📘 Introduction

In this project, we develop a binary text classification model using Recurrent Neural Networks (RNNs) in TensorFlow and Keras. The goal is to classify short text messages as either disaster-related (1) or not disaster-related (0), which is a common task in natural language processing (NLP) applications such as emergency response systems or real-time social media monitoring.

The dataset consists of short tweets with corresponding binary labels indicating whether the tweet refers to a disaster event. Each entry in the dataset includes the tweet text and its label. The key steps in the pipeline are:

- Data cleaning: removing noise, special characters, and transforming text to lowercase.
- Tokenization and vectorization: converting raw text into sequences of integers using Keras' Tokenizer and TextVectorization.
- Model architecture: building an RNN-based model with layers such as Embedding, LSTM, Dropout, and Dense layers to process sequential data and learn temporal dependencies.
- Training and evaluation: compiling the model with binary_crossentropy loss and evaluating its performance on validation data using metrics like accuracy, precision, recall, and F1-score.
- Model interpretation: visualizing training history and evaluating prediction quality on unseen data.

The main objective is to build a robust baseline model that can distinguish between relevant and irrelevant emergency tweets, thereby laying the groundwork for more advanced NLP classification systems.

In [3]:
```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf

import skillsnetwork
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.text import Tokenizer
# from tensorflow.keras.losses import mean_squared_error  # Removed because mean_sq
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Embedding,Masking,LSTM, GRU,
from tensorflow.keras.optimizers import Adam
```

```python
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Embedding, SimpleRNN
from tensorflow.keras.datasets import reuters
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import TextVectorization
from sklearn.metrics import accuracy_score,precision_recall_fscore_support
import tensorflow_hub as hub


# You can also use this section to suppress warnings generated by your code:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

sns.set_context('notebook')
sns.set_style('white')
np.random.seed(2024)
```

In [4]:

```python
#Helper Functions
# function to compute the accuracy, precision, recall and F1 score of a model's pre
def calculate_results(y_true, y_pred):
    model_accuracy = accuracy_score(y_true, y_pred)
    model_precision, model_recall, model_f1,_ = precision_recall_fscore_support(y_t
    model_results = {"accuracy":model_accuracy,
                     "precision":model_precision,
                     "recall" :model_recall,
                     "f1":model_f1}
    return model_results
```

In [5]:

```python
import skillsnetwork
import zipfile

# Download and extract to a directory
await skillsnetwork.download(
    "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevelope
)


with zipfile.ZipFile("nlp_disaster.zip", "r") as zip_ref:
    zip_ref.extractall("nlp_disaster")
```

```
Downloading nlp_disaster.zip:   0%|          | 0/607343 [00:00<?, ?it/s]
Saved as 'nlp_disaster.zip'
```

In [6]:

```python
train_df = pd.read_csv("nlp_disaster/train.csv")
# shuffle the dataset
train_df_shuffled = train_df.sample(frac=1, random_state=42)
train_df_shuffled.head()
```

Out[6]:

	id	keyword	location	text	target
2644	3796	destruction	NaN	So you have a new weapon that can cause un-ima...	1
2227	3185	deluge	NaN	The f$&@ing things I do for #GISHWHES Just...	0
5448	7769	police	UK	DT @georgegalloway: RT @Galloway4Mayor: ÛÏThe...	1
132	191	aftershock	NaN	Aftershock back to school kick off was great. ...	0
6845	9810	trauma	Montgomery County, MD	in response to trauma Children of Addicts deve...	0

In [7]:
```
from sklearn.model_selection import train_test_split

#We will use 90% of the entire labelled dataset for training, and 10% of it for tes
# split the data into 90% training and 10% testing
X_train, X_test, y_train, y_test = train_test_split(train_df_shuffled["text"].to_nu
                                                    train_df_shuffled["target"].to_
                                                    test_size = 0.1,
                                                    random_state=42)
X_train.shape, y_train.shape
```

