A business case

Audiobooks

Name: Hung Pham
Date: Mar 15, 2022

Table of Contents

1. Introduction

The audiobook dataset is a collection of data from the Audiobook app and each record in the dataset was generated when a customer made at least one purchase.

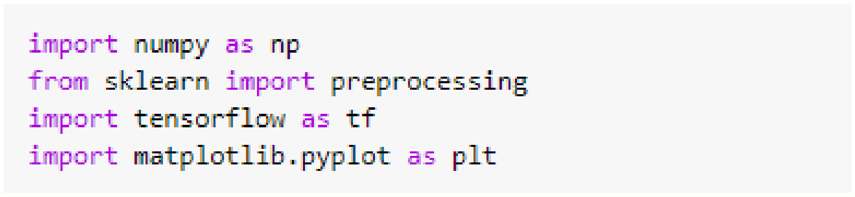
In this project, I use machine learning to make a model based on the dataset which was collected by the Audiobook app to predict returning customers who have a high chance of making more purchases. The model will show us which factors are crucial for a customer to buy again and which factors are less important to determine the buying decision. The dataset is consist of twelve columns including id, book-length, price-overall, price-average, review, minutes listened, completion, support requests, last visited minus purchase date, and targets. All of them are presented by numbers and some of them are not important or not useful for the training model so that we can filter them out.

The reason to make this model is that we can save a lot of time and money if we can predict what kinds of customers are going to buy again because we do not want to advertise to the people who are not likely to buy. Thus, we can use the model to identify the potential customers who are likely to buy again and focus only on those buying customers to reduce costs and increase productivity.

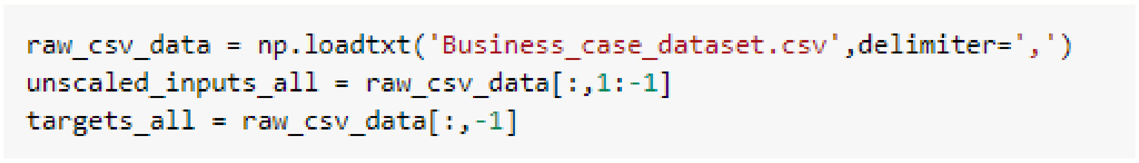
The main goal of this research is to train a machine learning model by using a different approach to generate a good model to predict if a customer will make a purchase again.

2. Methodology

2.1. Load and Filter data



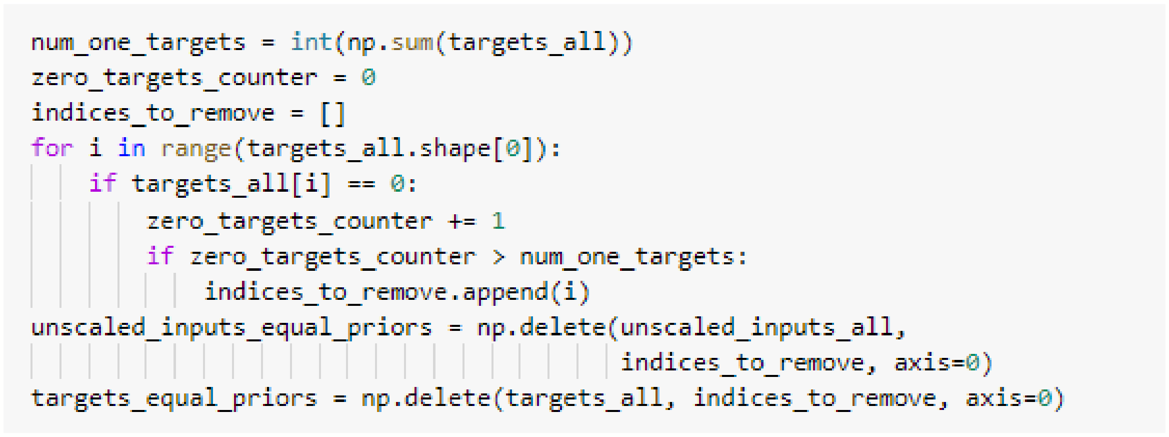
```python
import numpy as np
from sklearn import preprocessing
import tensorflow as tf
import matplotlib.pyplot as plt
```



```python
raw_csv_data = np.loadtxt('Business_case_dataset.csv',delimiter=',')
unscaled_inputs_all = raw_csv_data[:,1:-1]
targets_all = raw_csv_data[:,-1]
```

The ids are randomly generated numbers so I can safely remove them by taking all the values except the first column and the last column [:,1:-1] which is our target. There are missing data in the Review column and in a training model, I don't want empty values; Then I replace the empty values with the average value.

