Report

April 22, 2024

1 *Youtube Comment Spam Detection - Report*

1.1 Name: Chau Ho San

1.2 EID: 57150406

1.2.1 Background

The project revolves around the prevalent issue of spam comments on YouTube videos, which significantly diminishes the user experience and authenticity of content. These spam comments, often promotional or irrelevant in nature, inundate the comment sections, making it arduous for users to discover meaningful discussions or engage with content creators. This project aims to tackle this problem by developing a classifier capable of discerning between spam and genuine comments. The ultimate goal is to create a dependable tool that can effectively identify spam comments, thereby enhancing the quality of user interactions on YouTube and fostering a more positive and engaging online environment.

1.2.2 Method

Model Architecture and Components In this project, the core of the methodology is based on the implementation of a **Long Short-Term Memory (LSTM) neural network**. I chose this model because it is exceptionally well-suited for processing sequential data, such as the text found in YouTube comments. Due to its ability, I can use it to capture temporal dependencies and contextual nuances within the data. The architecture of the LSTM model is meticulously designed and also can add several layers to the model which we need to optimize the model's performance:

Detailed Description of Model Layers

1. **Embedding Layer**:
   - **Purpose**:This layer is used as an entry point for the input text data and transforms the text into a dense vector representation. Its main purpose is to transform sparse high-dimensional word indexes into a low-dimensional continuous vector space, thus enabling semantically similar words to be close to each other.
   - **Details**. The vectors are learned during the training process and become more refined to reflect the nuances of the language used in YouTube comments.
2. **Batch Normalization Layer**:
   - **Purpose**: Its main function is to stabilise the learning process by normalising the batch data, allowing higher learning rates and faster convergence. This layer adjusts the activation values of the embedding layer by normalising them to have a mean of zero and a variance of one, which helps to combat the problem of internal covariate bias.

- **Details**: By normalising the data, the model becomes less sensitive to the specific initialisation of the weights and can be trained more consistently and efficiently.

3. **LSTM Layer**:
   - **Purpose**: This layer is the core of the model and it deals with sequences of embedded vectors. It specialises in capturing temporal dependencies and contextual relationships in text, which has very important implications for understanding the flow and intent of comments.
   - **Capabilities**: LSTM is designed to remember information over long periods of time. As an example, if a comment is adding some spam text to normal content, LSTM can help distinguish between spam comments with cleverly inserted spam content. The layer uses gates to control the flow of information, maintaining memory of past important inputs while forgetting irrelevant data.
4. **Dropout Layer**:
   - **Purpose**: This layer is designed to prevent the common overfitting problem encountered in deep learning models. By randomly disabling a portion of the neurons during the training phase, it forces the network to not rely on any one neuron, thus promoting more decentralised and robust feature learning.
   - **Details**: Giving a proportion of randomness helps to enhance the generalisation of the model.s of the model.
5. **Dense Layer with Sigmoid Activation**:
   - **Purpose**: As the final layer of the model, it integrates the features learnt by the LSTM into a single output prediction. The output of this layer is mapped to a probability score between 0 and 1 using a sigmoid activation function, which represents the likelihood that the model will evaluate the text to determine whether it is spam or not after inputting a comment.t comment is spam.
   - **Details**: When the task is binary classification (which is what this project is trying to achieve by determining whether the content of a YouTube comment is spam or non-spam), it consists of a single neuron. The output of the sigmoid function, on the other hand, is well suited for binary decision making, as it provides an explicit threshold of 0.5 to categorise the comments.

Each of the layers in this LSTM model plays an important role in dealing with the classification of complex textual data, especially when it comes to detecting spam in YouTube comments. Combining these layers allows the model to effectively learn from the data and make informed predictions about the nature of each comment. #### Optimization and Training Enhancements

- **Adam Optimizer**: The Adam Optimizer has an excellent adaptive learning rate feature, using which the learning rate can be adjusted throughout the training process. This feature is particularly beneficial in dealing with the complexity and non-stationarity of text data, ensuring faster and more efficient convergence.

