Practical 7

Aim: Write a program to implement a simple form of a recurrent neural network.

a. **E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day**
b. **LSTM for sentiment analysis on datasets like UMICH SI650 for similar.**

Code: Part A

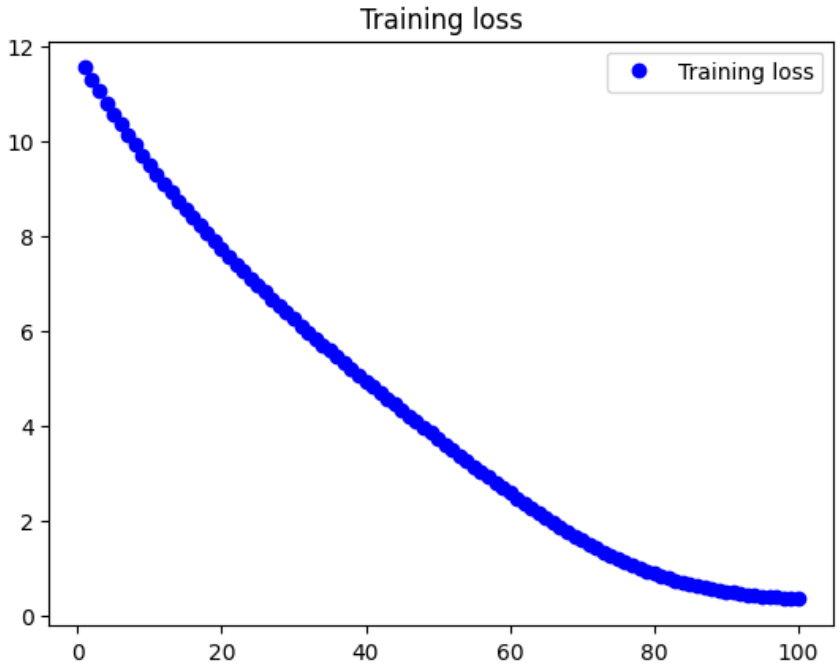
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

```
rain_data = np.array([2.3, 1.5, 3.1, 2.0, 2.5, 1.7, 2.9, 3.5, 3.0, 2.1,
2.5, 2.2, 2.8, 3.2, 1.8, 2.7, 1.9, 3.1, 3.3, 2.0,
2.5, 2.2, 2.4, 3.0, 2.1, 2.5, 3.2, 3.1, 1.9, 2.7,
2.2, 2.8, 3.1, 2.0, 2.5, 1.7, 2.9, 3.5, 3.0, 2.1,
2.5, 2.2, 2.8, 3.2, 1.8, 2.7, 1.9, 3.1, 3.3, 2.0])

def create_sequences(values, time_steps):
  x = []
  y = []
  for i in range(len(values)-time_steps):
    x.append(values[i:i+time_steps])
    y.append(values[i+time_steps])
  return np.array(x), np.array(y)
```

```
time_steps = 4
x_train, y_train = create_sequences(rain_data, time_steps)

model = tf.keras.models.Sequential([tf.keras.layers.SimpleRNN(8,
input_shape=(time_steps, 1)),tf.keras.layers.Dense(1)])
model.compile(optimizer="adam", loss="mse")

history = model.fit(x_train.reshape(-1, time_steps, 1), y_train,
epochs=100)

loss = history.history["loss"]
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, "bo", label="Training loss")
plt.title("Training loss")
plt.legend()
plt.show()
```

Epoch 95/100

```
2/2 [==============================] - 0s 7ms/step - loss: 0.4141
Epoch 96/100
2/2 [==============================] - 0s 8ms/step - loss: 0.3994
Epoch 97/100
2/2 [==============================] - 0s 8ms/step - loss: 0.3874
Epoch 98/100
2/2 [==============================] - 0s 8ms/step - loss: 0.3746
Epoch 99/100
2/2 [==============================] - 0s 8ms/step - loss: 0.3632
Epoch 100/100
2/2 [==============================] - 0s 8ms/step - loss: 0.3546
```




```
test_sequence = np.array([2.5, 2.2, 2.8, 3.2])
x_test = np.array([test_sequence])
y_test = model.predict(x_test.reshape(-1, time_steps, 1))

print("Previous days' rain data:", test_sequence)
print("Expected rain amount for next day:", y_test[0][0])
prediction = model.predict(np.array([test_sequence]).reshape(1,
time_steps, 1))
print("Prediction:", prediction[0][0])
```

```
1/1 [==============================] - 0s 242ms/step
Previous days' rain data: [2.5 2.2 2.8 3.2]
Expected rain amount for next day: 2.2730155
1/1 [==============================] - 0s 27ms/step
Prediction: 2.2730155
```

