Deep Learning Software

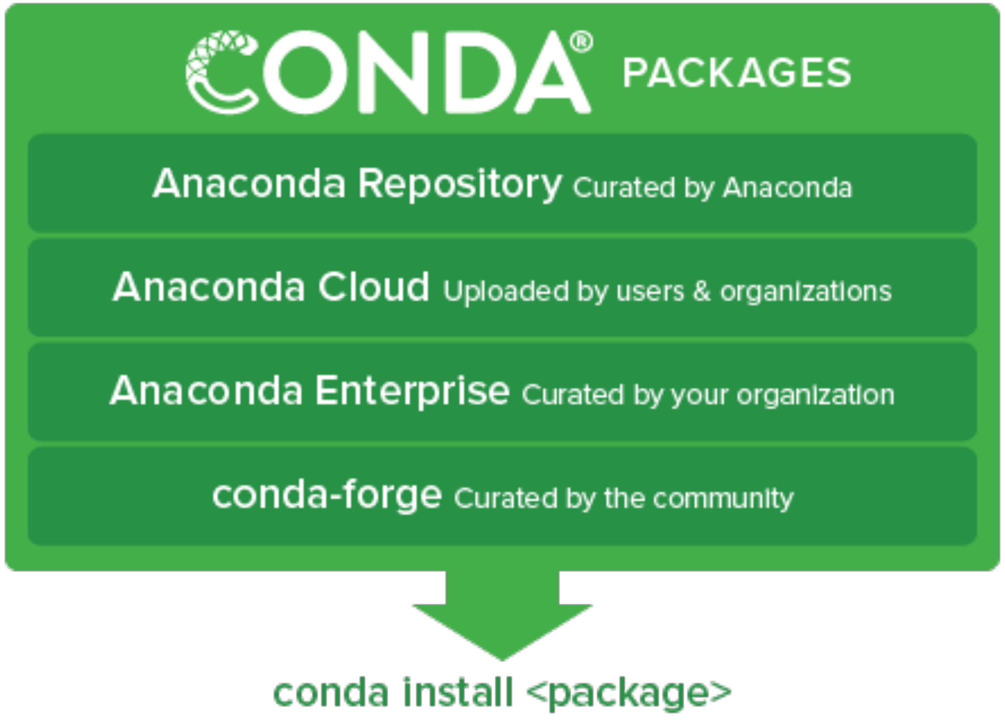
Security and Fairness of Deep Learning SP18

Today

- HW1 is out, due Feb 15th
- Anaconda and Jupyter Notebook
- Deep Learning Software
  - Keras
  - Theano
  - Numpy

Anaconda

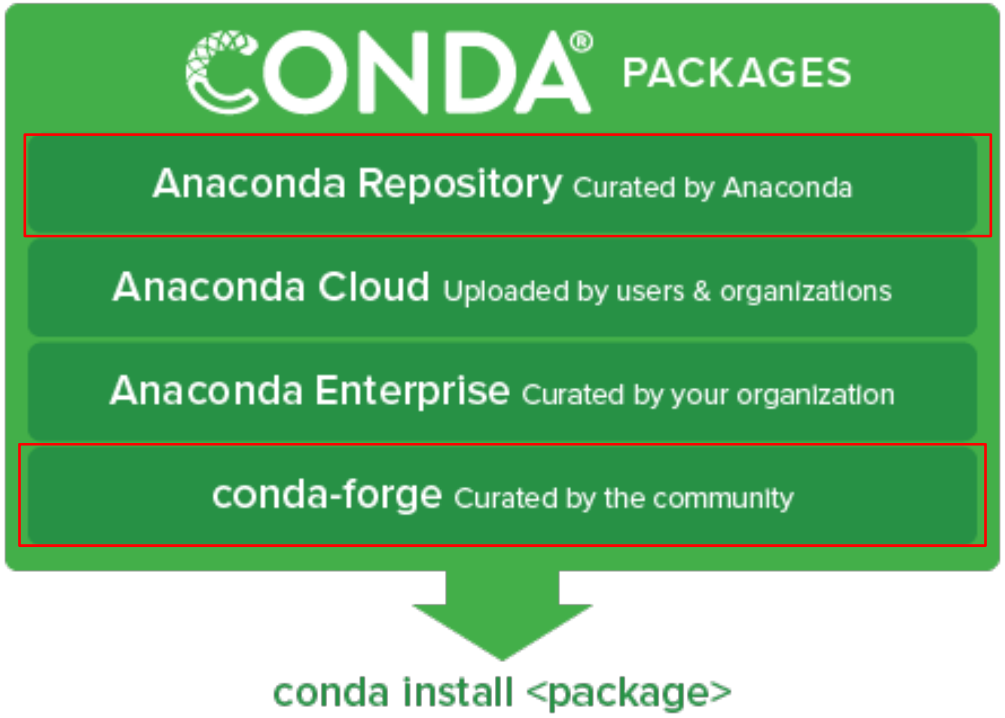
- A package management system for Python



CONDA® PACKAGES
Anaconda Repository Curated by Anaconda
Anaconda Cloud Uploaded by users & organizations
Anaconda Enterprise Curated by your organization
conda-forge Curated by the community
conda install <package>

