In [68]:
```python
import numpy as np
import pandas as pd
import tensorflow as tf

import re
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

import matplotlib.pyplot as plt
```

In [69]:
```python
pd.set_option('display.max_colwidth', -1)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: Future
Warning: Passing a negative integer is deprecated in version 1.0 and w
ill not be supported in future version. Instead, use None to not limit
the column width.
  """Entry point for launching an IPython kernel.
```

Load Data

Now I'll load the training dataset.

In [70]:
```python
train_data = pd.read_csv(
    '/kaggle/input/nlp-getting-started/train.csv',
    usecols=['text', 'target'],
    dtype={'text': str, 'target': np.int64}
)

len(train_data)
```

Out[70]:
```
7613
```

In [71]:
```python
train_data['text'].head().values
```

Out[71]:
```
array(['Our Deeds are the Reason of this #earthquake May ALLAH Forgive
us all',
       'Forest fire near La Ronge Sask. Canada',
       "All residents asked to 'shelter in place' are being notified b
y officers. No other evacuation or shelter in place orders are expecte
d",
       '13,000 people receive #wildfires evacuation orders in Californ
ia ',
       'Just got sent this photo from Ruby #Alaska as smoke from #wild
fires pours into a school '],
      dtype=object)
```

And load the test dataset for later.

In [72]:

```
test_data = pd.read_csv(
    '/kaggle/input/nlp-getting-started/test.csv',
    usecols=['text', 'id'],
    dtype={'text': str, 'id': str}
)
```

Mislabelled examples

There are a number of examples in the training dataset that are mislabelled. The keyword can be used to find these.

Thanks to Dmitri Kalyaevs whose notebook is where I found to do this: https://www.kaggle.com/dmitri9149/transformer-svm-semantically-identical-tweets (https://www.kaggle.com/dmitri9149/transformer-svm-semantically-identical-tweets)

In [73]:

```python
indices = [4415, 4400, 4399,4403,4397,4396, 4394,4414, 4393,4392,4404,4
train_data.loc[indices]
```

Out[73]:

	text	target
4415	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/xV3D9bPjHi #prebreak #best	1
4400	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/aAtt5aMnmD #prebreak #best	0
4399	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/nQiObcZKrT #prebreak #best	0
4403	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/qj3PVgaVN7 #prebreak #best	1
4397	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/J5onxFwLAo #prebreak #best	0
4396	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/wvTPuRYx63 #prebreak #best	0
4394	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/cx6auPneMu #prebreak #best	0
4414	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/cOMuiOk3mP #prebreak #best	1
4393	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/s4PNIhJQX7 #prebreak #best	0
4392	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/J2aQs5loxu #prebreak #best	1
4404	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/6AqrNanKFD #prebreak #best	0
4407	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/gexHzU1VK8 #prebreak #best	0
4420	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/UMgD92wLjA #prebreak #best	1
4412	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/G62txymzBv #prebreak #best	0
4408	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/MIs0RjxuIr #prebreak #best	0
4391	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/gUJNPLJVvt #prebreak #best	0
4405	#hot Funtenna: hijacking computers to send data as sound waves [Black Hat 2015] http://t.co/8JcYXhq1AZ #prebreak #best	0

In [74]:

```python
train_data.loc[indices, 'target'] = 0
```

In [75]:

```python
indices = [6840,6834,6837,6841,6816,6828,6831]
train_data.loc[indices]
```

Out[75]:

	text	target
6840	Hollywood Movie About Trapped Miners Released in Chile: 'The 33' Hollywood movie about trapped miners starring... http://t.co/0f8XA4Ih1U	1
6834	Hollywood Movie About Trapped Miners Released in Chile: 'The 33' Hollywood movie about trapped miners starring... http://t.co/x8moYeVjsJ	0
6837	Hollywood Movie About Trapped Miners Released in Chile: 'The 33' Hollywood movie about trapped miners starring... http://t.co/tyyfG4qQvM	1
6841	Hollywood Movie About Trapped Miners Released in Chile: 'The 33' Hollywood movie about trapped miners starring... http://t.co/3Yu26V19zh	0
6816	Hollywood Movie About Trapped Miners Released in Chile: 'The 33' Hollywood movie about trapped miners starring... http://t.co/KK8cnppZMk	0
6828	Hollywood Movie About Trapped Miners Released in Chile http://t.co/EXQKmlg4NJ	0
6831	Hollywood Movie About Trapped Miners Released in Chile http://t.co/qkrLtrd39B	1

In [76]:

```python
train_data.loc[indices, 'target'] = 0
```

In [77]:

```
indices = [601,576,584,608,606,603,592,604,591, 587]
train_data.loc[indices]
```

