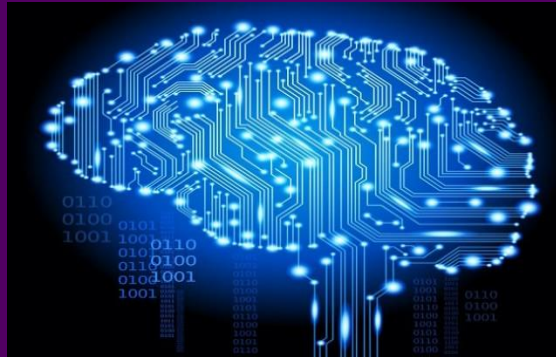


Deep Learning

MSiA 432



Theory and Applications



NORTHWESTERN
UNIVERSITY



NORTHWESTERN
UNIVERSITY

MSIA 432: Deep Learning. Spring 2018.
Instructor: Dr. Ellick Chan. TA: Jaehoon Koo.

COMPUTING RESOURCES



NORTHWESTERN
UNIVERSITY

MSIA 432: Deep Learning. Spring 2018.
Instructor: Dr. Ellick Chan. TA: Jaehoon Koo.

Overview

- GPU Cluster (Deepdish)
- Jupyter server
- Intel Devcloud



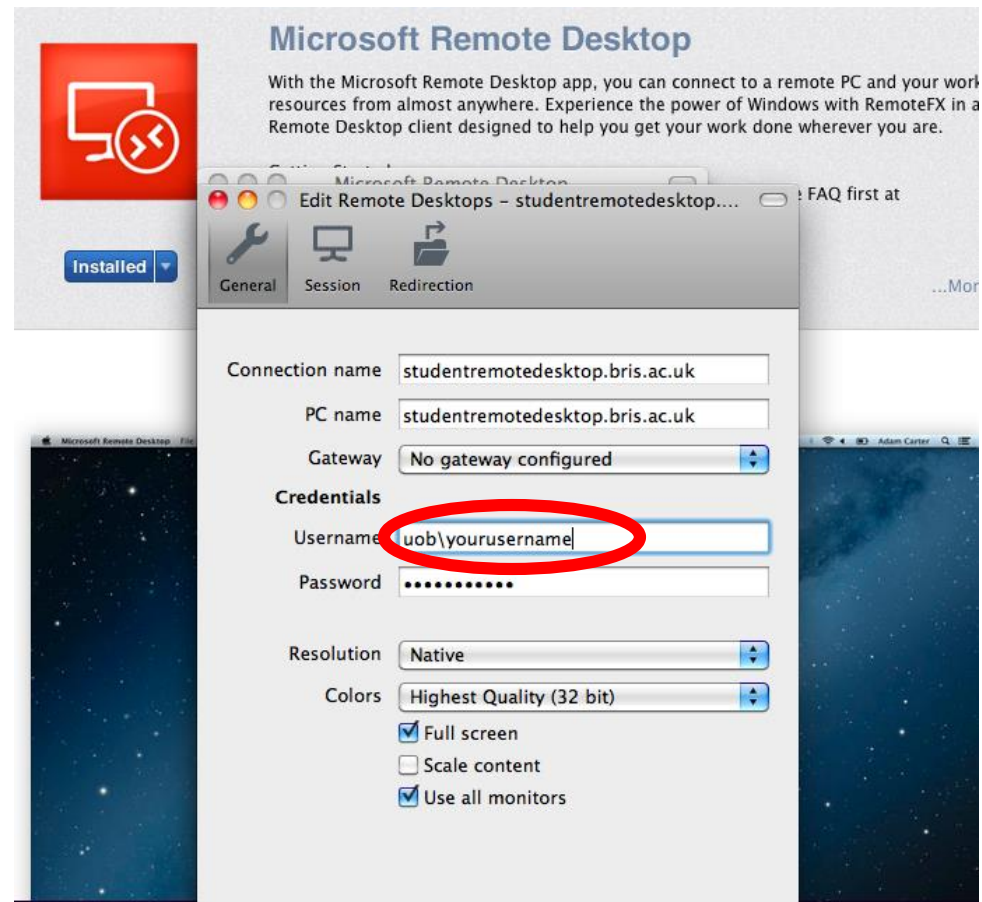
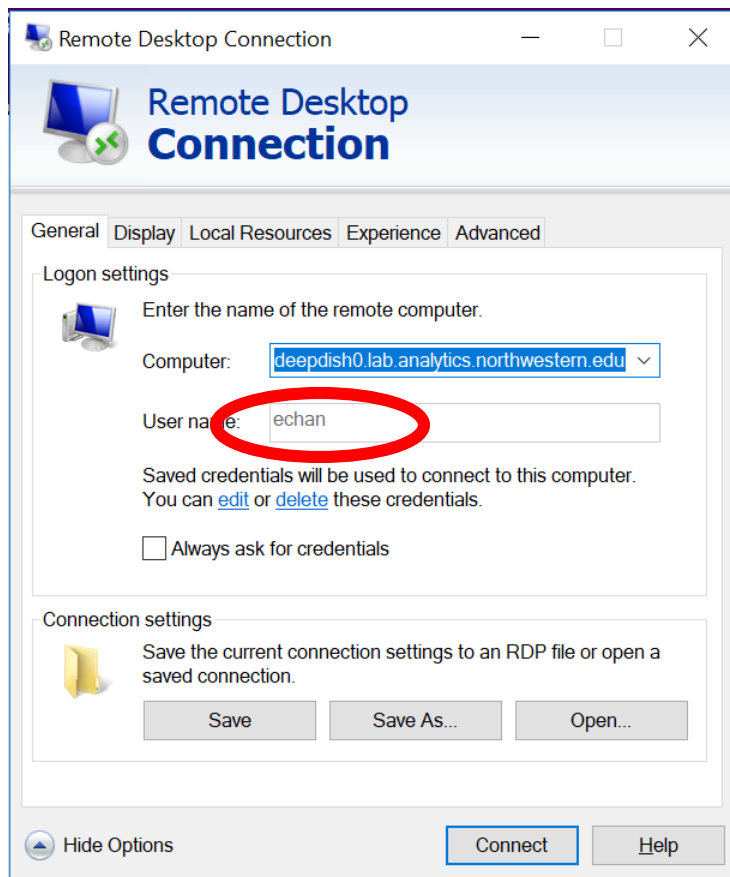
Deepdish GPU cluster

