Article
Zhong Li · Mar 14, 2019 10m read

A Deep Learning Demo Kit with Python3 Binding to HealthShare (Part I)

Keywords: Anaconda, Jupyter Notebook, Tensorflow GPU, Deep Learning, Python 3 and HealthShare

1. Purpose and Objectives

This "Part I" is a quick record on how to set up a "simple" but popular deep learning demo environment step-by-step with a Python 3 binding to a HealthShare 2017.2.1 instance . I used a Win10 laptop at hand, but the approach works the same on MacOS and Linux.

Last week it was noticed that Python overtook Java by becoming the most popular language in PYPL Index. Tensorflow is a powerful computation engine, very popular in research and academic worlds too. HealthShare is a data platform that provides an unified care record of patients for care providers.

Could we bind them all together into a single kit, and what could be the simplest possible approach to achieve this? Here let's try a first step on this topic together, before we consider what demo we can experiment on next.

2. Scope and Disclaimer

The demo kit we are trying to build here includes the following components:

- Anaconda - a data science platform popular with well structured package management.
- Tensorflow - a machine learning platform with powerful computation engines.
- Theano- a Python library of computation engine for ML. Optional, and alternative to Tensorflow.
- Keras - a Python Deep Learning Library that sits on a Tensorflow or Theano engine.
- Jupyter Notebook - included in the Anaconda package
- Python 3 - included in the Anaconda package
- HealthShare - we will use a HealthShare 2017.2.1 instance.

... ... and a Win10 laptop.

My Dell laptop happens to come with a low-end GPU GeForce 840M, not powerful but nonetheless it increases the demo's training speed by 3x times.

Other than that all the steps below would work well in a MacOS(mine is a Mac Mini), or a Linux environment (I tried e.g. an Ubuntu on AWS too).

A HealthShare 2017 instance is used as a demo DB in convenience but the basic experiments we start here would work with Cache or Ensemble instances, and eventually IRIS too I believe.

Disclaimer: this is an article to collect and/or call for community interests by starting with a PoC demo kit together. It's NOT an official release or announcement by any means or by any legal brand. The content or scope will be revised anytime as needed as we move forward in our spare times.

3. Prerequisite

There is no prerequisite of knowledge or experiences of ANN, AI, Deep Learning or Tensorflow etc needed here. A Python beginner is good enough to start with.

GPU is not mandatory on your laptop or server. But if you happen to have one, you might need to update its drive. For example, I downloaded the driver here

Slight reminder: see caveat 7.1 at the end.

4. Demo Environment Setup - Steps

4.1 Installing Anaconda environment:

4.1.1 Download and installation:

Follow this link to download Anaconda release: By today (Mar 2019) the most up-to-date version to download is Anaconda 2018.12 with Python 3.7.

Follow this link as a simple step-by-step screen-by-screen instruction to install an Anaconda base environment.

*Note: You can follow the same link to download it for **MacOS** or **Linux** hosts.*

4.1.2 Test Anaconda installation:

Follow this link to test the installation step-by-step, to confirm that Python and Jupyter notebook are all working as expected, although we would not need the Spyder IDE.

Make sure you can start the Jupyter Notebook from the Windows Start or a Windows Command line., per the instruction in the link.

4.2 Install a Tensorflow Environment

We will install Tensorflow engine as Anaconda packages, by using the Conda command within Anaconda Prompt Window.

The steps are:

1. Start Anaconda Prompt command window; check its Conda version and Python version:

```
(base) C:\>conda -V

conda 4.5.12

(base) C:\>python -V
Python 3.7.1
```

2. Create a new environment called "tensorflow," then activate it:

```
(base) C:\>conda create -n tensorflow
Solving environment: done

.... ... ... ...
(base) C:\>conda activate tensorflow

(tensorflow) C:\>
```

3. Install Tensorflow packages via a specified Conda channel:

```
(tensorflow) C:\>conda install -c conda-forge tensorflow
```

This step will take quite a few minutes to download, resolve, upgrade/downgrade quite a long list of specific packages.

