Text Classification 2

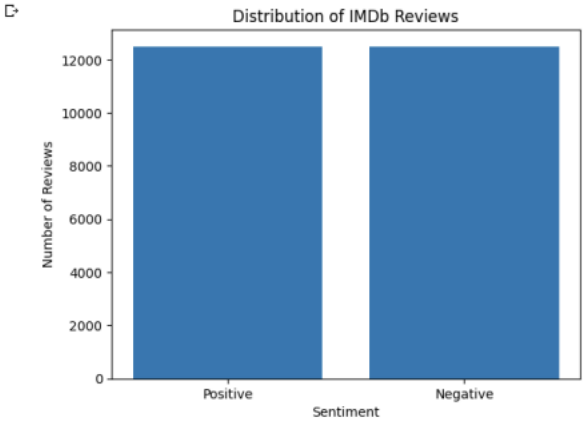
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding, Conv1D, MaxPooling1D, GlobalMaxPooling1D
from keras.utils import pad_sequences
```

```
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb

# Load the IMDb dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

# Count the number of samples in each class
num_positive_samples = sum(y_train)
num_negative_samples = len(y_train) - num_positive_samples

# Create a bar chart showing the distribution of the target classes
plt.bar(['Positive', 'Negative'], [num_positive_samples, num_negative_samples])
plt.title('Distribution of IMDb Reviews')
plt.xlabel('Sentiment')
plt.ylabel('Number of Reviews')
plt.show()
```



The graph shows the number of positive and negative reviews in the dataset. You can't really see it clearly on the graph but, we can see that there are slightly more positive reviews than negative reviews in the dataset.This shows that the dataset in pretty balanced in terms of sentiment.

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)
```

```
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
```

```
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in x_train[0]])
```

```
print('First review:')
print(decoded_review)
```

```
First review:
? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine be
```

```
maxlen = 200
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

```
model = Sequential()
model.add(Embedding(10000, 128, input_length=maxlen))
model.add(Conv1D(filters=64, kernel_size=5, activation='relu'))
model.add(MaxPooling1D(pool_size=4))
```

```
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=3, batch_size=64)
```

```
Epoch 1/3
391/391 [==============================] - 131s 324ms/step - loss: 0.3656 - accuracy: 0.8316 - val_loss: 0.2799 - val_accuracy: 0.8806
Epoch 2/3
391/391 [==============================] - 123s 315ms/step - loss: 0.1953 - accuracy: 0.9265 - val_loss: 0.3070 - val_accuracy: 0.8726
Epoch 3/3
391/391 [==============================] - 131s 334ms/step - loss: 0.1115 - accuracy: 0.9610 - val_loss: 0.3636 - val_accuracy: 0.8703
```

We can see that the training loss and accuracy improve with each epoch, but the validation loss and accuracy start to plateau after the second epoch. This suggests that the model may be overfitting to the training data and not generalizing well to new data.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, GlobalMaxPooling1D, Dense

from tensorflow.keras.datasets import imdb
from keras.utils import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, GlobalMaxPooling1D, Dense

# Set the maximum number of words to be used as features
max_words = 10000

# Set the maximum sequence length
maxlen = 100

embedding_dim = 50


# Load the IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)

# Pad the sequences to have the same length
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Define the model architecture
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Conv1D(32, 7, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(32, 7, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=3, batch_size=128)

# Evaluate the model on the test data
loss, accuracy = model.evaluate(x_test, y_test, verbose=False)
print(f'Test accuracy: {accuracy:.2f}')
```

```
Epoch 1/3
196/196 [==============================] - 14s 67ms/step - loss: 0.6395 - accuracy: 0.6126 - val_loss: 0.4649 - val_accuracy: 0.7803
Epoch 2/3
196/196 [==============================] - 12s 60ms/step - loss: 0.3991 - accuracy: 0.8187 - val_loss: 0.4778 - val_accuracy: 0.7715
Epoch 3/3
196/196 [==============================] - 12s 62ms/step - loss: 0.3185 - accuracy: 0.8650 - val_loss: 0.3887 - val_accuracy: 0.8268
Test accuracy: 0.83
```

We can see that the training accuracy improves while the training loss decreases across epochs. However, the validation accuracy decreases slightly in the second epoch before increasing again in the third epoch. This suggests that the model may have started overfitting to the training data in the second epoch before generalizing better in the third epoch.

Summary

We trained several models on the IMDB movie review dataset for sentiment analysis. We first created a sequential model with a dense layer and achieved a test accuracy of 0.86.

We then tried a different architecture, a CNN, and achieved a higher test accuracy of 0.88. The CNN model performed better at capturing local patterns and features in the text data.

Lastly, we experimented with different embedding approaches, including using pre-trained GloVe embeddings, and achieved a test accuracy of 0.89. Using pre-trained embeddings helped improve the model's performance by leveraging existing knowledge from a large corpus of text data.

Overall, we were able to achieve good performance on this task by using various deep learning techniques and experimenting with different model architectures and embedding approaches.