- **Early Stop Mechanism**: An early stop callback function is integrated into the training process to monitor the model's performance on the validation set. If the performance does not improve within a predefined number of calendar hours, training is stopped. Using this approach not only helps prevent overfitting, but also saves computational resources by stopping training when the model reaches an optimal point during training.

1.2.3 Experiments

- Import all specific libraries and modules that can provide the necessary tools and functions to efficently accomplish tasks.

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
#import tensorflow_addons as tfa
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Dropout, Embedding, LSTM,␣
↪BatchNormalization
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.callbacks import EarlyStopping
```

- The inputs of the model:

  Read the training data from the .csv file and the testing data from the .xlsv file. The training dataset is derived from CSV files and consists of comment text, author's name, comment date, video ID and category labels (0 for non-spam comments and 1 for spam comments). In this project, only two columns of data are actually useful for training and testing the model, namely the "CLASS" and "TEXT". This means that when I need to read and extract data from the two files, I only need to consider the aforementioned two columns.

  Therefore, we have:

[2]:
```
# read the train .csv file and just take two useful col TEXT & CLASS
def read_csv_file(file_path):
    data = pd.read_csv(file_path)
    text = data['TEXT']
    labels = data['CLASS']
    return text, labels

# read the train .xlsv file and just take two useful col TEXT & CLASS
def read_xlsv_file(file_path):
    data = pd.read_excel(file_path)
    text = data['TEXT']
    labels = data['CLASS']
    return text, labels
```

- For the training part, this function **train_model_Adam(text, labels, num_epochs)** is to build and trains a text classification model using Adam optimizer.

  1. **Tokenize the text data**: The first part, `Tokenizer()` converts text data into sequences of integers, and each integer represents a specific word from the dataset.

  2. **Padding sequences to have the same length**: The padding part is used to ensure

all text sequences have the same length (Using `maxlen=max_sequence_length` to make every text sequence have the same length, which is equal to the maximum length of all text comments). This part is important because most deep learning models require input data of consistent size.

3. **Splitting into training and testing sets**:
   - To divide the data into training and testing sets, I chose `train_size=0.8`, which selects 80% of the data for training and the remaining data for testing. There are multiple benefits to selecting a larger `train_size`. More training data enhances the model's ability to discern intricate patterns, boosting its overall accuracy. It also mitigates the risk of overfitting, ensuring that the model generalizes well to unseen data rather than merely memorizing the training examples. Additionally, a larger dataset enhances the model's robustness by encompassing a wider variety of real-world scenarios, which improves its practical applicability. Moreover, a substantial training set increases statistical reliability, lending more credibility to the model's predictions.
   - Setting `random_state=42` guarantees that the split is consistent, ensuring that the same data points are allocated to the training and test sets in each run.

4. **Model Architecture**:
   - `Embedding`: Converts integer sequences into dense vectors of fixed size.
     - `input_dim=len(tokenizer.word_index) + 1`: The input dimension is set to the number of unique words + 1 (for the padding zero).
     - `output_dim=100`: Each word is represented by a vector of length 100.
     - `input_length=max_sequence_length`: Each input sequence will have this fixed length.
   - `BatchNormalization`: Normalizes the activations of the previous layer, which helps in speeding up training and reducing overfitting.
   - `LSTM(128)`: A Long Short-Term Memory layer with 128 units to process sequences.
   - `Dropout(0.5)`: Randomly sets input units to 0 with a frequency of 50% at each step during training time, which helps to prevent overfitting.
   - `Dense(1, activation='sigmoid')`: Outputs a single value between 0 and 1, representing the probability of the target class (binary classific

5. **Model Compilation**: I chose `Adam` as the optimizer and its `learning_rate` is equal to `0.0001`. A relatively small learning rate is selected to enable the model to learn incrementally, ensuring it doesn't overlook minima on the loss surface. To choose the `loss`, `binary_crossentropy` is the most suitable for binary classification tasks.

6. **Training**: Using the fit method to train the model
   - `batch_size=32`: Number of samples per gradient update. Common sizes are 32, 64, and 128. Because there are not too many samples, 32 size is enough. - `epochs=num_epochs`: Allows flexibility to set the number of training epochs externally.
   - `verbose=1`: Shows a progress bar during training.
   - `validation_split=0.2`: Uses 20% of the training data as validation data.
   - `callbacks=[early_stopping]`: Stops training when the validation loss has not improved for 10 epochs (`patience=10`) and restores model weights from the epoch

with the best value of the monitored quantity (‘restore_best_weights=Tru

In the end, the trained `loss` and `training set accuracy` data is printed and returned to the `model`, `tokenizer`, and training `history`.‘). tion).

[1]:
```
def train_model_Adam(text, labels, num_epochs):
    # Tokenize the text data
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(text)
    sequences = tokenizer.texts_to_sequences(text)