4. Test the Tensorflow installation is successful

```
 tensorflow) C:\>python
Python 3.6.7 (default, Feb 28 2019, 07:28:18) ... .... ... ...
>>> import tensorflow as tf
>>> se=tf.Session()
>>> print(se.run(tf.constant('Hello Tensorflow')))
b'hello TensorFlow!'

>>> print(tf.version)
1.10.0

>>> quit()

(tensorflow) C:\>
```

4.3 Install a Tensorflow GPU environment

Similar step to install a Tensorflow GPU environment.

1. Create a new environment called Tensorflow, then activate it

```
(base) C:\>conda create -n tensorflow-gpu
Solving environment: done

... ...
(base) C:\>conda activate tensorflow-gpu

(tensorflow-gpu) C:\>
```

2. Install "tensorflow-gpu" packages (instead of "tensorflow") without specifying a channel

```
(tensorflow-gpu) C:\>conda install tensorflow-gpu
```

This will take a while to download, resolve and install all packages.

3. Validate the installation

```
>>> import tensorflow as tf
>>> se = tf.Session()
2019-03-14 14:02:52.423895: I tensorflow/core/platform/cpufeatureguard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
2019-03-14 14:02:53.081332: I tensorflow/core/commonruntime/gpu/gpudevice.cc:1433] Found device 0 with properties:
name: GeForce 840M major: 5 minor: 0 memoryClockRate(GHz): 1.124
pciBusID: 0000:03:00.0
totalMemory: 2.00GiB freeMemory: 1.65GiB
2019-03-14 14:02:53.087514: I tensorflow/core/commonruntime/gpu/gpudevice.cc:1512] Adding visible gpu devices: 0
```

2019-03-14 14:08:18.789761: I tensorflow/core/commonruntime/gpu/gpudevice.cc:984] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-03-14 14:08:18.794278: I tensorflow/core/commonruntime/gpu/gpudevice.cc:990] 0
2019-03-14 14:08:18.796348: I tensorflow/core/commonruntime/gpu/gpudevice.cc:1003] 0: N
2019-03-14 14:08:18.804907: I tensorflow/core/commonruntime/gpu/gpudevice.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1395 MB memory) -> physical GPU (device: 0, name: GeForce 840M, pci bus id: 0000:03:00.0, compute capability: 5.0)
Note: this step could take quite a few minutes since it is rebuilding the CUDA gpu libraries. If there is an issue, you can try to restart the Anaconda Prompt by "Run as Administrator", then retry the step here. It resolved my issues in a couple of occasions.

```
>>> print(se.run(tf.constant('Hello')))
b'Hello'
>>> print(tf.version)
1.13.1
```

Now let's list the environments of this Anaconda:

```
(base) c:\>conda env list
# conda environments:
#
base * C:\ProgramData\Anaconda3
tensorflow C:\Users\zhongli\AppData\Local\conda\conda\envs\tensorflow
tensorflow-gpu C:\Users\zhongli\AppData\Local\conda\conda\envs\tensorflow-gpu
```

4.4 Install a Theano environment (Optional)

You can skip this step if you don't plan to experiment with Keras configuration and will always stick Tensorflow as its underlining engine.

Otherwise, please create/activate a new environment then run this: conda install -c conda-forge theano

4.5 Install Keras

Keras will be the top layer module for this deep learning demo stack.

Let's install it into the "tensorflow-gpu" environment we just created. Note: we are installing "keras-gpu" here instead of "keras"

```
(tensorflow-gpu) c:\>conda install -c anaconda keras-gpu
```

It would install the related packages. Now you can do a quick validation:

```
(tensorflow-gpu) c:\>python
Python 3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import keras as k
Using TensorFlow backend.
```

In "Part II", We will use Julyter Notebook to run a deep learning model to validate all the installation above. But before then, let's finish our HealthShare instance installation and its Python 3 binding.

4.6. Install a HealthShare instance

I downloaded a standard HealthShare installation file from Intersystems WRC's download page:

`hscore15.032_hsaa15.032_hspi15.032_hsviewer15.032_linkage15.032` Windows 64 Bit EXE

Also as usual a temporary license is needed, and can be acquired by contacting Intersystems Sales or Support.

After installation, its version string is something like: **Cache for Windows (x86-64) 2017.2.1 (Build 801318095U) Mon May 7 2018 14:28:58 EDT [HealthShare Modules:Core:16.0.7669 + Linkage Engine:16.0.7669 + Patient Index:16.0.7669 + Clinical Viewer:16.0.7669 + Active Analytics:16.0.7669]**

4.7 Install a Python Binding to HealthShare into Tensorflow Environment.

4.7.1 Identify the HealthShare instance path

The official product documentation on Python Binding to Intersystems Cache is here.

