CHAPTER 12: RNN

Theory

In this chapter, we will study RNN (Recurrent Neural Network) which is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. The advantage of RNN is it can use their internal state (memory) to process variable-length sequences of inputs. RNN is widely used for NLP (Natural Language Processing), unsegmented, connected handwriting recognition, speech recognition and so on.

Recap

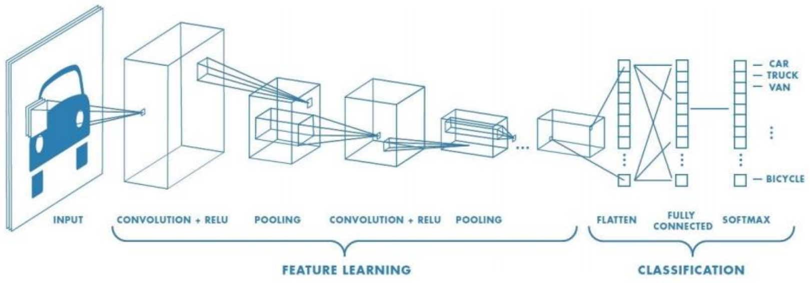
Let’s remember the contents of the previous chapter. The most important thing in the last chapter is to have an understanding of the concept of CNN and layers such as the Convolution layer, Pooling Layer, Fully-Connected Layer and so on.

Convolution Neural Network

Convolution Neural Network, (CNN or ConvNet) is a kind of deep neural network and widely used to visual imagery analyzing such as image classification, object detection.

CNN Layers

There are Convolution, Activation, Pooling, Fully-connected layers in the CNN.



Recurrent Neural Network

We can meet lots of sequence data in speech recognition, natural language processing, and weather forecast data and so on.

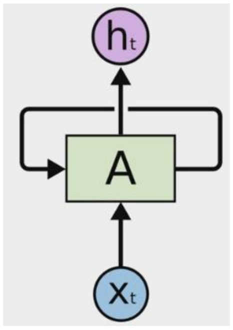
These sequence data have below properties:

- We don’t understand with only one sample

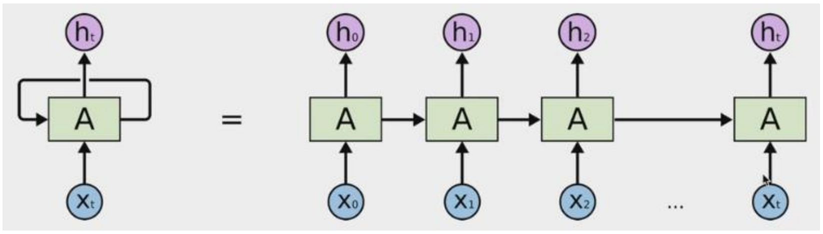
- We can understand based on the previous samples with the current sample

- We can’t implement this with NN/CNN

To overcome these issues, this kind of neural network was developed which can save the state of the network and use it in the next step.



When the model has multiple inputs, the above architecture can be implemented such as the below diagram.

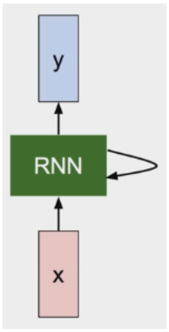


We can process a sequence of vectors x by applying a recurrence formula at every time steps.

$$h_t = f_w\left(h_{t-1}, x_t\right)$$

In here, x_t is input vector at some time step, h_{t-1} is an old state, h_t is a new state, and f_w is a function with parameters W.

For example, we can use a simple structure recurrent network.



$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right)$$

$$y_t = W_{hy}h_t$$

The function tanh is similar to Sigmoid, and working well in the practice.

RNN Applications

RNN can be widely used in practice.

- Language Modeling

- Speech Recognition

- Machine Translation

- Conversation Modeling

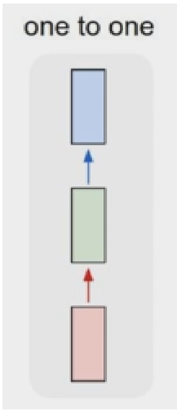
- Question Answering

- Image/Video Captioning

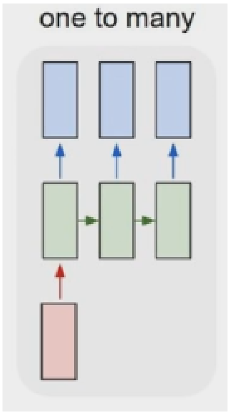
- Image/Music/Dance Generation

We can construct the RNN with several kinds of structures depend on the goal of the problem.

