CNTK

INSTALL CNTK

Select version which you want to install

CNTK 2.2 | CNTK 2.1 | CNTK 2.0

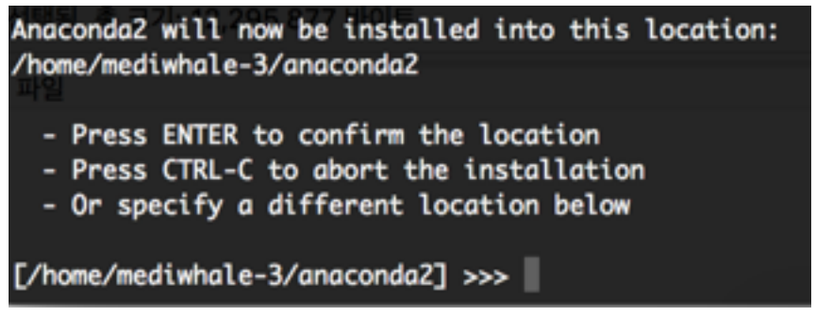
Python	Flavor	URL
2.7	CPU-Only	https://cntk.ai/PythonWheel/CPU-Only/cntk-2.2-cp27-cp27mu-linux_x86_64.whl
	GPU	https://cntk.ai/PythonWheel/GPU/cntk-2.2-cp27-cp27mu-linux_x86_64.whl
	GPU-1bit-SGD	https://cntk.ai/PythonWheel/GPU-1bit-SGD/cntk-2.2-cp27-cp27mu-linux_x86_64.whl
3.4	CPU-Only	https://cntk.ai/PythonWheel/CPU-Only/cntk-2.2-cp34-cp34m-linux_x86_64.whl
	GPU	https://cntk.ai/PythonWheel/GPU/cntk-2.2-cp34-cp34m-linux_x86_64.whl
	GPU-1bit-SGD	https://cntk.ai/PythonWheel/GPU-1bit-SGD/cntk-2.2-cp34-cp34m-linux_x86_64.whl
3.5	CPU-Only	https://cntk.ai/PythonWheel/CPU-Only/cntk-2.2-cp35-cp35m-linux_x86_64.whl
	GPU	https://cntk.ai/PythonWheel/GPU/cntk-2.2-cp35-cp35m-linux_x86_64.whl
	GPU-1bit-SGD	https://cntk.ai/PythonWheel/GPU-1bit-SGD/cntk-2.2-cp35-cp35m-linux_x86_64.whl
3.6	CPU-Only	https://cntk.ai/PythonWheel/CPU-Only/cntk-2.2-cp36-cp36m-linux_x86_64.whl
	GPU	https://cntk.ai/PythonWheel/GPU/cntk-2.2-cp36-cp36m-linux_x86_64.whl
	GPU-1bit-SGD	https://cntk.ai/PythonWheel/GPU-1bit-SGD/cntk-2.2-cp36-cp36m-linux_x86_64.whl

Ubuntu 16.04
Python 2.7

Pip install htttp://cntk.ai …

INSTALL STEP ANACONDA

Step1 If you require a Python 2.7 root environment, we recommend you install Anaconda2 4.3.0 Python for Linux (64-bit).
Below we assume that the prerequisites above are satisfied.



Step2

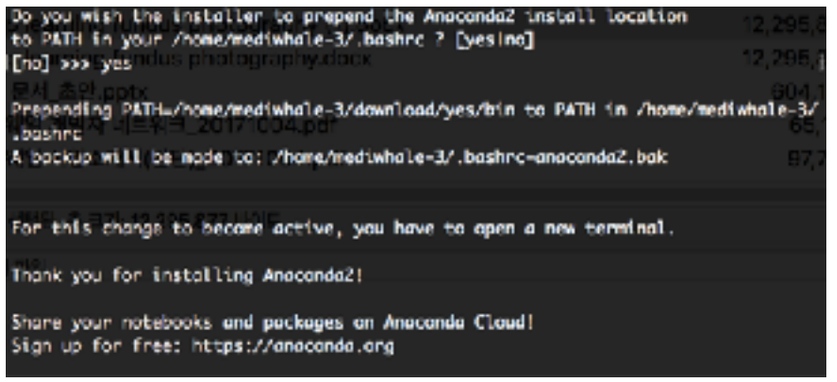
```
bash Anaconda3-4.2.0-Linux-x86_64.sh
```

Step3

```
Output
Anaconda3 will now be installed into this location:
/home/sammy/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/sammy/anaconda3] >>>
```



Step4 `PATH=/home/sammy/anaconda3/bin` 의 환경변수를 `/home/sammy/user/.bashrc` 에 추가한다.

`A backup will be made to: /home/sammy/.bashrc-anaconda3.bak` 백업 파일은 위와 같은 경로에 저장된다..