Out[77]:

	text	target
601	FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/c0p3SEsqWm via @usatoday	1
576	FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/HQsU8LWltH via @usatoday	1
584	FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/qZQc8WWwcN via @usatoday	0
608	FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/4M5UHeyfDo via @USATODAY	1
606	FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/hrqCJdovJZ	0
603	FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/MqbYrAvK6h	1
592	FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/pWAMG8oZj4	1
604	#FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/S4SiCMYRmH	1
591	#world FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/5zDbTktwW7	0
587	#world FedEx no longer to transport bioterror germs in wake of anthrax lab mishaps http://t.co/wvExJjRG6E	1

In [78]:

```
train_data.loc[indices, 'target'] = 1
```

In [79]:

```python
indices = [3913,3914,3936,3921,3941,3937,3938,3136,3133,3930,3933,3924,
train_data.loc[indices]
```

Out[79]:

	text	target
3913	Spot Flood Combo 53inch 300W Curved Cree LED Work Light Bar 4X4 Offroad Fog Lamp - Full re□Û_ http://t.co/xxkHjySn0p http://t.co/JEVHKNJGBX	1
3914	Spot Flood Combo 53inch 300W Curved Cree LED Work Light Bar 4X4 Offroad Fog Lamp - Full re□Û_ http://t.co/jCDd6SD6Qn http://t.co/9gUCkjghms	0
3936	Spot Flood Combo 53inch 300W Curved Cree LED Work Light Bar 4X4 Offroad Fog Lamp - Full re□Û_ http://t.co/5xmCE6JufS http://t.co/3Zo7PX3p1V	0
3921	Spot Flood Combo 53inch 300W Curved Cree LED Work Light Bar 4X4 Offroad Fog Lamp - Full re□Û_ http://t.co/mTmola0Oo0 http://t.co/Nn4ZtCmSRU	0
3941	Spot Flood Combo 53inch 300W Curved Cree LED Work Light Bar 4X4 Offroad Fog Lamp - Full re□Û_ http://t.co/6zCfHi7Srw http://t.co/vWYkDaU1vm	0
3937	Spot Flood Combo 53inch 300W Curved Cree LED Work Light Bar 4X4 Offroad Fog Lamp - Full re□Û_ http://t.co/O097vSOtxk http://t.co/I23Xy7iEjj	0
3938	Spot Flood Combo 53inch 300W Curved Cree LED Work Light Bar 4X4 Offroad Fog Lamp - Full re□Û_ http://t.co/fDSaoOiskJ http://t.co/2uVmq4vAfQ	0
3136	Survival Kit Whistle Fire Starter Wire Saw Cree Torch Emergency Blanket S knife - Full re□Û_ http://t.co/2OroYUNYM2 http://t.co/C9JnXz3DXC	0
3133	Survival Kit Whistle Fire Starter Wire Saw Cree Torch Emergency Blanket S knife - Full re□Û_ http://t.co/cm7HqwWUlZ http://t.co/KdwAzHQTov	1
3930	2pcs 18W CREE Led Work Light Offroad Lamp Car Truck Boat Mining 4WD FLOOD BEAM - Full rea□Û_ http://t.co/O1SMUh2unn http://t.co/xqj6WgiuQH	0
3933	2pcs 18W CREE Led Work Light Offroad Lamp Car Truck Boat Mining 4WD FLOOD BEAM - Full rea□Û_ http://t.co/1QT51r5h98 http://t.co/OQH1JbUEnl	0
3924	2pcs 18W CREE Led Work Light Offroad Lamp Car Truck Boat Mining 4WD FLOOD BEAM - Full rea□Û_ http://t.co/ApWXS5Mm44 http://t.co/DS76loZLSu	1
3917	2pcs 18W CREE Led Work Light Offroad Lamp Car Truck Boat Mining 4WD FLOOD BEAM - Full rea□Û_ http://t.co/VDeFmulx43 http://t.co/yqpAljSa5g	0

In [80]:

```python
train_data.loc[indices, 'target'] = 0
```

In [81]:

```
indices = [246,270,266,259,253,251,250,271]
train_data.loc[indices]
```

Out[81]:

	text	target
246	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... http://t.co/6LoJOoROuk via @Change	0
270	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... https://t.co/0fekgyBY5F via @Change	0
266	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... https://t.co/x2Wn7O2a3w via @Change	0
259	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... https://t.co/MatIJwkzbh via @Change	0
253	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... http://t.co/KPQk0C4G0M via @Change	1
251	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... https://t.co/sW1sBua3mN via @Change	1
250	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... https://t.co/m8MvDSPJp7 via @Change	0
271	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... http://t.co/SB5R7ShcCJ via @Change	1

In [82]:

```
train_data.loc[indices, 'target'] = 0
```

In [83]:

```python
indices = [6119,6122,6123,6131,6160,6166,6167,6172,6212,6221,6230,6091,
train_data.loc[indices]
```

