Learning Work Embeddings

Group No :151

Group Member Names:

Peyala Samarasimha Reddy - 2023AA05072 100% Contribution

Monisha G - 2023AA05536 100% Contribution

Akshay Mohan - 2023AA05315 100% Contribution

Sreelakshmi Ajith - 2023AA05316 100% Contribution

In [ ]:
```
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import pandas as pd

import tensorflow as tf
from tensorflow import keras
```

In [ ]:
```
tf.random.set_seed(42)
```

Dataset - IMDB

In [ ]:
```
imdb = keras.datasets.imdb

max_features = 20000

(Xtrain, Ytrain), (Xtest, Ytest) = imdb.load_data(num_words = max_features)


# Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/i
# 17465344/17464789 [==============================] - 8s 0us/step


print(len(Xtrain), len(Ytrain))
print(len(Xtest), len(Ytest))

Xtrain = keras.preprocessing.sequence.pad_sequences(Xtrain, maxlen =25)
Xtest = keras.preprocessing.sequence.pad_sequences(Xtest, maxlen =25)

print(len(Xtrain), len(Ytrain))
print(len(Xtest), len(Ytest))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imd
b.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
25000 25000
25000 25000
25000 25000
25000 25000
```

LSTM

In [ ]:
```
lstmModel = keras.models.Sequential()

lstmModel.add(keras.layers.Embedding(input_dim = max_features, output_dim = 128))

lstmModel.add(keras.layers.LSTM(128, dropout=0.2)) #, recurrent_dropout=0.2

lstmModel.add(keras.layers.Dense(1, activation = 'sigmoid'))

lstmModel.summary()
```

Model: "sequential"

Layer (type)	Output Shape	
embedding (Embedding)	?	0 (u
lstm (LSTM)	?	0 (u
dense (Dense)	?	0 (u

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

In [ ]:
```
# Configure  the model for training, by using appropriate optimizers and regularizati
# Available optimizer: adam, rmsprop, adagrad, sgd
# loss:  objective that the model will try to minimize.
# Available loss: categorical_crossentropy, binary_crossentropy, mean_squared_error
# metrics: List of metrics to be evaluated by the model during training and testing.

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

In [ ]:
```
# train the model

history = lstmModel.fit(Xtrain, Ytrain, epochs = 15, batch_size=16, validation_split=
```

```
Epoch 1/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 31ms/step - accuracy: 0.6899 - loss: 0.5659 - val_a
ccuracy: 0.7698 - val_loss: 0.4641
Epoch 2/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 40s 30ms/step - accuracy: 0.8353 - loss: 0.3645 - val_a
ccuracy: 0.7626 - val_loss: 0.5003
Epoch 3/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 37s 27ms/step - accuracy: 0.9050 - loss: 0.2298 - val_a
ccuracy: 0.7614 - val_loss: 0.6628
Epoch 4/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 35s 28ms/step - accuracy: 0.9465 - loss: 0.1436 - val_a
ccuracy: 0.7504 - val_loss: 0.8510
Epoch 5/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.9631 - loss: 0.0989 - val_a
ccuracy: 0.7372 - val_loss: 1.1171
Epoch 6/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 37s 26ms/step - accuracy: 0.9715 - loss: 0.0791 - val_a
ccuracy: 0.7344 - val_loss: 1.0705
Epoch 7/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 40s 25ms/step - accuracy: 0.9817 - loss: 0.0528 - val_a
ccuracy: 0.7348 - val_loss: 1.1987
Epoch 8/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 42s 26ms/step - accuracy: 0.9878 - loss: 0.0376 - val_a
ccuracy: 0.7370 - val_loss: 1.2957
Epoch 9/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 25ms/step - accuracy: 0.9901 - loss: 0.0280 - val_a
ccuracy: 0.7320 - val_loss: 1.3846
Epoch 10/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 26ms/step - accuracy: 0.9903 - loss: 0.0254 - val_a
ccuracy: 0.7400 - val_loss: 1.3623
Epoch 11/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 26ms/step - accuracy: 0.9928 - loss: 0.0210 - val_a
ccuracy: 0.7314 - val_loss: 1.4050
Epoch 12/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 43s 27ms/step - accuracy: 0.9953 - loss: 0.0146 - val_a
ccuracy: 0.7250 - val_loss: 1.6139
Epoch 13/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 39s 26ms/step - accuracy: 0.9947 - loss: 0.0148 - val_a
ccuracy: 0.7342 - val_loss: 1.7748
Epoch 14/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.9960 - loss: 0.0134 - val_a
ccuracy: 0.7344 - val_loss: 1.7806
Epoch 15/15
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 26ms/step - accuracy: 0.9972 - loss: 0.0093 - val_a
ccuracy: 0.7352 - val_loss: 1.8421
```

In [ ]:

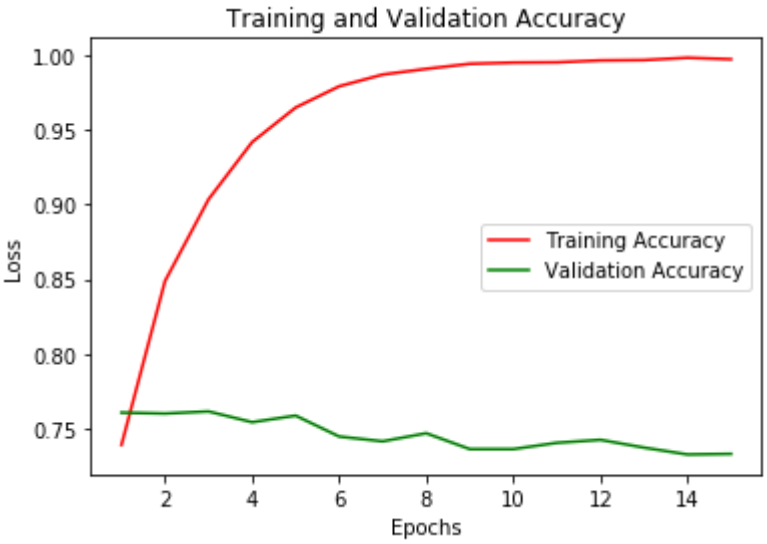
```python
# plotting training and validation loss