2.2. Preprocessing the Dataset



```python
num_one_targets = int(np.sum(targets_all))
zero_targets_counter = 0
indices_to_remove = []
for i in range(targets_all.shape[0]):
    if targets_all[i] == 0:
        zero_targets_counter += 1
        if zero_targets_counter > num_one_targets:
            indices_to_remove.append(i)
unscaled_inputs_equal_priors = np.delete(unscaled_inputs_all,
                                         indices_to_remove, axis=0)
targets_equal_priors = np.delete(targets_all, indices_to_remove, axis=0)
```

I want to make our dataset balanced but there are more targets with 1 than 0, so I want to remove some zero values to balance the dataset.

2.3. Standardizing the inputs

```
scaled_inputs = preprocessing.scale(unscaled_inputs_equal_priors)
```

It is extremely important to perform a standardization to the inputs before training the model since the values are in different ranges which could cause some values to become dominating over other values or it could make it difficult for the machine to understand the data. Thus, applying standardization could improve the performance of the model as well as reduce the time training and errors.

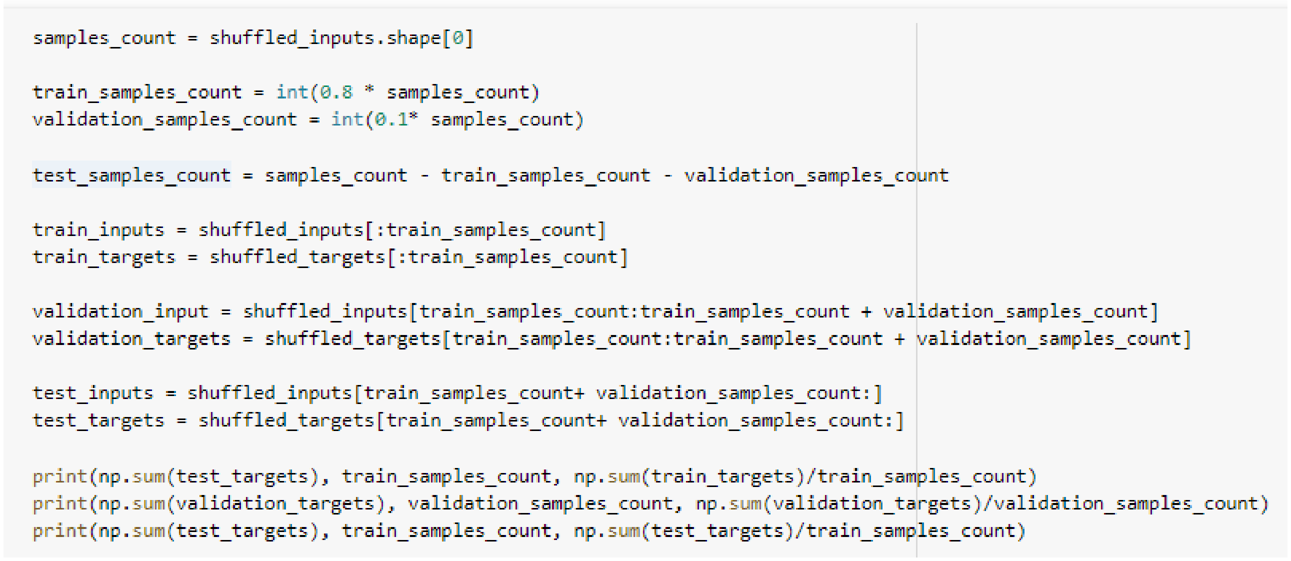
2.4. Shuffle the data

```
shuffled_indices = np.arange(scaled_inputs.shape[0])
np.random.shuffle(shuffled_indices)

shuffled_inputs  = scaled_inputs[shuffled_indices]
shuffled_targets = targets_equal_priors[shuffled_indices]
```

The audiobook dataset is arranged but I am going to use the batching process which updates the model parameters regularly. Hence, shuffling the data is important as

I want the dataset to be as random as possible to reduce errors and overfitting.