Step5

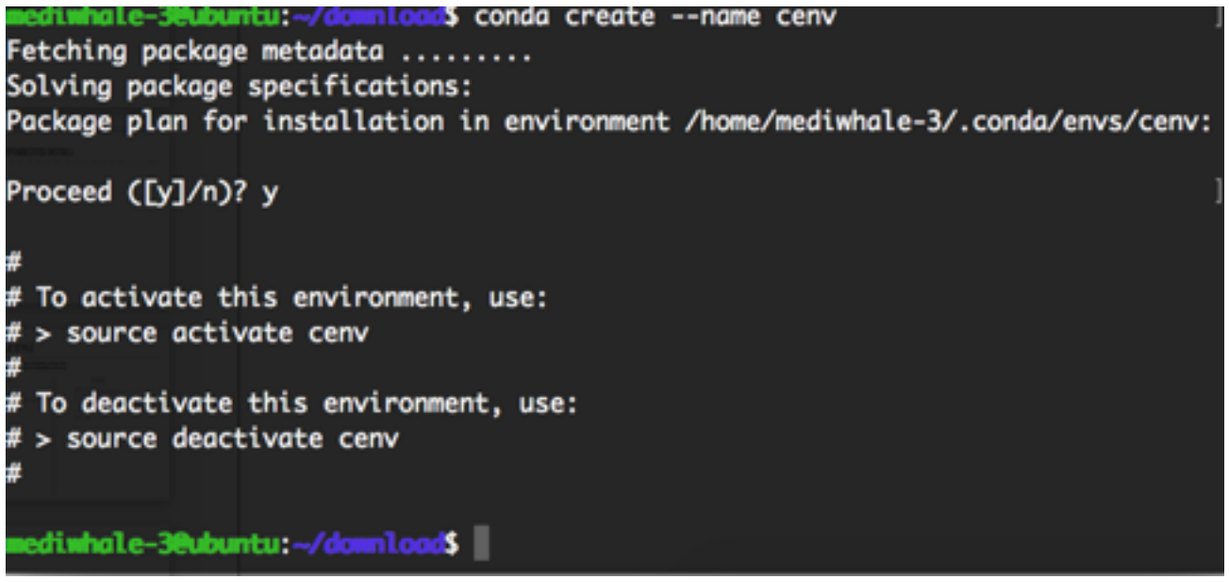
`source ~/.bashrc`

```
conda list

Output
# packages in environment at /home/sammy/anaconda3:
#
_license                  1.1                      py35_1
_nb_ext_conf              0.3.0                    py35_0
alabaster                 0.7.9                    py35_0
...
```

Step6

```
conda create --name cenv python=3(default 2)
```



Step7

```
source activate cenv
```

Step8

```
source deactivate
```

ANACONDA | PIP INSTALL

1.cpu only

https://docs.microsoft.com/en-us/cognitive-toolkit/setup-linux-python?tabs=cntkpy22

2.cpu and gpu

If you plan on using a GPU enabled version of CNTK, you will need a CUDA 8 compliant graphics card and up-to-date graphics drivers installed on your system. Also, we assume Anaconda2 is installed and that it is listed before any other Python installations in your PATH.

INSTALL ERROR

```
ImportError: libmpi_cxx.so.1: cannot open shared object file: No such file or directory
```

```
sudo apt-get install openmpi-bin
```

```
ImportError: libjasper.so.1: cannot open shared object file: No such file or directory
```

```
sudo apt-get install libjasper-dev
```

Jepg-2000 형태를 다룰수 있는 라이브러리

```
>>> import cntk
>>> cntk.__version__
'2.2'
```

CNTK'S VARIABLE STYLE

Most of the data containers like parameters, constants, values, etc. implement the asarray() method, which returns a NumPy interface.

Tensorflow

```
import tensorflow as tf
>>> c = tf.constant(3, shape=(2,3))
>>> print c
>>> Tensor("Const:0", shape=(), dtype=int32)
>>> sess=tf.Session()
>>> sess.run(c)
```

```
import tensorflow as tf
>>> v=np.zeros([3,2])
>>> v=tf.Variable(c)

>>> v1=tf.placeholder(dtype=tf.float32 , shape=[3,2])
```

Cntk Concepts

```
import cntk as C
>>> c = C.constant(3, shape=(2,3))
>>> c.asarray()
array([[ 3.,  3.,  3.],
       [ 3.,  3.,  3.]], dtype=float32)
```

기본적으로 반환 형태가 **numpy** 라는 파이선 라이브러리를 사용해 반환

사용하기 쉽다.

```
import cntk as C
>>> c = C.input_variable([3,2])
```

HOW TO MAKE LAYER?

