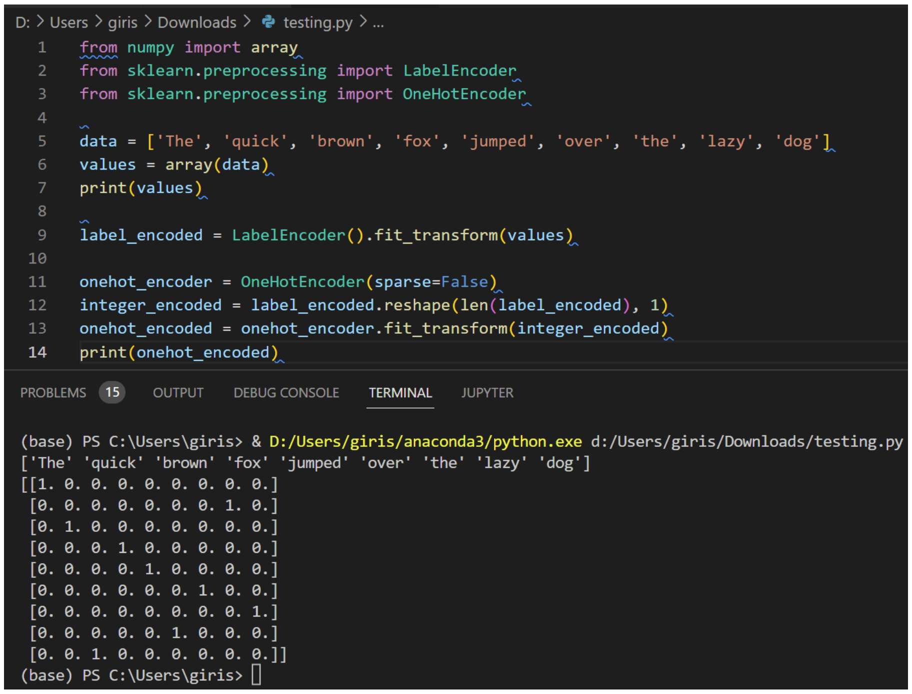
cs577 Assignment 5

Girish Rajani-Bathija
A20503736
Department of Computer Science
Illinois Institute of Technology
November 22, 2022

1. One hot encoding vector of each word in “The quick brown fox jumped over the lazy dog”



D: > Users > giris > Downloads > testing.py > ...

```python
from numpy import array
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

data = ['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog']
values = array(data)
print(values)

label_encoded = LabelEncoder().fit_transform(values)

onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = label_encoded.reshape(len(label_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
```

```
(base) PS C:\Users\giris> & D:/Users/giris/anaconda3/python.exe d:/Users/giris/Downloads/testing.py
['The' 'quick' 'brown' 'fox' 'jumped' 'over' 'the' 'lazy' 'dog']
[[1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0.]]
(base) PS C:\Users\giris> 
```

2. Plausible word embedding encoding of the words from question 1:
First index - animal, second index - color, third index - action

The - [0.1, 0.0, 0.7]
quick - [0.1, 0.0, 0.6]
brown - [0.2, 0.9, 0.0]
fox - [0.9, 0.0, 0.1]
jumped - [0.0, 0.0, 0.8]
over - [0.1, 0.0, 0.9]
the - [0.1, 0.0, 0.7]
lazy - [0.1, 0.0, 0.9]
dog - [0.9, 0.0, 0.1]

3. Gate Update equations in LSTM:

Input:
i_t = activation_1(dot(state_t, Ui) + dot(input_t, Wi) + bi) ***sigmoid activation***
Forget:
f_t = activation_1(dot(state_t, Uf) + dot(input_t, Wf) + bf) ***sigmoid activation***

Output:
o_t = activation_1(dot(state_t, Uo) + dot(input_t, Wo) + bo) ***sigmoid activation***

Block:
b_t = activation_2(dot(state_t, Ub) + dot(input_t, Wb) + bb) ***tanh activation***

Compute Carry Channel:
c_t = i_t * b_t + c_t * f_t

Compute State Channel:
state_t = o_t * activation_2(c_t) ***tanh activation***

4. The input, forget, and output gates use sigmoid activation while the block gate and state channel computation uses tanh activation. The input gate uses sigmoid $\in[0,1]$ and controls if we write to the channel or not. The forget gate also uses sigmoid in that if the output is 0 then it can forget and if the output is 1, it can remember. This simply means forgetting or remembering individual components in a vector. The output gate also uses sigmoid so it controls how much of the hidden state we reveal. The block gate is tanh activation $\in[-1,1]$ and it controls if we add or subtract from the channel.