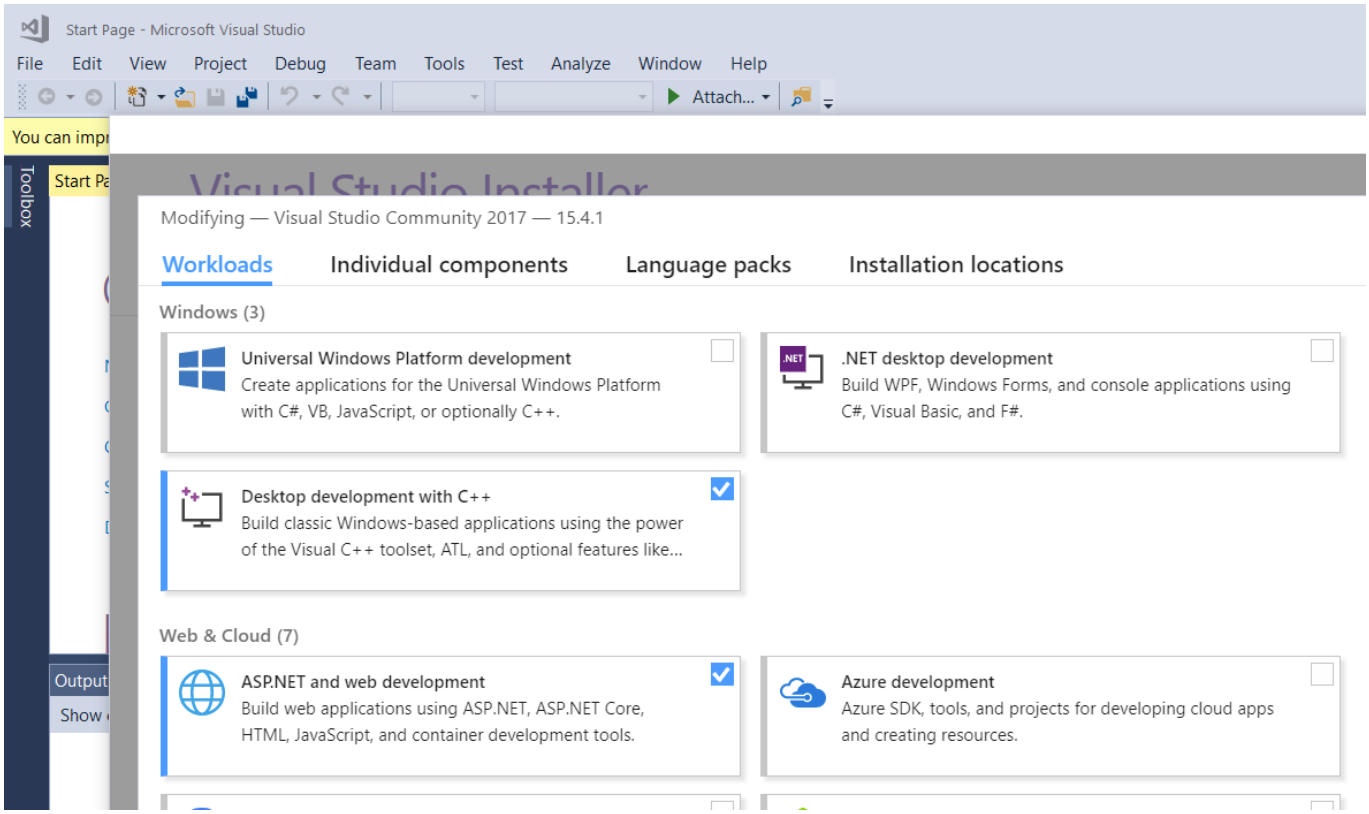
For example, my HS instance is installed locally at here: **C:\InterSystems\HS20181**

So first thing I did was to add this ISC Python DLL path into Windows's "Environment Path":
C:\InterSystems\HS20181\bin

4.7.2 Set up VC++ compiler

Then I ran this command "**vcvarsall.bat**" to make sure the path is set up for MS VC++ compiler.