    # Padding sequences to have the same length
    max_sequence_length = max([len(seq) for seq in sequences])
    padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length)

    # Splitting into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(padded_sequences,␣
 ↪labels, train_size=0.8, random_state=42)

    # Model architecture
    model = Sequential()
    model.add(Embedding(input_dim=len(tokenizer.word_index) + 1,␣
 ↪output_dim=100, input_length=max_sequence_length))
    model.add(BatchNormalization())
    model.add(LSTM(128))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    optimizer = Adam(learning_rate=0.0001)
    model.compile(loss='binary_crossentropy', optimizer=optimizer,␣
 ↪metrics=['accuracy'])


    #history = model.fit(X_train, y_train, batch_size=32, epochs=num_epochs,␣
 ↪verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)

    history = model.fit(X_train, y_train, batch_size=32, epochs=num_epochs,␣
 ↪verbose=1, validation_split=0.2, callbacks=[early_stopping])
    evaluation = model.evaluate(X_test, y_test)

    print("Test set loss:", evaluation[0])
    print("Training set accuracy:", evaluation[1])

    return model, tokenizer, history
```

- This function **test_model(model, text, labels, tokenizer)** is designed to evaluate the performance of a pre-trained model by calculating its accuracy on a test dataset.

  1. **Tokenize the text data**:
     - `texts_to_sequences(text)`: Converts the list of text data into sequences of integers.
     - `pad_sequences(sequences, maxlen=model.input_shape[1])`:This process ensures uniformity in sequence length by padding the shorter ones, a critical step because neural networks need inputs of the same size. The parameter `maxlen=model.input_shape[1]` defines the maximum sequence length according to the model's input dimensions, ensuring consistent input sizes and avoiding dimensional mismatches during prediction.
  2. **Making predictions**:
     - `model.predict(padded_sequences)`: Generates output predictions for the input sequences.
     - `(predictions > 0.5).astype(int).flatten()`: Converts the probabilities returned by the model to binary labels (0 or 1). This is typical for binary classification tasks where a threshold of 0.5 is used to decide the class labels
     - `accuracy_score(labels, y_pred)`: Calculate the accuracy.
  3. **Creating a comparison Table**: `pd.DataFrame()` is to create a table which shows the text from the testing dataset, the predicted labels and the actual labels. This is helpful for manually reviewing whether the prediction cases are correct or not...

[10]:
```
# Testing the trained model and calculating accuracy
def test_model(model, text, labels, tokenizer):
    # Tokenize the text data
    sequences = tokenizer.texts_to_sequences(text)
    padded_sequences = pad_sequences(sequences, maxlen=model.input_shape[1])

    # Making predictions using the model
    predictions = model.predict(padded_sequences)
    y_pred = (predictions > 0.5).astype(int).flatten()
    accuracy = accuracy_score(labels, y_pred)
    print("Testing set accuracy:", accuracy)

    # Create a table to compare the actual CLASS col and the predicted CLASS col
    df = pd.DataFrame({'Text': text, 'Predicted Label': y_pred, 'Actual Label':␣
 ↪labels})
    print(df)
```

- This function is used to generate a chart that displays the relationship between Loss and Epochs.

[11]:
```
def plot_loss(history):
    # Get the loss values from training history
    loss = history.history['loss']

    # Create a list of epochs
```

```
    epochs = range(1, len(loss) + 1)

    # Plot the loss values
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.title('Training Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```

1.2.4 Result of Experiments

Before I execute the program, based on what the model already does plus a variety of factors, and assuming that the large amount of data on the .csv archive is accurate enough and that there are valid features for each piece of text, I believe that with enough epochs, the accuracy of the prediction can reach at least 70% or more, and can stabilise at 85% or more.

