**Problem Definition and Design Thinking**

**Problem Definition:**

The problem is to develop a fake news detection model using a Kaggle dataset. The goal is to distinguish between genuine and fake news articles based on their titles and text. This project involves using natural language processing (NLP) techniques to preprocess the text data, building a machine learning model for classification, and evaluating the model's performance.

**Design Thinking:**

**1. Data Collection:**

Collect a labeled dataset of news articles, where each article is tagged as Fake or True. we take it from such datasets on platforms like Kaggle.

**2. Data Preprocessing:**

Preprocess the text data:

Clean the text by removing special characters, and punctuation.

Tokenize the text into words or sub word tokens. Remove stop words.

We can add the label in the dataset like fake as 0 and true as 1.

**3. Data Splitting:**

Split your dataset into training and test sets. A common split might be 80% for training, 20% for testing.

**4. Word Embeddings:**

Convert text data into numerical vectors using word embeddings like Word2Vec and Glove. These embeddings capture semantic meaning.

**5. RNN Model Design:**

Choose an RNN architecture such as simple RNN or LSTM. These RNN variants are good choices for sequential data like text. Define the input layer to accept the word embeddings.

Include techniques like dropout, early stopping, regularization to prevent overfitting. Add a fully connected layer(s) with appropriate activation functions like sigmoid, Tanh, ReLu etc....

Experiment with different hyperparameters, including learning rate, batch size, number of hidden units, and dropout rates to find the best-performing model.

**6. Model Training:**

Train your RNN model on the training data.

Use appropriate loss functions such as binary cross-entropy for binary classification.

Optimize your model using gradient descent-based optimizers like Adam or RMSprop.

**7.Model Evaluation:**

Evaluate your RNN model's performance on the test set using metrics like accuracy, precision, recall, F1-score, and confusion matrix.

**DATA COLLECTION**

**80% Training**

**20% Testing**

**DATA SPLITTING**

**DATA PREPROCESSING**

**WORD EMBEDDINGS**

**RNN MODEL DESIGN**

**MODEL TRAINING**

**MODEL**

**EVALUATION**

Link of the dataset used: <https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset>

**Description of dataset**

The "fake-and-real-news-dataset" is a popular dataset that is often used for fake news detection and natural language processing (NLP) tasks. It is typically used for building and training deep learning models to classify true and fake news. Here are some key characteristics of this dataset:

1. Data Source: This dataset is often available on platforms like Kaggle.
2. Data Content: The dataset contains a collection of text and title, which are labeled as either "true" or "fake. Each message in the dataset is associated with a class label 0 and 1 .
3. Data Format: The dataset is usually provided in a structured format, often as a CSV file.
4. Size: The dataset typically contains few thousands of text and title , with a relatively balanced distribution of true and fake news .
5. Applications: The applications of this dataset include fake news detection and helps to aware of fake news.

Data preprocessing:

**1.Data Collection**

Gather a dataset that contains labeled examples of real and fake news articles.

**2. Handling Missing Values:**

* Check for missing values in dataset and decide how to handle them (e.g., remove rows with missing data or impute values).

**3. labeling:**

Convert the class labels (true/fake) into numerical values. For instance, you can map "true" to 1 and "fake" to 0.

**4. information about dataset:**

Checking how many columns in the dataset and non null values count.

**5.Importing necessary modules:**

Importing necessary modules like numpy, pandas, tensorflow, wordcloud, stopwords, nltk etc.

**6.stop words:**

Remove common words (stop words) like "the," "and," "in" as they often don't carry important information for fake news detection.

**7.Lowercasing:**

Convert all text to lowercase to ensure uniformity.

**8. *Tokenization:***

Split text into individual words or tokens.

9. **. Handling Text Encoding:**

Convert text data into a numerical format that can be fed into deep learning models. use one-hot encoding for BoW or embeddings for word vectors.

**10. padding:**

Add padding technique or truncate text sequences to a fixed length,

because we are using neural network.

**Visualization**

**Countplot**

which is built on top of Matplotlib and is used for data visualization.

Helps to visualize the distribution of categorical data.

* We use seaborn library for visualization. we plot a countplot for the distribution of fake and real news.
* We plot the countplot for the distribution of subjects.

**Wordcloud**

Word clouds are graphical representations of words, where the size of each word is proportional to its frequency or importance in a given text or dataset.

Word clouds are commonly used to visualize the most frequently occurring words in a corpus of text. This can give you a quick overview of the most important or common terms in a document or a collection of documents.

* We plot the wordcloud for the fake news distribution and real news distribution

**Histogram**

* A histogram is used to visualize the distribution of numerical data by representing the frequencies of data points within specific intervals (bins).
* We using histogram for visualize the distribution of number of words in a text.