I use **Visual Studio 2017 -> Tools -> Get Tools and Features** menu, then double check the "Desktop Development with C++" is installed (ticked) as shown below - if not, it should be installed. After then, you will be able to find "**vcvarsall.bat**" in this default directory, e.g. **C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build**



4.7.3 Install the Python 3 Binding to HealthShare instance

Now I will set up the "Python 3 binding to HealthShare" directly in my "tensorflow-gpu" environment that I installed as in above sections.

Note: I ran "**python setup3.py install**" instead of "...**setup.py** ...":

```
(tensorflow-gpu) c:\>cd c:\InterSystems\HS20181\dev\python

(tensorflow-gpu) c:\InterSystems\HS20181\dev\python>python setup3.py install
enter directory where you installed Cache'c:\InterSystems\HS20181
libdir=c:\InterSystems\HS20181/dev/cpp/lib
include dir=c:\InterSystems\HS20181/dev/cpp/include
libs=['cbind']
running install
running bdist_egg
running egg_info
... ...
installing library code to build\bdist.win-amd64\egg
running install_lib
running build_py
running build_ext
creating build\bdist.win-amd64\egg
creating build\bdist.win-amd64\egg\intersys
copying build\lib.win-amd64-3.6\intersys\pythonbind.py -> build\bdist.win-amd64\egg\intersys
... ...

creating stub loader for intersys\pythonbind31.cp36-win_amd64.pyd
byte-compiling build\bdist.win-amd64\egg\intersys\pythonbind31.py to pythonbind31.cpython-36.pyc
creating build\bdist.win-amd64\egg\EGG-INFO
... ...

Processing pythonbind3-1.0-py3.6-win-amd64.egg
creating c:\users\zhongli\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-
packages\pythonbind3-1.0-py3.6-win-amd64.egg
Extracting pythonbind3-1.0-py3.6-win-amd64.egg to c:\users\zhongli\appdata\local\conda\conda\envs\tensorflow-
gpu\lib\site-packages
Adding pythonbind3 1.0 to easy-install.pth file

Installed c:\users\zhongli\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-
packages\pythonbind3-1.0-py3.6-win-amd64.egg
Processing dependencies for pythonbind3==1.0
Finished processing dependencies for pythonbind3==1.0
```

4.7.4 Verify the Binding is working with HealthShare database instance

Now let's run the "**test.py**" sample within "**samples3**" path :

```
(tensorflow-gpu) c:\InterSystems\HS20181\dev\python\samples3>python test.py
Simple Python binding sample
Cache server port (default 1972)? 56778
Connection string: localhost[56778]:SAMPLES
Connecting to Cache server
Connected successfully
```

```
Creating database
Opening Sample.Person instance with ID 1 with default concurrency and timeout
Getting the value of the Name property
Value: Zevon,Mary M.
Test completed successfully
```

The above confirmed that my Python 3 binding into the specific HealthShare database (SAMPLES namespace: Sample.Person table) has been working within a tensorflow-gpu environment.

5. What's next

Now we have a basic environment with some popular tools to start to try some real deep learning demos.

Next in "Part II" We will use Jupyter Notebook to run the sample codes, and try to explore the key concept of various models and their capabilities, starting from the most classic one that almost everybody started with. We will always try to find the simplest approach or shortcut to explore how to train/predict some common sample data, before we even attempt to imagine or collect some real cases together in the long reach.

6. Acknowledgement

Most of the work here were re-done over the peaceful Christmas and New Year holiday. For this demo kit listed here, the only thing extra was to run the Python binding to a HealthShare instance.

7. Caveat

7.1 GPU driver update: I was reminded that updating GPU driver directly occasionally caused issue for other applications such as Matlab or common gaming platforms, so you need to be comfortable fiddling through their fixes.

7**.2 Simple shortcut**: Whenever possible, we will always try the simplest code or shortcuts. For any possible real cases or implications, the formal methodologies on the model parameter optimisation as well as data normalisation would always be heavy-weight tasks that need not be considered here for demo data set.

#Artificial Intelligence (AI) #Machine Learning (ML) #Python #HealthShare

Source URL:https://community.intersystems.com/post/deep-learning-demo-kit-python3-binding-healthshare-part-i