google and hope for something less irritating',
        'puisquil y a de multiples sites web sur chaque sujet je clique dhabitude sur le bouton
retour arrière lorsque jatterris sur nimporte quelle page qui contient des publicités surgissantes
je me rends juste sur la prochaine page proposée par google et espère tomber sur quelque chose de
moins irritant'],
       ['if someone who doesnt know your background says that you sound like a native speaker it
means they probably noticed something about your speaking that made them realize you werent a nati
ve speaker in other words you dont really sound like a native speaker',
        'si quelquun qui ne connaît pas vos antécédents dit que vous parlez comme un locuteur nati
f cela veut dire quil a probablement remarqué quelque chose à propos de votre élocution qui la fai
t prendre conscience que vous nêtes pas un locuteur natif en dautres termes vous ne parlez pas vra
iment comme un locuteur natif'],
       ['it may be impossible to get a completely errorfree corpus due to the nature of this kind
of collaborative effort however if we encourage members to contribute sentences in their own
languages rather than experiment in languages they are learning we might be able to minimize error
s',
        'il est peutêtre impossible dobtenir un corpus complètement dénué de fautes étant donnée
la nature de ce type dentreprise collaborative cependant si nous encourageons les membres à
produire des phrases dans leurs propres langues plutôt que dexpérimenter dans les langues quils ap
prennent nous pourrions être en mesure de réduire les erreurs']],
      dtype='<U349')
```

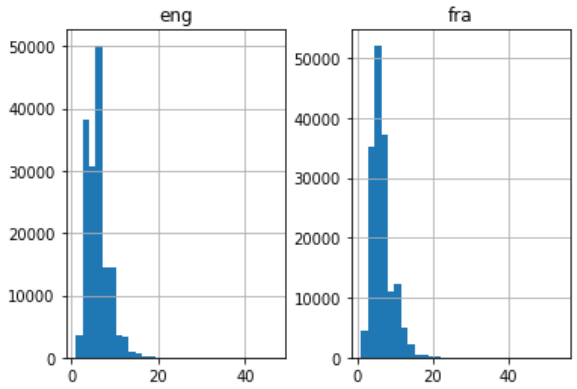
A Seq2Seq model requires that we convert both the input and the output sentences into integer sequences of fixed length.

But before we do that, let's visualise the length of the sentences. We will capture the lengths of all the sentences in two separate lists for English and French, respectively.

In [12]:

```
# empty lists
eng_l = []
fra_l = []

# populate the lists with sentence lengths
for i in fra_eng[:,0]:
      eng_l.append(len(i.split()))

for i in fra_eng[:,1]:
      fra_l.append(len(i.split()))

length_df = pd.DataFrame({'eng':eng_l, 'fra':fra_l})

length_df.hist(bins = 30)
```

```
plt.show()
```



In [13]:

```
fra_eng[115,0].split()
```

Out[13]:

```
['Go', 'away']
```

In [14]:

```
len(fra_eng[115,0].split())
```

Out[14]:

```
2
```

Quite intuitive – the maximum length of the French sentences is 15 and that of the English phrases is 15.

Next, vectorize our text data by using Keras's Tokenizer() class. It will turn our sentences into sequences of integers. We can then pad those sequences with zeros to make all the sequences of the same length.

Note that we will prepare tokenizers for both the French and English sentences:

In [15]:

```
# function to build a tokenizer
def tokenization(lines):
      tokenizer = Tokenizer()
      tokenizer.fit_on_texts(lines)
      return tokenizer
```

In [16]:

```
# prepare english tokenizer
eng_tokenizer = tokenization(fra_eng[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1

eng_length = 15
print('English Vocabulary Size: %d' % eng_vocab_size)
```

```
English Vocabulary Size: 14347
```

In [17]:

```
# prepare Deutch tokenizer
fra_tokenizer = tokenization(fra_eng[:, 1])
fra_vocab_size = len(fra_tokenizer.word_index) + 1

fra_length =15
print('french Vocabulary Size: %d' % fra_vocab_size)
```

```
french Vocabulary Size: 32113
```

The below code block contains a function to prepare the sequences. It will also perform sequence padding to a maximum sentence length as mentioned above.

In [18]:

```
# encode and pad sequences
def encode_sequences(tokenizer, length, lines):
         # integer encode sequences
         seq = tokenizer.texts_to_sequences(lines)
         # pad sequences with 0 values
         seq = pad_sequences(seq, maxlen=length, padding='post')
         return seq
```

We will now split the data into train and test set for model training and evaluation, respectively.

In [19]:

```
import sklearn
```

In [20]:

```
from sklearn.model_selection import train_test_split

# split data into train and test set
train, test = train_test_split(fra_eng, test_size=0.1, random_state = 12)
```

It's time to encode the sentences. We will encode French sentences as the input sequences and English sentences as the target sequences. This has to be done for both the train and test datasets.

In [21]:

```
# prepare training data
trainX = encode_sequences(fra_tokenizer, fra_length, train[:, 1])
trainY = encode_sequences(eng_tokenizer, eng_length, train[:, 0])

# prepare validation data
testX = encode_sequences(fra_tokenizer, fra_length, test[:, 1])
testY = encode_sequences(eng_tokenizer, eng_length, test[:, 0])
```

We'll start off by defining our Seq2Seq model architecture:

For the encoder, we will use an embedding layer and an LSTM layer For the decoder, we will use another LSTM layer followed by a dense layer

In [22]:

```
# build NMT model
def define_model(in_vocab,out_vocab, in_timesteps,out_timesteps,units):
      model = Sequential()
      model.add(Embedding(in_vocab, units, input_length=in_timesteps, mask_zero=True))
      model.add(LSTM(units))
      model.add(RepeatVector(out_timesteps))
      model.add(LSTM(units, return_sequences=True))
      model.add(Dense(out_vocab, activation='softmax'))
      return model
```

We are using the RMSprop optimizer in this model as it's usually a good choice when working with recurrent neural networks.

In [23]:

```
# model compilation
model = define_model(fra_vocab_size, eng_vocab_size, fra_length, eng_length, 512)
```

```
WARNING:tensorflow:From C:\Users\admin\Anaconda3\envs\hello-tf\lib\site-
```

```
packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [24]:

```
rms = optimizers.RMSprop(lr=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')
```

Please note that we have used 'sparse_categorical_crossentropy' as the loss function. This is because the function allows us to use the target sequence as is, instead of the one-hot encoded format. One-hot encoding the target sequences using such a huge vocabulary might consume our system's entire memory.

We are all set to start training our model!

We will train it for 30 epochs and with a batch size of 512 with a validation split of 10%. 90% of the data will be used for training the model and the rest for evaluating it. You may change and play around with these hyperparameters.

We will also use the ModelCheckpoint() function to save the model with the lowest validation loss.

In [66]:

```
filename = 'model.h1.06_apr_19'
checkpoint = ModelCheckpoint(filename, monitor='val_loss', verbose=1, save_best_only=True, mode='mi
n')

# train model
history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1], 1),
                    epochs=30, batch_size=512, validation_split = 0.1,callbacks=[checkpoint],
                    verbose=1)
```

```
Train on 130305 samples, validate on 14479 samples
Epoch 1/30
130305/130305 [==============================] - 1594s 12ms/step - loss: 2.8369 - val_loss: 2.5074

Epoch 00001: val_loss improved from inf to 2.50737, saving model to model.h1.06_apr_19
Epoch 2/30
130305/130305 [==============================] - 1581s 12ms/step - loss: 2.3979 - val_loss: 2.2816

Epoch 00002: val_loss improved from 2.50737 to 2.28162, saving model to model.h1.06_apr_19
Epoch 3/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 2.1429 - val_loss: 2.0406

Epoch 00003: val_loss improved from 2.28162 to 2.04061, saving model to model.h1.06_apr_19
Epoch 4/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 1.9156 - val_loss: 1.8708

Epoch 00004: val_loss improved from 2.04061 to 1.87082, saving model to model.h1.06_apr_19
Epoch 5/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 1.7210 - val_loss: 1.6923

Epoch 00005: val_loss improved from 1.87082 to 1.69232, saving model to model.h1.06_apr_19
Epoch 6/30
130305/130305 [==============================] - 1573s 12ms/step - loss: 1.5518 - val_loss: 1.5951

Epoch 00006: val_loss improved from 1.69232 to 1.59513, saving model to model.h1.06_apr_19
Epoch 7/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 1.4048 - val_loss: 1.4701

Epoch 00007: val_loss improved from 1.59513 to 1.47005, saving model to model.h1.06_apr_19
Epoch 8/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 1.2795 - val_loss: 1.3906

Epoch 00008: val_loss improved from 1.47005 to 1.39058, saving model to model.h1.06_apr_19
Epoch 9/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 1.1699 - val_loss: 1.3231

Epoch 00009: val_loss improved from 1.39058 to 1.32315, saving model to model.h1.06_apr_19
Epoch 10/30
130305/130305 [==============================] - 1573s 12ms/step - loss: 1.0738 - val_loss: 1.2722

Epoch 00010: val_loss improved from 1.32315 to 1.27222, saving model to model.h1.06_apr_19
```