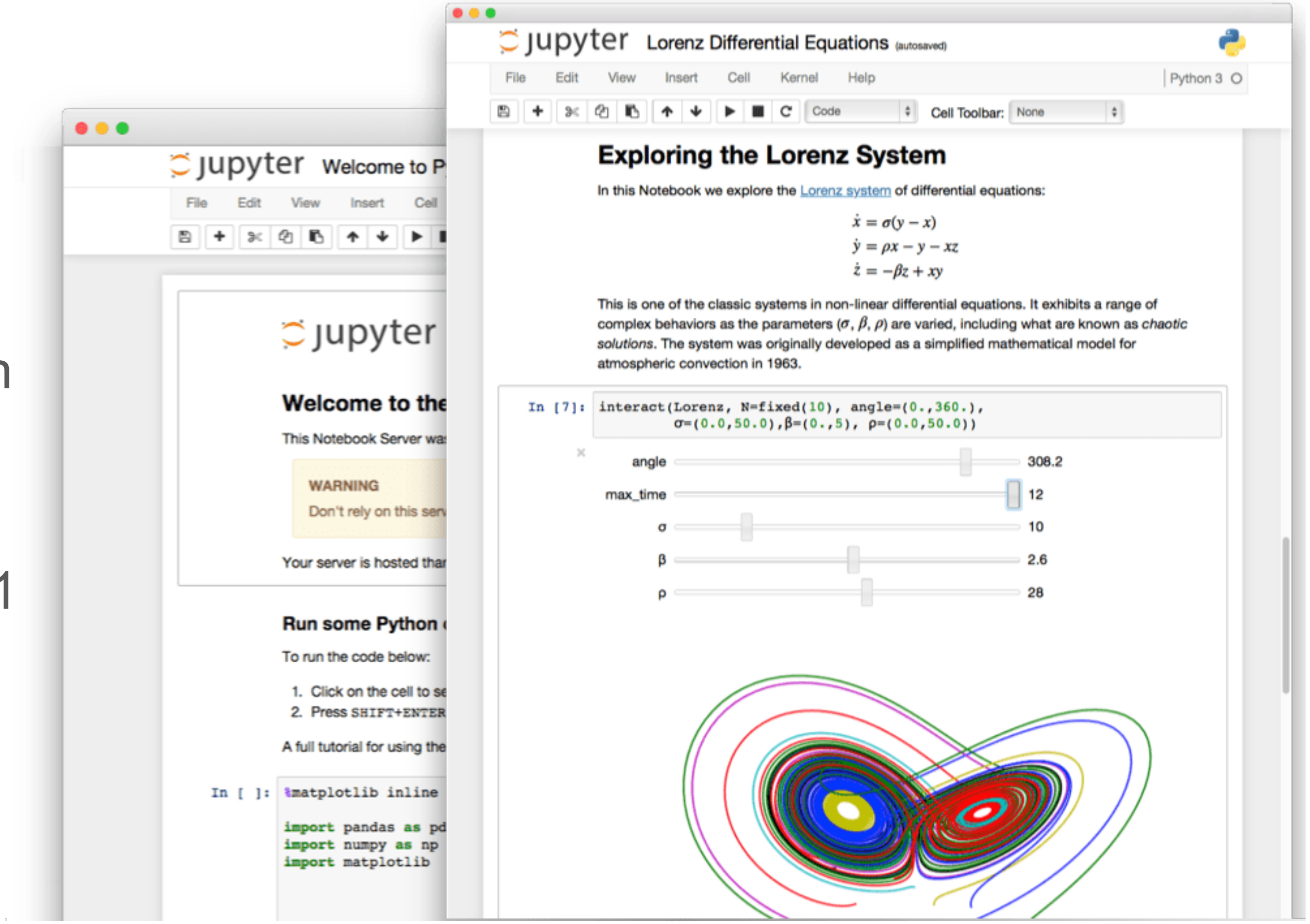
Anaconda

- A package management system for Python


CONDA® PACKAGES
Anaconda Repository Curated by Anaconda
Anaconda Cloud Uploaded by users & organizations
Anaconda Enterprise Curated by your organization
conda-forge Curated by the community
conda install <package>

Jupyter notebook

- A web application that where you can code, interact, record and plot.
- Allow for remote interaction when you are working on the cloud
- You will be using it for HW1



