BSc in Information Technology
Specialising in Data Science

Semester 2

Lab Exercise 5

IT3071 – Machine Learning and Optimization Methods 2023

Build a LSTM model using tensorlfow for Text Generation

Long short term memory

A memory holding one.
Can predict spellings of a text.

Objective:

- How to frame the problem of text sequences to a recurrent neural network generative model.
- How to develop an LSTM to generate reasonable text sequences for a given problem.

1. Import necessary libraries

2. Load ascii text and covert to lowercase

```
filename = "data.txt"
raw_text = open(filename, 'r', encoding='utf-8').read()
raw_text = raw_text.lower()
```

3. create mapping of unique chars to integers

```
# create mapping of unique chars to integers
chars = sorted(list(set(raw_text)))
print(chars)
char_to_int = dict((c, i) for i, c in enumerate(chars))
```

4. Getting the details of the dataset

```
n_chars = len(raw_text)
n_vocab = len(chars)
print("Total Characters: ", n_chars)
print("Total Vocab(Unique characters): ", n_vocab)
```

5. Prepare the dataset of input to output pairs encoded as integers.

```
# prepare the dataset of input to output pairs encoded as integers
seq_length = 15 #can be changed
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
    seq_in = raw_text[i:i + seq_length]
    seq_out = raw_text[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
n_patterns = len(dataY)
print("Total Patterns: ", n_patterns)
```

6. Transform the list of input sequences into the form [samples, time steps, features] that is expected by an LSTM network and rescale the integers to the range [0,1] to make the patterns easier to learn by the LSTM network that uses the sigmoid activation function by default.

```python
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
# normalize - rescaling the integer values
X = X / float(n_vocab)
print(X)
```

7. Convert the output values (single characters converted to integers) into a one hot encoding.

```python
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
#print(y)
```

8. Define the LSTM model.

```python
# define the LSTM model
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2]))) #It can have 1 or more training samples
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
# define the checkpoint
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]
```

9. Fitting the model to data

```python
# Change the hyperparameter values and train model
epochs = 10
batch_size = 128
```

```python
model.fit(X, y, epochs=epochs, batch_size=batch_size, callbacks=callbacks_list)
```

10. Generate Text with the trained LSTM model

```python
# load the network weights
filename = "weights-improvement-10-2.3100.hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

```python
# generate a random seed
print(len(dataX))
start = numpy.random.randint(0, len(dataX)-1)
print(start)
pattern = dataX[start] #dataX contains list of patterns
print("Seed:")
print("\"", ''.join([int_to_char[value] for value in pattern]), "\"")
```

```python
# generate characters
length = 10
final = []
for i in range(length):
    # reshaping the seed sequence before passing it into the LSTM model
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    #print(x)
    # normalizing the integer values
    x = x / float(n_vocab)
   # print(x)
    # making prediction
    prediction = model.predict(x, verbose=0)
    # Get the predicted value with maximum probability
    index = numpy.argmax(prediction)

    # Convert the predicted integer to char
    result = int_to_char[index]
    #print(result)
    final.append(result)
    # Adding the predicted character to the sequence sequence
    pattern.append(index)
    # Removing the first character from the seed sequence
    pattern = pattern[1:len(pattern)]
print(final)
```