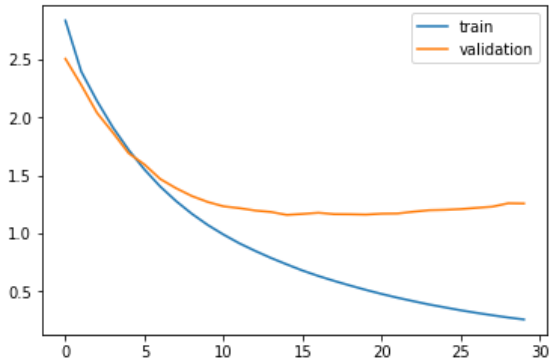
```
Epoch 11/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.9906 - val_loss: 1.2346

Epoch 00011: val_loss improved from 1.27222 to 1.23456, saving model to model.h1.06_apr_19
Epoch 12/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.9153 - val_loss: 1.2189

Epoch 00012: val_loss improved from 1.23456 to 1.21889, saving model to model.h1.06_apr_19
Epoch 13/30
130305/130305 [==============================] - 1573s 12ms/step - loss: 0.8500 - val_loss: 1.1984

Epoch 00013: val_loss improved from 1.21889 to 1.19841, saving model to model.h1.06_apr_19
Epoch 14/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.7892 - val_loss: 1.1859

Epoch 00014: val_loss improved from 1.19841 to 1.18594, saving model to model.h1.06_apr_19
Epoch 15/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.7337 - val_loss: 1.1606

Epoch 00015: val_loss improved from 1.18594 to 1.16065, saving model to model.h1.06_apr_19
Epoch 16/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.6802 - val_loss: 1.1690

Epoch 00016: val_loss did not improve from 1.16065
Epoch 17/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.6334 - val_loss: 1.1791

Epoch 00017: val_loss did not improve from 1.16065
Epoch 18/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.5909 - val_loss: 1.1671

Epoch 00018: val_loss did not improve from 1.16065
Epoch 19/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.5513 - val_loss: 1.1661

Epoch 00019: val_loss did not improve from 1.16065
Epoch 20/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.5137 - val_loss: 1.1634

Epoch 00020: val_loss did not improve from 1.16065
Epoch 21/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.4786 - val_loss: 1.1712

Epoch 00021: val_loss did not improve from 1.16065
Epoch 22/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.4461 - val_loss: 1.1728

Epoch 00022: val_loss did not improve from 1.16065
Epoch 23/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.4163 - val_loss: 1.1881

Epoch 00023: val_loss did not improve from 1.16065
Epoch 24/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.3877 - val_loss: 1.2011

Epoch 00024: val_loss did not improve from 1.16065
Epoch 25/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.3626 - val_loss: 1.2044

Epoch 00025: val_loss did not improve from 1.16065
Epoch 26/30
130305/130305 [==============================] - 1574s 12ms/step - loss: 0.3376 - val_loss: 1.2107

Epoch 00026: val_loss did not improve from 1.16065
Epoch 27/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.3154 - val_loss: 1.2215

Epoch 00027: val_loss did not improve from 1.16065
Epoch 28/30
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.2949 - val_loss: 1.2317

Epoch 00028: val_loss did not improve from 1.16065
Epoch 29/30
130305/130305 [==============================] - 1572s 12ms/step - loss: 0.2754 - val_loss: 1.2609

Epoch 00029: val_loss did not improve from 1.16065
Epoch 30/30
```

```
130305/130305 [==============================] - 1571s 12ms/step - loss: 0.2586 - val_loss: 1.2589

Epoch 00030: val_loss did not improve from 1.16065
```