5. Problems with simple RNN and how they are addressed in LSTM:
   a. SimpleRNN has problem retaining information about inputs seen many time steps before and so it does not learn well long-term dependencies. LSTM has the ability to remember things that happened long time ago. It has two channels for memory, one for short term memory and one for long term.
   b. Long term dependencies are a problem due to vanishing gradients similar to what happens in deep feedforward networks. LSTM solves this vanishing gradients problem by adding a way to carry information across many timesteps. The LSTM has a gradient flow path that passes the gradient without multiplying it again and again, which will prevent the vanishing gradients problem.

6. The need for bidirectional RNN rises when the context of the input is needed and this is particularly useful in some Natural Language Processing tasks. An example of this can be seen below:

Suppose we are given the sentence below and would like to predict the next word:
“The boy went to the …”. In a single direction RNN, it will try and predict the next word only from the context of the sentence, whereas a bidirectional RNN will have additional context from the future and can get a much better understanding. For example, in addition to the sentence mentioned, the bidirectional RNN may also see “made a sand castle,” and from this additional context, a more accurate prediction can be made.

Bidirectional RNN is expected to perform better than a single direction RNN when processing the sequence of words such as sentiments and NLP tasks such as sentence classification, translation, and entity recognition.

Bidirectional RNN is expected to perform worse than a single direction RNN when predicting the temperature.

7. Network architecture for sequence-to-sequence learning:

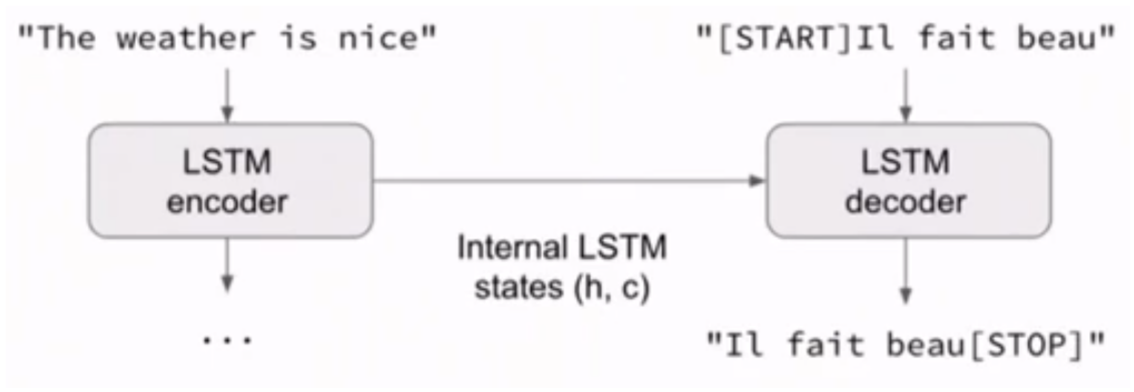


Figure 1 - Illustration of Encoder-Decoder setup

From Figure 1 above, we can see that the sequence-to-sequence network architecture consists of a RNN layer (or stack) for encoding the input as well as another RNN layer (or stack) to decode the input. The encoder will process the input sequence and return its own internal state while discarding the outputs of the encoder RNN and keeping only the state. The state is as the “context” for the decoder in the next step.

The decoder is trained to predict the next characters of the target sequence, given previous characters of the target sequence. It is trained to turn these target sequence into the same sequences offset by one timestep in the future (“teacher forcing”). The decoder uses as initial state the state vectors from the encoder, which is how the decoder knows what it is supposed to generate. The decoder essentially learns to generate targets[t+1…] given targets[...t], conditioned on the input sequence.