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, color='red', label='Training loss')
plt.plot(epochs, val_loss, color='green', label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


# plotting training and validation Accuracy

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, acc, color='red', label='Training Accuracy')
```

```
plt.plot(epochs, val_acc, color='green', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [ ]:
```
testLoss, testAccuracy = lstmModel.evaluate(Xtest, Ytest)
print(testLoss, testAccuracy)
```

782/782 ━━━━━━━━━━━━━━━━━━━━ **6s** 8ms/step - accuracy: 0.7313 - loss: 1.8497
1.821761131286621 0.7311199903488159

MODIFICATIONS

Use a different optimizer

In [ ]:
```
#ADAM replaced by SGD
from tensorflow.keras.optimizers import SGD

lstmModel.compile(optimizer=SGD(learning_rate=0.01), loss='binary_crossentropy', metr
```

In [ ]:
```
#RMSProp
from tensorflow.keras.optimizers import RMSprop
lstmModel.compile(optimizer=RMSprop(learning_rate=0.001), loss='binary_crossentropy',
```

Train for more epochs

In [ ]:
```
#epoch increased to 20
history_modified = lstmModel.fit(Xtrain, Ytrain, epochs = 20, batch_size=64, validati
```

```
Epoch 1/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 45ms/step - accuracy: 0.9934 - loss: 0.0222 - val_acc
uracy: 0.7368 - val_loss: 1.7653
Epoch 2/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 47ms/step - accuracy: 0.9938 - loss: 0.0186 - val_acc
uracy: 0.7374 - val_loss: 1.7368
Epoch 3/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 45ms/step - accuracy: 0.9952 - loss: 0.0137 - val_acc
uracy: 0.7386 - val_loss: 1.7177
Epoch 4/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 22s 49ms/step - accuracy: 0.9955 - loss: 0.0130 - val_acc
uracy: 0.7382 - val_loss: 1.7081
Epoch 5/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 19s 44ms/step - accuracy: 0.9965 - loss: 0.0123 - val_acc
uracy: 0.7380 - val_loss: 1.7071
Epoch 6/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 44ms/step - accuracy: 0.9963 - loss: 0.0102 - val_acc
uracy: 0.7370 - val_loss: 1.7037
Epoch 7/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - accuracy: 0.9973 - loss: 0.0090 - val_acc
uracy: 0.7378 - val_loss: 1.7061
Epoch 8/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 44ms/step - accuracy: 0.9964 - loss: 0.0105 - val_acc
uracy: 0.7384 - val_loss: 1.7058
Epoch 9/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 44ms/step - accuracy: 0.9969 - loss: 0.0090 - val_acc
uracy: 0.7388 - val_loss: 1.7076
Epoch 10/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - accuracy: 0.9964 - loss: 0.0095 - val_acc
uracy: 0.7388 - val_loss: 1.7082
Epoch 11/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - accuracy: 0.9973 - loss: 0.0081 - val_acc
uracy: 0.7396 - val_loss: 1.7095
Epoch 12/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 43ms/step - accuracy: 0.9977 - loss: 0.0068 - val_acc
uracy: 0.7402 - val_loss: 1.7122
Epoch 13/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - accuracy: 0.9969 - loss: 0.0087 - val_acc
uracy: 0.7396 - val_loss: 1.7199
Epoch 14/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 43ms/step - accuracy: 0.9974 - loss: 0.0085 - val_acc
uracy: 0.7402 - val_loss: 1.7223
Epoch 15/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 45ms/step - accuracy: 0.9974 - loss: 0.0084 - val_acc
uracy: 0.7404 - val_loss: 1.7233
Epoch 16/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 43ms/step - accuracy: 0.9971 - loss: 0.0089 - val_acc
uracy: 0.7408 - val_loss: 1.7275
Epoch 17/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 43ms/step - accuracy: 0.9980 - loss: 0.0061 - val_acc