Let's compare the training loss and the validation loss.

In [67]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train','validation'])
plt.show()
```



As you can see in the above plot, the validation loss stopped decreasing after 10 epochs.

Finally, we can load the saved model and make predictions on the unseen data – testY.

In [26]:

```
#Load Weights file
weightsfile= "model.h1.06_apr_19"

#FInd the number of the epoch from the file name
model.load_weights(weightsfile)

# compile network
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')


model.fit(trainX[0:10000], trainY[0:10000].reshape(trainY[0:10000].shape[0], trainY[0:10000].shape[
1], 1),epochs=1,
          batch_size=512, validation_split = 0.1, verbose=1)
# fit network
# model.fit(trainY, trainX.reshape(trainX.shape[0], trainX.shape[1], 1),epochs=1,
 #         batch_size=512, validation_split = 0.1,callbacks=[checkpoint], verbose=1)
```

```
Train on 9000 samples, validate on 1000 samples
Epoch 1/1
9000/9000 [==============================] - 108s 12ms/step - loss: 0.6825 - val_loss: 0.6667
```

Out[26]:

```
<keras.callbacks.History at 0x4546fbe0>
```

In [27]:

```
model = load_model('model.h1.06_apr_19')
preds = model.predict_classes(testX.reshape((testX.shape[0],testX.shape[1])))
```

These predictions are sequences of integers. We need to convert these integers to their corresponding words. Let's define a function to do this:

In [28]:

```
def get_word(n, tokenizer):
      for word, index in tokenizer.word_index.items():
          if index == n:
              return word
      return None
```

Convert predictions into text (English)

In [29]:

```
preds_text = []
for i in preds:
       temp = []
       for j in range(len(i)):
            t = get_word(i[j], eng_tokenizer)
            if j > 0:
                if (t == get_word(i[j-1], eng_tokenizer)) or (t == None):
                    temp.append('')
                else:
                    temp.append(t)
            else:
                   if(t == None):
                          temp.append('')
                   else:
                          temp.append(t)

       preds_text.append(' '.join(temp))
```

In [30]:

```
pred_df = pd.DataFrame({'actual' : test[:,0], 'predicted' : preds_text})
```

We can randomly print some actual vs predicted instances to see how our model performs:

In [42]:

```
# print 15 rows randomly
pred_df.sample(5)
```

Out[42]:

	actual	predicted
14148	I want you to run to the store for me	i want you to look in the store me
8417	I dont feel like doing my math homework now	i dont feel like mood to
15495	What did you want to see me about	what did you want to me
8689	We were prisoners	we were calm
9274	Tom came too late	tom arrived too

Our Seq2Seq model does a decent job. But there are several instances where it misses out on understanding the key words.

In [ ]:

```
```

8 Future Work

- API will be created with the model developed.
- Simple HTML page will be created which include one text input box, which takes english text as input.
- The model API created, will translate the input text into output text(target language) and finally show the translated sentence.

References

[1] https://www.analyticsvidhya.com/blog/2019/01/neural-machine-translation-keras/.

[2] Activation function. https://github.com/shreyans29/thesemicolon.

[3] Logistic regression and artificial neural network. https://statinfer.com/.

[4] Machine translation. https://en.wikipedia.org/wiki/Machine_translation.