- 2 servers: `deepdish0.lab.analytics.northwestern.edu`, `deepdish1.lab.analytics.northwestern.edu`
- Each server has:
 - 4 NVIDIA GeForce 1080 GPUs
 - 128 GB memory
 - Intel i7 CPU with 6 cores (12 threads)
- Accessible via RDP
 - Windows – mstsc (remote desktop connection)
 - Mac – Microsoft remote desktop client
 - Linux – rdesktop client
- Also available via ssh



Accessing via RDP

- Be sure to provide the credentials at the time of login

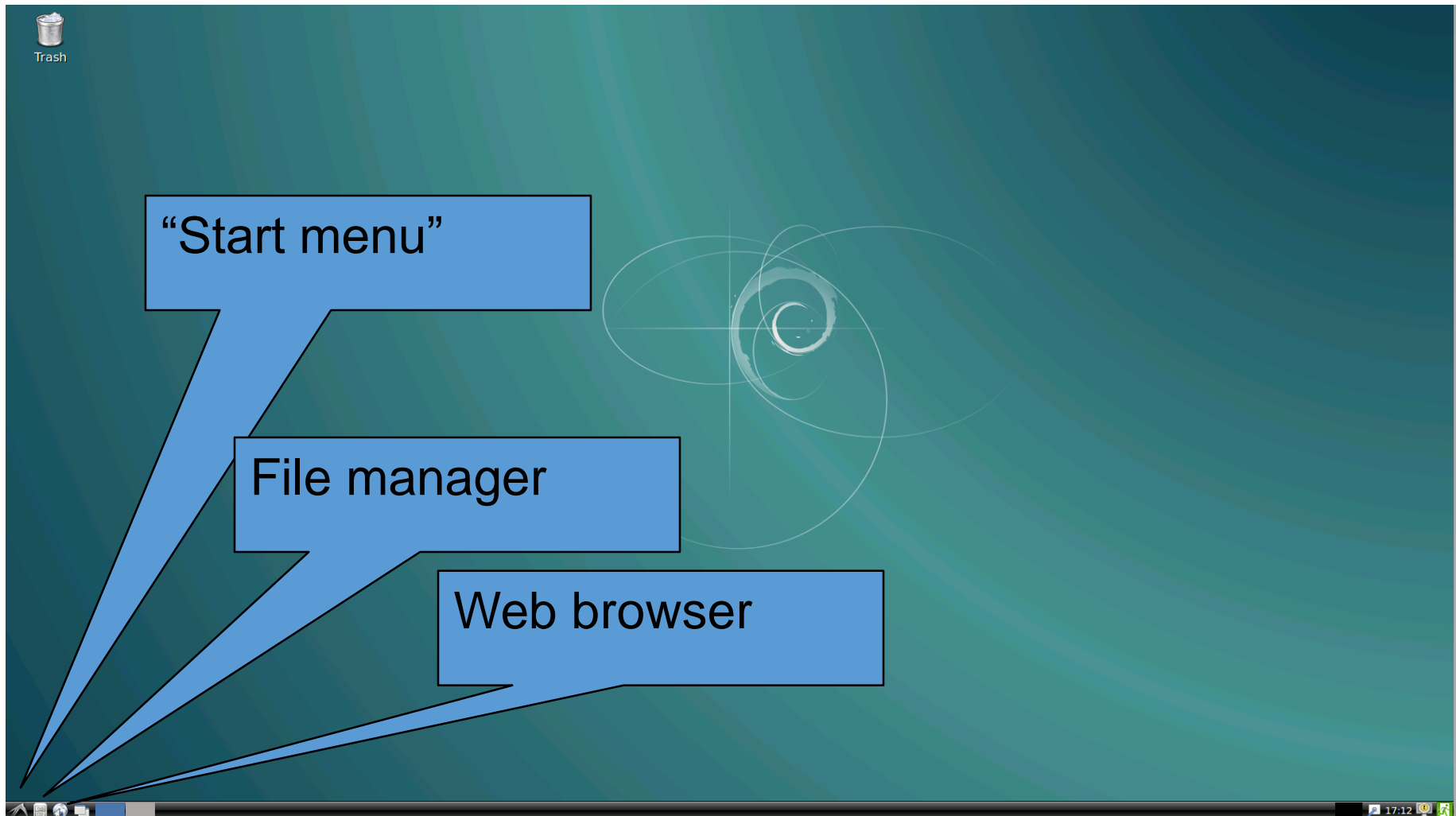




NORTHWESTERN
UNIVERSITY

MSIA 432: Deep Learning. Spring 2018.
Instructor: Dr. Ellick Chan. TA: Jaehoon Koo.