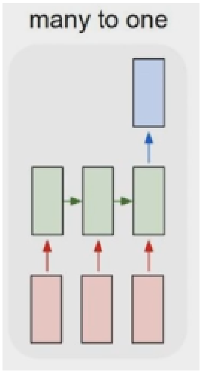
For the regression problem, we can use “one by one” type RNN which is called Vanilla Neural Network, the extension of the linear regression model.



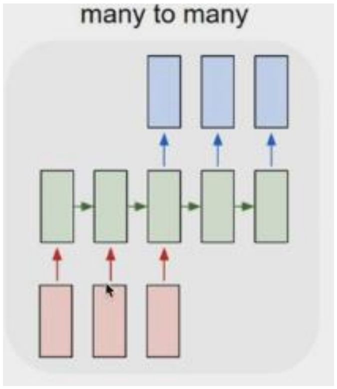
For image captioning, we can use “one to many” type RNN which input an image and output the sequence of words.



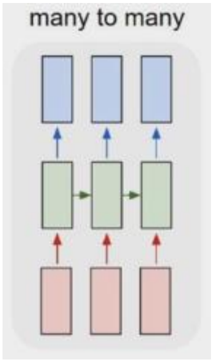
And for the sentiment classification problem which converts the sequence of words into sentiment, “many to one” type of RNN can be used.



And “many to many” type of RNN can be used for machine translation problems which convert the sequence of words into another sequence of words.



For the video classification on frame level, we can use another type of “many to many” RNN.



AIM

The aim of the following lab exercise is to implement the RNN which can predict the next character from the sequence of characters using Pycharm and Tensorflow, and then train the model and evaluate the trained result.

The following steps are required.

Task 1: Preparation of the development environment

Task 2: Making script for RNN

Task 3: Running script and get results

LAB EXERCISE 12: RNN

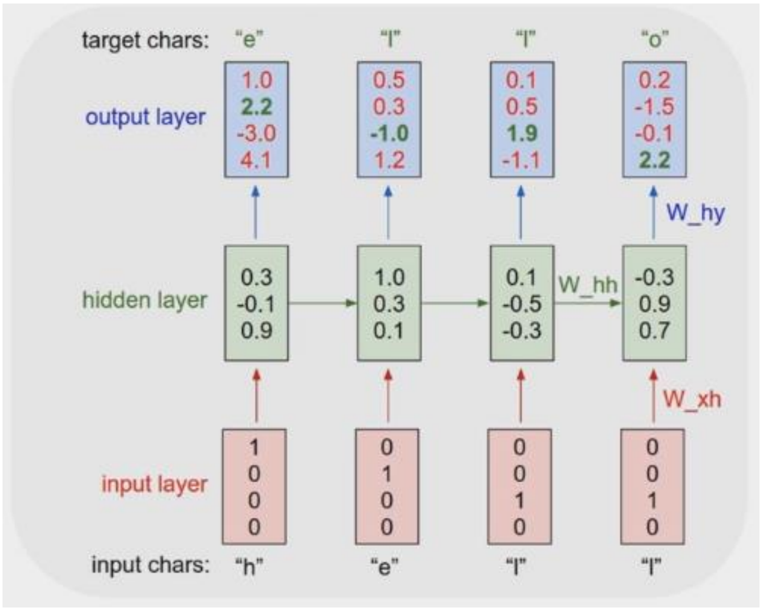
Preparation
Making Script
Running and Get Result

1. Preparation of the development environment
2. Making RNN script using Sequence data
3. Running script and get results

Task 1: Preparation of the development environment

STEP 1: Problem Statement

In this lab, let's create a python script to build and train the RNN.



The model should predict the next character "o" if the sequence of characters "h", "e", "l", "l" is inputted.

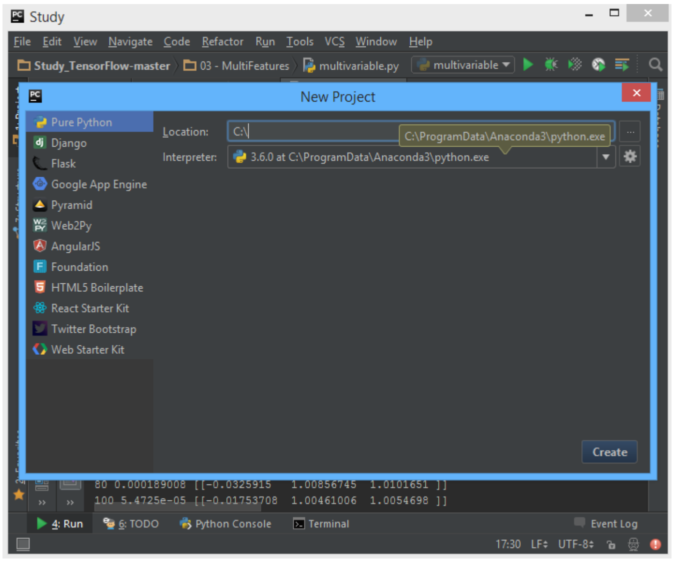
STEP 2: Function for create of RNN cell

In this script, we can use the below function of tensorflow package for creating of RNN cell.

```
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
```

STEP 3: Run PyCharm and create the project

First, run PyCharm and create a new project using the File/New Project menu. Select the project location and interpreter there and click Create Button. Then the empty project will be created.