**Training and Testing**

80% of the data used for training and 20% of data used for testing

We use LSTM model for fake news detection:

**LSTM:**

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture used in deep learning. It is designed to model and process sequential data, like time series and natural language, by maintaining and updating internal memory states over long sequences. LSTMs are capable of capturing long-term dependencies and are commonly used in tasks such as speech recognition, language translation, fake news detection and sentiment analysis.

**Model Training:**

* This creates a sequential neural network model for binary text classification.
* It begins with an embedding layer with 128-dimensional word embeddings, followed by a bidirectional LSTM layer with 128 units.
* A dense layer with 128 neurons and ReLU activation is included, and the output layer consists of a single neuron with a sigmoid activation function for binary classification.
* The model is compiled using the Adam optimizer and binary cross-entropy loss. This architecture is designed to process and classify text data efficiently.

**Bi-directional RNN:**

A Bi-directional RNN (Recurrent Neural Network) is a neural network architecture that processes sequential data in both forward and backward directions simultaneously, allowing it to capture information from past and future contexts, which is especially useful for tasks like natural language understanding where context matters.

**Prediction:**

* Generally, snippet post-processes a list of numerical predictions (`pred`) for a binary classification task.
* It converts probability scores into binary class predictions by considering a threshold of 0.5. If a probability score is greater than 0.5, it is categorized as class 1 (positive), and if it's equal to or less than 0.5, it is categorized as class 0 (negative).
* The resulting binary predictions are stored in the `prediction` list. This post-processing step is common in binary classification tasks to make decisions based on model outputs.

**Model Evaluation:**

After training, evaluate the model's performance on the testing dataset using appropriate evaluation metrics.

**Evaluation Metrics:**

**Accuracy:** Measures the overall correctness of the model's predictions. However, accuracy may not be the best metric when classes are imbalanced.

**Confusion Matrix:** A table showing true positive, true negative, false positive, and false negative predictions, which can be used to compute various metrics.

**#CODE** :

#importing all the necessary modules and models.

import tensorflow as tf

import numpy as np

from wordcloud import WordCloud, STOPWORDS

import nltk

nltk.download('punkt')

import re

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize

import gensim

from gensim.utils import simple\_preprocess

from gensim.parsing.preprocessing import STOPWORDS

from tensorflow.keras.preprocessing.text import one\_hot, Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Embedding, Input, LSTM, Conv1D, MaxPool1D, Bidirectional

from tensorflow.keras.models import Model

from google.colab import drive

drive.mount('/content/drive')

#read the csv file

import pandas as pd

fake=pd.read\_csv("/content/drive/MyDrive/rough/Fake.csv")

true=pd.read\_csv("/content/drive/MyDrive/rough/True.csv")

#label fake as 0 and true as 1

fake["label"]=0

true["label"]=1

#concatenating two files

df= pd.concat([fake,true], axis=0)

#define the shape of the dataset

df.shape

#analysing of the dataset

df.info()

#Obtain the total words present in the dataset

list\_of\_words = []

for i in df.after\_clean:

    for j in i:

        list\_of\_words.append(j)

#This code snippet calculates summary statistics for the 'clean\_joined' column in a DataFrame 'df' by applying a lambda function to concatenate the elements of the 'after\_clean' column and then using the `describe` method.

df['clean\_joined'] = df['after\_clean'].apply(lambda x: " ".join(x))

#visualizing the dataset

import seaborn as sns

import matplotlib.pyplot as plt

sns.countplot(x=df["label"])

plt.title('distribution of real and fake news')

plt.show()

sns.countplot(y = df['subject'])

plt.title("distribution of subjects")

#wordcloud

plt.figure(figsize = (20,20))

wc = WordCloud(max\_words = 2000 , width = 1600 , height = 800 , stopwords = stop\_words).generate(" ".join(df[df.label == 0].clean\_joined))

plt.imshow(wc, interpolation = 'bilinear')

# Create a tokenizer to tokenize the words and create sequences of tokenized words

tokenizer = Tokenizer(num\_words = total\_words)

tokenizer.fit\_on\_texts(x\_train)

train\_sequences = tokenizer.texts\_to\_sequences(x\_train)

test\_sequences = tokenizer.texts\_to\_sequences(x\_test)

#

# Sequential Model

model = Sequential()

# embeddidng layer

model.add(Embedding(total\_words, output\_dim = 128))

# model.add(Embedding(total\_words, output\_dim = 240))

# Bi-Directional RNN and LSTM

model.add(Bidirectional(LSTM(128))) # no of neurons

# Dense layers

model.add(Dense(128, activation = 'relu'))

model.add(Dense(1,activation= 'sigmoid')) # reason: we do binary classification here

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])

model.summary()

#getting the accuracy

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(list(y\_test), prediction)

print("Model Accuracy : ", accuracy)

#confusion matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(list(y\_test), prediction)

plt.figure(figsize = (15, 15))

sns.heatmap(cm, annot = True)