```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense
from keras.preprocessing import sequence
```

Create and investigate a SimpleRNN. The model stacks several recurrent layers one after the other in order to increase the representatio
power of the network. All intermediate layers return a full sequence of successive outputs for each timestamp. The final layer returns th
at the last timestamp (contains info of all timestamps from 0 to t).

```
model = Sequential()
model.add(Embedding(10000,32))
model.add(SimpleRNN(32,return_sequences=True))
model.add(SimpleRNN(32,return_sequences=True))
model.add(SimpleRNN(32,return_sequences=True))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_5 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_6 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_7 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_8 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```

Apply the SimpleRNN model to classify IMDB movie reviews as positive (1) or negative (0).

```
from keras.datasets import imdb
```

Include only the top 10,000 most frequently occuring words in the reviews.

```
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
Downloading data from https://s3.amazonaws.com/text-datasets/imdb.npz
17465344/17464789 [==============================] - 1s 0us/step
```

```
train_data.shape
```

```
(25000,)
```

Investigate the data by decode the first review back to English words:

```
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 train_data[0]])
print(decoded_review)
```

```
? this film was just brilliant casting location scenery story direction everyone's really suited the part they played a
```

train_labels and test_labels are lists of 0s and 1s indicating which reviews are positive (1) and which are negative (0). The labels are sca

```
print(train_labels[0])
```

```
1
```

train_data and test_data are lists of reviews; each review is a list of word indices (specifies which word, lower the index higher the frequ

```
print(train_data[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670,
```

Transforms data into a 2D Numpy array of shape (10000, 500). Consider 10,000 features, each feature up to 500 of the most common w

```
train_data = sequence.pad_sequences(train_data, maxlen=500)
test_data = sequence.pad_sequences(test_data, maxlen=500)
```

Train the model with Embedding and SimpleRNN layers.

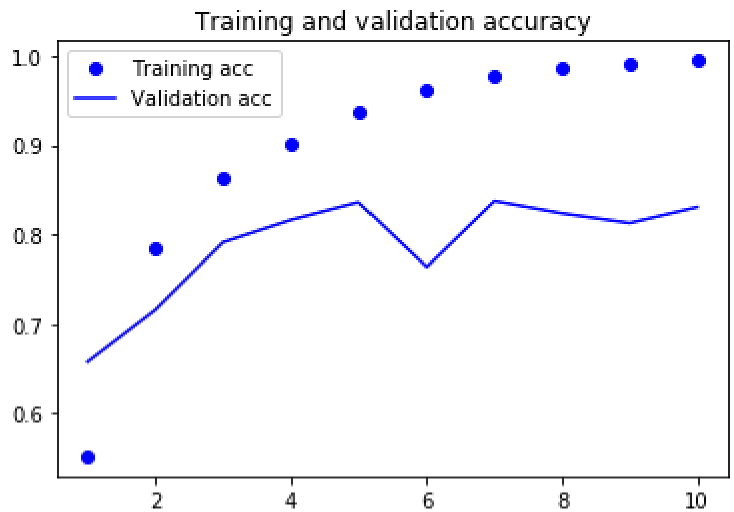
```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(train_data, train_labels, epochs=10, batch_size=128, validation_split=0.2)
```

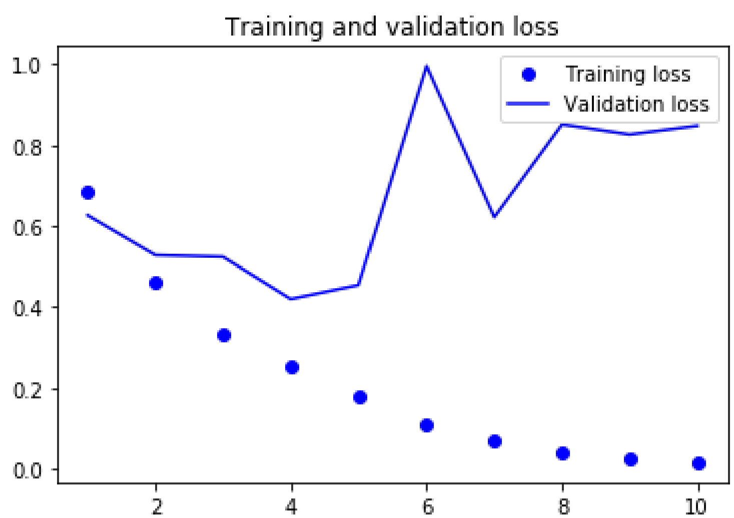
```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 132s 7ms/step - loss: 0.6830 - acc: 0.5517 - val_loss: 0.6264 - val_acc:
Epoch 2/10
20000/20000 [==============================] - 131s 7ms/step - loss: 0.4626 - acc: 0.7853 - val_loss: 0.5284 - val_acc:
Epoch 3/10
20000/20000 [==============================] - 130s 6ms/step - loss: 0.3313 - acc: 0.8641 - val_loss: 0.5242 - val_acc:
Epoch 4/10
20000/20000 [==============================] - 129s 6ms/step - loss: 0.2523 - acc: 0.9016 - val_loss: 0.4199 - val_acc:
Epoch 5/10
20000/20000 [==============================] - 129s 6ms/step - loss: 0.1784 - acc: 0.9375 - val_loss: 0.4543 - val_acc:
Epoch 6/10
20000/20000 [==============================] - 128s 6ms/step - loss: 0.1092 - acc: 0.9624 - val_loss: 0.9958 - val_acc:
Epoch 7/10
20000/20000 [==============================] - 128s 6ms/step - loss: 0.0685 - acc: 0.9783 - val_loss: 0.6217 - val_acc:
Epoch 8/10
20000/20000 [==============================] - 127s 6ms/step - loss: 0.0381 - acc: 0.9866 - val_loss: 0.8509 - val_acc:
Epoch 9/10
20000/20000 [==============================] - 127s 6ms/step - loss: 0.0261 - acc: 0.9914 - val_loss: 0.8263 - val_acc:
Epoch 10/10
20000/20000 [==============================] - 128s 6ms/step - loss: 0.0160 - acc: 0.9951 - val_loss: 0.8476 - val_acc:
```

Display the training and validation loss accuracy.

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
```

```
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





Unfortunately, the recurrent network does not perform well - it achieves a maximum validation accuracy of only 83.74% during the 7th e
seems the SimpleRNN model is too simplistic to be of real use. Indeed, the SimpleRNN network has a major issue: it is unable to mainta
learn long-term dependencies. In other words, the SimpleRNN suffers from the vanishing gradient problem, where the network becomes
increasingly untrainable as layers are added to the network.