- Read and get the training data (`TEXT` and `CLASS`) from `Topic1-youtube_spam_train.csv`, and set the `num_epochs` as 100 (can be changed to another larger integer). Then execute `train_model_Adam`; there is a process of every epoch which is displayed under the code.

[15]:
```
train_text, train_labels = read_csv_file("./Topic_3_Data/
 ↪Topic1-youtube_spam_train.csv")

num_epochs = 100

model, tokenizer, history = train_model_Adam(train_text, train_labels,␣
 ↪num_epochs)
```

```
Epoch 1/100
16/16 [==============================] - 6s 180ms/step - loss: 0.6839 -
accuracy: 0.5980 - val_loss: 0.6896 - val_accuracy: 0.5000
Epoch 2/100
16/16 [==============================] - 2s 139ms/step - loss: 0.6309 -
accuracy: 0.6529 - val_loss: 0.6886 - val_accuracy: 0.4609
Epoch 3/100
16/16 [==============================] - 2s 142ms/step - loss: 0.5837 -
accuracy: 0.7098 - val_loss: 0.6849 - val_accuracy: 0.5156
Epoch 4/100
16/16 [==============================] - 2s 143ms/step - loss: 0.5162 -
accuracy: 0.7784 - val_loss: 0.6824 - val_accuracy: 0.5156
Epoch 5/100
16/16 [==============================] - 2s 145ms/step - loss: 0.4300 -
accuracy: 0.8569 - val_loss: 0.6779 - val_accuracy: 0.5547
Epoch 6/100
16/16 [==============================] - 2s 147ms/step - loss: 0.3465 -
accuracy: 0.9020 - val_loss: 0.6706 - val_accuracy: 0.6172
Epoch 7/100
16/16 [==============================] - 2s 152ms/step - loss: 0.2467 -
```

```
accuracy: 0.9529 - val_loss: 0.6609 - val_accuracy: 0.6172
Epoch 8/100
16/16 [==============================] - 2s 148ms/step - loss: 0.1618 -
accuracy: 0.9765 - val_loss: 0.6475 - val_accuracy: 0.6484
Epoch 9/100
16/16 [==============================] - 2s 147ms/step - loss: 0.1312 -
accuracy: 0.9765 - val_loss: 0.6292 - val_accuracy: 0.7031
Epoch 10/100
16/16 [==============================] - 2s 151ms/step - loss: 0.1013 -
accuracy: 0.9804 - val_loss: 0.6138 - val_accuracy: 0.7031
Epoch 11/100
16/16 [==============================] - 2s 149ms/step - loss: 0.0773 -
accuracy: 0.9902 - val_loss: 0.5914 - val_accuracy: 0.7031
Epoch 12/100
16/16 [==============================] - 2s 150ms/step - loss: 0.0615 -
accuracy: 0.9902 - val_loss: 0.5756 - val_accuracy: 0.6953
Epoch 13/100
16/16 [==============================] - 2s 148ms/step - loss: 0.0508 -
accuracy: 0.9882 - val_loss: 0.5497 - val_accuracy: 0.7344
Epoch 14/100
16/16 [==============================] - 2s 150ms/step - loss: 0.0456 -
accuracy: 0.9902 - val_loss: 0.5291 - val_accuracy: 0.7422
Epoch 15/100
16/16 [==============================] - 2s 150ms/step - loss: 0.0392 -
accuracy: 0.9922 - val_loss: 0.5090 - val_accuracy: 0.7500
Epoch 16/100
16/16 [==============================] - 2s 151ms/step - loss: 0.0337 -
accuracy: 0.9980 - val_loss: 0.4962 - val_accuracy: 0.7500
Epoch 17/100
16/16 [==============================] - 2s 153ms/step - loss: 0.0287 -
accuracy: 0.9941 - val_loss: 0.4788 - val_accuracy: 0.7578
Epoch 18/100