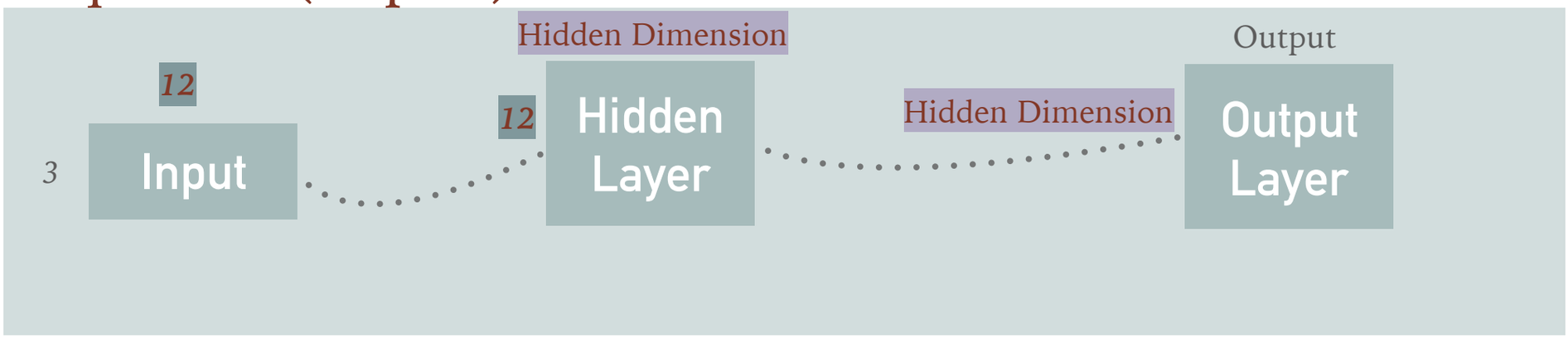
from cntk.layers import Dense , Sequential

```
print Dense(hidden_dimension ,activation=cntk.sigmoid)
print Dense(outputs)
my_model = Sequential(layers=[Dense(hidden_dimension ,activation=cntk.sigmoid),Dense(outputs)])
print my_model
z = my_model(features)
"""
```

```
my_model = Sequential ([
    For(range(6), lambda: \
        Dense(2048, activation=sigmoid))
    Dense(9000, activation=softmax)
])
```

Using for loop , can make layer more simply!

Sequential(Input)



MODEL MAKE

Tensorflow

```
out_ch=28
w1=tf.get_variable("w1" , [7,7,color_ch , out_ch] , init
er())
b1=tf.Variable(tf.constant(0.1) ,out_ch)
s1=[1,1,1,1]
p1='SAME'
layer1=tf.nn.conv2d(x_ , w1 , s1 , p1 )+b1
layer1=tf.nn.relu(layer1)
```

```
out_ch2=64
w2=tf.get_variable("w2" , [5,5,out_ch, out_ch2] ,
initializer=tf.contrib.layers.xavier_initializer())
b2=tf.Variable(tf.constant(0.1) ,out_ch2)
s2=[1,1,1,1]
layer2=tf.nn.conv2d(layer1, w2 , s2, padding='SAME')+b2
layer2=tf.nn.relu(layer2)
```

```
end_conv_layer=layer5
flatten_layer=tf.contrib.layers.flatten(end_conv_layer)
length=flatten_layer.get_shape()[1]
fc_w1=tf.get_variable("fc_w1" ,[length,n_classes])
fc_b1=tf.Variable(tf.constant(0.1) , n_classes)
y_conv=tf.matmul(flatten_layer ,fc_w1 )+fc_b1
```

CNTK Concept

```
def create_model(features):
    with C.layers.default_options(init=C.glorot_uniform(), activation=C.relu):
            h = features
            h = C.layers.Convolution2D(filter_shape=(5,5),
                                       num_filters=8,
                                       strides=(2,2),
                                       pad=True, name='first_conv')(h)
            h = C.layers.Convolution2D(filter_shape=(5,5),
                                       num_filters=16,
                                       strides=(2,2),
                                       pad=True, name='second_conv')(h)
        r = C.layers.Dense(num_output_classes, activation=None, name='classify')(h)
        return r
```

*Context Manager*을 이용해 레이어에 일괄적용할 수 있다
코드를 직관적이고 단순하게 할수 있다.

LEARNING RATE

Tensorflow

```
def begin(self):
    self._lrn_rate = 0.1
def before_run(self, run_context):
    return tf.train.SessionRunArgs(cls_resnet.global_step , feed_dict={cls_resnet.lrn_rate : self._lrn_rate})
def after_run(self , run_context , run_values):
    train_step = run_values.results
    if train_step < 40000:
        self._lrn_rate=0.1
    elif train_step < 60000:
        self._lrn_rate = 0.01
    elif train_step < 80000:
        self._lrn_rate = 0.001
    else:
        self._lrn_rate = 0.0001
```

CNTK Concept

lr_per_minibatch = learning_rate_schedule(0.125 , UnitType.minibatch)

class **UnitType** [source]

Bases: `enum.Enum`

Deprecated:: 2.2

Indicates whether the values in the schedule are specified on the per-sample or per-minibatch basis.