Code: Part B

```
from keras.layers import Activation, Dense, Dropout, SpatialDropout1D
from keras.layers import Embedding
from keras.layers import LSTM
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
import collections
import matplotlib.pyplot as plt
import nltk
import os
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
```

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

x_train =train['sentence']
y_train =train['label']

vectorizer = CountVectorizer()
x_train = vectorizer.fit_transform(train.sentence)
x_test = vectorizer.transform(test.sentence)

word_freq = pd.DataFrame({'Word': vectorizer.get_feature_names_out(),
'Count': x_train.toarray().sum(axis=0)})
word_freq['Frequency'] = word_freq['Count'] / word_freq['Count'].sum()

word_freq_sort = word_freq.sort_values(by='Frequency', ascending=False)
word_freq_sort.head(10)

maxlen = 0
word_freqs = collections.Counter()
num_recs = 0

nltk.download('punkt')

f = open("train.csv", "rb")
for line in f:
    sentence = str(line).split(",")[0]
    words = nltk.word_tokenize(sentence.encode().decode("ascii",
"ignore").lower())
```

```
    if len(words) > maxlen:
        maxlen = len(words)
    for word in words:
        word_freqs[word] += 1
    num_recs += 1
f.close()
print("Max number of words in a sentence: ", maxlen)
print("Number of unqiue words: ", len(word_freqs))
```

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.

Max number of words in a sentence: 41
Number of unqiue words: 1545

```
MAX_FEATURES = 1500
MAX_SENTENCE_LENGTH = 40
vocab_size = min(MAX_FEATURES, len(word_freqs)) + 2
word2index = {x[0]: i+2 for i, x in
enumerate(word_freqs.most_common(MAX_FEATURES))}
word2index["PAD"] = 0
word2index["UNK"] = 1
index2word = {v:k for k, v in word2index.items()}

X = np.empty((num_recs, ), dtype=list)
Xt = []
y = np.zeros((num_recs, ))
i = 0
```

```
f = open("train.csv", "r")
next(f)
fixed = []
for line in f:
    line_arr = line.split(",")
    if len(line_arr) > 2:
        line_arr[:len(line_arr)-1] =
[",".join(line_arr[:len(line_arr)-1])]
        fixed.append(line_arr)
    else:
        fixed.append(line_arr)

for fline in fixed:
        sentence = fline[0]
        label = int(fline[1])
```

```
        words = nltk.word_tokenize(sentence.encode().decode("ascii",
"ignore").lower())
        seqs = []
        for word in words:
            if word in word2index:
                seqs.append(word2index[word])
            else:
                seqs.append(word2index["UNK"])
        X[i] = seqs
        y[i] = int(label)
        i += 1
f.close()
```

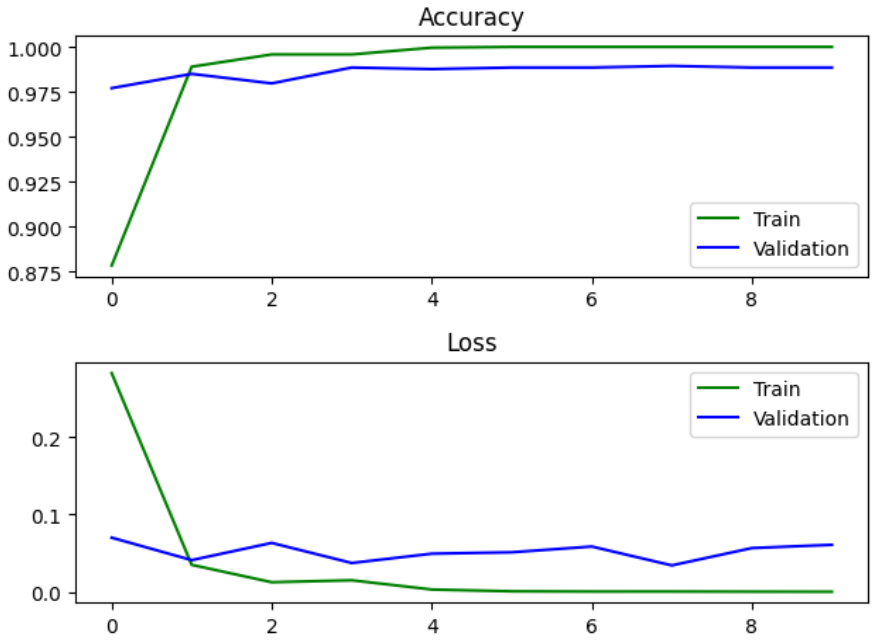
```
# Convert every occurence of None to an empty list

for i in range(0,X.shape[0]):
    if X[i] == None:
        X[i] = ()