Out[83]:

	text	target
6119	That horrible sinking feeling when you□Ûªve been at home on your phone for a while and you realise its been on 3G this whole time	1
6122	The #Tribe just keeps sinking everyday it seems faster. As for this year it's been a titanic disaster.	1
6123	that horrible sinking feeling when you□Ûªve been at home on your phone for a while and you realise its been on 3G this whole time	1
6131	'Amateur Night' Actress Reprises Role for 'Siren' - HorrorMovies.ca #horror http://t.co/W9Cd6OFfcj	1
6160	@SoonerMagic_ I mean I'm a fan but I don't need a girl sounding off like a damn siren	1
6166	Reasons I should have gone to Warped today: tony played issues showed up sleeping w sirens played attila is there issues issues issues	1
6167	Thu Aug 06 2015 01:20:32 GMT+0000 (UTC)\n#millcityio #20150613\ntheramin sirens	1
6172	my dad said I look thinner than usual but really im over here like http://t.co/bnwyGx6luh	1
6212	@PianoHands You don't know because you don't smoke. The way to make taxis and buses come is to light a cigarette to smoke while you wait.	1
6221	I get to smoke my shit in peace	1
6230	Manuel hoping for an early Buffalo snowstorm so his accuracy improves.	1
6091	that horrible sinking feeling when you□Ûªve been at home on your phone for a while and you realise its been on 3G this whole time	1
6108	Do you feel like you are sinking in low self-image? Take the quiz: http://t.co/bJoJVM0pjX http://t.co/wHOc7LHb5F	1

In [84]:

```python
train_data.loc[indices, 'target'] = 0
```

In [85]:
```python
indices = [7435,7460,7464,7466,7469,7475,7489,7495,7500,7525,7552,7572,
train_data.loc[indices]
```

Out[85]:

	text	target
7435	Gunshot wound #9 is in the bicep. The only one of the ten wounds that is not in the chest/torso area. #KerrickTrial #JonathanFerrell	1
7460	@IcyMagistrate ÛÓher upper armÛÒ those /friggin/ icicle projectilesÛÒ and leg from various other wounds the girl looks like a miniature moreÛÓ	1
7464	Crawling in my skin\nThese wounds they will not hea	1
7466	Sorrower - Fresh Wounds Over Old Scars (2015 Death Metal) http://t.co/L056yj2IOi http://t.co/uTMWMjiRty	1
7469	@Captainn_Morgan car wreck ??	1
7475	Watertown Gazette owner charged in wreck http://t.co/JHc2RT0V9F	1
7489	@GeorgeFoster72 and The Wreck of the Edmund Fitzgerald	1
7495	Greer man dies in wreck http://t.co/n2qZbMZuly	1
7500	Omg if Cain dies i will be an emotional wreck #emmerdale	1
7525	The first piece of wreckage from the first-ever lost Boeing 777 which vanished back in early March along with the 239 people on board has	1
7552	Israel wrecked my home. Now it wants my land. \nhttps://t.co/g0r3ZR1nQj	1
7572	@Kirafrog @mount_wario Did you get wrecked again?	1
7591	Heat wave warning aa? Ayyo dei. Just when I plan to visit friends after a year.	1
7599	1.3 #Earthquake in 9Km Ssw Of Anza California #iPhone users download the Earthquake app for more information http://t.co/V3aZWOAmzK	1

In [86]:
```python
train_data.loc[indices, 'target'] = 0
```

Split training dataset

To see if the model overfits the data during training I will take a slice of the training data as a validation dataset.

In [87]:
```python
val_data = train_data.tail(1500)
train_data = train_data.head(6113)
```

Clean text

As with all datasets, text-based data needs a bit of cleaning too. Some common cleaning steps are:

Removing Noise: Remove elements that hold little meaning, such as URLs and HTML tags in the text. Punctuation is also usually removed at this stage, although the TensorFlow tokenizer I use later does this by default, so I leave out the logic at this step.

Remove Stopwords: Certain words like "a," "the," and "are" are very common and hold little meaning for a sentence. Removing them speeds up training and helps with accuracy.

Stemming or Lemmatization: Many words are derived from a root or stem word. For example, words like "working" and "worked" stem from the word "work." Reverting all words to their stem can help in some tasks, though for this task, I found that it made very little difference.

In [88]:

```
def remove_url(sentence):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'', sentence)
```

In [89]:

```
def remove_at(sentence):
    url = re.compile(r'@\S+')
    return url.sub(r'', sentence)
```

In [90]:

```
def remove_html(sentence):
    html = re.compile(r'<.*?>')
    return html.sub(r'', sentence)
```

In [91]:

```
def remove_emoji(sentence):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictog
                           u"\U0001F680-\U0001F6FF"  # transport & map 
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)

    return emoji_pattern.sub(r'', sentence)
```

In [92]:

```
def remove_stopwords(sentence):
    words = sentence.split()
    words = [word for word in words if word not in stopwords.words('eng

    return ' '.join(words)
```

In [93]:
```
stemmer = SnowballStemmer('english')

def stem_words(sentence):
    words = sentence.split()
    words = [stemmer.stem(word) for word in words ]

    return ' '.join(words)
```

For speed I have wrapped all of these cleaning functions into one. This is applied to all three datasets.