Study
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Study_TensorFlow-master 03 - MultiFeatures multivariable.py multivariable
New Project
Pure Python
Django
Flask
Google App Engine
Pyramid
Web2Py
AngularJS
Foundation
HTML5 Boilerplate
React Starter Kit
Twitter Bootstrap
Web Starter Kit
Create
4: Run 6: TODO Python Console Terminal Event Log
17:30 LF: UTF-8:

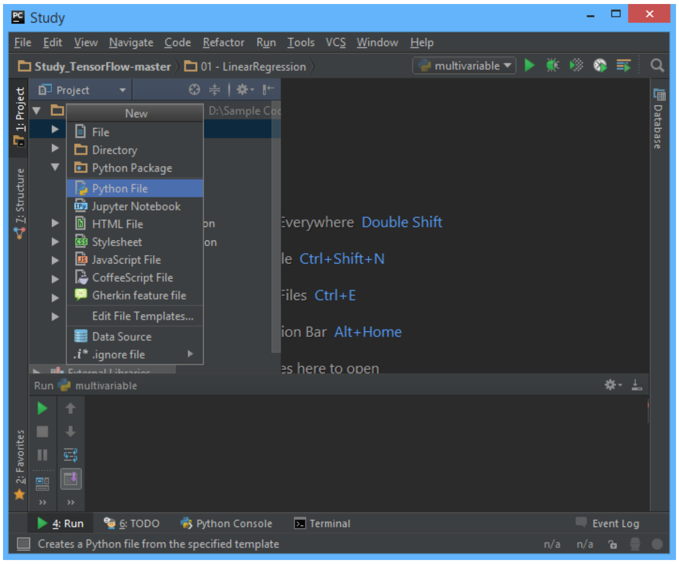
Location: C:\
Interpreter: 3.6.0 at C:\ProgramData\Anaconda3\python.exe

```
80 0.000189008 [[-0.0325915    1.00856745  1.0101651 ]]
100 5.4725e-05 [[-0.01753708   1.00461006  1.0054698 ]]
```

STEP 4: Create a python script file

Create a new python file using File/New/Python File.

And give the file name as “RNN.py”



Task1 is completed.

Task 2: Making RNN script using Sequence data

STEP 5: Import packages

First, import the Tensorflow and Numpy package in order to use it in Python script using the code.

```
import tensorflow as tf
import numpy as np
```

STEP 6: Create a training data

We can create the training data reflects on 'hello'.

```
char_rdic = list('helo')  # id -> char
char_dic = {w: i for i, w in enumerate(char_rdic)}  #
char -> id

x_data = np.array([
    [1, 0, 0, 0],  # h
    [0, 1, 0, 0],  # e
    [0, 0, 1, 0],  # l
    [0, 0, 1, 0],  # l
],
    dtype='f')

sample = [char_dic[c] for c in "hello"]  # to index
```

STEP 7: Define model configuration

And define rnn size, step size, batch size and so on.

```
char_vocab_size = len(char_dic)
rnn_size = char_vocab_size  # 1 hot coding (one of 4)
time_step_size = 4  # 'hell' -> predict 'ello'
batch_size = 1  # one sample
```

STEP 8: Define the RNN model

Next, let's define the RNN model.

```
rnn_cell = tf.nn.rnn_cell.BasicRNNCell(rnn_size)
state = tf.zeros([batch_size, rnn_cell.state_size])
X_split = tf.split(0, time_step_size, x_data)
outputs, state = tf.nn.rnn(rnn_cell, X_split, state)
```

STEP 9: Define the weights

And the weights can be defined as below lines.

```
logits = tf.reshape(tf.concat(1, outputs), [-1,
rnn_size])
targets = tf.reshape(sample[1:], [-1])
weights = tf.ones([time_step_size * batch_size])
```

STEP 10: Define the cost

We can define the cost function using the below lines.

```
loss = tf.nn.seq2seq.sequence_loss_by_example([logits],
[targets], [weights])
cost = tf.reduce_sum(loss) / batch_size
train_op = tf.train.RMSPropOptimizer(0.01,
0.9).minimize(cost)
```

Task2 is completed.

Task 3: Running script and get results

Let’s complete the code and run it.