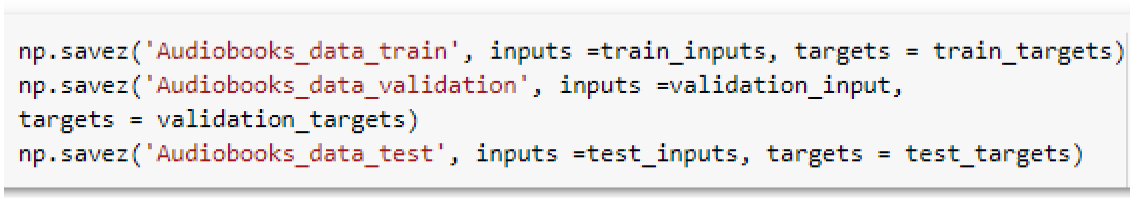
2.5. Divide the Dataset into Train, Validation, and Test



```
samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1* samples_count)

test_samples_count = samples_count - train_samples_count - validation_samples_count

train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]

validation_input = shuffled_inputs[train_samples_count:train_samples_count + validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count + validation_samples_count]

test_inputs = shuffled_inputs[train_samples_count+ validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+ validation_samples_count:]

print(np.sum(test_targets), train_samples_count, np.sum(train_targets)/train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets)/validation_samples_count)
print(np.sum(test_targets), train_samples_count, np.sum(test_targets)/train_samples_count)
```

I split the dataset into three sets train data, validation data, and test data. Most of the data will be used for training, a small amount of data is used for validation and testing. In this project, I will use 80% of the dataset for training, 10% will be used for validation and 10% will be used for testing. The training set is used to train the model. The validation set is used to validate the model and determine if the model is overfitting or going in the wrong direction. So that I can make an adjustment. The testing set is used to determine the accuracy of the model and how well the model performed in new data.

2.6. Save the dataset



```
np.savez('Audiobooks_data_train', inputs =train_inputs, targets = train_targets)
np.savez('Audiobooks_data_validation', inputs =validation_input,
targets = validation_targets)
np.savez('Audiobooks_data_test', inputs =test_inputs, targets = test_targets)
```

Save the three sets of data in .npz format to use later.

2.7. Load the saved data and plotting function

```
npz = np.load('Audiobooks_data_train.npz')
train_inputs, train_targets = npz['inputs'].astype(float), npz['targets'].astype(int)

npz = np.load('Audiobooks_data_validation.npz')
validation_inputs, validation_targets = npz['inputs'].astype(float), npz['targets'].astype(int)

npz = np.load('Audiobooks_data_test.npz')
test_inputs, test_targets = npz['inputs'].astype(float), npz['targets'].astype(int)
```

```
def plot(history,name):
  print("- Optimizer: "+ name)
  fig , ax = plt.subplots(1,2)
  train_acc = history.history['accuracy']
  train_loss = history.history['loss']
  val_acc = history.history['val_accuracy']
  val_loss = history.history['val_loss']
  fig.set_size_inches(10,5)
  ax[0].plot(train_acc , 'go-' , label = 'Training Accuracy')
  ax[0].plot(val_acc , 'ro-' , label = 'Validation Accuracy')
  ax[0].set_title('Training & Validation Accuracy')
  ax[0].legend()
  ax[0].set_xlabel("Epochs")
  ax[0].set_ylabel("Accuracy")

  ax[1].plot(train_loss , 'g-o' , label = 'Training Loss')
  ax[1].plot(val_loss , 'r-o' , label = 'Validation Loss')
  ax[1].set_title('Testing Accuracy & Loss')
  ax[1].legend()
  ax[1].set_xlabel("Epochs")
  ax[1].set_ylabel("Training & Validation Loss")
  plt.show()
```

I load the data that I saved before in the previous step and define a plot function to visualize the training losses and validation losses.