Welcome to the

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

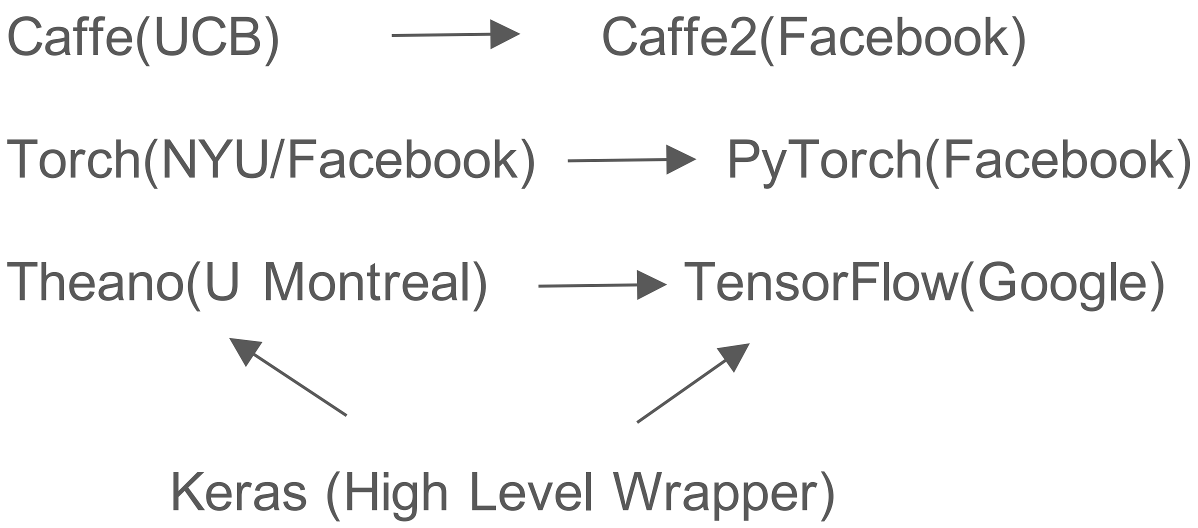
```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

angle 308.2
max_time 12
σ 10
β 2.6
ρ 28

Deep Learning Software

Caffe
Chainer
DL4J
Deeplearning4j
K
KERAS
Microsoft
CNTK
MatConvNet
MINERVA
mxnet
Purine
TensorFlow
theano
torch

Deep Learning Software



Caffe(UCB) → Caffe2(Facebook)

Torch(NYU/Facebook) → PyTorch(Facebook)

Theano(U Montreal) → TensorFlow(Google)

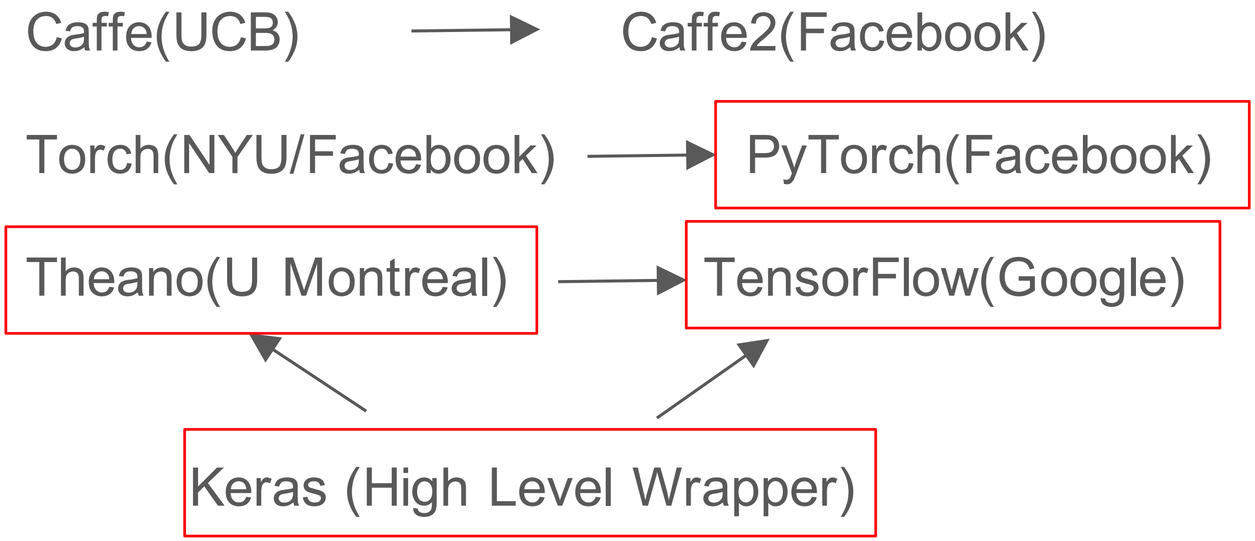
Keras (High Level Wrapper)

Paddle (Baidu)

CNTK(Microsoft)

MXNet(Amazon)

Deep Learning Software: Most Popular



Caffe(UCB)

Caffe2(Facebook)

Torch(NYU/Facebook)

PyTorch(Facebook)

Theano(U Montreal)

TensorFlow(Google)

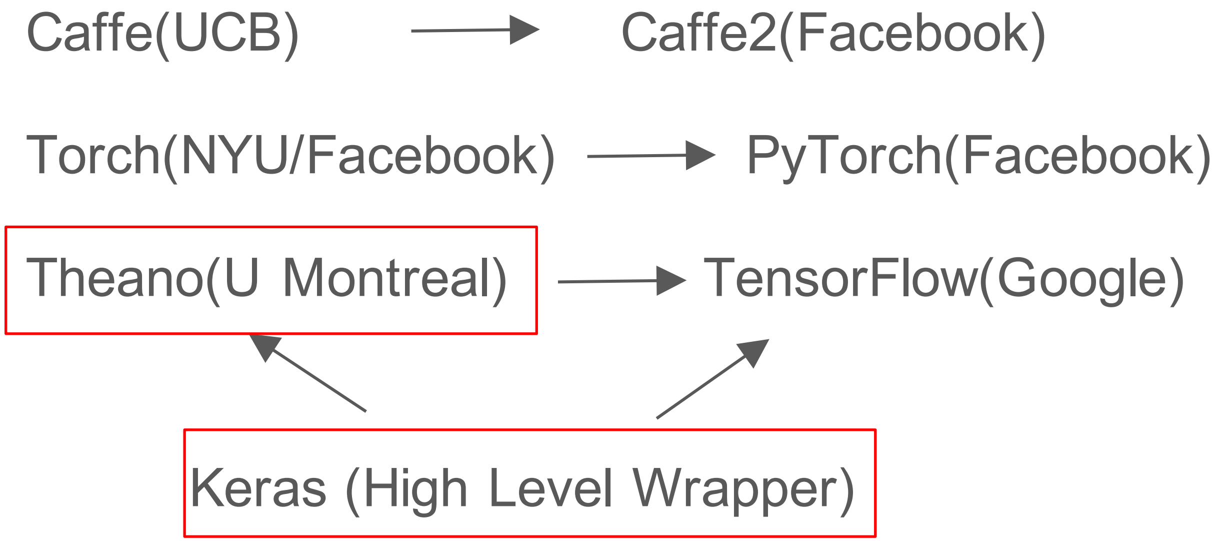
Keras (High Level Wrapper)

Paddle (Baidu)

CNTK(Microsoft)

MXNet(Amazon)

Deep Learning Software: Today



Caffe(UCB) → Caffe2(Facebook)

Torch(NYU/Facebook) → PyTorch(Facebook)

Theano(U Montreal) → TensorFlow(Google)

Keras (High Level Wrapper)

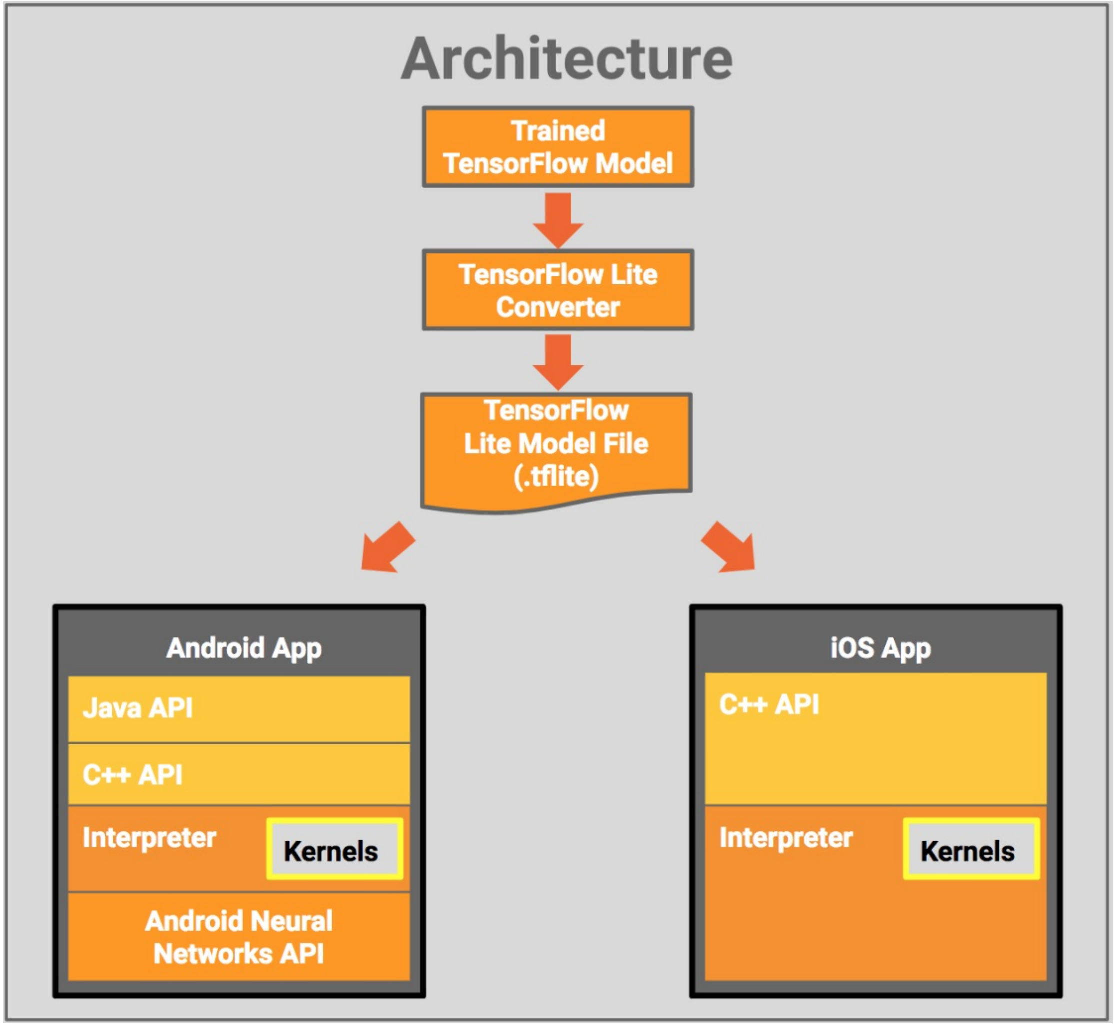
Paddle (Baidu)

CNTK(Microsoft)

MXNet(Amazon)

Mobile Platform

- Tensorflow Lite:
  - Released last November



Why do we use deep learning frameworks?

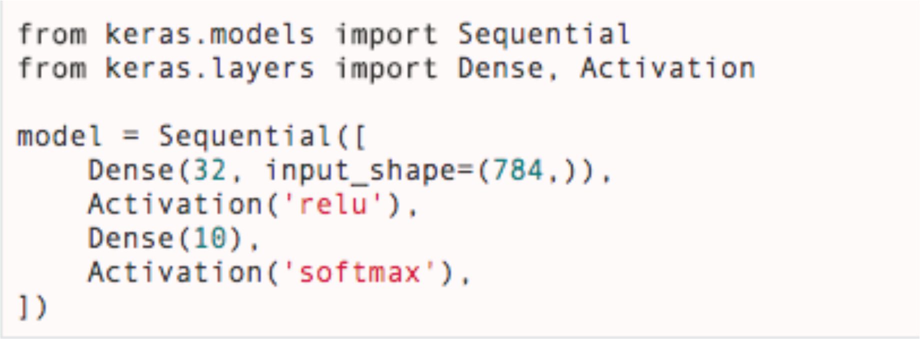
- Easily build big computational graphs
  - Not the case in HW1
- Easily compute gradients in computational graphs
- GPU support (cuDNN, cuBLA...etc)
  - Not required in HW1

Keras

- A high-level deep learning framework
- Built on other deep-learning frameworks
    - Theano
    - Tensorflow
    - CNTK
- Easy and Fun!

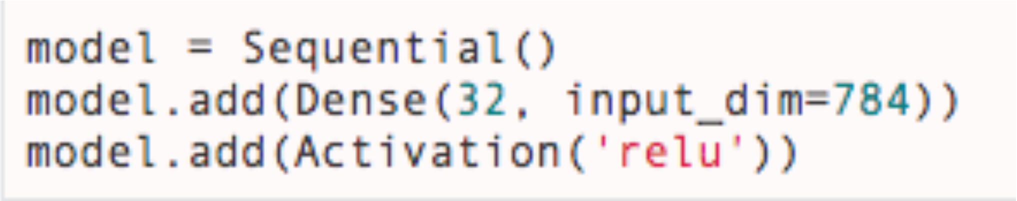
Keras: A High-level Wrapper

- Pass on a layer of instances in the constructor



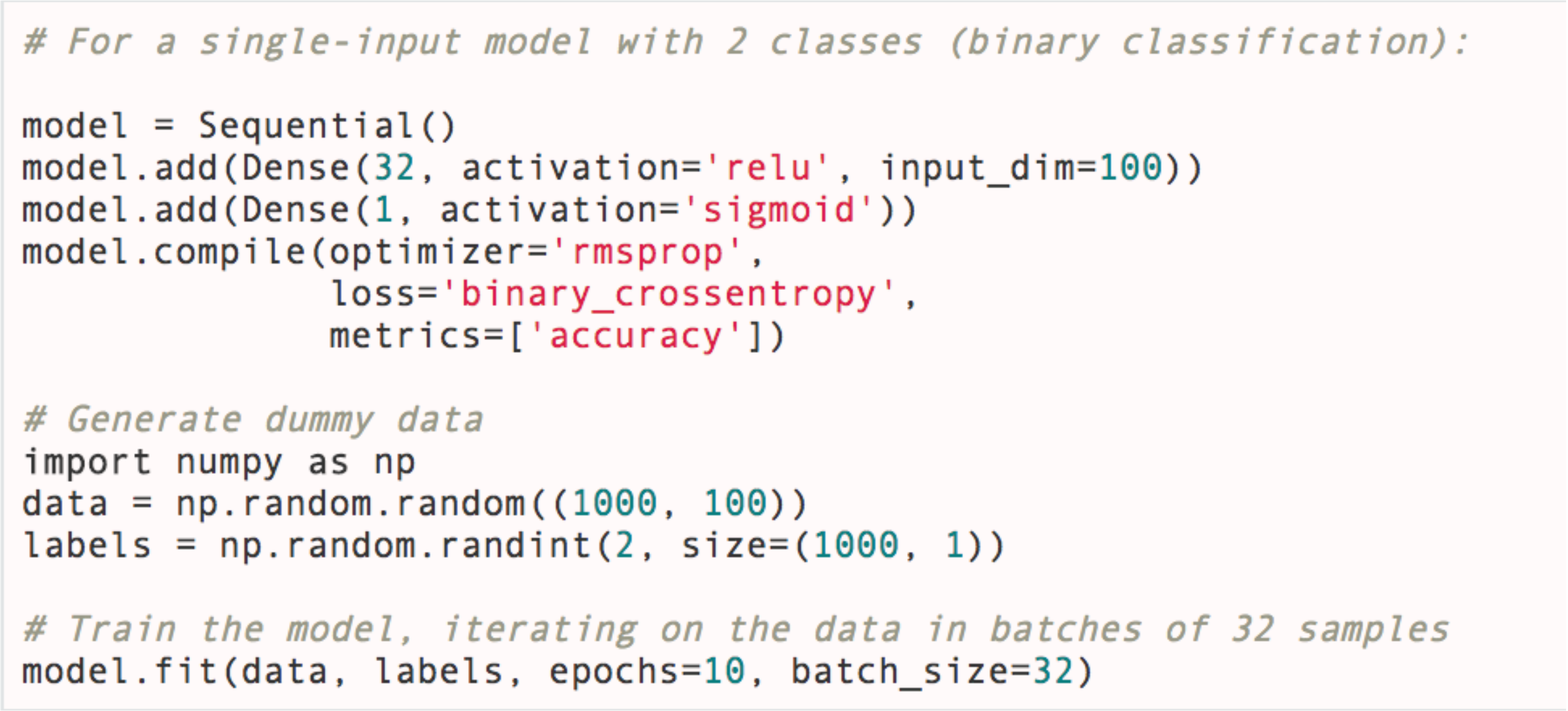
```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