`minibatch= 'minibatch'`

Schedule contains per-minibatch values (and need to be re-scaled by the learner using the actual minibatch size in samples).

`sample= 'sample'`

Schedule contains per-sample values.

```
lrs = cntk.learning_rate_schedule([0.001]*12 + [0.0005]*6
```

LEARNING RATE

`learning_rate_schedule`(*lr, unit, epoch_size=None*) [source]

Deprecated:: 2.2

Create a learning rate schedule (using the same semantics as `training_parameter_schedule()`).

Parameters:

- **lr** (*float or list*) – see parameter `schedule` in `training_parameter_schedule()`.
- **unit** (`UnitType`) – see parameter `unit` in `training_parameter_schedule()`.

  deprecated:: 2.2

  Use minibatch_size parameter to specify the reference minbiatch size instead.

- **epoch_size** (*int*) – see parameter `epoch_size` in `training_parameter_schedule()`.

Returns: learning rate schedule

See also

`training_parameter_schedule()`

LOGGING

```
summary_hook = tf.train.SummarySaverHook(save_steps=100, output_dir=FLAGS.train_dir,\
                                         summary_op=tf.summary.merge([cls_resnet.summaries, tf.summary.scalar('Precision',

logging_hook = tf.train.LoggingTensorHook(tensors={'step': cls_resnet.global_step ,
                                                   'loss': cls_resnet.cost,
                                                   'precision':precision} , every_n_iter=100)
```

---

cntk.logging.progress_print module

cntk.logging.progress_print module

class ProgressPrinter(*freq=None, first=0, tag='', log_to_file=None, rank=None, gen_heartbeat=False, num_epochs=None, test_freq=None, test_first=0, metric_is_pct=True, distributed_freq=None, distributed_first=0*) [source]

Bases: cntk.cntk_py.ProgressWriter

Allows printing various statistics (e.g. loss and metric) as training/evaluation progresses.

Parameters:
- freq (*int* or *None*, default *None*) – determines how often printing of training progress will occur. A value of 0 means a geometric schedule (1,2,4,...). A value > 0 means an arithmetic schedule (print for minibatch number: `freq`, print for minibatch number: `2 * freq`, print for minibatch number: `3 * freq`,...). A value of None means no per-minibatch log.

```
aggregate_loss 149.235839844
     2.94        3.17       0.856        0.85         180
     4.19        5.13       0.871       0.883         420
     5.21        6.09       0.866        0.86         900
     4.86        4.53       0.859       0.853        1860
     3.66        2.49       0.765       0.673        3780
     2.45        1.26        0.57       0.378        7620
     1.59       0.749       0.397       0.225       15300
     1.06        0.52       0.275       0.153       30660
    0.731       0.406       0.198        0.12       61380
    0.539       0.347        0.15       0.102      122820
    0.423       0.308        0.12      0.0903      245700
    0.344       0.266      0.0983      0.0765      491460
    0.267        0.19      0.0761       0.054      982980
    0.187       0.107      0.0531        0.03     1966020
    0.118       0.049       0.033      0.0129     3932100
   0.0672      0.0163      0.0178     0.00253     7864260
   0.0357     0.00422     0.00894    0.000116    15728580
0.00339293479919
 error rate on an unseen minibatch : 0.0199

Out[8]:

(0.03144461323818896, 0.0199)
```

CNTK TRAINER

```
with tf.train.MonitoredTrainingSession(
    checkpoint_dir = FLAGS.log_root ,
    hooks=[logging_hook , _LearningRateSetterHook()],
    chief_only_hooks=[summary_hook],
    save_summaries_steps=0,
    config=tf.ConfigProto(allow_soft_placement=True)) as mon_sess:
 while not mon_sess.should_stop():
    mon_sess.run(cls_resnet.train_op)
```

cntk.train.trainer module

A trainer encapsulates the overall training process and employs one or more `learners` to tune the parameters of a specified model using gradients of parameters w.r.t. a training objective.

class **Trainer**(*model, criterion, parameter_learners, progress_writers=None*) [source]

Bases: `cntk.cntk_py.Trainer`

Class for training the model parameters of a models' specified loss function, using the specified set of `parameter_learners` for updating the model's parameters using computed gradients. An optional specified metric function, which can be non-differentiable, can be used for tracking the trained model's quality.