2.8. Train the model

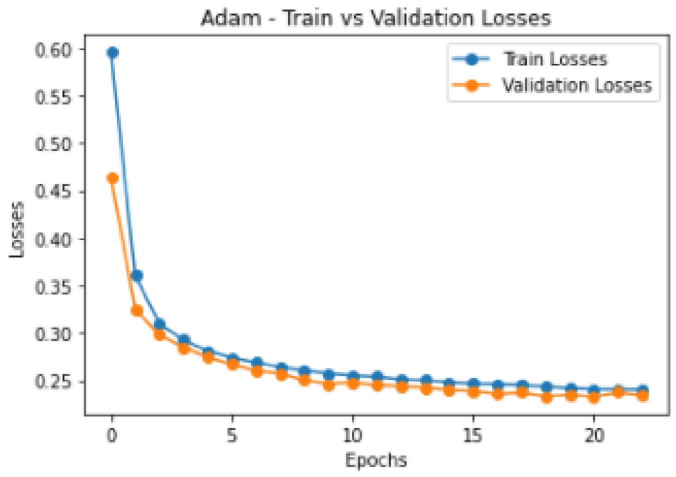
```
def train(epochs=100, batch_size=9, optimizer="adam", sequential= [], patience = 2):
  model = tf.keras.Sequential(sequential)
  model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
  history = model.fit(train_inputs,train_targets,
                      batch_size= batch_size, epochs = epochs,
                      callbacks= [tf.keras.callbacks.EarlyStopping(patience =patience)],
                      validation_data=(validation_inputs, validation_targets),
                      validation_steps=10,
                      verbose = 0
                     )
  test_loss, test_accuracy = model.evaluate(test_inputs, test_targets)
  return test_loss, test_accuracy , history
```

Define a train function so that I can change the parameters later in the experiment steps.

```
input_size = 10
output_size = 2
hidden_nodes = 8
sequential = []
for i in range(3):
  sequential.append(tf.keras.layers.Dense(hidden_nodes, activation='relu'))
sequential.append(tf.keras.layers.Dense(output_size, activation='softmax'))
test_loss, test_accuracy, history = train(epochs=100, batch_size=9, optimizer="adam",
                                          sequential= sequential, patience = 2)
print('\nTest loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss, test_accuracy*100.))
plot_loss("", history)
```

This is an example of using the 8 hidden nodes per layer and 2 hidden layers with ‘relu’ activation function and ‘softmax’, 100 epochs, 9 batches, early stopping is 2, and “Adam” optimizer.

```
Test loss: 0.19. Test accuracy: 94.87%
```



3. Experiment

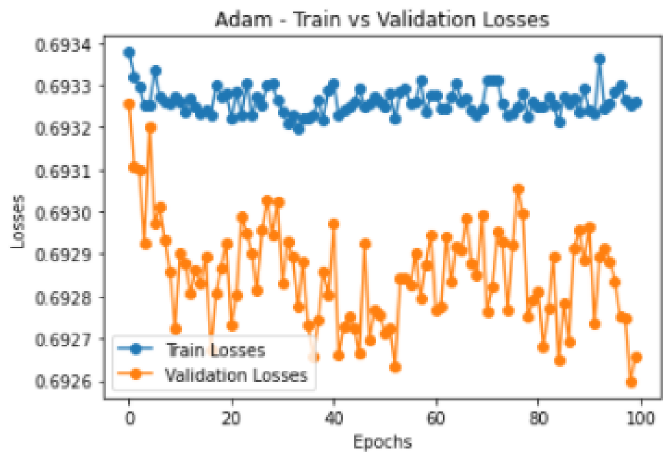
In this experiment, I will perform tests on different scenarios to find an optimal result that is not underfitting or overfitting as well as it should have high accuracy of more than 93% with this Audiobook dataset. First, I will load the data files that I saved earlier in .npz format, then I will choose some default parameters, then change one of those hyperparameters parameters to see if the result is satisfied then move on to the next to get a better result.