Desktop interface

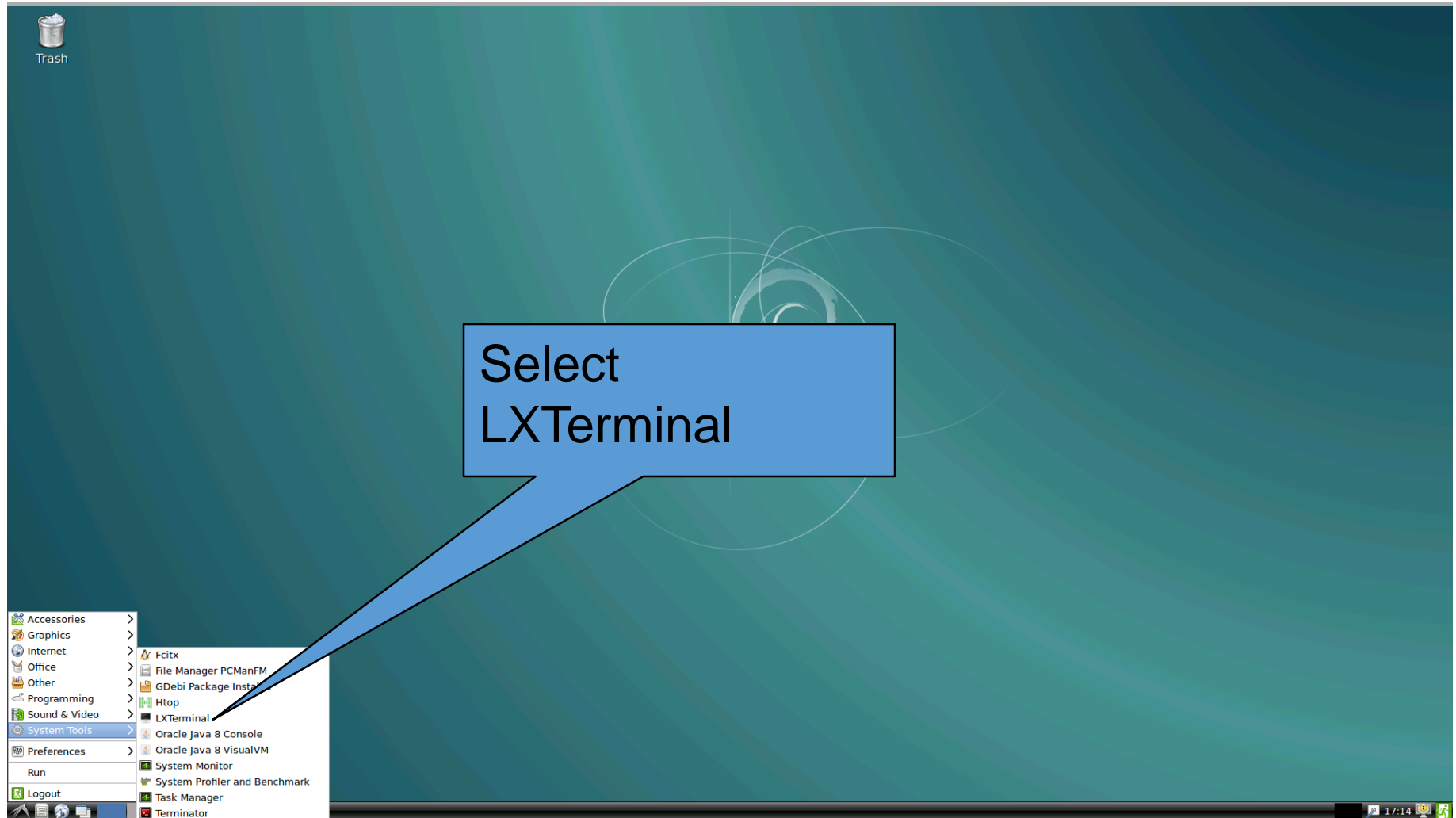




NORTHWESTERN
UNIVERSITY

MSIA 432: Deep Learning. Spring 2018.
Instructor: Dr. Ellick Chan. TA: Jaehoon Koo.