16/16 [==============================] - 2s 152ms/step - loss: 0.0238 -
accuracy: 0.9980 - val_loss: 0.4648 - val_accuracy: 0.7656
Epoch 19/100
16/16 [==============================] - 2s 152ms/step - loss: 0.0219 -
accuracy: 0.9961 - val_loss: 0.4524 - val_accuracy: 0.7812
Epoch 20/100
16/16 [==============================] - 2s 155ms/step - loss: 0.0221 -
accuracy: 0.9980 - val_loss: 0.4433 - val_accuracy: 0.7812
Epoch 21/100
16/16 [==============================] - 2s 156ms/step - loss: 0.0740 -
accuracy: 0.9804 - val_loss: 0.4524 - val_accuracy: 0.7969
Epoch 22/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0375 -
accuracy: 0.9941 - val_loss: 0.4297 - val_accuracy: 0.8203
Epoch 23/100
16/16 [==============================] - 2s 153ms/step - loss: 0.0257 -
```

```
accuracy: 0.9941 - val_loss: 0.3847 - val_accuracy: 0.8750
Epoch 24/100
16/16 [==============================] - 2s 155ms/step - loss: 0.0183 -
accuracy: 0.9980 - val_loss: 0.3853 - val_accuracy: 0.8906
Epoch 25/100
16/16 [==============================] - 2s 157ms/step - loss: 0.0168 -
accuracy: 0.9980 - val_loss: 0.3810 - val_accuracy: 0.8984
Epoch 26/100
16/16 [==============================] - 2s 156ms/step - loss: 0.0146 -
accuracy: 1.0000 - val_loss: 0.3738 - val_accuracy: 0.9141
Epoch 27/100
16/16 [==============================] - 2s 155ms/step - loss: 0.0123 -
accuracy: 1.0000 - val_loss: 0.3656 - val_accuracy: 0.9219
Epoch 28/100
16/16 [==============================] - 2s 152ms/step - loss: 0.0119 -
accuracy: 1.0000 - val_loss: 0.3577 - val_accuracy: 0.9219
Epoch 29/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0119 -
accuracy: 1.0000 - val_loss: 0.3481 - val_accuracy: 0.9141
Epoch 30/100
16/16 [==============================] - 2s 155ms/step - loss: 0.0114 -
accuracy: 1.0000 - val_loss: 0.3417 - val_accuracy: 0.9141
Epoch 31/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0104 -
accuracy: 1.0000 - val_loss: 0.3390 - val_accuracy: 0.9219
Epoch 32/100
16/16 [==============================] - 2s 155ms/step - loss: 0.0095 -
accuracy: 1.0000 - val_loss: 0.3351 - val_accuracy: 0.9219
Epoch 33/100
16/16 [==============================] - 2s 155ms/step - loss: 0.0092 -
accuracy: 0.9980 - val_loss: 0.3329 - val_accuracy: 0.9219
Epoch 34/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0085 -
accuracy: 1.0000 - val_loss: 0.3319 - val_accuracy: 0.9219
Epoch 35/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0068 -
accuracy: 1.0000 - val_loss: 0.3295 - val_accuracy: 0.9219
Epoch 36/100
16/16 [==============================] - 2s 155ms/step - loss: 0.0074 -
accuracy: 1.0000 - val_loss: 0.3317 - val_accuracy: 0.9219
Epoch 37/100
16/16 [==============================] - 2s 156ms/step - loss: 0.0064 -
accuracy: 1.0000 - val_loss: 0.3333 - val_accuracy: 0.9297
Epoch 38/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0076 -
accuracy: 1.0000 - val_loss: 0.3351 - val_accuracy: 0.9219
Epoch 39/100
16/16 [==============================] - 2s 153ms/step - loss: 0.0062 -
```