- Or: simply add layers. Make sure the dimensions match.



```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Keras: Compile and train!

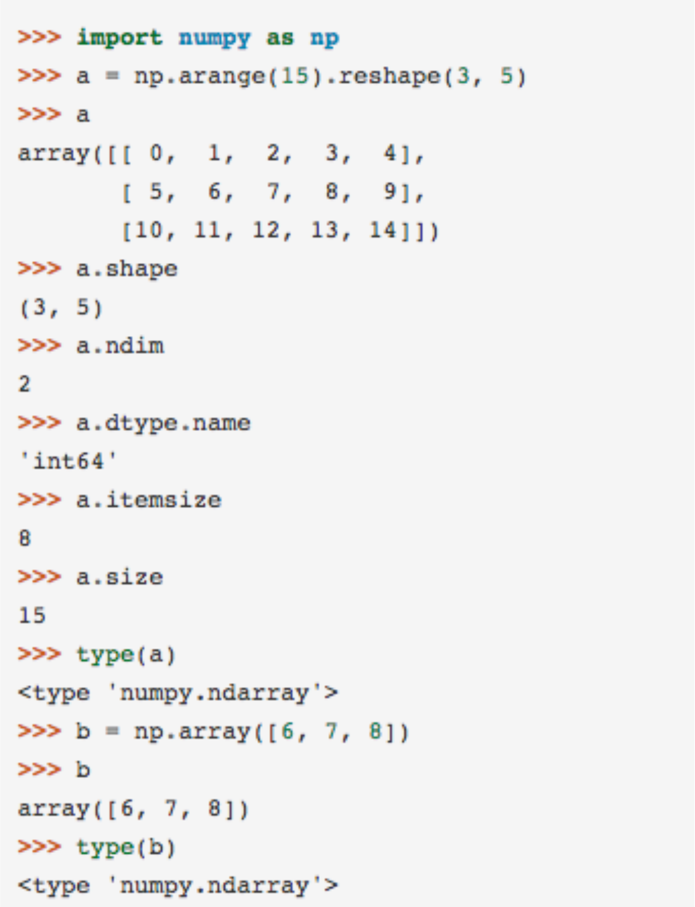