Starting a terminal

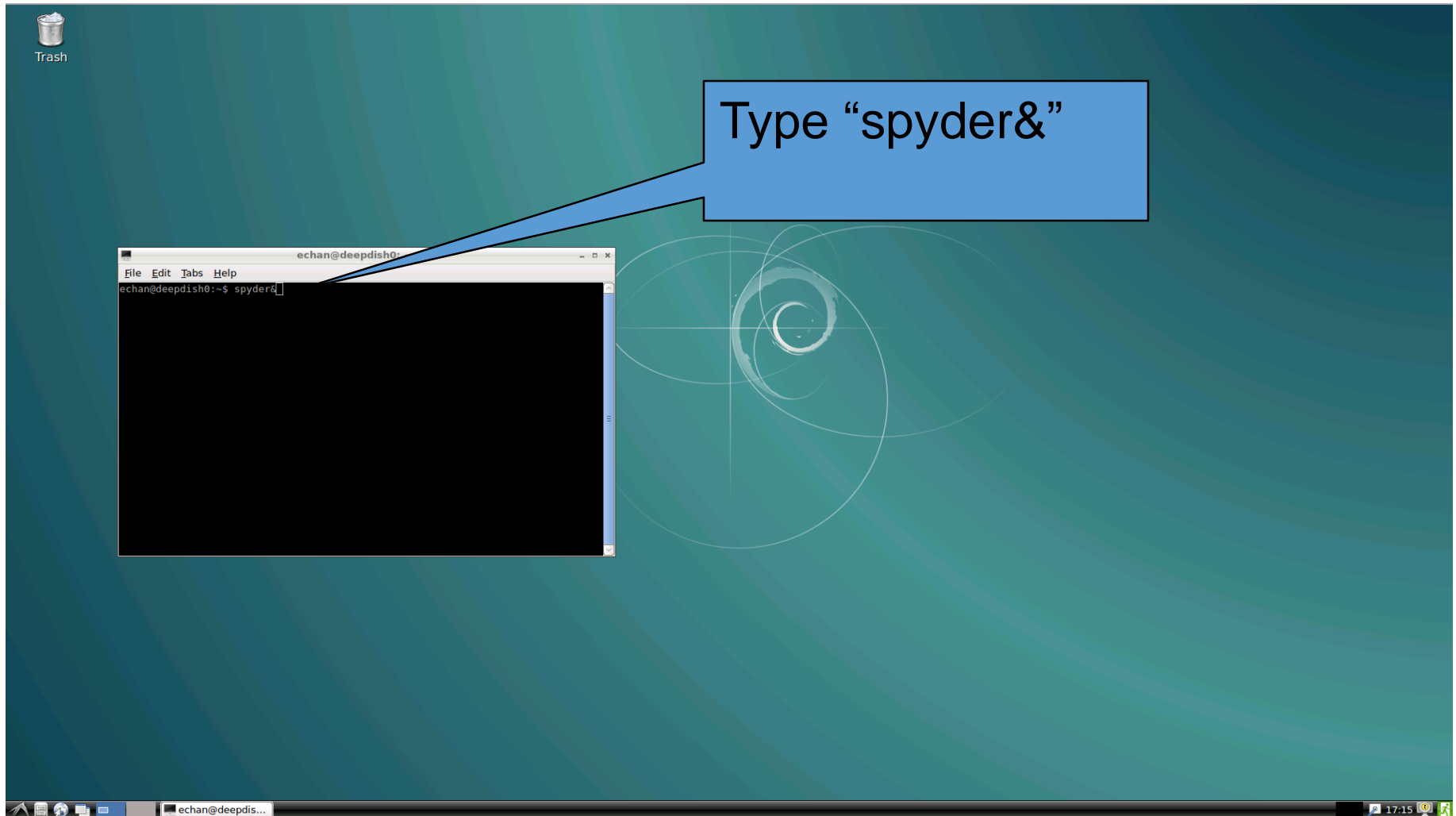




NORTHWESTERN
UNIVERSITY

MSIA 432: Deep Learning. Spring 2018.
Instructor: Dr. Ellick Chan. TA: Jaehoon Koo.

Starting Spyder





NORTHWESTERN
UNIVERSITY

MSIA 432: Deep Learning. Spring 2018.
Instructor: Dr. Ellick Chan. TA: Jaehoon Koo.

Using Spyder

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script for generating text from Nietzsche's writings. The script includes comments, imports, and code for loading data, preprocessing, and vectorizing sentences. Two blue callout boxes with arrows point to specific icons in the Spyder toolbar: one labeled "Open file" points to the file icon, and another labeled "Run code" points to the green play button icon. The IPython console on the right shows the Python version (3.6.0) and IPython version (5.1.0) information. The status bar at the bottom indicates the current file is "lstm_text_generation-charheatmaps.py" and shows various settings like permissions, end-of-lines, encoding, and memory usage.