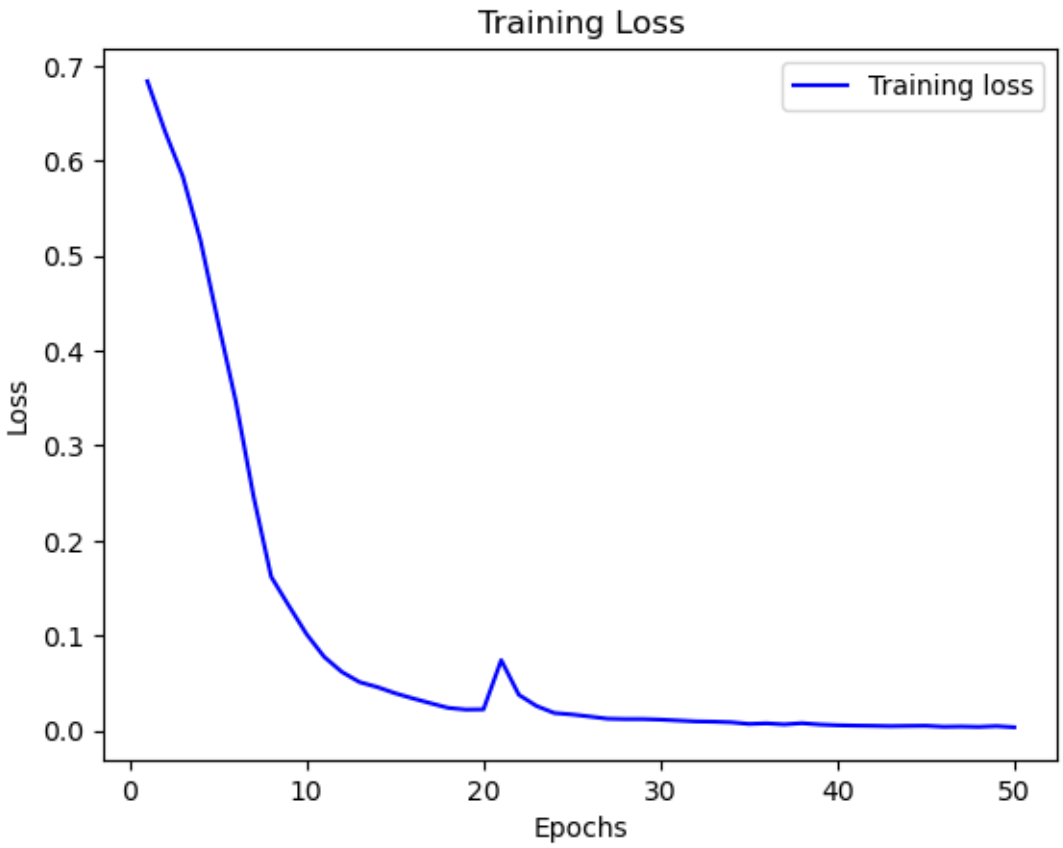
```
accuracy: 1.0000 - val_loss: 0.3323 - val_accuracy: 0.9141
Epoch 40/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0054 -
accuracy: 1.0000 - val_loss: 0.3267 - val_accuracy: 0.9141
Epoch 41/100
16/16 [==============================] - 2s 152ms/step - loss: 0.0051 -
accuracy: 1.0000 - val_loss: 0.3275 - val_accuracy: 0.9219
Epoch 42/100
16/16 [==============================] - 2s 153ms/step - loss: 0.0048 -
accuracy: 1.0000 - val_loss: 0.3306 - val_accuracy: 0.9219
Epoch 43/100
16/16 [==============================] - 3s 164ms/step - loss: 0.0045 -
accuracy: 1.0000 - val_loss: 0.3370 - val_accuracy: 0.9219
Epoch 44/100
16/16 [==============================] - 2s 156ms/step - loss: 0.0047 -
accuracy: 1.0000 - val_loss: 0.3407 - val_accuracy: 0.9219
Epoch 45/100
16/16 [==============================] - 2s 156ms/step - loss: 0.0049 -
accuracy: 1.0000 - val_loss: 0.3450 - val_accuracy: 0.9141
Epoch 46/100
16/16 [==============================] - 2s 153ms/step - loss: 0.0039 -
accuracy: 1.0000 - val_loss: 0.3522 - val_accuracy: 0.9219
Epoch 47/100
16/16 [==============================] - 2s 153ms/step - loss: 0.0041 -
accuracy: 1.0000 - val_loss: 0.3588 - val_accuracy: 0.9141
Epoch 48/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0037 -
accuracy: 1.0000 - val_loss: 0.3654 - val_accuracy: 0.9141
Epoch 49/100
16/16 [==============================] - 2s 157ms/step - loss: 0.0045 -
accuracy: 1.0000 - val_loss: 0.3647 - val_accuracy: 0.9141
Epoch 50/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0033 -
accuracy: 1.0000 - val_loss: 0.3690 - val_accuracy: 0.9141
5/5 [==============================] - 0s 68ms/step - loss: 0.2055 - accuracy:
0.9125
Test set loss: 0.20549964904785156
Training set accuracy: 0.9125000238418579
```