Parameters:

- **model** (`Function`) – root node of the function to train
- **criterion** (tuple of `Function` or `Variable`) – Function with one or two outputs, representing loss and, if given, evaluation metric (in this order). Alternatively, a tuple(loss Function, evaluation Function) is also accepted.
- **parameter_learners** (*list*) – list of learners from `cntk.learners`
- **progress_writers** (*progress writer or list of them*) – optionally, list of progress writers from `cntk.logging` to automatically track training progress.

trainer = cntk.Trainer(z , (ce , pe) , [sgd(z.parameters , lr=lr_per_minibatch)] ,[progress_printer])

```
@typemap
def sgd(parameters, lr,                                                    [docs]
        l1_regularization_weight=0.0, l2_regularization_weight=0.0,
        gaussian_noise_injection_std_dev=0.0, gradient_clipping_threshold_per_sample=np.inf,
        gradient_clipping_with_truncation=True, use_mean_gradient=None,
        minibatch_size=None, epoch_size=None):
    '''sgd(parameters, lr, l1_regularization_weight=0, l2_regularization_weight=0, gaussian_noi
    Creates an SGD learner instance to learn the parameters. See [1] for more
    information on how to set the parameters.

Args:
    parameters (list of parameters): list of network parameters to tune.
```

MINIBATCH

```
return C.io.MinibatchSource(ctf,
        randomize = is_training, max_sweeps = C.io.INFINITELY_REPEAT if is_training else 1)
```

class **MinibatchSource**(*deserializers*, *max_samples=cntk.io.INFINITELY_REPEAT*, *max_sweeps=cntk.io.INFINITELY_REPEAT*, *randomization_window_in_chunks=cntk.io.DEFAULT_RANDOMIZATION_WINDOW*, *randomization_window_in_samples=0*, *randomization_seed=0*, *trace_level=cntk.logging.get_trace_level()*, *multithreaded_deserializer=None*, *frame_mode=False*, *truncation_length=0*, *randomize=True*) [source]

Bases: `cntk.cntk_py.MinibatchSource`

Parameters:

- **deserializers** (a single deserializer or a *list*) – deserializers to be used in the composite reader
- **max_samples** (*int*, defaults to `cntk.io.INFINITELY_REPEAT`) – The maximum number of input samples (not 'label samples') the reader can produce. After this number has been reached, the reader returns empty minibatches on subsequent calls to `next_minibatch()`. *max_samples* and *max_sweeps* are mutually exclusive, an exception will be raised if both have non-default values. **Important:** Click here for a description of input and label samples.

MAKE MODEL FLOW

Model Define

```
#layer 1

input_features=tf.placeholder(x_shape)

w1=tf.get_variable(w_shape , initializer )

b1= tf. get_variabe(w_shape , initalizer )

layer1=tf.matmul(x,w)+b

layer1=tf.relu(layer1)

#layer 2

w2=tf.get_variable(w_shape , initializer )

b2= tf. get_variabe(w_shape , initalizer )

layer2=tf.matmul(layer1,w2)+b2

Layer2 = tf.nn.relu(layer2)
```

Model Define

```
my_model = Sequential(layers=[Dense(hidden_dimension ,activation=cntk.sigmoid)\

,Dense(outputs)])

my_model(input_featrues)
```

INPUT DATA AND RUN GRAPH

Tensorflow Concept

```
cost= tf.nn.sofrmax_cross_entropy_with_logits(labels , pred)

train=GradientOptimizer(learning_rate).minimize(cost)
```

```
sess.run(train , feed_dict ={train_images , train_labels})
```

CNTK Concept

```
ce = cntk.cross_entropy_with_softmax(z , label)

pe = cntk.classification_error(z, label)

trainer = cntk.Trainer(model=z , criterion=(ce , pe) ,
                       parameter_learners=[sgd(z.parameters ,lr=lr_per_minibatch)] ,
                       progress_writers=[progress_printer])
```

```
trainer.train_minibatch({features : train_images , label : train_labels})
```

CNTK EXAMPLE

2 layers : Fully connected Layers

```
def ffnet():
    inputs = 2
    outputs = 2
    layers = 2
    hidden_dimension = 50

    # input variables denoting the features and label data
    features = C.input_variable((inputs), np.float32)
    label = C.input_variable((outputs), np.float32)

    # Instantiate the feedforward classification model
    my_model = Sequential ([
                    Dense(hidden_dimension, activation=C.sigmoid),
                    Dense(outputs)])
    z = my_model(features)

    ce = C.cross_entropy_with_softmax(z, label)
    pe = C.classification_error(z, label)

    # Instantiate the trainer object to drive the model training
    lr_per_minibatch = learning_rate_schedule(0.125, UnitType.minibatch)
    progress_printer = ProgressPrinter(0)
    trainer = C.Trainer(z, (ce, pe), [sgd(z.parameters, lr=lr_per_minibatch)], [progress_pr

    # Get minibatches of training data and perform model training
    minibatch_size = 25
    num_minibatches_to_train = 1024

    aggregate_loss = 0.0
    for i in range(num_minibatches_to_train):
        train_features, labels = generate_random_data(minibatch_size, inputs, outputs)
        # Specify the mapping of input variables in the model to actual minibatch data to b
        trainer.train_minibatch({features : train_features, label : labels})
        sample_count = trainer.previous_minibatch_sample_count
        aggregate_loss += trainer.previous_minibatch_loss_average * sample_count

    last_avg_error = aggregate_loss / trainer.total_number_of_samples_seen

    test_features, test_labels = generate_random_data(minibatch_size, inputs, outputs)
    avg_error = trainer.test_minibatch({features : test_features, label : test_labels})
    print(' error rate on an unseen minibatch: {}'.format(avg_error))
    return last_avg_error, avg_error

np.random.seed(98052)
ffnet()
```