Out[7]:
```
((6851,), (6851,))
```

In [8]:
```
X_train[0:5]
```

Out[8]:
```
array(['@mogacola @zamtriossu i screamed after hitting tweet',
       'Imagine getting flattened by Kurt Zouma',
       '@Gurmeetramrahim #MSGDoing111WelfareWorks Green S welfare force ke appx 65
000 members har time disaster victim ki help ke liye tyar hai....',
       "@shakjn @C7 @Magnums im shaking in fear he's gonna hack the planet",
       'Somehow find you and I collide http://t.co/Ee8RpOahPk'],
      dtype=object)
```

`TextVectorization` is a preprocessing layer which maps text features to integer sequences. We also specify `lower_and_strip_punctuation` as the standardization method to apply to the input text. The text will be lowercased and all punctuation removed. Next we split on the whitespace, and pass `None` to `ngrams` so no ngrams are created.

In [9]:
```
text_vectorizer = TextVectorization(max_tokens=None,
                                    #remove punctuation and make letters lowercase
                                    standardize="lower_and_strip_punctuation",
                                    #whitespace delimiter
                                    split="whitespace",
                                    #dont group anything, every token alone
                                    ngrams = None,
                                    output_mode ="int",
                                    #length of each sentence == length of largest s
```

```
                                    output_sequence_length=None
                                    )
```

In [10]:
```
# define hyperparameters

# number of words in the vocabulary
max_vocab_length = 10000
# tweet average length
max_length = 15
```

Below we define an Embedding layer with a vocabulary of 10,000, a vector space of 128 dimensions in which words will be embedded, and input documents that have 15 words each.

In [11]:
```
embedding = layers.Embedding(input_dim= max_vocab_length,
                             output_dim=128,
                             input_length=max_length)
```

The hub.KerasLayer wraps a SavedModel (or a legacy TF1 Hub format) as a Keras Layer. The universal-sentence-encoder is an encoder of greater-than-word length text trained on a variety of data. It can be used for text classification, semantic similarity, clustering, and other natural language tasks.

In [12]:
```
encoder_layer = hub.KerasLayer("https://tfhub.dev/google/universal-sentence-encoder
                               input_shape=[],
                               dtype = tf.string,
                               trainable=False,
                               name="pretrained")
```

```
WARNING:tensorflow:Please fix your imports. Module tensorflow.python.training.tracki
ng.data_structures has been moved to tensorflow.python.trackable.data_structures. Th
e old module will be deleted in version 2.11.
```

The encoder_layer will take as input variable length English text and the output is a 512 dimensional vector.

We will add a Dense layer with unit 1 to create a simple binary text classifier on top of any TF-Hub module. Next, we will compile and fit it using 20 epochs.