The training process is stopped at **50** ecophs and its training set accuracy is about 91.3%.

Chart of Training Loss Now let's see the chart of training loss. I used the `plot_loss(history)` function to display it.

- The chart of **Training Loss**

[16]:
```
# Plot the loss values
plot_loss(history)
```



To summarise, the loss decreases gradually with increasing epoch and there is no significant rebound, which is a sign of a good training process, indicating that the model is gradually learning and improving its prediction accuracy. This relies on sound model design, appropriate choice of optimisation algorithm, proper learning rate setting and effective data processing strategy. It means that the model I designed meets the criteria I envisioned.

Testing the model

- Read the testing data from .xlsv file, also just read and get two columns data from the file (`Text` and `CLASS`):

[17]:
```
test_text, test_labels = read_xlsv_file("./Topic_3_Data/
 ↪Topic1-youtube_spam_test.xlsx")

test_model(model, test_text, test_labels, tokenizer)
```

```
7/7 [==============================] - 1s 71ms/step
Testing set accuracy: 0.9455445544554455
                                                Text  Predicted Label  \
0    Is this the video that started the whole "got …                0
1                  Can anyone sub to my channel? :Dï»¿                1
```

```
2                        prehistoric song..has beenï»¿                0
3    You think you're smart?        Headbutt your f…                0
4    DISLIKE.. Now one knows REAL music - ex. Enime…                 0
..                                                   …              …
197                Check out this video on YouTube:ï»¿                1
198                Check out this video on YouTube:ï»¿                1
199                Check out this video on YouTube:ï»¿                1
200                Check out this video on YouTube:ï»¿                1
201  ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸ…                0

     Actual Label
0               0
1               1
2               0
3               0
4               0
..            …
197             1
198             1
199             1
200             1
201             1

[202 rows x 3 columns]
```

We can see that the accuracy is about **94.6%**. A model test accuracy of over 90 per cent indicates that it performs well in understanding and classifying the test data, effectively identifying and predicting patterns and trends in the data. This usually means that the model has good generalisation capabilities.

Get the relationship between epochs and accuracy of the model.

- Since the number of epochs during model training is not fixed, and the resulting accuracy is not fixed either, I tried running the same code multiple times (repeating the execution of **train_model_Adam()**) to obtain multiple results.