노드 형식의 ***Graph***구조를 가짐

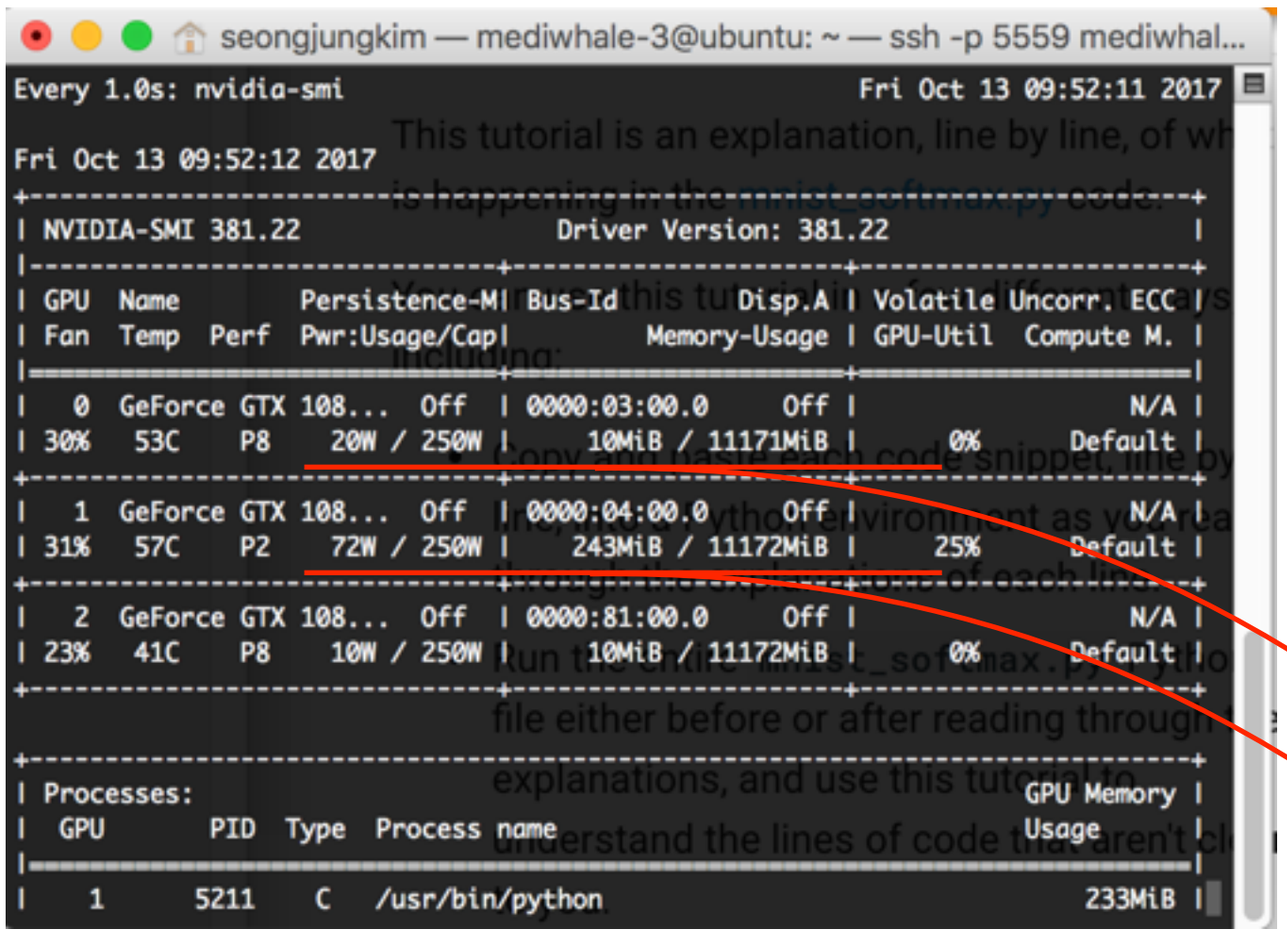
MNIST DATA 을 이용한 CNTK 실습

- Make Fully Connected Layers
- Make Convolution Layers

텐서 플로랑 충돌이 일어나지 않을까?

Virtualenv 을 사용 , 가상환경을 만들어 기존에 시스템이 망가지지 않고 새로운 시스템만 만들어 사용.

Anaconda 을 사용 , 가상환경을 만들어 기존 시스템이 망가트리지 않고 새로운 시스템을 만든다.