STEP 11: Initialize the tensorflow variables

Initialize all of the tensorflow variables.

```
with tf.Session() as sess:
    # you need to initialize all variables
    tf.initialize_all_variables().run()
```

STEP 12: Train the model

If you add the training code then the project will be complete here. We print the result.

```
for i in range(100):
    sess.run(train_op)
    result = sess.run(tf.arg_max(logits, 1))
    print("%r, %r" % (result, [char_rdic[t] for t in
result]))
```

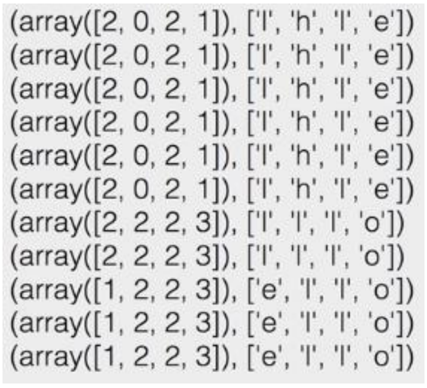
Then the full code is below.

```
# Recurrent Neural Network
import numpy as np
import tensorflow as tf

char_rdic = list('helo')  # id -> char
char_dic = {w: i for i, w in enumerate(char_rdic)}  #
char -> id

x_data = np.array([
    [1, 0, 0, 0],  # h
    [0, 1, 0, 0],  # e
    [0, 0, 1, 0],  # l
    [0, 0, 1, 0],  # l
],
    dtype='f')

sample = [char_dic[c] for c in "hello"]  # to index

# Configuration
char_vocab_size = len(char_dic)
rnn_size = char_vocab_size  # 1 hot coding (one of 4)
time_step_size = 4  # 'hell' -> predict 'ello'
batch_size = 1  # one sample

# RNN model
rnn_cell = tf.nn.rnn_cell.BasicRNNCell(rnn_size)
state = tf.zeros([batch_size, rnn_cell.state_size])
X_split = tf.split(0, time_step_size, x_data)
outputs, state = tf.nn.rnn(rnn_cell, X_split, state)

# logits: list of 2D Tensors of shape [batch_size x
num_decoder_symbols]
# targets: list of 1D batch-sized int32 Tensors of the
same length as logits.
# weights: list of 1D batch-sized float-Tensors of the
same length as logits.
logits = tf.reshape(tf.concat(1, outputs), [-1,
rnn_size])
targets = tf.reshape(sample[1:], [-1])
weights = tf.ones([time_step_size * batch_size])

loss = tf.nn.seq2seq.sequence_loss_by_example([logits],
[targets], [weights])
cost = tf.reduce_sum(loss) / batch_size
train_op = tf.train.RMSPropOptimizer(0.01,
0.9).minimize(cost)

# Launch the graph in a session
```

```
with tf.Session() as sess:
    # you need to initialize all variables
    tf.initialize_all_variables().run()
    for i in range(100):
        sess.run(train_op)
        result = sess.run(tf.arg_max(logits, 1))
        print("%r, %r" % (result, [char_rdic[t] for t in
result]))
```

Let’s run this code. Then we can see the results such as below.

As we can know from the results, this RNN model predicts the next character ‘o’ successfully.



```
(array([2, 0, 2, 1]), ['l', 'h', 'l', 'e'])
(array([2, 0, 2, 1]), ['l', 'h', 'l', 'e'])
(array([2, 0, 2, 1]), ['l', 'h', 'l', 'e'])
(array([2, 0, 2, 1]), ['l', 'h', 'l', 'e'])
(array([2, 0, 2, 1]), ['l', 'h', 'l', 'e'])
(array([2, 0, 2, 1]), ['l', 'h', 'l', 'e'])
(array([2, 2, 2, 3]), ['l', 'l', 'l', 'o'])
(array([2, 2, 2, 3]), ['l', 'l', 'l', 'o'])
(array([1, 2, 2, 3]), ['e', 'l', 'l', 'o'])
(array([1, 2, 2, 3]), ['e', 'l', 'l', 'o'])
(array([1, 2, 2, 3]), ['e', 'l', 'l', 'o'])
```

LAB CHALLENGE

Challenge

In this lab, we have implemented the RNN model using simple sequence data “hello”.

Use another long sequence data set such as “dictionary”, “recurrent model” and so on, and expand the size of RNN cells to 6 from 4.

SUMMARY

The RNN (Recurrent Neural Network) is a class of artificial neural networks where connections between nodes from the directed graph along a temporal sequence.

RNN model is used widely in language modeling, speech recognition, machine translation, conversation modeling, question answering, image/video captioning and so on.

REFERENCES

- https://en.wikipedia.org/wiki/RNN
- https://en.wikipedia.org/wiki/TensorFlow
- https://en.wikipedia.org/wiki/Artificial_intelligence

INDEX