X = sequence.pad_sequences(X, maxlen=MAX_SENTENCE_LENGTH)

# Get shape of X
X.shape

X_train, X_test, Y_train, Y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
EMBEDDING_SIZE = 128
HIDDEN_LAYER_SIZE = 64
BATCH_SIZE = 32
NUM_EPOCHS = 10

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_SIZE,
input_length=MAX_SENTENCE_LENGTH))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(HIDDEN_LAYER_SIZE, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1))
model.add(Activation("sigmoid"))

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

```
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE,
epochs=NUM_EPOCHS, validation_data=(X_test, Y_test))

# Plot loss and accuracy values over time
plt.subplot(211)
plt.title("Accuracy")
plt.plot(history.history["accuracy"], color="g", label="Train")
plt.plot(history.history["val_accuracy"], color="b",
label="Validation")
plt.legend(loc="best")

plt.subplot(212)
plt.title("Loss")
plt.plot(history.history["loss"], color="g", label="Train")
plt.plot(history.history["val_loss"], color="b", label="Validation")
plt.legend(loc="best")

plt.tight_layout()
plt.show()

score, acc = model.evaluate(X_test, Y_test, batch_size=BATCH_SIZE)
print("Test score: %.3f, accuracy: %.3f" % (score, acc))
```

```
Epoch 1/10
142/142 [==============================] - 29s 162ms/step - loss: 0.2826 -
accuracy: 0.8785 - val_loss: 0.0700 - val_accuracy: 0.9771
Epoch 2/10
142/142 [==============================] - 11s 77ms/step - loss: 0.0349 - accuracy:
0.9890 - val_loss: 0.0410 - val_accuracy: 0.9850
Epoch 3/10
142/142 [==============================] - 12s 85ms/step - loss: 0.0125 - accuracy:
0.9958 - val_loss: 0.0632 - val_accuracy: 0.9797
Epoch 4/10
142/142 [==============================] - 12s 85ms/step - loss: 0.0150 - accuracy:
0.9958 - val_loss: 0.0373 - val_accuracy: 0.9885
Epoch 5/10
142/142 [==============================] - 12s 83ms/step - loss: 0.0030 - accuracy:
0.9996 - val_loss: 0.0493 - val_accuracy: 0.9877
Epoch 6/10
142/142 [==============================] - 13s 90ms/step - loss: 7.1040e-04 -
accuracy: 1.0000 - val_loss: 0.0511 - val_accuracy: 0.9885
Epoch 7/10
142/142 [==============================] - 13s 91ms/step - loss: 4.2562e-04 -
accuracy: 1.0000 - val_loss: 0.0586 - val_accuracy: 0.9885
Epoch 8/10
142/142 [==============================] - 10s 72ms/step - loss: 4.4279e-04 -
accuracy: 1.0000 - val_loss: 0.0342 - val_accuracy: 0.9894
Epoch 9/10
142/142 [==============================] - 13s 94ms/step - loss: 2.7130e-04 -
```

accuracy: 1.0000 - val_loss: 0.0566 - val_accuracy: 0.9885
Epoch 10/10
142/142 [==============================] - 12s 84ms/step - loss: 1.4941e-04 - accuracy: 1.0000 - val_loss: 0.0608 - val_accuracy: 0.9885



36/36 [==============================] - 0s 11ms/step - loss: 0.0608 - accuracy: 0.9885
Test score: 0.061, accuracy: 0.989

```
for i in range(5):
    idx = np.random.randint(len(X_test))
    xtest = X_test[idx].reshape(1, 40)
    ylabel = Y_test[idx]
    ypred = model.predict(xtest)[0][0]
    sent = " ".join([index2word[x] for x in xtest[0].tolist() if x != 0])
    print("%.0f - %d - %s" % (ypred, ylabel, sent))
```

1/1 [==============================] - 0s 280ms/step
1 - 1 - mission impossible iii was awesome .
1/1 [==============================] - 0s 23ms/step
1 - 1 - mission impossible 3 was excellent .
1/1 [==============================] - 0s 24ms/step
1 - 1 - love luv lubb the da vinci code !
1/1 [==============================] - 0s 23ms/step
1 - 1 - brokeback mountain was an awesome movie .
1/1 [==============================] - 0s 25ms/step
1 - 1 - i love brokeback mountain ....