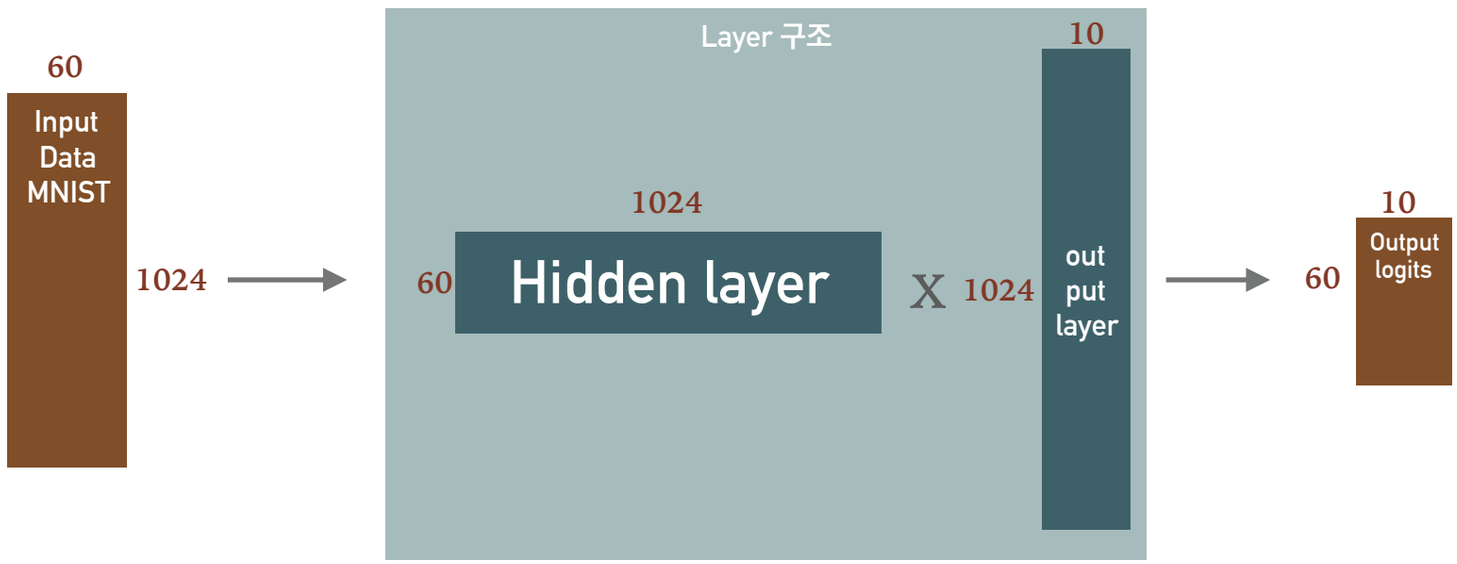


tensorflow의 main gpu는 0번

cntk의 **main gpu**를 **1**번으로 설정해

tensorflow와 같이 돌릴수 있다.

사용시간 분석



Tensorflow

```
Every 1.0s: nvidia-smi                                  Fri Oct 13 13:56:39 20
Fri Oct 13 13:56:39 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 381.22                 Driver Version: 381.22                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 0000:03:00.0     Off |                  N/A |
| 32%   58C    P2    93W / 250W |  10689MiB / 11171MiB |     27%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 0000:04:00.0     Off |                  N/A |
| 29%   53C    P2    61W / 250W |  10623MiB / 11172MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  GeForce GTX 108...  Off  | 0000:81:00.0     Off |                  N/A |
| 23%   44C    P2    58W / 250W |  10623MiB / 11172MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

328.12

CNTK

```
Every 1.0s: nvidia-smi                                  Fri Oct 13 14:00:03 2017
Fri Oct 13 14:00:03 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 381.22                 Driver Version: 381.22                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 0000:03:00.0     Off |                  N/A |
| 36%   59C    P8    22W / 250W |     10MiB / 11171MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 0000:04:00.0     Off |                  N/A |
| 42%   71C    P2    88W / 250W |    339MiB / 11172MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  GeForce GTX 108...  Off  | 0000:81:00.0     Off |                  N/A |
| 23%   42C    P8    10W / 250W |     10MiB / 11172MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

321.659117937

TIME

	Tensorflow	*CNTK*
2fully	328.12	321.6591
3conv 1 fully	1521.6934	1521.1144
5conv 1 fully	1821.1	1781.4772