```python
# For a single-input model with 2 classes (binary classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

Epoch: 1 epoch means going through
all the training dataset once

Numpy

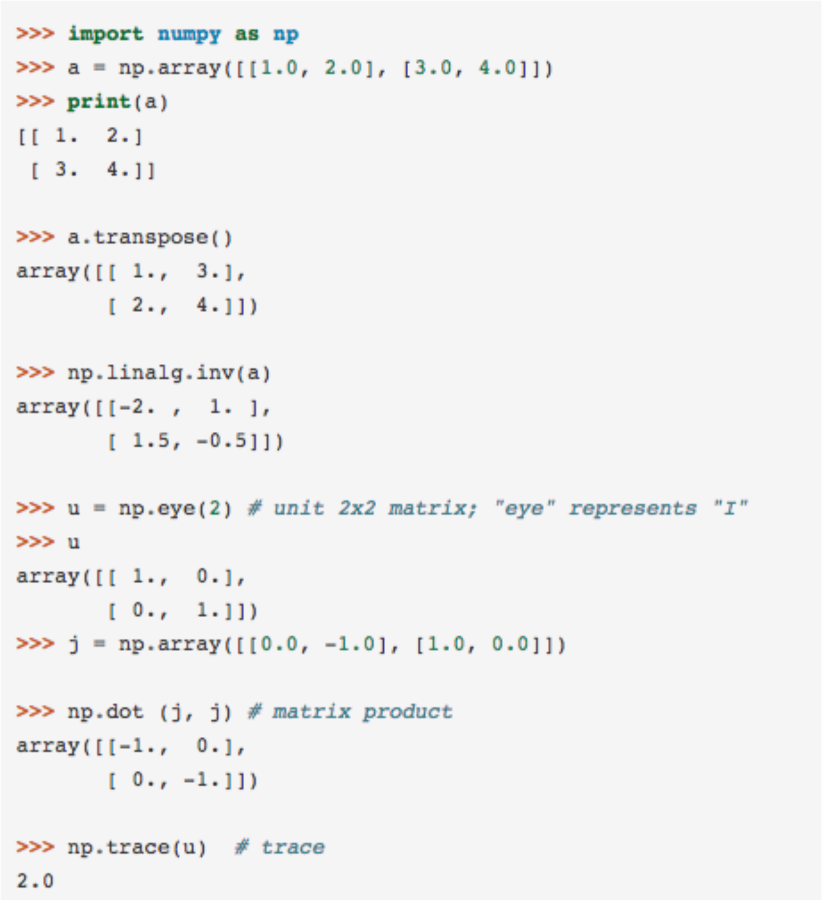
- The **fundamental** package in Python for:
  - Scientific Computing
  - Data Science
- Think in terms of vectors/Matrices
  - Refrain from using for loops!
  - Similar to Matlab



```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

Numpy

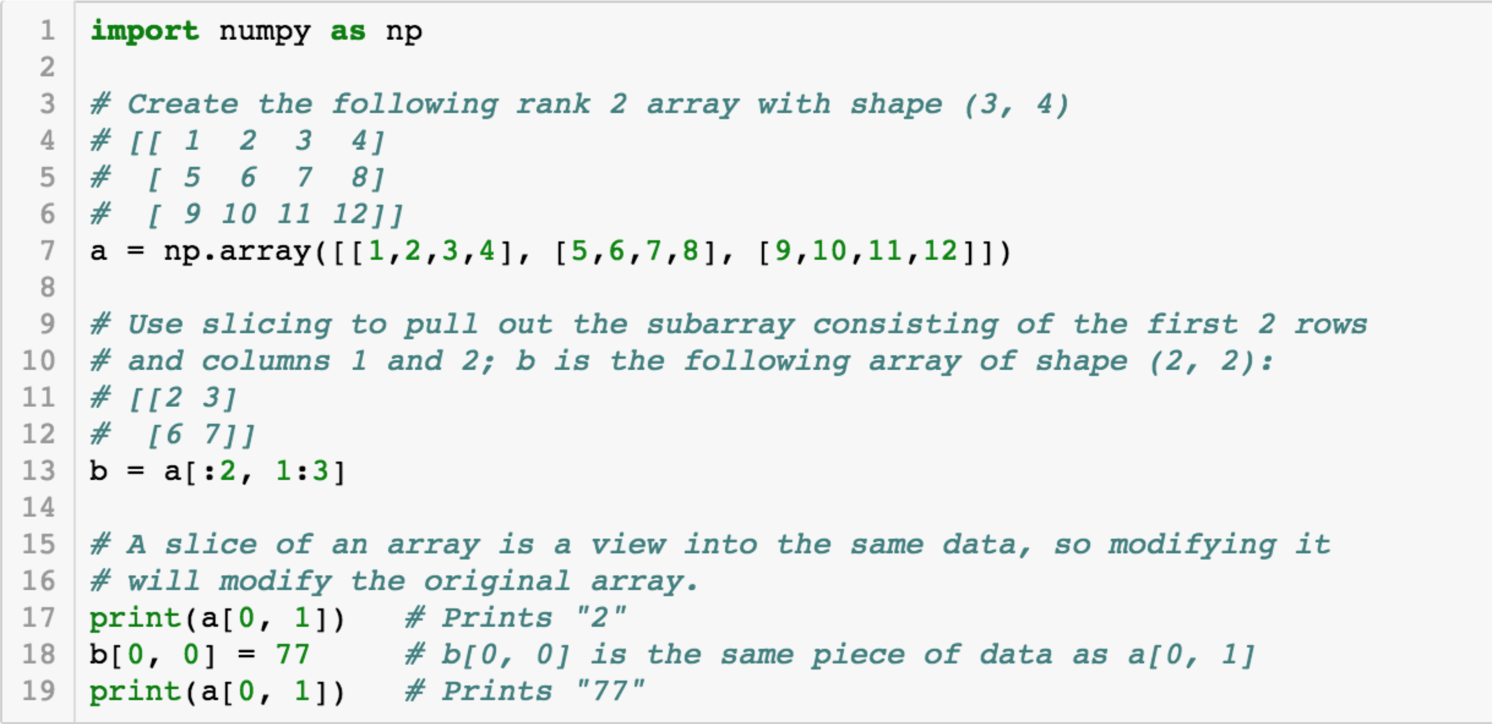
- Basic vector operations
  - Sum, mean, argmax….
- Linear Algebra operations



