**Lab Assignment 5**

**Neural Network & Deep Learning**

**Hyper parameter tuning**

**Step 1**: Import the IMDB data.

**Step 2**: Pre-processing and prepare the data for giving to the neural network.

1. Encoding the integer sequences into a binary matrix.
2. Split it into train and test.
3. Set aside validation data from the training set.

**Step 3**: Building the sequential neural network model.

1. You may choose the layers.
2. Use appropriate activation and loss functions.

**Step 4**: Compile and fit the model to the training dataset. Use validation also.

**Step 5**: Plot training and validation loss.

**Step 6**: Use regularizes to improve the performance.

**Step 7:** Record the best performance.

LAB Manual

**Experiment No. 5**

**Neural Network & Deep Learning**

PART B

|  |  |
| --- | --- |
| Roll No: C0009 | Name: Samarth Borade |
| Class : B | Batch : EB1 |
| Date of Experiment: 02/02/24 | Date of Submission |
| Grade : |  |

**B.1 Software Code written by student:**

#Samarth Borade

#BTI SEM 10

#C009

#EXP 5 : Implementing parameter tuning

import pandas as pd

from tensorflow.keras.preprocessing.text import Tokenizer

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

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, Flatten, Dropout

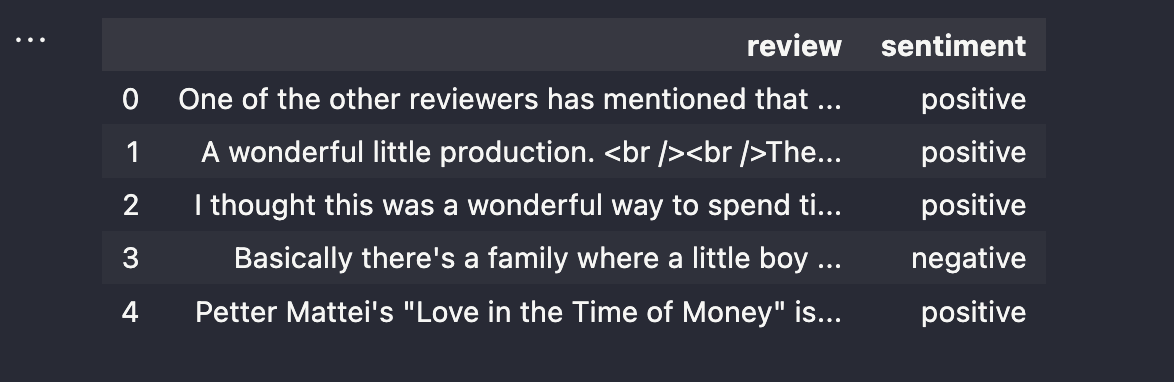
from tensorflow.keras import regularizers

import matplotlib.pyplot as plt

# Step 1: Import the IMDB data

df = pd.read\_csv('IMDB Dataset.csv')

df.head(5)

****

# Step 2: Pre-processing and prepare the data

tokenizer = Tokenizer(*num\_words*=10000)

tokenizer.fit\_on\_texts(df['review'])

sequences = tokenizer.texts\_to\_sequences(df['review'])

data = pad\_sequences(sequences, *maxlen*=200)

labels = (df['sentiment'] == 'positive').astype(int)

\

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, *test\_size*=0.2, *random\_state*=42)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, *test\_size*=0.2, *random\_state*=42)

# Step 3: Building the sequential neural network model without regularization

model\_no\_reg = Sequential()

model\_no\_reg.add(Embedding(10000, 32, *input\_length*=200))

model\_no\_reg.add(Flatten())

model\_no\_reg.add(Dense(32, *activation*='relu'))

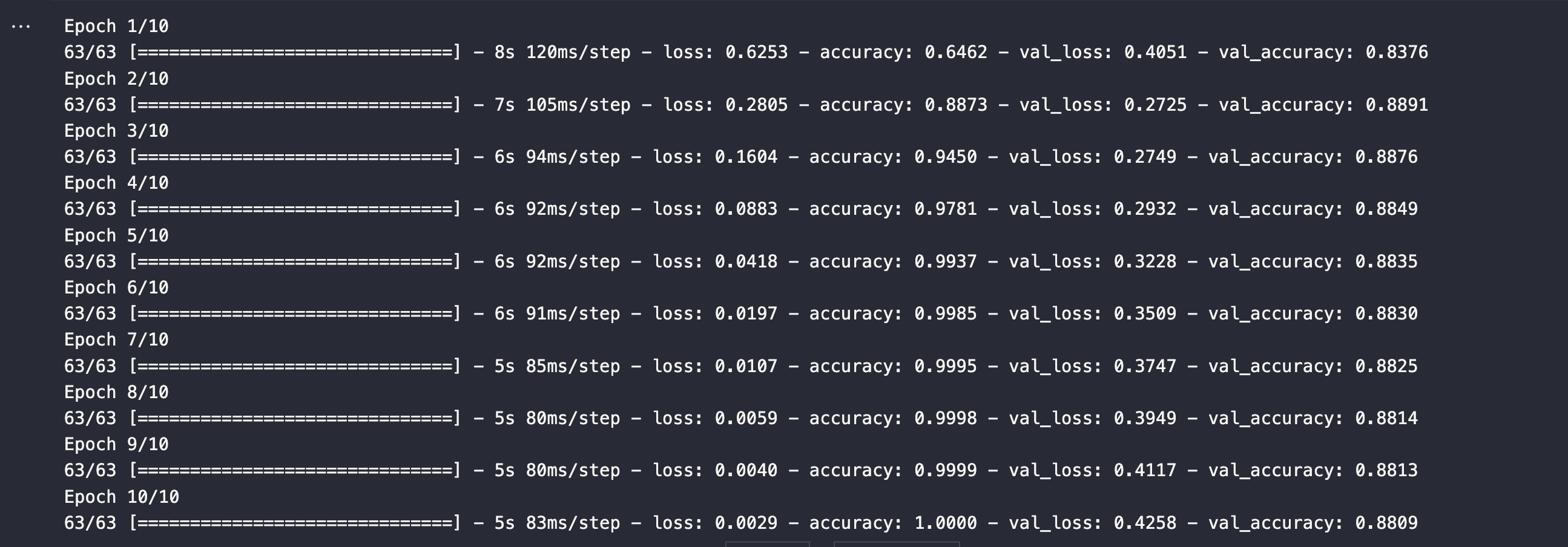
model\_no\_reg.add(Dense(1, *activation*='sigmoid'))