In [94]:
```
def clean_text(data):
    data['text'] = data['text'].apply(lambda x : remove_url(x))
    data['text'] = data['text'].apply(lambda x : remove_at(x))
    data['text'] = data['text'].apply(lambda x : remove_html(x))
    data['text'] = data['text'].apply(lambda x : remove_emoji(x))
    data['text'] = data['text'].apply(lambda x : remove_stopwords(x))
    data['text'] = data['text'].apply(lambda x : stem_words(x))

    return data
```

In [95]:
```
train_data = clean_text(train_data)
val_data = clean_text(val_data)
test_data = clean_text(test_data)

train_data['text'].head().values
```

Out[95]:
```
array(['our deed reason #earthquak may allah forgiv us',
       'forest fire near la rong sask. canada',
       'all resid ask shelter place notifi officers. no evacu shelter
place order expect',
       '13,000 peopl receiv #wildfir evacu order california',
       'just got sent photo rubi #alaska smoke #wildfir pour school'],
      dtype=object)
```

Encode sentences

When working with text data, machine learning models need a way to understand and process sentences. Instead of working with raw text, we convert sentences into arrays of numbers, where each number represents a word in the sentence. This process is crucial for training models effectively. To achieve this, we use a tokenizer.

A tokenizer essentially assigns a unique number (an index) to each word in the text. Think of it as creating a dictionary where every word has a corresponding number. This way, the model can treat words as categorical values with numerical representations. TensorFlow provides a built-in tokenizer for this purpose.

To set up this tokenizer, we gather all the sentences from our datasets and use them to build a vocabulary. This vocabulary contains all the unique words present in the text data. Combining data from all three datasets ensures that our tokenizer knows all the words used in the tweets.

Once we have our tokenizer set up, we can use it to encode sentences. Encoding means replacing each word in a sentence with its corresponding index in the vocabulary. Additionally, we make sure all the encoded sentences are of the same length, typically equal to the length of the longest sentence in the training dataset. To achieve this, we pad shorter sentences with zeros. This padding ensures consistent input size for the model, and

In [96]:
```
def define_tokenizer(train_sentences, val_sentences, test_sentences):
    sentences = pd.concat([train_sentences, val_sentences, test_sentenc

    tokenizer = tf.keras.preprocessing.text.Tokenizer()
    tokenizer.fit_on_texts(sentences)

    return tokenizer

def encode(sentences, tokenizer):
    encoded_sentences = tokenizer.texts_to_sequences(sentences)
    encoded_sentences = tf.keras.preprocessing.sequence.pad_sequences(e

    return encoded_sentences
```

In [97]:
```
tokenizer = define_tokenizer(train_data['text'], val_data['text'], test_

encoded_sentences = encode(train_data['text'], tokenizer)
val_encoded_sentences = encode(val_data['text'], tokenizer)
encoded_test_sentences = encode(test_data['text'], tokenizer)
```

The tokeniser provides some interesting information about the sentences it encodes. To get the index number assigned to a word I can look up the word in the tokenizers word index (which is just a python dict with the words as keys and the index numbers as values).

In [98]:
```
tokenizer.word_index['disaster']
```

Out[98]:
```
740
```

The word index can also be used to find out how many words are in the vocabulary.

In [99]:
```
len(tokenizer.word_index)
```

Out[99]:
```
16551
```

As well as some configurations that are used to tokenize sentences such as whether the tokenizer changes all characters to lowercase, what split it performs to get the words from the sentences and what characters it filters out.

In [100]:
```
print('Lower: ', tokenizer.get_config()['lower'])
print('Split: ', tokenizer.get_config()['split'])
print('Filters: ', tokenizer.get_config()['filters'])
```

```
Lower:  True
Split:
Filters:  !"#$%&()*+,-./:;<=>?@[\]^_`{|}~
```

Import GloVe Embedding

While I could train my own word embedding for the model it might help to use a pre-trained word embedding. This enables me to take advantage of an embedding that has ungone more rogourous training. Additionally it will also include words that I may not have in my training dataset (but may appear in the test dataset) which helps with overfitting.

The first thing to do then is to load the embedding. I'll use GloVe for this task.

In [101]:
```
embedding_dict = {}

with open('../input/glove-global-vectors-for-word-representation/glove.(
    for line in f:
        values = line.split()
        word = values[0]
        vectors = np.asarray(values[1:],'float32')
        embedding_dict[word] = vectors

f.close()
```

To ensure the encoding of the tokenizer and the embeddings are synchronized I use the below function to update the encoded words in the embedding with the encoding from the tokenizer.