```
>>> import numpy as np
>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])
>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]

>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]])

>>> np.linalg.inv(a)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

>>> u = np.eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> np.dot (j, j) # matrix product
array([[-1.,  0.],
       [ 0., -1.]])

>>> np.trace(u)  # trace
2.0
```

Numpy

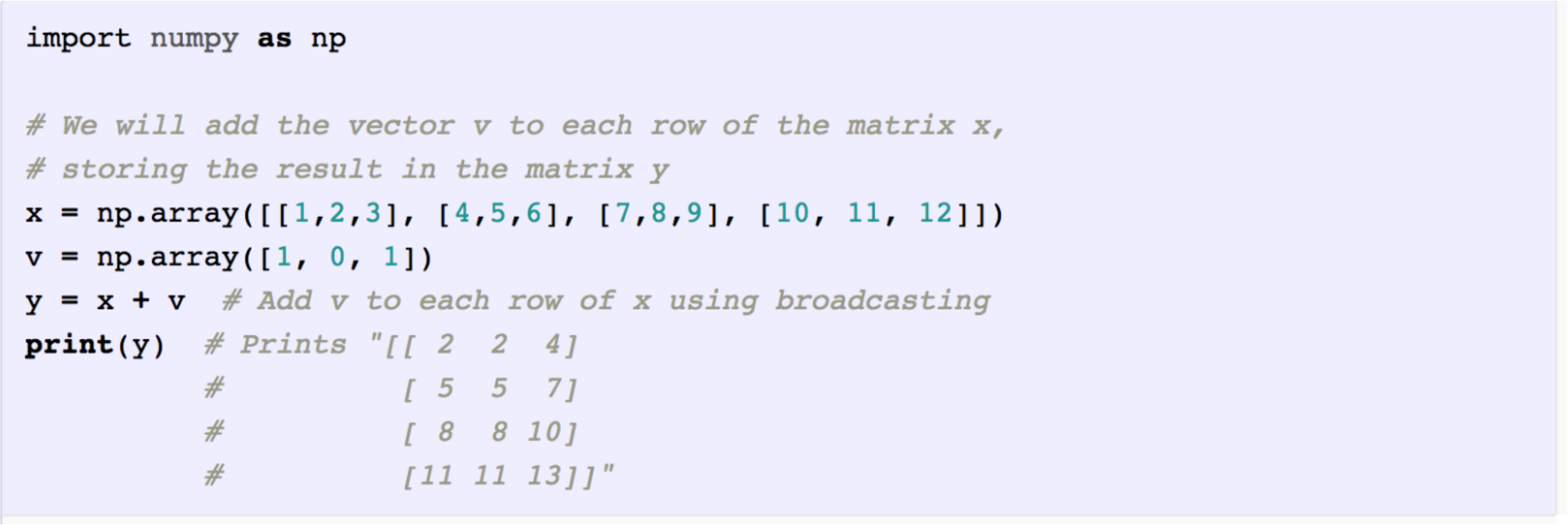
- Indexing, Slicing, Iterating



```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

Numpy

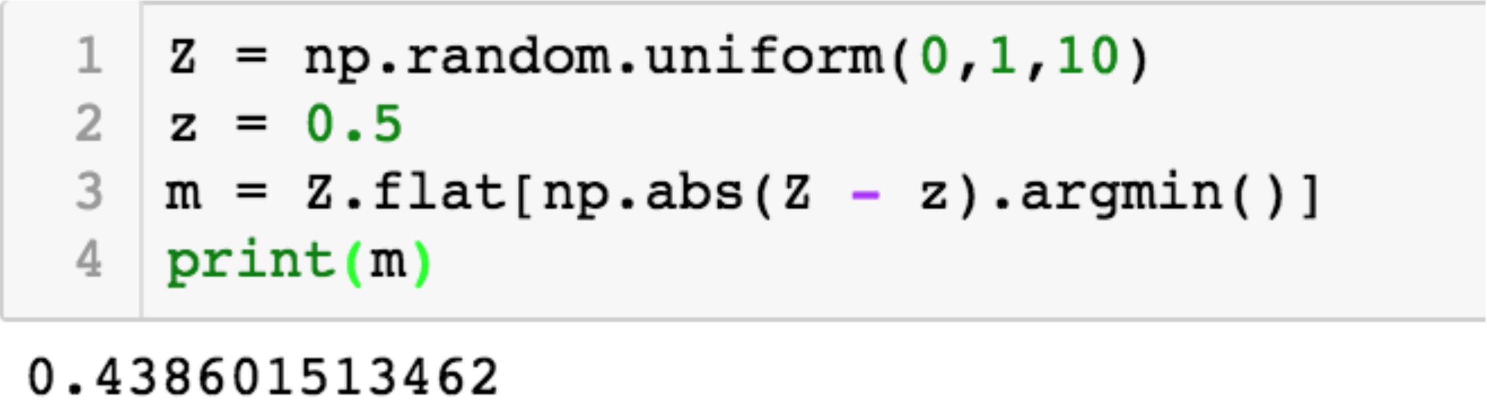
- Broadcasting



```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

Numpy Example

- Find the nearest value from a given value in an array

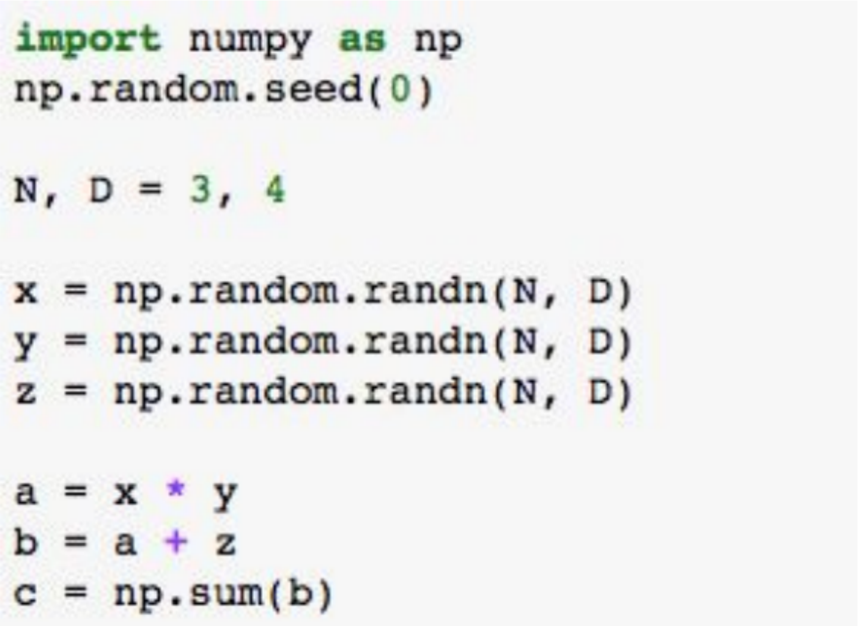