- The input size is 10, the output size is 2
- The hidden layers are 3, hidden nodes are 50,
- 5 'relu' and 1 'softmax' activation functions
- Optimizer is "Adam" and early stopping is 2
- Epochs are 100, and batch size is 9

3.1. Change the number of Hidden Layers

After changing some values of hidden layers, I realized that if I increase the number of layers to a big number like 50, the complexity of the model and training time is also increased. The model could easily become overfitting. (Patience =100)

```
Test loss: 0.69. Test accuracy: 44.20%
```

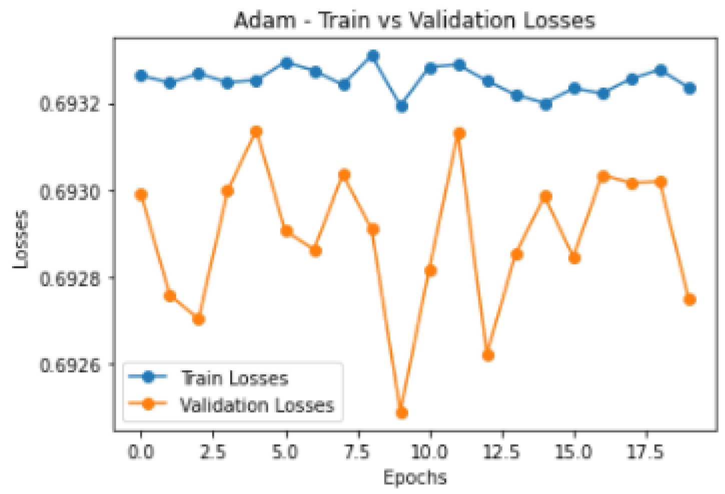


The optimal hidden layers in my tests were range from 3 to 5.

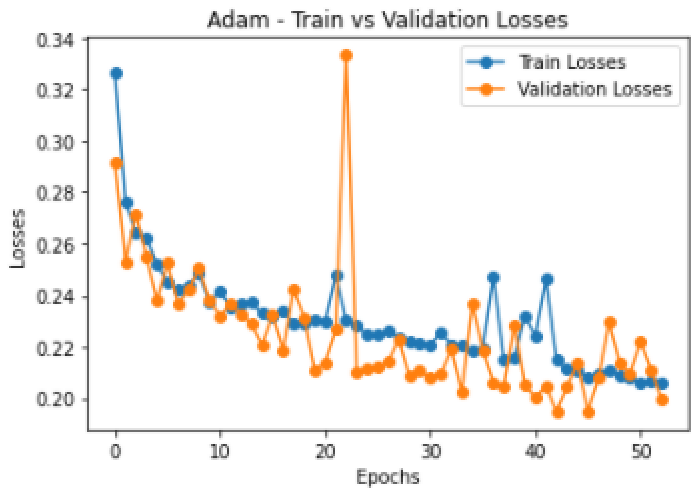
3.2. Change the number of Nodes

First, I change the number of nodes to 2 nodes per layer. After running the training, we can see that the model is underfitting and not accurate as the accuracy dropped on a vast scale

```
14/14 [==============================] - 0s 3ms/step - loss: 0.6944 - accuracy: 0.4420

Test loss: 0.69. Test accuracy: 44.20%
```



I change the number of nodes to in one layer to 500 nodes, then the model becomes overfitting and high variance as we can see in the picture below

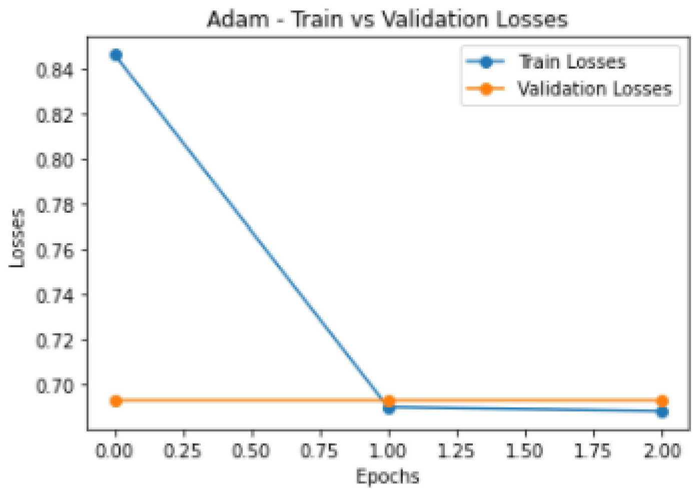


As I tested with different hidden nodes, I see that the values between 8 to 20 work extremely well with this dataset.

3.3. Change the Activation Functions

I change the activation function from “softmax” to “selu” or “tanh” then the loss increased and the accuracy decreased massively.

```
Test loss: 0.69. Test accuracy: 64.29%
```



So the optimal case here is to use the “softmax” activation function for the last hidden layer.