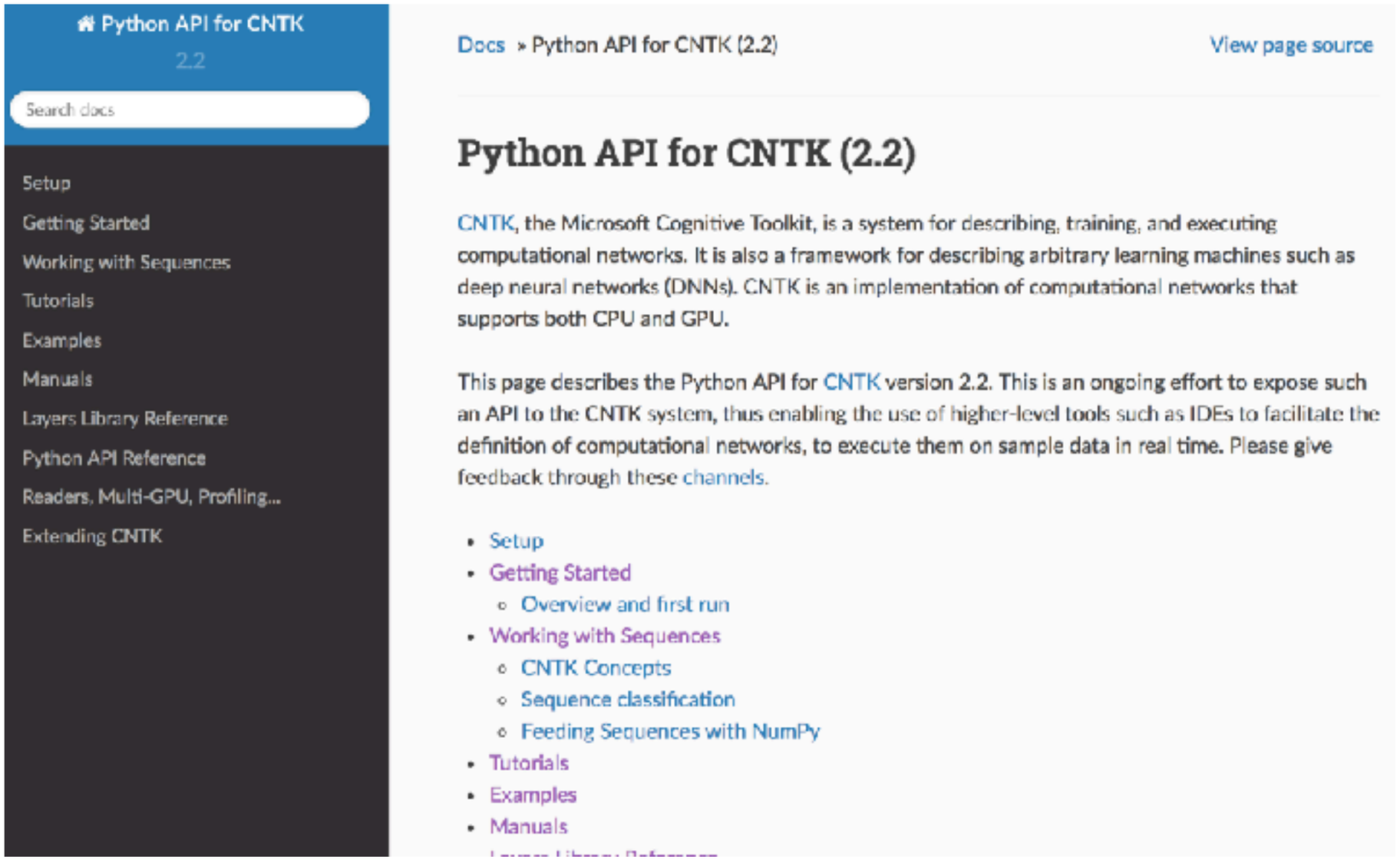
CNTK TUTORIAL

https://cntk.ai/pythondocs/gettingstarted.html

https://notebooks.azure.com/cntk/libraries/tutorials

에저 환경에서 튜토리얼을 실행해볼수 있다

https://www.cntk.ai/pythondocs/



CNTK 장점

1.간단하다.

2.직관적이다.

3.GPU 효율이 좋다. GPU을 병렬로 사용하는데 효과적이다.

4.Open source 고 Main Page(cntk.ai)source 에 대한 주석이 잘 정리되어 있어

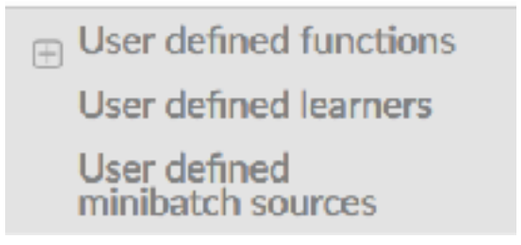
코드 읽기가 편하다.

5. 기존 라이브러리에 연구자들이 연구를 분석하기 쉽도록 많은 주석이 제공되어 연구 분석에 용이하다.

CNTK 단점

1. 기존 라이브러리에 제약이 많다.(Hook 형식의 library 들이 많아서)

2. 코드를 수정해 사용자의 편의에 맞게 수정하는게 까다롭다.



(이 점은 User defined functions , learner , minibatch 부분들을 좀더 자세히 살펴보고 해야 말할 수 있을것 같다.)