In [102]:
```
num_words = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((num_words, 100))

for word, i in tokenizer.word_index.items():
    if i > num_words:
        continue

    emb_vec = embedding_dict.get(word)

    if emb_vec is not None:
        embedding_matrix[i] = emb_vec
```

Define pipeline

With the sentences encoded they can now be prepared to be fed into the model. Tensorflow provides an api to format data in its own format. While data can be inserted in a more common format (such as numpy arrays), tensorflow seems to prefer its own format and provides a few handy bits of functionality as incentives.

Firstly then I will convert the encoded sentences and labels into tensors.

In [103]:
```
tf_data = tf.data.Dataset.from_tensor_slices((encoded_sentences, train_
```

Now the data is in the tensorflow format a few handy methods can be added to improve the training. This includes shuffling the data per training step, processing the next batch of data for training while the current batch of data is training and defining each batch as a padded batch.

In [104]:
```
def pipeline(tf_data, buffer_size=100, batch_size=32):
    tf_data = tf_data.shuffle(buffer_size)
    tf_data = tf_data.prefetch(tf.data.experimental.AUTOTUNE)
    tf_data = tf_data.padded_batch(batch_size, padded_shapes=([None],[]

    return tf_data

tf_data = pipeline(tf_data, buffer_size=1000, batch_size=32)
```

In [105]:
```
print(tf_data)
```

```
<PaddedBatchDataset shapes: ((None, None), (None,)), types: (tf.int32,
tf.int64)>
```

A similar pipeline is defined for the validation dataset. The difference is the lack of shuffling to speed up the validation.

In [106]:
```
tf_val_data = tf.data.Dataset.from_tensor_slices((val_encoded_sentences
```

In [107]:
```
def val_pipeline(tf_data, batch_size=1):
    tf_data = tf_data.prefetch(tf.data.experimental.AUTOTUNE)
    tf_data = tf_data.padded_batch(batch_size, padded_shapes=([None],[]

    return tf_data

tf_val_data = val_pipeline(tf_val_data, batch_size=len(val_data))
```

In [108]:
```
print(tf_val_data)
```

```
<PaddedBatchDataset shapes: ((None, None), (None,)), types: (tf.int32,
tf.int64)>
```

Train Model

To build our model, we start by defining its key components:

Embedding Layer: This layer is essential for the model to understand the meaning of words. It creates a set of abstract features for each word in the dataset. These features can represent various aspects of a word, such as its gender or sentiment. It also helps the model recognize relationships between words. Think of it as a way for the model to convert words into numerical representations that capture their essence.

RNN Layer: RNN stands for Recurrent Neural Network, which is a type of layer used in deep learning. It's a bit complex to dive into all the details here, but in simple terms, RNNs are excellent at handling sequences of data, like sentences. TensorFlow offers different types of RNN layers, including simple RNN, GRU (Gated Recurrent Unit), and LSTM (Long Short-Term Memory). Among these, LSTM has been found to work best for this task. You can think of this layer as the part of the model that understands the context and relationships between words in a sentence.

Dense Layer: This is the final layer of our model. It takes the output from the LSTM layer and makes a decision about the sentence. In this case, it assigns a class to the sentence: 1 if the sentence is indicating a real disaster and 0 if it's not. Essentially, it's the part of the model that decides whether a tweet is talking about a real emergency or not.

In [109]:
```
embedding = tf.keras.layers.Embedding(
    len(tokenizer.word_index) + 1,
    100,
    embeddings_initializer = tf.keras.initializers.Constant(embedding_m
    trainable = True
)
```

In [110]:
```
model = tf.keras.Sequential([
    embedding,
    tf.keras.layers.SpatialDropout1D(0.2),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, dropout=0.2
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Then compile the model defining the training function (adam) and the loss function (log loss). I have also added a metrics parameter so that the models accuracy is printed per epoch.

In [111]:
```
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(0.0001),
    metrics=['accuracy', 'Precision', 'Recall']
)
```

To avoid the model stepping over the optimum I'll add learning rate decay logic to reduce the learning rate if the loss plateaus for two or more epochs. I'll also add early stopping if loss hasn't fallen for five epochs. This saves some processing.

In [112]:
```
callbacks = [
    tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', patience=2, ve
    tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5, verbos
]
```

Finally begin training the model.

In [113]:

```
history = model.fit(
    tf_data,
    validation_data = tf_val_data,
    epochs = 20,
    callbacks = callbacks
)
```