3.4. Change the Optimizers

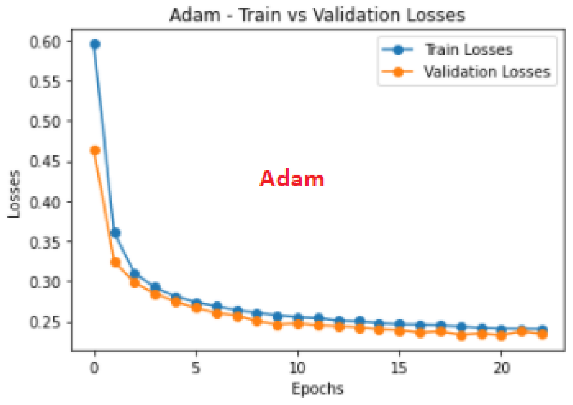
There are many optimizers to choose from including

“Adagrad","Adadelta","Adam","Adadelta","SGD", "RMSprop","FTRL","Nadam”

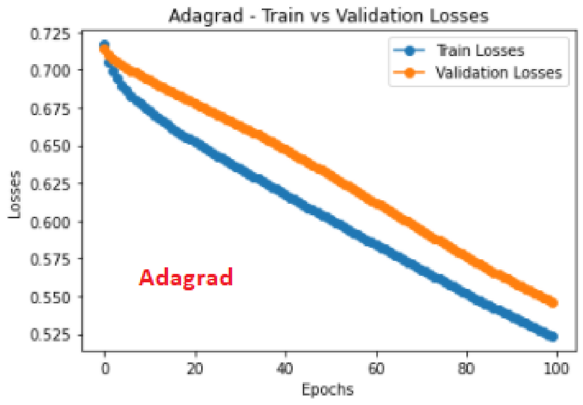
But in this experiment, I will choose the first 6 optimizers to perform tests.

Optimizers

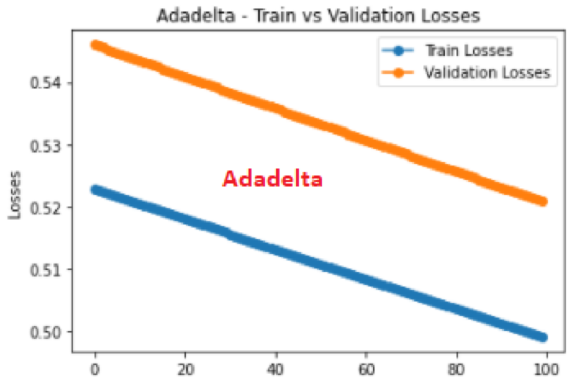
```
Test loss: 0.19. Test accuracy: 94.87%
```



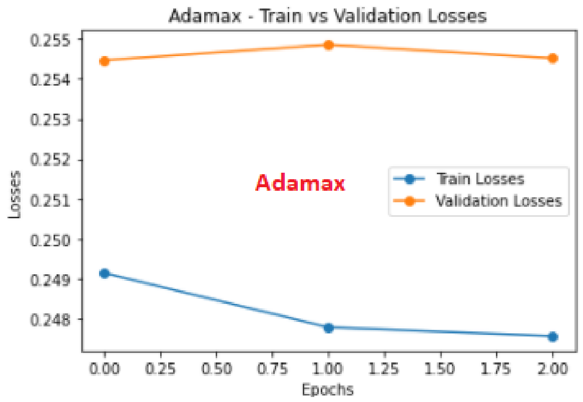
```
Test loss: 0.54. Test accuracy: 73.44%
```



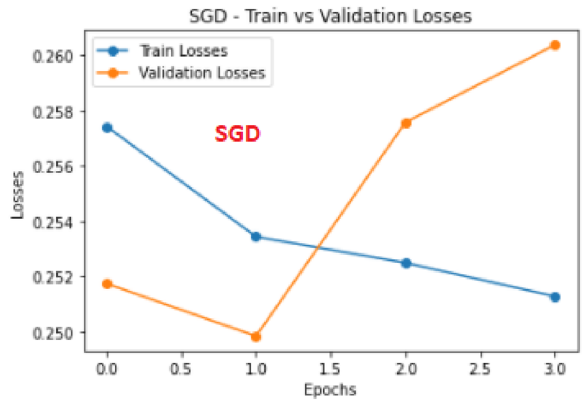
```
Test loss: 0.52. Test accuracy: 76.12%
```