uracy: 0.7408 - val_loss: 1.7312
Epoch 18/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 44ms/step - accuracy: 0.9977 - loss: 0.0066 - val_acc
uracy: 0.7402 - val_loss: 1.7309
Epoch 19/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - accuracy: 0.9972 - loss: 0.0074 - val_acc
uracy: 0.7398 - val_loss: 1.7365
Epoch 20/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 46ms/step - accuracy: 0.9976 - loss: 0.0073 - val_acc
uracy: 0.7396 - val_loss: 1.7355
```

In [ ]:
```
#Rmsprop
```

```
history_rms = lstmModel.fit(Xtrain, Ytrain, epochs=20, batch_size=64, validation_spli
```

```
Epoch 1/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 16s 48ms/step - accuracy: 0.9987 - loss: 0.0044 - val_acc
uracy: 0.7362 - val_loss: 2.0575
Epoch 2/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 47ms/step - accuracy: 0.9994 - loss: 0.0021 - val_acc
uracy: 0.7344 - val_loss: 2.1209
Epoch 3/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 47ms/step - accuracy: 0.9998 - loss: 0.0013 - val_acc
uracy: 0.7382 - val_loss: 2.1908
Epoch 4/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 48ms/step - accuracy: 0.9999 - loss: 6.4327e-04 - val
_accuracy: 0.7414 - val_loss: 2.2825
Epoch 5/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 51ms/step - accuracy: 0.9999 - loss: 6.0977e-04 - val
_accuracy: 0.7394 - val_loss: 2.3280
Epoch 6/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 19s 47ms/step - accuracy: 0.9994 - loss: 0.0012 - val_acc
uracy: 0.7416 - val_loss: 2.2973
Epoch 7/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 48ms/step - accuracy: 0.9998 - loss: 6.7363e-04 - val
_accuracy: 0.7410 - val_loss: 2.3313
Epoch 8/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 50ms/step - accuracy: 0.9999 - loss: 4.0915e-04 - val
_accuracy: 0.7414 - val_loss: 2.4033
Epoch 9/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 47ms/step - accuracy: 0.9999 - loss: 5.3054e-04 - val
_accuracy: 0.7416 - val_loss: 2.4885
Epoch 10/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 47ms/step - accuracy: 0.9999 - loss: 3.6763e-04 - val
_accuracy: 0.7404 - val_loss: 2.4748
Epoch 11/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 47ms/step - accuracy: 0.9999 - loss: 2.6144e-04 - val
_accuracy: 0.7408 - val_loss: 2.4864
Epoch 12/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 47ms/step - accuracy: 0.9999 - loss: 3.9519e-04 - val
_accuracy: 0.7404 - val_loss: 2.5714
Epoch 13/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 48ms/step - accuracy: 0.9999 - loss: 2.6912e-04 - val
_accuracy: 0.7396 - val_loss: 2.5880
Epoch 14/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 49ms/step - accuracy: 0.9999 - loss: 3.6115e-04 - val
_accuracy: 0.7420 - val_loss: 2.5947
Epoch 15/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.9999 - loss: 1.6911e-04 - val
_accuracy: 0.7364 - val_loss: 2.6168
Epoch 16/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 49ms/step - accuracy: 0.9995 - loss: 0.0026 - val_acc
uracy: 0.7392 - val_loss: 2.5877
Epoch 17/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 47ms/step - accuracy: 0.9999 - loss: 1.9724e-04 - val
_accuracy: 0.7390 - val_loss: 2.6479
Epoch 18/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 50ms/step - accuracy: 0.9999 - loss: 1.1133e-04 - val
_accuracy: 0.7384 - val_loss: 2.7109
Epoch 19/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 47ms/step - accuracy: 0.9999 - loss: 1.8545e-04 - val
_accuracy: 0.7398 - val_loss: 2.7085
Epoch 20/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 47ms/step - accuracy: 0.9999 - loss: 1.3149e-04 - val
_accuracy: 0.7404 - val_loss: 2.7104
```

