Exploring Three Python Libraries

Jinqian Pan | panjin@kean.edu | CPS3320
https://github.com/JinqianPan/CPS-3320/tree/master/Program02

Keras

Using pip to install: pip install Keras

Central Purpose:

As the API of advanced neural network, Keras is easy to be used to building models, which means it is friendliness, high modularity and scalability. Because it is used to build model for neural network, Keras supports to convolution neural network and cyclic neural network and it also could combine these two kinds of neural network for some complex neural networks. Keras could run on the CPU and GPU to calculate complicated operations.

- **User friendly**. Keras is an API designed for humans, not machines. It puts the user experience first and foremost. Keras follows the best practice of reducing cognitive difficulties.
- **modularization**. A model is understood as a sequence or diagram of independent, fully configurable modules. These modules can be assembled with as few restrictions as possible.
- **Scalability**. New modules are easy to add (as new classes and functions), and existing modules already provide ample examples.

Useful ways:

The most useful function of Keras is building the model for neural network models, so that the useful and interesting ways of Keras are building various of models.

1. **For Convolutional Neural Network (CNN)**

Example code:

```
model2.add(Conv2D(32,(3,3)))
model2.add(Activation('relu'))
```

These two lines of code show the ways to create layers in the convolutional neural network. The first one for creating a 2D Convolutional layer and the second line creates an activation layer, which use the ReLU, for the model.

2. **For Recurrent Neural Network (RNN)**

Example code:

```
model3.add(Dense(1, activation='sigmoid'))
```

The code also shows how to add a layer for model; however, this time the code create the layer for RNN. This line uses Sigmoid as the activation function.

Functions (with link of official documentation):

https://keras.io/why-use-keras/

1. **Full connection layer**: the most commonly used one in the neural network to activate the neurons in the neural network.

Dense（units, activation=‘relu’, use_bias=True）

Parameter Description:
units: Dimension of the output of the full connection layer, i.e. the number of neurons in the next layer
activation: activation function, relu is used by default
use_bias: use bias

2. **Activation layer**: applies the activation function to the output of the previous layer.

Activation(activation)

Parameter Description:
Activation: the activation function you want to use, such as relu, tanh, sigmoid, etc

3. **Dropout layer**: select a certain proportion of neurons in the upper layer to be inactivated at random without updating, but the weight is still reserved to prevent over fitting.

Dropout(rate)

Parameter Description:
Rate: the proportion of deactivation, 0-1 floating point number.

4. **Flatten layer:** a high-dimensional matrix with a dimension greater than or equal to 3 is "flattened" into a two-dimensional matrix. That is, keep the first dimension (such as the number of batches), and then multiply the values of the remaining dimensions to the second dimension of the squash matrix.

Flatten ( )

PyTorch

Using pip to install: pip install torch torchvision

Central Purpose:

PyTorch is similar as Keras and Tensorflow; All of them are used for deep learning. PyTorch can better explain the process of neural network. The neural network built by pytorch is dynamic. Compared with static Tensorflow, it can deal with some problems more effectively. Keras is easier to learn and experiment with standard layers, plug and play; PyTorch provides a lower-level method that is more flexible for users with more mathematical backgrounds.

Useful ways:

1. Graphics settings

PyTorch has its own code to connect CPU and GPU.

Example Code:

For only one Graphic

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

For more than one Graphics

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0,1'
```

2. One Hot Encoding

One Hot Encoding is a useful tool to analysis the data, and using PyTorch can also do this kind of encoding.

Example Code:

```
tensor = torch.tensor([0, 2, 1, 3])
N = tensor.size(0)
num_classes = 4
one_hot = torch.zeros(N, num_classes).long()
one_hot.scatter_(dim=1, index=torch.unsqueeze(tensor, dim=1), src=torch.ones(N,
num_classes).long())
```

Functions (with link of official documentation):

https://pytorch.org/docs/stable/index.html

1. **For Tensor**

torch.Tensor.new_tensor(data, dtype=None, device=None, requires_grad=False) → Tensor
This code returns a new Tensor with data as the tensor data. By default, the returned Tensor has the same torch.dtype and torch.device as this tensor.

Parameters:
data (array_like): The returned Tensor copies data.
dtype (torch.dtype, optional): the desired type of returned tensor. Default: if None, same torch.dtype as this tensor.
device (torch.device, optional): the desired device of returned tensor. Default: if None, same torch.device as this tensor.
requires_grad (bool, optional): If autograd should record operations on the returned tensor. Default: False.

2. **For adding layers in model**

torch.nn.add_module(name, module)

Parameters:
name (string): name of the child module. The child module can be accessed from this module using the given name
module (Module): child module to be added to the module.

3. **For using CUDA**

torch.cuda.current_stream(device=None)

Parameters:
device (torch.device or python:int, optional): selected device. Returns the currently selected Stream for the current device, given by current_device(), if device is None (default).

PyMySQL

Using pip to install: pip install PyMySQL

Central Purpose:

Most of time Python needs work with data, and SQL language can not be avoided when we analysis the data. PyMySQL offers a connection between Python and MySQL, so that they can work together.

Useful ways:

1. Connect MySQL

The example code could link to the database and after that we can work with data

Example Code:

```
db = pymysql.connect(host="imc.kean.edu", user="panjin", password="1025761",
db="CPS3740" )
```

2. Execute by using SQL language

By using executing language, we can work by using SQL language.

Example Code:

```
cursor = db.cursor()
sql = "SELECT * FROM EMPLOYEE \
        WHERE INCOME > %s" % (1000)
cursor.execute(sql)
```

Functions (with link of official documentation):

https://pymysql.readthedocs.io/en/latest/

There are lots of parameters when we need to connect database; however, we usually just use host, user, password and database.

Code:

```
pymysql.connections.Connection(host=None, user=None, password='', database=None, port=0,
unix_socket=None, charset='', sql_mode=None, read_default_file=None, conv=None,
use_unicode=None, client_flag=0, cursorclass=<class 'pymysql.cursors.Cursor'>,
init_command=None, connect_timeout=10, ssl=None, read_default_group=None,
compress=None, named_pipe=None, autocommit=False, db=None, passwd=None,
local_infile=False, max_allowed_packet=16777216, defer_connect=False,
auth_plugin_map=None, read_timeout=None, write_timeout=None, bind_address=None,
binary_prefix=False, program_name=None, server_public_key=None)
```

Parameters:

host – Host where the database server is located

user – Username to log in as

password – Password to use.

database – Database to use, None to not use a particular one.

…