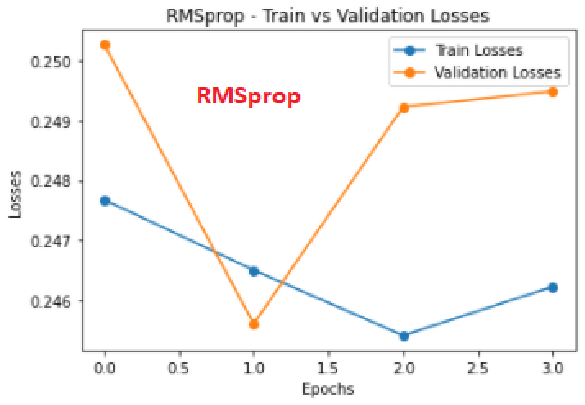
```
Test loss: 0.20. Test accuracy: 94.20%
```



```
Test loss: 0.20. Test accuracy: 94.64%
```



```
Test loss: 0.19. Test accuracy: 93.97%
```



As we can see, ‘Adam’ optimizer works really well on this dataset

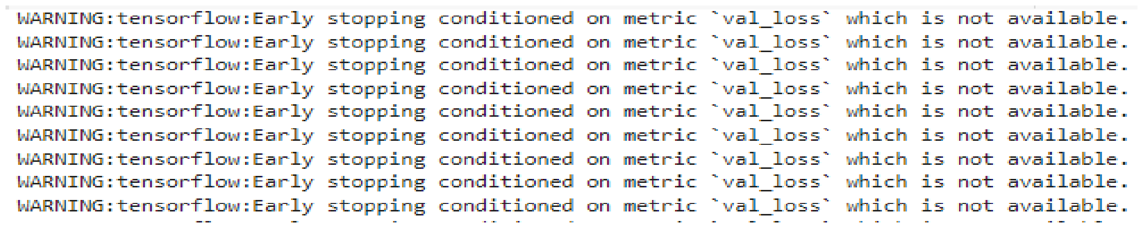
3.5. Change the Early Stopping Value

The early stopping to prevent the model to become overfitting. The model will stop training if the condition is met such as the training loss is too high and the validation loss is too low.

3.6. Change the Ratio of Train, Validation, and Test data

In this experiment, I will change the ratio between train, validation, and test to see if the accuracy is any better than before

- 97% or more on training, 1.5% validation, and 1.5% test

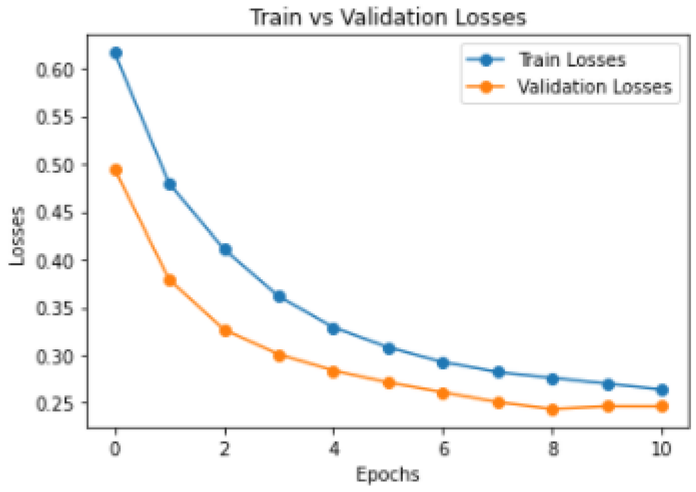


```
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not available.
```

