CISC 867: Project 3

Building a recurrent language model

Kariman Karm Mohamed Mousaa

I. Project Overview

A language model can predict the probability of the next word in the sequence, based on the words already observed in the sequence. In this problem, we will need to prepare text for developing a word-based language model, design and fit a neural language model with a learned embedding and a recurrent hidden layer, and to use the learned language model to generate new text with similar statistical properties as the source text.

Evaluation Metrics

To evaluate the model, we will use the accuracy matrix, it is how close the value is to its true value, it used for classification problems to be evaluated.

The Formula:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

TP+TN: the number of correct predictions.

TP+FP+TN+FN: Total number of predictions.

II. Analysis

Before exploring the data, we need to import the basic libraries (NumPy, pandas, matplotlib, seaborn, sklearn, tenserflaw, keras) and the dataset.

1-Load text: load doc into memory by function to open and read it.

-Part of data in the file:

```
BOOK I.

I went down yesterday to the Piraeus with Glaucon the son of
Ariston,
that I might offer up my prayers to the goddess (Bendis, the
Thracian
```

Clean Text

1-turn a doc into clean tokens.

2-replace '--' with a space ' '

3-split into tokens by white space

4-remove punctuation from each token

5- remove remaining tokens that are not alphabetic

6- make lower case

7-organize into sequences of tokens to can get each of 50 input words and 1 output word.

8- select sequence of tokens

9- convert into a line

10-store in sequence

11- save tokens to file, one dialog per line

12-save sequences to file

We got:

```
Total Tokens: 118684
Unique Tokens: 7409
Total Sequences: 118633
```

You will see each line is shifted along one word, with a new word at the end to be predicted; for example, here are the first 3 lines in truncated form:

book i i … catch sight of
i i went … sight of us
i went down … of us from

Train Language Model

1- Load Sequences
2- Encode Sequences by using Tokenizer, we made integer encode sequences of words.
3- Assign length of tokenizer. word_index+1 to vocabulary size.
4- Sequence Inputs and Output, we separate into input and output then transform output by one-hot-encoding and number of classes=vocab size.
5- Splitting train data as train and validation data with test size=0.2, random state=13.

Models:

1- Long Short-Term Networks (LSTM):

it's a building unit for layers of a recurrent neural network (RNN), the success of LSTMs is in their claim to be one of the first implements to overcome the technical problems and deliver on the promise of recurrent neural networks.

We define model then Embedding with vocab size, 50, input length=sequence length and then built 2 layers from LSTM with 100 and return sequences=True

And then Dense with 100 and activation='relu' and output layer Dense with vocab size and activation='softmax'.

We used 300,100 then 50 values to trial the model.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 50)            370500

 lstm (LSTM)                 (None, 50, 100)           60400

 lstm_1 (LSTM)               (None, 100)               80400

 dense (Dense)               (None, 100)               10100

 dense_1 (Dense)             (None, 7410)              748410

=================================================================
Total params: 1,269,810
Trainable params: 1,269,810
Non-trainable params: 0
_________________________________________________________________
None
```

Then we compile model using loss='categorical cross entropy', optimizer=Adam, metrics=accuracy.

And fit the model with batch size=120, epochs=20.

2- Gated Recurrent Unit (GRU) model:

GRU is basically an LSTM without an output gate. They perform similarly to LSTMs for most tasks but do better on certain tasks with smaller datasets and less frequent data.

We define model then Embedding with vocab size, 50, input length=sequence length and then built 2 layers from GRU with 100 and return sequences=True

And then Dense with 100 and activation='relu' and output layer Dense with vocab size and activation='softmax'.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 50)            370500

 gru (GRU)                   (None, 50, 100)           45600

 gru_1 (GRU)                 (None, 100)               60600

 dense (Dense)               (None, 100)               10100

 dense_1 (Dense)             (None, 7410)              748410

=================================================================
Total params: 1,235,210
Trainable params: 1,235,210
Non-trainable params: 0
_________________________________________________________________
```

Then we compile model using loss='categorical cross entropy', optimizer=Adam, metrics=accuracy.

And fit the model with batch size=120, epochs=20.

Model Evaluation:

1- LSTM:

With value =300

Accuracy: 0.2067

With value =100

Accuracy: 0.1760

With value=50

Accuracy=0.1642

```
791/791 [==============================] - 30s 38ms/step - loss: 4.6840 - accuracy: 0.1793 - val_loss: 5.7896 - val_accuracy: 0.1635
Epoch 12/20
791/791 [==============================] - 30s 38ms/step - loss: 4.6165 - accuracy: 0.1829 - val_loss: 5.8681 - val_accuracy: 0.1644
Epoch 13/20
791/791 [==============================] - 30s 38ms/step - loss: 4.5655 - accuracy: 0.1835 - val_loss: 5.9054 - val_accuracy: 0.1634
Epoch 14/20
791/791 [==============================] - 30s 38ms/step - loss: 4.5020 - accuracy: 0.1862 - val_loss: 6.0180 - val_accuracy: 0.1666
Epoch 15/20
791/791 [==============================] - 30s 38ms/step - loss: 4.4443 - accuracy: 0.1878 - val_loss: 6.1143 - val_accuracy: 0.1632
Epoch 16/20
791/791 [==============================] - 30s 37ms/step - loss: 4.3892 - accuracy: 0.1908 - val_loss: 6.2374 - val_accuracy: 0.1641
Epoch 17/20
791/791 [==============================] - 29s 37ms/step - loss: 4.3372 - accuracy: 0.1931 - val_loss: 6.3305 - val_accuracy: 0.1662
Epoch 18/20
791/791 [==============================] - 29s 37ms/step - loss: 4.2885 - accuracy: 0.1956 - val_loss: 6.4965 - val_accuracy: 0.1642
Epoch 19/20
791/791 [==============================] - 29s 37ms/step - loss: 4.4556 - accuracy: 0.1824 - val_loss: 6.2496 - val_accuracy: 0.1527
Epoch 20/20
791/791 [==============================] - 29s 37ms/step - loss: 4.3844 - accuracy: 0.1859 - val_loss: 6.7616 - val_accuracy: 0.1642
```

2- GRU:

With value =300

Accuracy: 0.2042

With value =100

Accuracy: 0.1695

With value=50

Accuracy= 0.1608

So we can see LSTM is the greatest performance.

Note: The model was run on broken parts due to the difficulty of implementing it due to the space and time, which took many days.

Finally, we can use the Language Model,

To do that we need to do the following:

1- used it from random import randint.
2- load doc into memory.
3- generate a sequence from a language model.
4- generate a fixed number of words.
5- encode the text as integer.
6- truncate sequences to a fixed length.
7- predict probabilities for each word.
8- map predicted word index to word.
9- append to input.
10- load cleaned text sequences.
11- load the model.
12- load the tokenizer.
13- select a seed text.
14- generate new text.

References:

1- **https://www.kaggle.com/code/thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru/notebook.**
2- **https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6.**
3- **https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.**