Open file

Run code

```
'''Example script to generate text from Nietzsche's writings.
2
3 At least 10 epochs are required before the generated
4 starts making coherent.
5
6 It is recommended to run this script on GPU, as recurrent
7 networks are computationally intensive.
8
9 If you try to run on new data, make sure your corpus
10 has at least 100000 characters. ~1M is better.
11 '''
12
13 from __future__ import print_function
14 from keras.models import Sequential
15 from keras.layers import Dense, SimpleRNN, LSTM
16 from keras.preprocessing.text import Tokenizer
17 from keras.preprocessing.sequence import pad_sequences
18 from keras.utils.np_utils import to_categorical
19 import numpy as np
20 import random
21 import sys
22 from collections import Counter
23 from sklearn.metrics import accuracy_score
24
25 #modeltype = 'lstm'
26 modeltype = 'cnn'
27 modelName = 'model-text-' + socket.gethostname() + '-' + modeltype + '.h5'
28 print('Model:', modelName)
29
30 path = get_file('nietzsche.txt', origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt')
31 text = open(path).read().lower()
32 print('corpus length:', len(text))
33
34 #chars = sorted(list(set(text)))
35 # Sort characters by frequency
36 chars = dict(sorted(Counter(text).items(), key=lambda x:x[1], reverse=True)).keys()
37 print('total chars:', len(chars))
38 char_indices = dict((c, i) for i, c in enumerate(chars))
39 indices_char = dict((i, c) for i, c in enumerate(chars))
40
41 # cut the text in semi-redundant sequences of maxlen characters
42 maxlen = 40
43 step = 3
44 sentences = []
45 next_chars = []
46 for i in range(0, len(text) - maxlen, step):
47     sentences.append(text[i: i + maxlen])
48     next_chars.append(text[i + maxlen])
49 print('nb sequences:', len(sentences))
50
51 def vectorize(sentences):
52     X = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
53     y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
54     for i, sentence in enumerate(sentences):
55         for t, char in enumerate(sentence):
56             X[i, t, char_indices[char]] = 1
57             y[i, char_indices[next_chars[t]]] = 1
58
59 if modeltype == 'cnn': return np.expand_dims(X, axis=3), y
60 return X, y
```

Python 3.6.0 [Anaconda custom (64-bit)] (default, Dec 23 2016, 12:22:00)
Type "copyright", "credits" or "license()" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

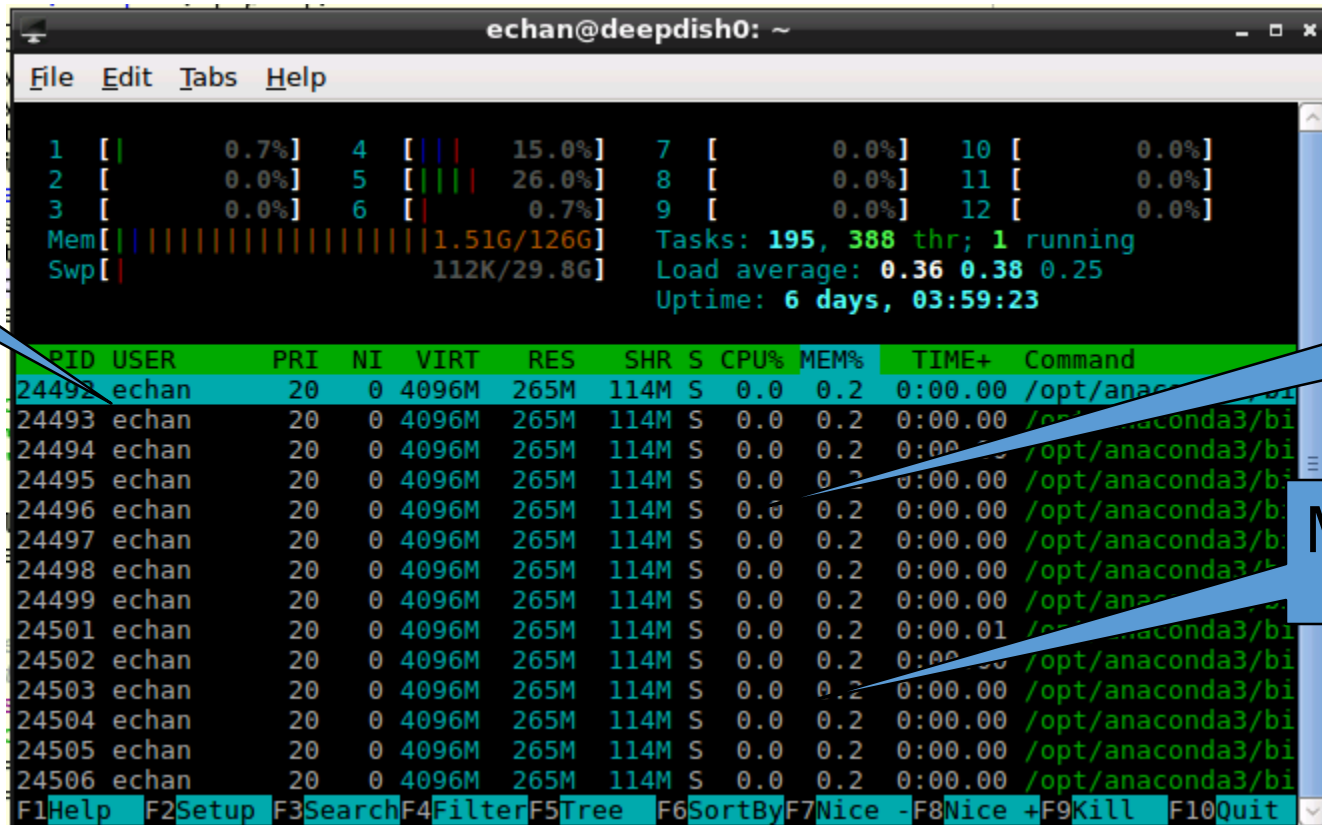
In [1]:

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 22 Column: 32 Memory: 1% CPU: 4%



Monitoring system resources

- System resources are scarce. Be “nice” to other groups.
- CPU/memory - htop





GPU

- Each user is assigned a single GPU at login per session. Please be courteous and only run a single job at a time to ensure fair allocation. We occasionally catch resource hogs and terminate their jobs.
- If a job fails, try the other server, but please terminate your current job.
- Tool to check usage is nvidia-smi

```
echan@deepdish0: ~  
File Edit Tabs Help  
echan@deepdish0:~$ nvidia-smi  
Mon Apr 16 17:24:03 2018  
  
NVIDIA-SMI 384.130 Driver Version: 384.130  
-----  
GPU Name Perf Persistence-M Bus-Id Disp.A Volatile Uncorr. ECC  
Fan Temp Perf Pwr:Usage/Cap Memory-Usage GPU-Util Compute M.  
-----  
0 GeForce GTX 1080 Off 00000000:05:00.0 Off 0% 36C P0 45W / 198W 0MiB / 8113MiB 0% Default  
1 GeForce GTX 1080 Off 00000000:06:00.0 Off 27% 32C P0 39W / 180W 0MiB / 8114MiB 0% Default  
2 GeForce GTX 1080 Off 00000000:09:00.0 Off 0% 36C P0 38W / 180W 0MiB / 8114MiB 1% Default  
3 GeForce GTX 1080 Off 00000000:0A:00.0 Off 0% 44C P2 135W / 198W 1427MiB / 8114MiB 72% Default  
-----  
Processes:  
GPU PID Type Process name GPU Memory  
-----  
3 24601 C /opt/anaconda3/bin/python 1417MiB  
echan@deepdish0:~$
```

Using GPU 3

GPU
currently
72% busy

PID is 24601

1417/8114
MB memory
used



Finding GPU hogs

- NVIDIA-SMI doesn't provide a direct way to map PID (process ID) to user
- Need to use `ps aux` to figure out the owner of the job

```
File Edit Tabs Help
echan@deepdish0:~$ nvidia-smi
Mon Apr 16 17:24:03 2018
+-----+
| NVIDIA-SMI 384.130              |
+-----+-----+
|   Name   | Persistence- |
| Temp  Perf | Pwr:Usage/Ca |
+-----+-----+
| GeForce GTX 1080 | Off          |
| 36C    P0    45W / 198W | 0MiB / 8113MiB | 0% | Default |
+-----+-----+
| GeForce GTX 1080 | Off          |
| 32C    P0    39W / 180W | 0MiB / 8114MiB | 0% | Default |
+-----+-----+
| GeForce GTX 1080 | Off          |
| 36C    P0    38W / 180W | 0MiB / 8114MiB | 1% | Default |
+-----+-----+
| GeForce GTX 1080 | Off          |
| 44C    P2   135W / 198W | 1427MiB / 8114MiB | 72% | Default |
+-----+-----+
| Processes: |
| GPU   PID  Type  Process name                               | GPU Memory |
|-----|-----|-----|-----|
| 3      24601 C    /opt/anaconda3/bin/python                  | Usage      |
+-----+-----+
echan@deepdish0:~$ ps aux | grep 24601
echan  24601 95.0  1.5 21006640 2104628 ?        Ssl  17:16  15:24 /opt/anaconda3/bin/python /opt/anacond
a3/lib/python3.6/site-packages/spyder/utils/ipython/start_kernel.py -f /run/user/1001/jupyter/kernel-f8
a7e4d74c07.json
echan  24818  0.0  0.0 14224 1032 pts/0      S+   17:32  0:00 grep --color=auto 24601
echan@deepdish0:~$
```

Execute: “`ps aux | grep <jobid>`”

Replace 24601
with the job #

Owner is listed
here. Echan in
this case.



GPU rules of the road

- Please have only one job per user execute on the cluster at a time. GPU resources are limited.
- If a user accidentally runs too many jobs, please let them know they have runaway jobs and ask them to terminate the excess jobs.
- Tensorflow by default pre-allocates all GPU memory, and this causes problems with other users. Please apply the fix described on the next slide.



Tensorflow GPU preallocation fix

- Tensorflow by default pre-allocates all GPU memory. This can cause other jobs to fail to launch.
- <https://stackoverflow.com/questions/34199233/how-to-prevent-tensorflow-from-allocating-the-totality-of-a-gpu-memory>
- The preferred fix is to paste the following code at the top of your Python file:

```
### GPU memory fix
import tensorflow as tf, keras.backend.tensorflow_backend as ktf
def get_session(gpu_fraction=0.4):
    gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=gpu_fraction, allow_growth=True)
    return tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
ktf.set_session(get_session())
```