Cannot train the model due to the validation set was too small

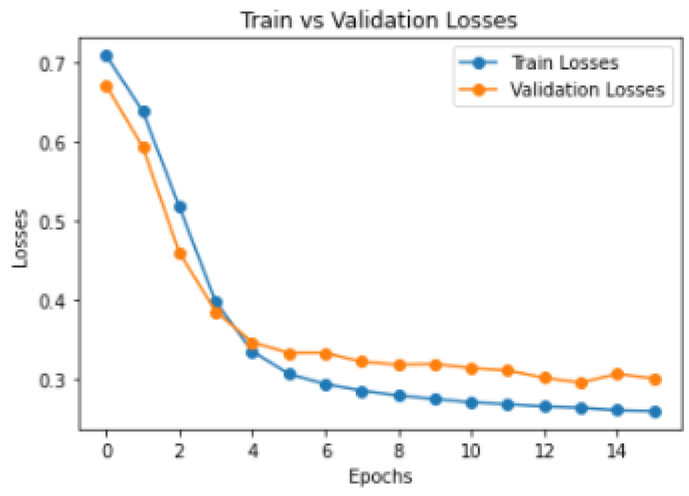
- 40% train, 30% validation, 30% test

```
Test loss: 0.29. Test accuracy: 88.91%
```



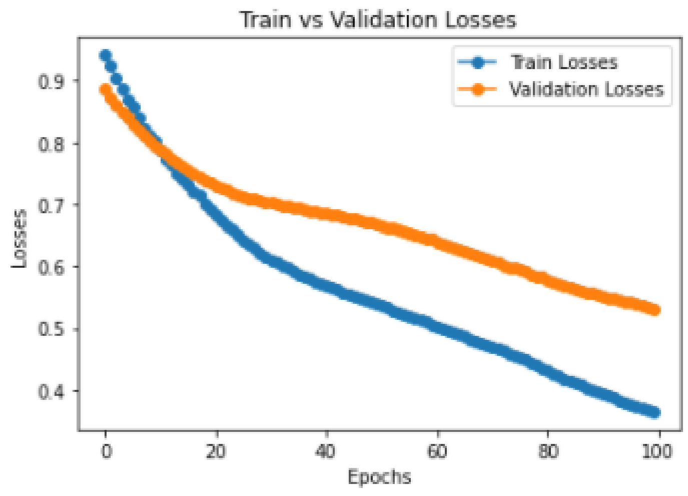
- 20% train, 40% validation, 40% test

```
Test loss: 0.30. Test accuracy: 88.89%
```



- 2% train, 49% validation, 49% test

```
Test loss: 0.55. Test accuracy: 72.25%
```



As we see in the picture above, the accuracy is decreased every time I make the train data smaller. Because the model does not have enough data to train, the accuracy might drop significantly. The good ratio would be 80/20/20 or 70/15/15

- Batch size

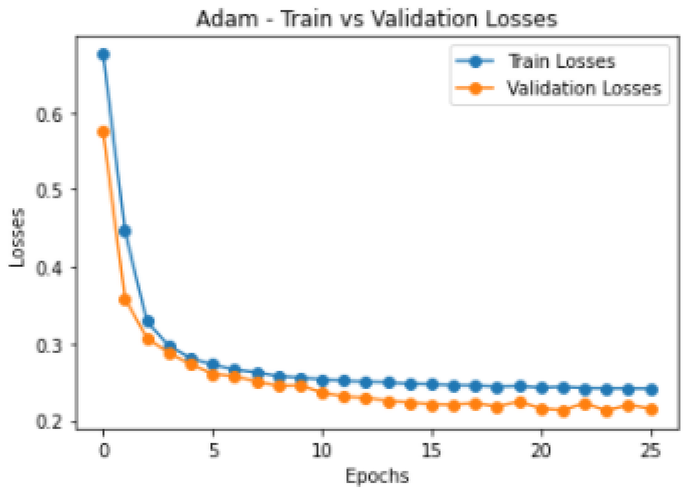
  High batch size could lower the accuracy of the model or even run out of data.

  Low batch size could lead to more noise when updating the weights.

4. Conclusion

After doing some experience to train the model with different approaches and parameters, I found that changing these parameters above can lead to different results. We can use different methods to prevent the model to become overfitting or underfitting such as using the early stopping function. I learned that before we train the model, we should clean up the data first because the dataset could contain a lot of noise and outliers that do not contribute to improving the model. After the experiment, I can generate a model with 95.09% accuracy and I believe that this model will work well with the future Audiobook data.

```
Test loss: 0.18. Test accuracy: 95.09%
```


Adam - Train vs Validation Losses
Train Losses
Validation Losses
Losses
Epochs