In [13]:
```
model = tf.keras.Sequential([
                             encoder_layer,
                             layers.Dense(1,activation="sigmoid")], name="model_pre
model.compile(loss="binary_crossentropy",
                     optimizer="adam",
                     metrics=["accuracy"])

model.fit(x=X_train,
              y=y_train,
              epochs=20,
              validation_data=(X_test,y_test))
```

```
WARNING:tensorflow:From c:\Users\cesar\tf_clean\lib\site-packages\tensorflow\python
\autograph\pyct\static_analysis\liveness.py:83: Analyzer.lamba_check (from tensorflo
w.python.autograph.pyct.static_analysis.liveness) is deprecated and will be removed
after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are u
sed, or at least in the same block. https://github.com/tensorflow/tensorflow/issues/
56089
```


```
WARNING:tensorflow:From c:\Users\cesar\tf_clean\lib\site-packages\tensorflow\python
\autograph\pyct\static_analysis\liveness.py:83: Analyzer.lamba_check (from tensorflo
w.python.autograph.pyct.static_analysis.liveness) is deprecated and will be removed
after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are u
sed, or at least in the same block. https://github.com/tensorflow/tensorflow/issues/
56089
```

```
Epoch 1/20
215/215 [==============================] - 3s 6ms/step - loss: 0.6500 - accuracy: 0.
7323 - val_loss: 0.6133 - val_accuracy: 0.7677
Epoch 2/20
215/215 [==============================] - 1s 5ms/step - loss: 0.5820 - accuracy: 0.
7900 - val_loss: 0.5632 - val_accuracy: 0.7861
Epoch 3/20
215/215 [==============================] - 1s 5ms/step - loss: 0.5387 - accuracy: 0.
7975 - val_loss: 0.5319 - val_accuracy: 0.7861
Epoch 4/20
215/215 [==============================] - 1s 5ms/step - loss: 0.5099 - accuracy: 0.
7999 - val_loss: 0.5100 - val_accuracy: 0.7887
Epoch 5/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4896 - accuracy: 0.
7997 - val_loss: 0.4955 - val_accuracy: 0.7927
Epoch 6/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4748 - accuracy: 0.
8016 - val_loss: 0.4848 - val_accuracy: 0.7927
Epoch 7/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4636 - accuracy: 0.
8024 - val_loss: 0.4770 - val_accuracy: 0.7913
Epoch 8/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4549 - accuracy: 0.
8047 - val_loss: 0.4715 - val_accuracy: 0.7927
Epoch 9/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4479 - accuracy: 0.
8062 - val_loss: 0.4671 - val_accuracy: 0.7953
Epoch 10/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4422 - accuracy: 0.
8083 - val_loss: 0.4638 - val_accuracy: 0.7927
Epoch 11/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4375 - accuracy: 0.
8088 - val_loss: 0.4610 - val_accuracy: 0.7953
Epoch 12/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4334 - accuracy: 0.
8102 - val_loss: 0.4590 - val_accuracy: 0.7953
Epoch 13/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4300 - accuracy: 0.
8110 - val_loss: 0.4573 - val_accuracy: 0.7979
Epoch 14/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4271 - accuracy: 0.
8124 - val_loss: 0.4555 - val_accuracy: 0.7966
Epoch 15/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4245 - accuracy: 0.
8137 - val_loss: 0.4540 - val_accuracy: 0.7966
Epoch 16/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4222 - accuracy: 0.
8155 - val_loss: 0.4529 - val_accuracy: 0.8005
Epoch 17/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4201 - accuracy: 0.
8145 - val_loss: 0.4524 - val_accuracy: 0.8045
Epoch 18/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4183 - accuracy: 0.
8159 - val_loss: 0.4513 - val_accuracy: 0.8058
Epoch 19/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4166 - accuracy: 0.
```

```
8173 - val_loss: 0.4507 - val_accuracy: 0.8058
Epoch 20/20
215/215 [==============================] - 1s 5ms/step - loss: 0.4151 - accuracy: 0.
8161 - val_loss: 0.4501 - val_accuracy: 0.8045
```

Out[13]:
```
<keras.callbacks.History at 0x2d294850af0>
```

In [14]:
```python
calculate_results(y_true=y_test,
                  y_pred=tf.squeeze(tf.round(model.predict(X_test))))
```

```
24/24 [==============================] - 0s 5ms/step
```

Out[14]:
```
{'accuracy': 0.8044619422572179,
 'precision': 0.804799099712002,
 'recall': 0.8044619422572179,
 'f1': 0.8036035500282784}
```

📈 Conclusion

The final model trained with an LSTM (Long Short-Term Memory) architecture achieved promising performance, effectively learning to classify disaster-related messages from noisy real-world text. The training process demonstrated a stable convergence, and the model was evaluated with standard classification metrics to assess its predictive strength.

While simple, this RNN-based model provides a solid foundation for more sophisticated architectures such as Bidirectional RNNs, GRUs, or Transformer-based models (e.g., BERT). Future improvements could also include using pre-trained embeddings like GloVe or FastText, experimenting with attention mechanisms, and expanding the dataset for better generalization.

Overall, this project demonstrates the feasibility of applying deep learning to binary text classification tasks and reinforces the usefulness of RNNs for processing sequential textual data.