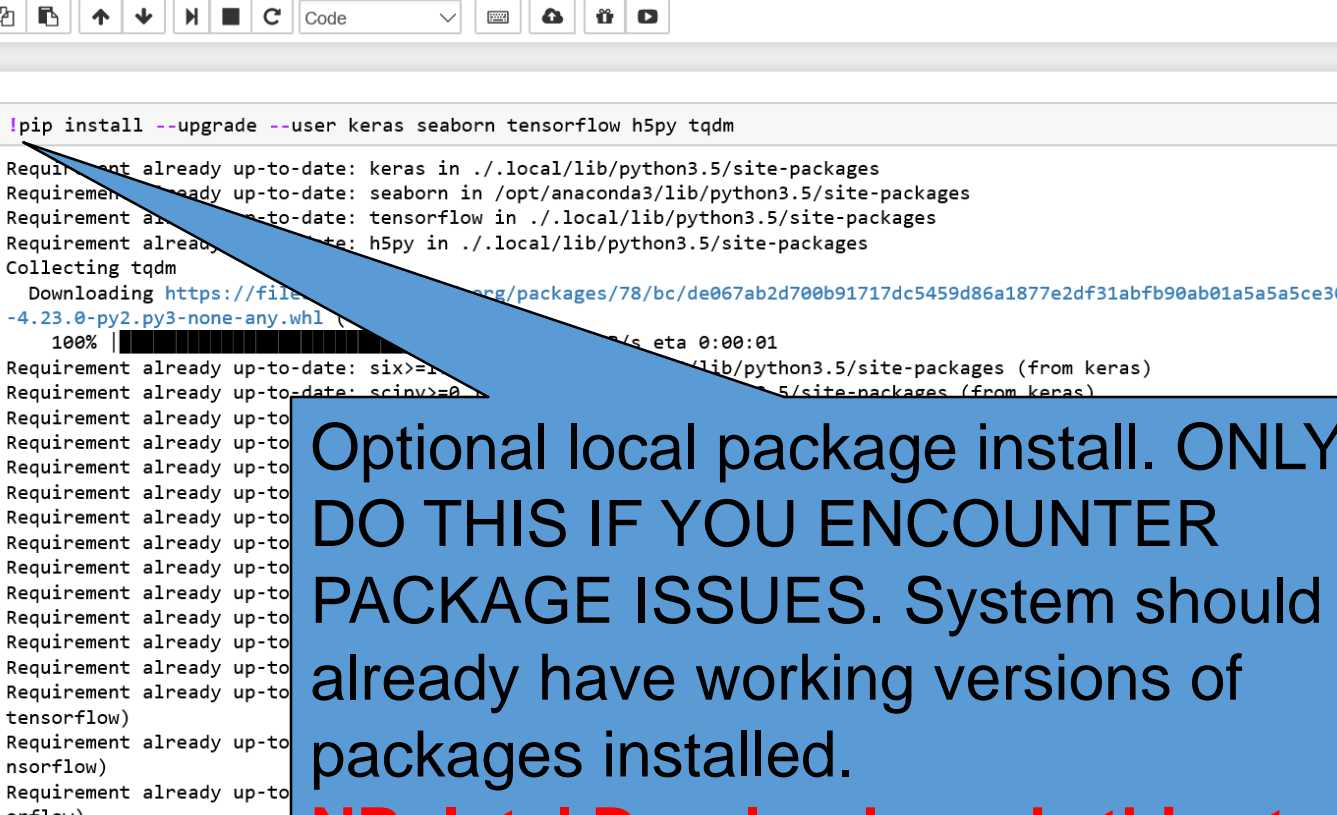


Jupyter notebooks

- Jupyter notebooks have been tested on:
 - <https://app1.lab.analytics.northwestern.edu:8000>
 - Intel AI Devcloud: <https://software.intel.com/en-us/ai-academy/tools/devcloud>
- Sometimes the latest packages are not installed by default, you can install them with “pip install --user <package>”. This installs the package in your home folder.
- Generally, you'll need: tensorflow, keras, tqdm, seaborn



Installing local versions of packages



The screenshot shows a Jupyter Notebook interface. At the top, the Jupyter logo and 'MSIA 432' are visible, along with a timestamp 'Last Checkpoint: a minute ago (unsaved changes)'. The top bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help' menus. A 'Trusted' button and a Python environment selector 'Python [conda env:anaconda3]' are on the right. The toolbar contains icons for saving, adding, deleting, and running cells, as well as a 'Code' dropdown and a 'Run' button. The main area displays a terminal output for the command `!pip install --upgrade --user keras seaborn tensorflow h5py tqdm`. The output shows that several packages (keras, seaborn, tensorflow, h5py) are already up-to-date, and tqdm is being collected. A large blue callout box with black text is overlaid on the right side of the terminal output, containing the following text:

Optional local package install. ONLY DO THIS IF YOU ENCOUNTER PACKAGE ISSUES. System should already have working versions of packages installed.