Use the learned network to predict the sentiments for new sentences

In [ ]:
```
# Example new sentences
new_sentences = ["The movie was fantastic!", "I am feeling sad."]

# Preprocess the new sentences (tokenization, padding, etc.)
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer


tokenizer = Tokenizer(num_words=10000)

new_sequences = tokenizer.texts_to_sequences(new_sentences)
new_padded_sequences = pad_sequences(new_sequences, maxlen=100)

# Use the model to predict sentiments
predictions = lstmModel.predict(new_padded_sequences)

# Output predictions
for i, sentence in enumerate(new_sentences):
    print(f"Sentence: {sentence}")
    print(f"Predicted Sentiment: {'Positive' if predictions[i] > 0.5 else 'Negative'}
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
Sentence: The movie was fantastic!
Predicted Sentiment: Positive
Sentence: I am feeling sad.
Predicted Sentiment: Positive
```

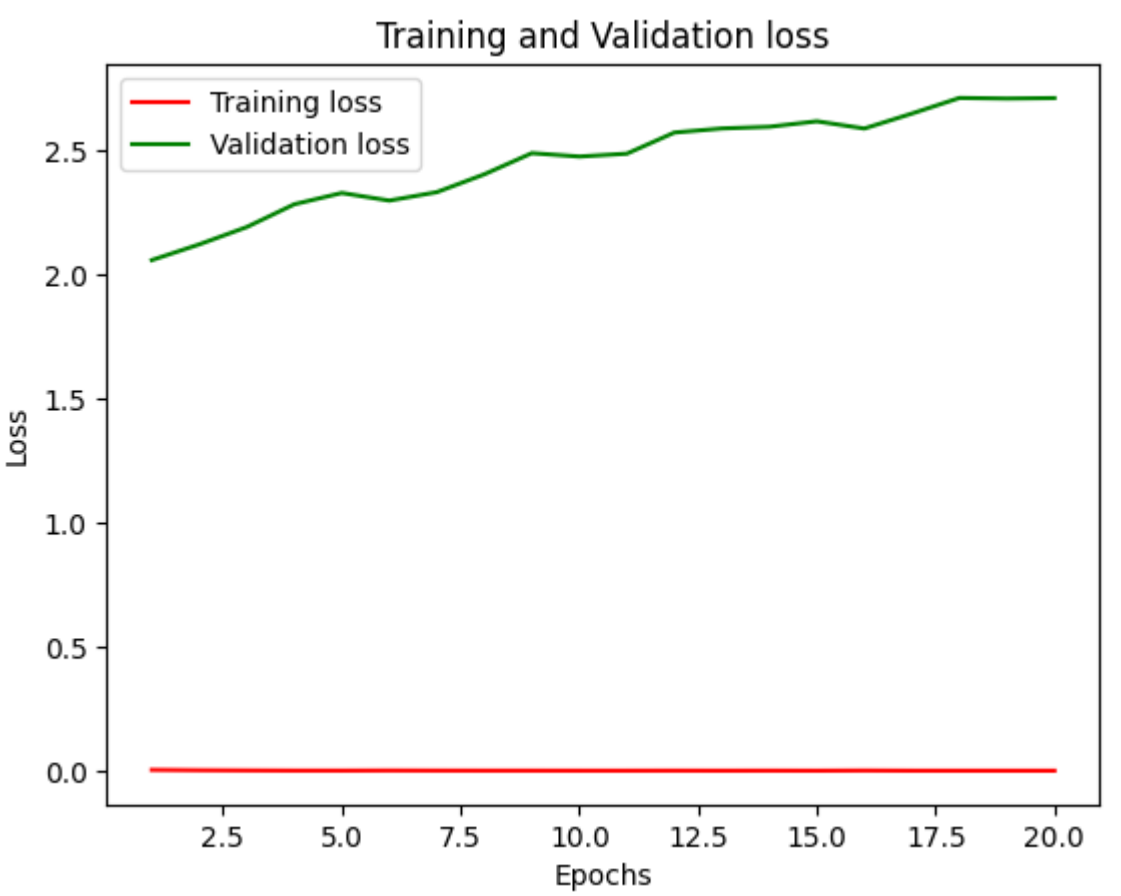