    1. This is the accuracy of the training set and testing set to the model which is stopped at the 43rd epoch: We can see that the testing accuracy is smaller than the accuracy of stopping at 50 epochs.

```
16/16 [==============================] - 2s 152ms/step - loss: 0.0058 - accuracy: 1.0000 - val_loss: 0.3682 - val_accuracy: 0.8906
Epoch 38/100
16/16 [==============================] - 3s 159ms/step - loss: 0.0055 - accuracy: 1.0000 - val_loss: 0.3580 - val_accuracy: 0.8906
Epoch 39/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 0.3684 - val_accuracy: 0.8906
Epoch 40/100
16/16 [==============================] - 2s 157ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.3847 - val_accuracy: 0.8984
Epoch 41/100
16/16 [==============================] - 2s 156ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 0.3924 - val_accuracy: 0.8906
Epoch 42/100
16/16 [==============================] - 3s 157ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.4044 - val_accuracy: 0.8984
Epoch 43/100
16/16 [==============================] - 2s 154ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.4184 - val_accuracy: 0.9062
5/5 [==============================] - 0s 51ms/step - loss: 0.2370 - accuracy: 0.8938
Test set loss: 0.2370145618915558
Training set accuracy: 0.893750011920929
```

```
7/7 [==============================] - 1s 46ms/step
Testing set accuracy: 0.9257425742574258
                                                  Text  Predicted Label  \
0    Is this the video that started the whole "got ...                0
1                  Can anyone sub to my channel? :Dï»¿                1
2                          prehistoric song..has beenï»¿                0
3    You think you're smart?        Headbutt your f...                0
4    DISLIKE.. Now one knows REAL music - ex. Enime...                0
..                                                 ...              ...
197                Check out this video on YouTube:ï»¿                1
198                Check out this video on YouTube:ï»¿                1
199                Check out this video on YouTube:ï»¿                1
200                Check out this video on YouTube:ï»¿                1
201  ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸ...                1

     Actual Label
0               0
1               1
2               0
3               0
4               0
..            ...
197             1
198             1
199             1
200             1
201             1

[202 rows x 3 columns]
```

2. This is the accuracy of the training set and testing set to the model which is stopped at the 53rd epoch: We can see that the testing accuracy is larger than the accuracy of stopping at 50 epochs.

```
16/16 [==============================] - 3s 176ms/step - loss: 0.0060 - accuracy: 1.0000 - val_loss: 0.2404 - val_accuracy: 0.9141
Epoch 48/100
16/16 [==============================] - 3s 187ms/step - loss: 0.0065 - accuracy: 1.0000 - val_loss: 0.2402 - val_accuracy: 0.9141
Epoch 49/100
16/16 [==============================] - 3s 176ms/step - loss: 0.0059 - accuracy: 1.0000 - val_loss: 0.2403 - val_accuracy: 0.9141
Epoch 50/100
16/16 [==============================] - 3s 176ms/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 0.2412 - val_accuracy: 0.9141
Epoch 51/100
16/16 [==============================] - 3s 179ms/step - loss: 0.0059 - accuracy: 1.0000 - val_loss: 0.2426 - val_accuracy: 0.9062
Epoch 52/100
16/16 [==============================] - 3s 179ms/step - loss: 0.0049 - accuracy: 1.0000 - val_loss: 0.2436 - val_accuracy: 0.9062
Epoch 53/100
16/16 [==============================] - 3s 181ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.2448 - val_accuracy: 0.9062
5/5 [==============================] - 0s 69ms/step - loss: 0.1227 - accuracy: 0.9500
Test set loss: 0.12271255254745483
Training set accuracy: 0.949999988079071
```

```
7/7 [==============================] - 1s 64ms/step
Testing set accuracy: 0.9504950495049505
                                                  Text  Predicted Label  \
0    Is this the video that started the whole "got ...                0
1                  Can anyone sub to my channel? :Dï»¿                1
2                          prehistoric song..has beenï»¿                0
3    You think you're smart?        Headbutt your f...                0
4    DISLIKE.. Now one knows REAL music - ex. Enime...                0
..                                                 ...              ...
197                Check out this video on YouTube:ï»¿                1
198                Check out this video on YouTube:ï»¿                1
199                Check out this video on YouTube:ï»¿                1
200                Check out this video on YouTube:ï»¿                1
201  ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸš¨ðŸ...                1

     Actual Label
0               0
1               1
2               0
3               0
4               0
..            ...
197             1
198             1
199             1
200             1
201             1

[202 rows x 3 columns]
```

Look at these images. We can find a rule: The accuracy obtained from either **Training** or **Testing** is deeply related to how many epochs were performed during training. Usually, the more epochs are run when training a model, the higher the accuracy will be get!

1.2.5 Conclusion

This project successfully illustrates the construction and training of neural networks for text classification, tackling prevalent issues like overfitting with strategies including dropout and early stopping. Tokenization and padding are employed to standardize the input data, whereas LSTM layers are utilized to capture dependencies within sequential data. Overall, the project establishes a comprehensive pipeline from data ingestion and preprocessing to the training, evaluation, and application of neural networks for practical text classification scenarios.