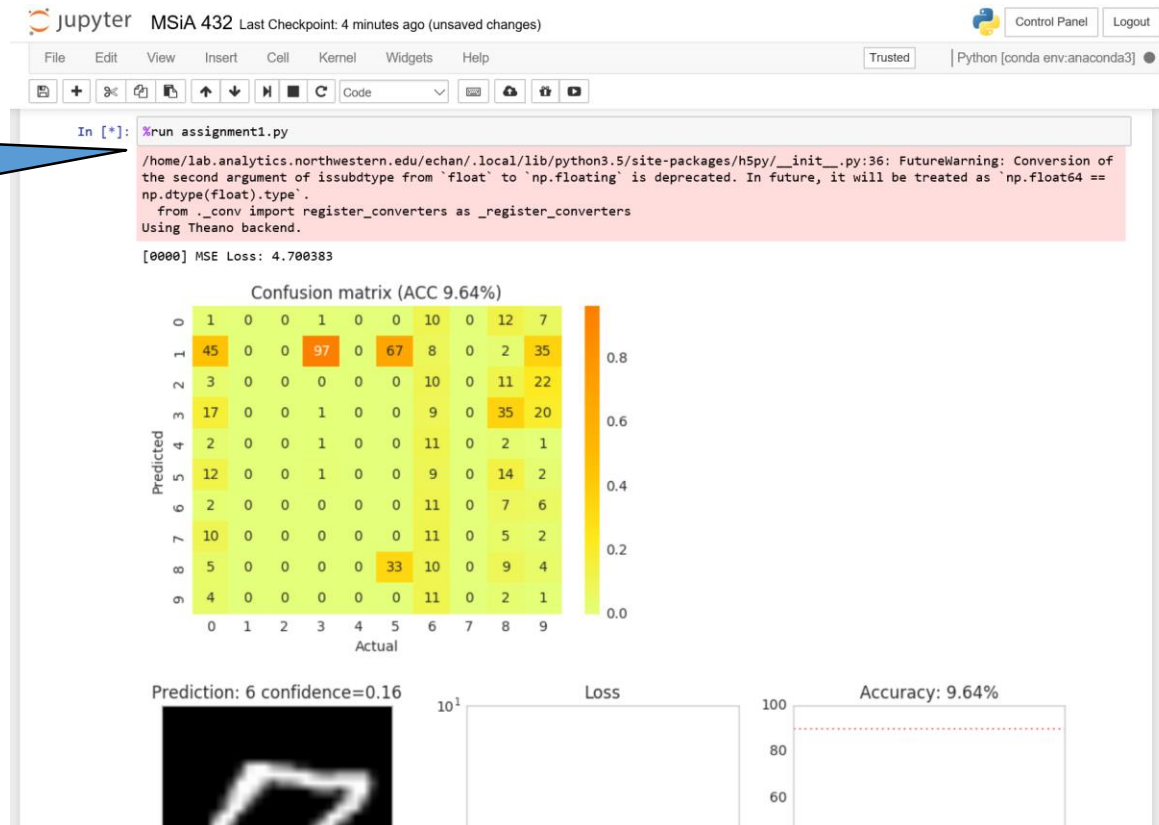
NB: Intel Devcloud needs this step



Running files

- Drag and drop/upload Python code to notebook
- Use “%run” magic to execute

%run
<file>





Loading files

- Either copy/paste file contents or use “%load”

%load
<file>

The screenshot shows a Jupyter Notebook window titled "MSIA 432 Last Checkpoint: 5 minutes ago (unsaved changes)". The interface includes a top bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help" menus. Below the menus is a toolbar with icons for saving, running, and other actions. The main area displays a code cell with the following content:

```
In [ ]: %load assignment1.py

# Homework 1 starter code: Simple neural network
import keras.datasets.mnist, numpy as np, os, matplotlib.pyplot as plt, matplotlib.cm as cm, seaborn, sklearn.manifold
os.makedirs('train', exist_ok=True)
(X, Y), (_, _) = keras.datasets.mnist.load_data()
#Y = np.random.randint(0, 9, size=Y.shape) # Uncomment this line for random labels extra credit
X = X.astype('float32').reshape((len(X), -1)).T / 255.0 # 784 x 60000
T = np.zeros((len(Y), 10), dtype='float32').T # 10 x 60000
for i in range(len(Y)): T[Y[i], i] = 1

%% Setup: 784 -> 256 -> 128 -> 10
W1 = 2*np.random.rand(784, 256).astype('float32').T - 1
W2 = 2*np.random.rand(256, 128).astype('float32').T - 1
W3 = 2*np.random.rand(128, 10).astype('float32').T - 1

lr = 1e-5 # Learning rate, decrease if optimization isn't working
def sigmoid(x): return 1.0/(1.0 + np.e**-x)
losses, accuracies, hw1, hw2, hw3, ma = [], [], [], [], [], []
for i in range(1000): # Do not change this, we will compare performance at 1000 epochs
    # Forward pass
    L1 = sigmoid(W1.dot(X))
    L2 = sigmoid(W2.dot(L1))
    L3 = sigmoid(W3.dot(L2))
    # Backward pass
    dw3 = (L3 - T) * L3*(1 - L3)
    dw2 = W3.T.dot(dw3)*(L2*(1-L2))
    dw1 = W2.T.dot(dw2)*(L1*(1-L1))
    # Update
    W3 -= lr*np.dot(dw3, L2.T)
    W2 -= lr*np.dot(dw2, L1.T)
    W1 -= lr*np.dot(dw1, X.T)

    loss = np.sum((L3 - T)**2)/len(T.T)
    print("[%04d] MSE Loss: %0.6f" % (i, loss))
```



Long-running jobs

- Jupyter wasn't really built for long-running jobs
- YMMV, but it seems to work OK for jobs that run a few hours. Sometimes the jobs time out.
- Use Spyder for longer-running jobs
- You have been warned...