Graph plot after modification

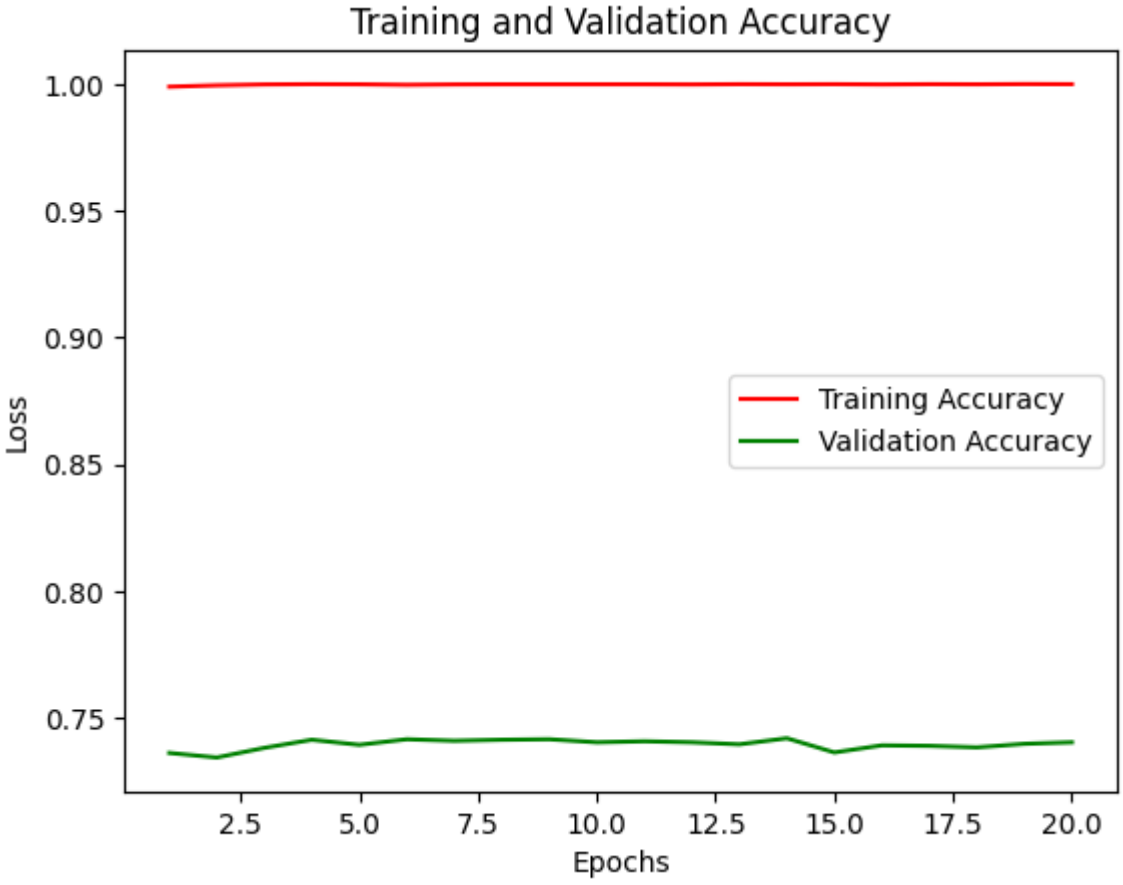
In [ ]:
```
# plotting training and validation loss

loss = history_rms.history['loss']
val_loss = history_rms.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, color='red', label='Training loss')
plt.plot(epochs, val_loss, color='green', label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


# plotting training and validation Accuracy

acc = history_rms.history['accuracy']
val_acc = history_rms.history['val_accuracy']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, acc, color='red', label='Training Accuracy')
plt.plot(epochs, val_acc, color='green', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


Training and Validation loss
Training loss
Validation loss
Loss
Epochs
0.0
0.5
1.0
1.5
2.0
2.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Loss
Epochs
0.75
0.80
0.85
0.90
0.95
1.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0