```
Z = np.random.uniform(0,1,10)
z = 0.5
m = Z.flat[np.abs(Z - z).argmin()]
print(m)
```

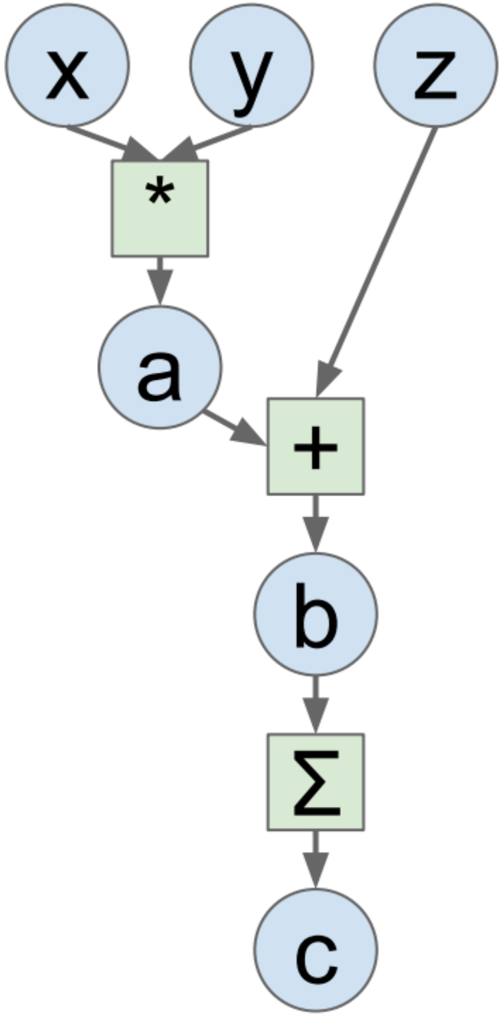
```
0.438601513462
```

Computational Graphs

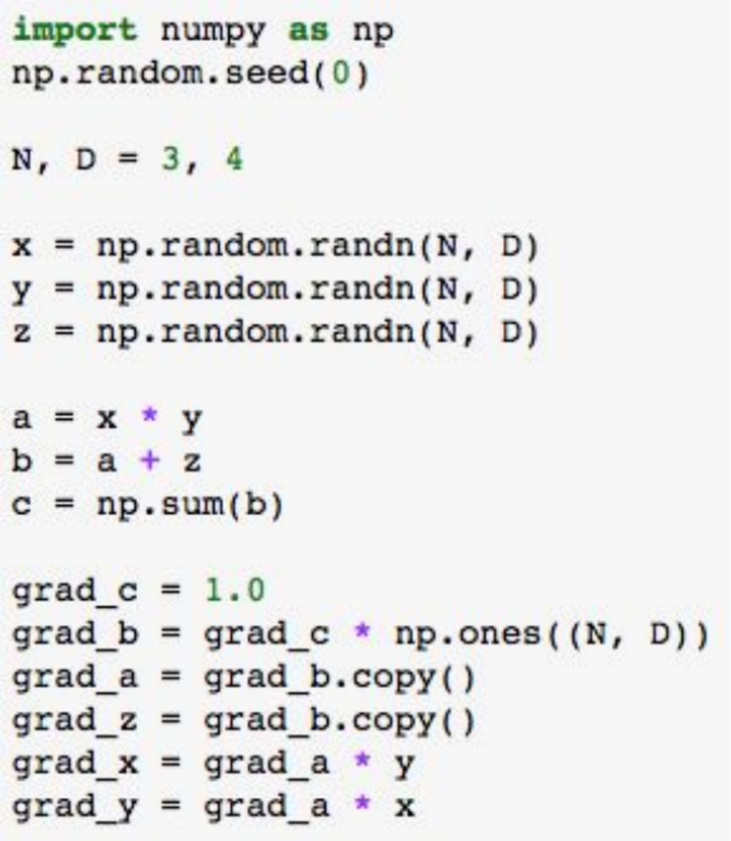
- f(x,y,z) = sum(x*y + z)
- x,y,z can be scalars, vectors, matrices, tensors.



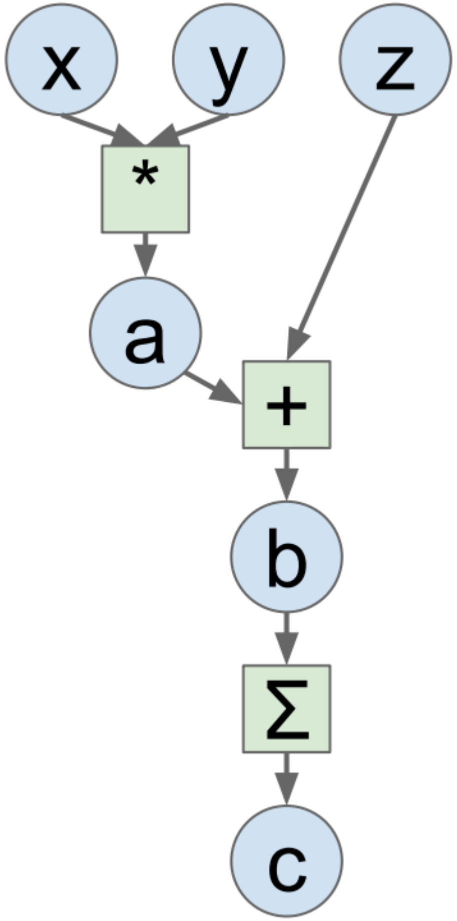
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```



Computational Graphs- Numpy

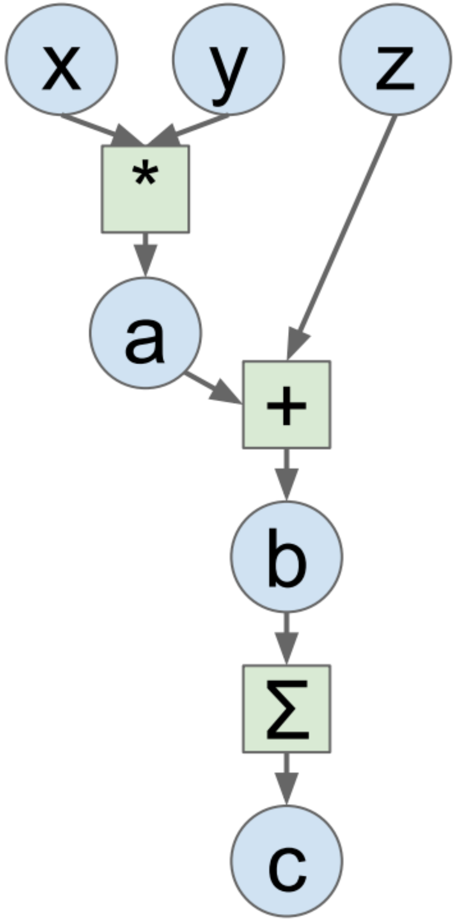