```
Train for 192 steps, validate for 1 steps
Epoch 1/20
192/192 [==============================] - 21s 108ms/step - loss: 0.70
71 - accuracy: 0.5956 - Precision: 0.5981 - Recall: 0.1002 - val_loss:
0.6335 - val_accuracy: 0.7707 - val_Precision: 0.8810 - val_Recall: 0.
5687
Epoch 2/20
192/192 [==============================] - 15s 80ms/step - loss: 0.653
9 - accuracy: 0.7273 - Precision: 0.7709 - Recall: 0.4949 - val_loss:
0.6175 - val_accuracy: 0.7833 - val_Precision: 0.8877 - val_Recall: 0.
5953
Epoch 3/20
192/192 [==============================] - 16s 83ms/step - loss: 0.644
2 - accuracy: 0.7491 - Precision: 0.8082 - Recall: 0.5243 - val_loss:
0.6117 - val_accuracy: 0.7947 - val_Precision: 0.8697 - val_Recall: 0.
6411
Epoch 4/20
192/192 [==============================] - 16s 83ms/step - loss: 0.638
3 - accuracy: 0.7587 - Precision: 0.8242 - Recall: 0.5376 - val_loss:
0.6112 - val_accuracy: 0.7920 - val_Precision: 0.8600 - val_Recall: 0.
6440
Epoch 5/20
192/192 [==============================] - 16s 81ms/step - loss: 0.634
2 - accuracy: 0.7662 - Precision: 0.8356 - Recall: 0.5489 - val_loss:
0.6106 - val_accuracy: 0.7887 - val_Precision: 0.8629 - val_Recall: 0.
6322
Epoch 6/20
192/192 [==============================] - 16s 81ms/step - loss: 0.632
6 - accuracy: 0.7690 - Precision: 0.8405 - Recall: 0.5524 - val_loss:
0.6073 - val_accuracy: 0.7980 - val_Precision: 0.8848 - val_Recall: 0.
6352
Epoch 7/20
192/192 [==============================] - 16s 83ms/step - loss: 0.631
4 - accuracy: 0.7726 - Precision: 0.8667 - Recall: 0.5391 - val_loss:
0.6069 - val_accuracy: 0.8007 - val_Precision: 0.8677 - val_Recall: 0.
6588
Epoch 8/20
192/192 [==============================] - 16s 83ms/step - loss: 0.628
5 - accuracy: 0.7787 - Precision: 0.8613 - Recall: 0.5610 - val_loss:
0.6067 - val_accuracy: 0.8060 - val_Precision: 0.8712 - val_Recall: 0.
6691
Epoch 9/20
192/192 [==============================] - 16s 84ms/step - loss: 0.624
8 - accuracy: 0.7859 - Precision: 0.8553 - Recall: 0.5872 - val_loss:
0.6044 - val_accuracy: 0.8047 - val_Precision: 0.8934 - val_Recall: 0.
6440
Epoch 10/20
192/192 [==============================] - 16s 83ms/step - loss: 0.621
5 - accuracy: 0.7918 - Precision: 0.8723 - Recall: 0.5880 - val_loss:
0.6047 - val_accuracy: 0.8087 - val_Precision: 0.8721 - val_Recall: 0.
6750
Epoch 11/20
192/192 [==============================] - 16s 85ms/step - loss: 0.621
4 - accuracy: 0.7934 - Precision: 0.8731 - Recall: 0.5919 - val_loss:
0.6033 - val_accuracy: 0.8067 - val_Precision: 0.8757 - val_Recall: 0.
6662
Epoch 12/20
```

```
192/192 [==============================] - 16s 81ms/step - loss: 0.617
7 - accuracy: 0.8027 - Precision: 0.8862 - Recall: 0.6060 - val_loss:
0.6034 - val_accuracy: 0.8113 - val_Precision: 0.8774 - val_Recall: 0.
6765
Epoch 13/20
192/192 [==============================] - 16s 85ms/step - loss: 0.617
0 - accuracy: 0.8040 - Precision: 0.8823 - Recall: 0.6131 - val_loss:
0.6028 - val_accuracy: 0.8087 - val_Precision: 0.8963 - val_Recall: 0.
6514
Epoch 14/20
192/192 [==============================] - 16s 86ms/step - loss: 0.616
6 - accuracy: 0.8065 - Precision: 0.8762 - Recall: 0.6256 - val_loss:
0.6043 - val_accuracy: 0.8147 - val_Precision: 0.8634 - val_Recall: 0.
7001
Epoch 15/20
192/192 [==============================] - 17s 86ms/step - loss: 0.613
4 - accuracy: 0.8068 - Precision: 0.8873 - Recall: 0.6162 - val_loss:
0.6015 - val_accuracy: 0.8147 - val_Precision: 0.8829 - val_Recall: 0.
6795
Epoch 16/20
192/192 [==============================] - 16s 84ms/step - loss: 0.614
5 - accuracy: 0.8150 - Precision: 0.8943 - Recall: 0.6322 - val_loss:
0.6011 - val_accuracy: 0.8187 - val_Precision: 0.8828 - val_Recall: 0.
6898
Epoch 17/20
192/192 [==============================] - 16s 84ms/step - loss: 0.612
5 - accuracy: 0.8106 - Precision: 0.8909 - Recall: 0.6232 - val_loss:
0.6004 - val_accuracy: 0.8187 - val_Precision: 0.8887 - val_Recall: 0.
6839
Epoch 18/20
192/192 [==============================] - 16s 82ms/step - loss: 0.611
3 - accuracy: 0.8194 - Precision: 0.8959 - Recall: 0.6428 - val_loss:
0.6002 - val_accuracy: 0.8207 - val_Precision: 0.8878 - val_Recall: 0.
6898
Epoch 19/20
192/192 [==============================] - 16s 84ms/step - loss: 0.606
9 - accuracy: 0.8245 - Precision: 0.9041 - Recall: 0.6491 - val_loss:
0.6053 - val_accuracy: 0.8187 - val_Precision: 0.8450 - val_Recall: 0.
7326
Epoch 20/20
192/192 [==============================] - 15s 81ms/step - loss: 0.611
3 - accuracy: 0.8240 - Precision: 0.9031 - Recall: 0.6487 - val_loss:
0.6027 - val_accuracy: 0.8213 - val_Precision: 0.8607 - val_Recall: 0.
7208
```

Evaluate

Let's take a look at the models output to get an idea how it did. The quickest and easiest evaluation method is to take a look at the metrics produced by the model. The final metrics can be extracted using the evaluate method. Since this competition uses an F1 score to rank submissions it may be worth having a look at it on the training dataset.

In [114]:

```
metrics = model.evaluate(tf_val_data)

precision = metrics[2]
recall = metrics[3]
f1 = 2 * (precision * recall) / (precision + recall)

print('F1 score: ' + str(f1))
```

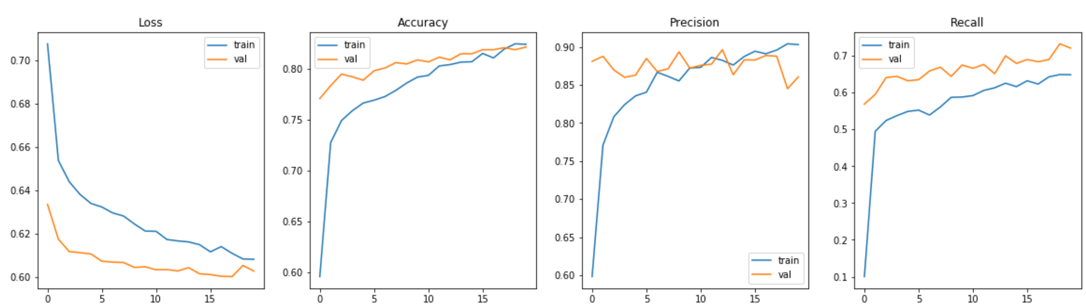
```
1/1 [==============================] - 0s 171ms/step - loss: 0.6027 -
accuracy: 0.8213 - Precision: 0.8607 - Recall: 0.7208
F1 score: 0.7845658882836413
```

Additionally the metrics produced per epoch when the model was training can be visualised to get a better idea for how the training went.

In [115]:

```
fig, axs = plt.subplots(1, 4, figsize=(20, 5))

axs[0].set_title('Loss')
axs[0].plot(history.history['loss'], label='train')
axs[0].plot(history.history['val_loss'], label='val')
axs[0].legend()

axs[1].set_title('Accuracy')
axs[1].plot(history.history['accuracy'], label='train')
axs[1].plot(history.history['val_accuracy'], label='val')
axs[1].legend()

axs[2].set_title('Precision')
axs[2].plot(history.history['Precision'], label='train')
axs[2].plot(history.history['val_Precision'], label='val')
axs[2].legend()

axs[3].set_title('Recall')
axs[3].plot(history.history['Recall'], label='train')
axs[3].plot(history.history['val_Recall'], label='val')
axs[3].legend()
```

Out[115]: <matplotlib.legend.Legend at 0x7f7a3a93be10>



It is also worth having a look at which sentences the model got wrong. To do this the model needs to produce predictions for the training dataset. This involves a slightly different pipeline.

In [116]:

```
predictions = model.predict(tf_val_data)
predictions = np.concatenate(predictions).round().astype(int)

val_data['predictions'] = predictions
```

First take a look at the false postives (when the model thought there was a disaster in the tweet but there was not).

In [117]:

```
false_positives = val_data[(val_data['predictions'] == 1) & (val_data['

print('Count of false positives: ' + str(len(false_positives)))
```

```
Count of false positives: 79
```

In [118]:

```
false_positives.head(10)
```

Out[118]:

	text	target	predictions
6159	outdoor siren test 2pm :: the fgcu siren test 2pm today. anoth messag sent test concluded.	0	1
6167	thu aug 06 2015 01:20:32 gmt+0000 (utc) #millcityio #20150613 theramin siren	0	1
6189	a rocket to the moon ? sleep with siren ?a rocket to the moon ????????????	0	1
6213	[55436] 1950 lionel train smoke locomot with magne-tract instruct	0	1
6230	manuel hope earli buffalo snowstorm accuraci improves.	0	1
6240	sassi citi girl countri hunk strand smoki mountain snowstorm #aom #ibooklov #bookboost	0	1
6245	cooler freddi jackson sippin milkshak snowstorm	0	1
6257	snowstorm plan outsid #rome st mari major tonight - annual occas artifici snow rememb summer snow 358 ad spot.	0	1
6258	final storm	0	1
6267	rt the person danc rain like walk storm. -anonym	0	1

And then do the same with the false negatives (when the model didn't think there was a disaster in a tweet when in fact there was).

In [119]:
```
false_negatives = val_data[(val_data['predictions'] == 0) & (val_data['

print('Count of false negatives: ' + str(len(false_negatives)))
```

```
Count of false negatives: 189
```

In [120]:
```
false_positives.tail(10)
```

Out[120]:

	text	target	predictions
7330	we fire safeti plan. rt mock wildfir near #vail agenc prepar worst.	0	1
7386	new roof hardi up..windstorm inspect tomorrow	0	1
7391	twia board approv 5 percent rate hike: the texa windstorm insur associ (twia) board director v...	0	1
7412	have ever seen presid kill wound child? or man crash sister plane claimin sent god?	0	1
7442	rt gunshot wound #9 bicep. 1 10 wound chest/torso area. #kerricktri #jonathanferrel	0	1
7459	court back session. testimoni continu med. examin discuss gunshot wound #kerricktri	0	1
7472	wreck? wreck wreck wreck wreck wreck wreck wreck wreck wreck wreck wreck wreck?	0	1
7487	my emot train wreck. my bodi train wreck. i'm wreck	0	1
7503	amazon prime day: 12 quick takeaway amazon□ûª magnific train wreck -	0	1
7525	the first piec wreckag first-ev lost boe 777 vanish back earli march along 239 peopl board	0	1

Submission

With the model trained it now takes a few more steps to load the test data and use the model to label the test sentences as either disaster or no disaster. First convert the data to a tensorflow dataset and apply the pipeline methods. The pipeline has been adjusted slightly to account for not wanting any shuffling and the different shape of the input (no label).

In [121]:
```
tf_test_data = tf.data.Dataset.from_tensor_slices((encoded_test_sentenc
```

In [122]:

```python
def test_pipeline(tf_data, batch_size=1):
    tf_data = tf_data.prefetch(tf.data.experimental.AUTOTUNE)
    tf_data = tf_data.padded_batch(batch_size, padded_shapes=([None]))

    return tf_data

tf_test_data = test_pipeline(tf_test_data)
```

Then use the model to apply labels to the test data.

In [123]:

```python
predictions = model.predict(tf_test_data)
```

The model outputs a probability per sentence. The easy way to set a threshold of 0.5 (i.e. if the probability is less than 0.5 set the label to 0 and vice versa) is to use the round method.

In [124]:

```python
predictions = np.concatenate(predictions).round().astype(int)
```

Write the submission to a csv file.

In [125]:

```python
submission = pd.DataFrame(data={'target': predictions}, index=test_data
submission.index = submission.index.rename('id')
submission.to_csv('submission.csv')
```

In [126]:

```python
submission.head()
```

Out[126]:

id	target
0	0
2	0
3	1
9	0
11	1

Appendix

Word mismatch

Earlier in the notebook I mentioned that the training, validation and test datasets are likely to contain words that the other datasets do not. If the model is only trained on the words in the training dataset there may be an overfitting problem when the model tries to read words it doesn't recognise in the validation and the test datasets.

The below function takes two datasets and counts how words are matching and not matching to see how severe the issue is.

In [127]:

```
def compare_words(train_words, test_words):
    unique_words = len(np.union1d(train_words, test_words))
    matching = len(np.intersect1d(train_words, test_words))
    not_in_train = len(np.setdiff1d(test_words, train_words))
    not_in_test = len(np.setdiff1d(train_words, test_words))

    print('Count of unique words in both arrays: ' + str(unique_words))
    print('Count of matching words: ' + str(matching))
    print('Count of words in first array but not in second: ' + str(not_
    print('Count of words in second array but not first: ' + str(not_in_
```

In [128]:

```
compare_words(encoded_sentences, val_encoded_sentences)
```

```
Count of unique words in both arrays: 13463
Count of matching words: 2799
Count of words in first array but not in second: 9006
Count of words in second array but not first: 1658
```

In [129]:

```
compare_words(encoded_sentences, encoded_test_sentences)
```

```
Count of unique words in both arrays: 15231
Count of matching words: 4856
Count of words in first array but not in second: 6949
Count of words in second array but not first: 3426
```