# Step 4: Compile and fit the model to the training dataset

model\_no\_reg.compile(*optimizer*='adam', *loss*='binary\_crossentropy', *metrics*=['accuracy'])

early\_stopping = EarlyStopping(*monitor*='val\_loss', *patience*=3, *restore\_best\_weights*=True)

history\_no\_reg = model\_no\_reg.fit(X\_train, y\_train, *epochs*=20, *batch\_size*=512, *validation\_data*=(X\_val, y\_val), *callbacks*=[early\_stopping])



# Step 5: Building the sequential neural network model with L2 regularization

model\_with\_reg = Sequential()

model\_with\_reg.add(Embedding(10000, 32, *input\_length*=200))

model\_with\_reg.add(Flatten())

model\_with\_reg.add(Dense(32, *activation*='relu', *kernel\_regularizer*=regularizers.l2(0.01)))

model\_with\_reg.add(Dense(1, *activation*='sigmoid'))

model\_with\_reg.compile(*optimizer*='adam', *loss*='binary\_crossentropy', *metrics*=['accuracy'])

history\_with\_reg = model\_with\_reg.fit(X\_train, y\_train, *epochs*=20, *batch\_size*=512, *validation\_data*=(X\_val, y\_val), *callbacks*=[early\_stopping])

A screenshot of a computer

Description automatically generated

# Plotting training and validation loss for models without and with regularization

plt.figure(*figsize*=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(history\_no\_reg.history['loss'], *label*='No Regularization - Training Loss')

plt.plot(history\_no\_reg.history['val\_loss'], *label*='No Regularization - Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Model without Regularization')

plt.legend()

A graph of a line

Description automatically generated with medium confidence A graph of loss and loss

Description automatically generated

**B.3 Observations and learning:**

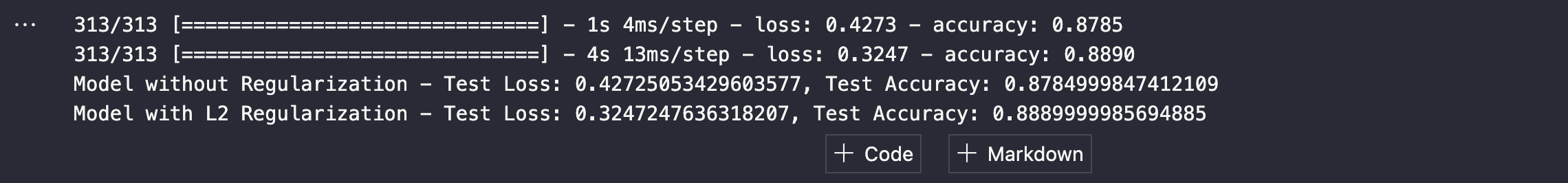
# Evaluate models on the test set

loss\_no\_reg, accuracy\_no\_reg = model\_no\_reg.evaluate(X\_test, y\_test)

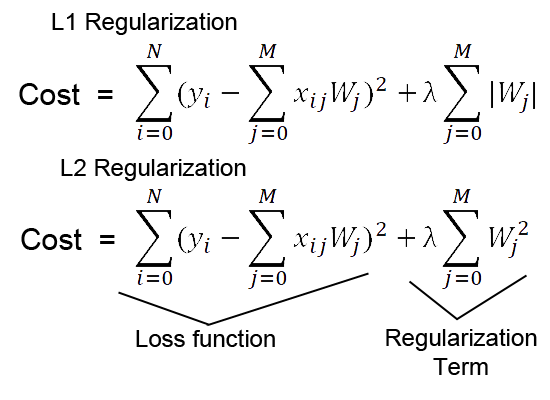
loss\_with\_reg, accuracy\_with\_reg = model\_with\_reg.evaluate(X\_test, y\_test)

print(f'Model without Regularization - Test Loss: {loss\_no\_reg}, Test Accuracy: {accuracy\_no\_reg}')

print(f'Model with L2 Regularization - Test Loss: {loss\_with\_reg}, Test Accuracy: {accuracy\_with\_reg}')

****

In examining the test results, it appears that the model without regularization yielded a slightly lower accuracy of 87.85% and a higher loss of 0.4273 compared to the model with L2 regularization, which achieved an accuracy of 88.90% and a lower loss of 0.3247.



**B.4 Conclusion:**

Through this experiment, it's evident that applying L2 regularization enhanced the model's performance by reducing loss and slightly improving accuracy. Regularization techniques like L2 regularization can effectively mitigate overfitting and enhance generalization capabilities, particularly in neural network models trained on datasets like the IMDB sentiment analysis dataset.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*