```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

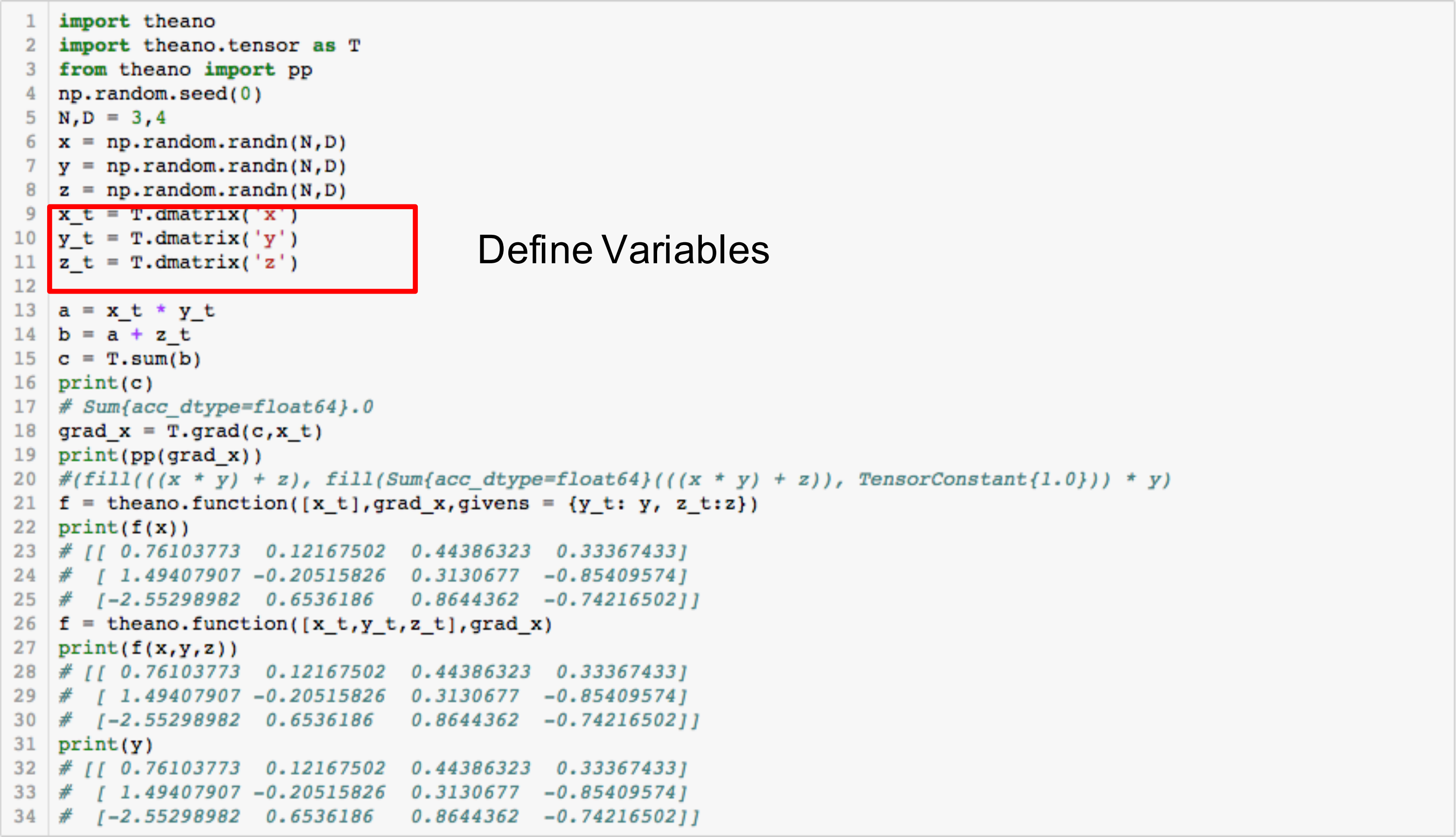


Computational Graphs - Theano

- Define a variable(input)
- Define new variables using Theano operations
- Define Output
- Define functions over these variables

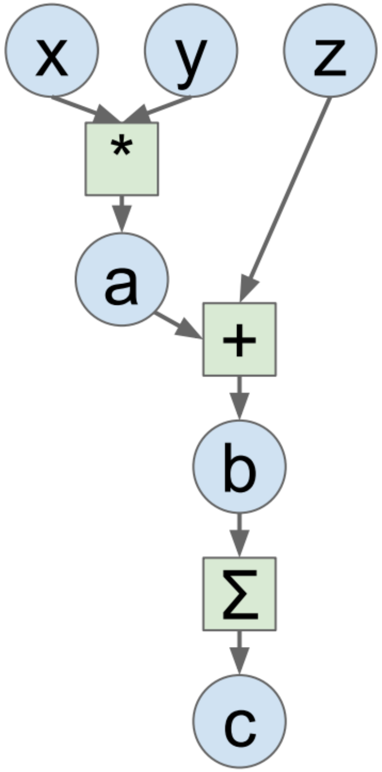


Computational Graph-Theano

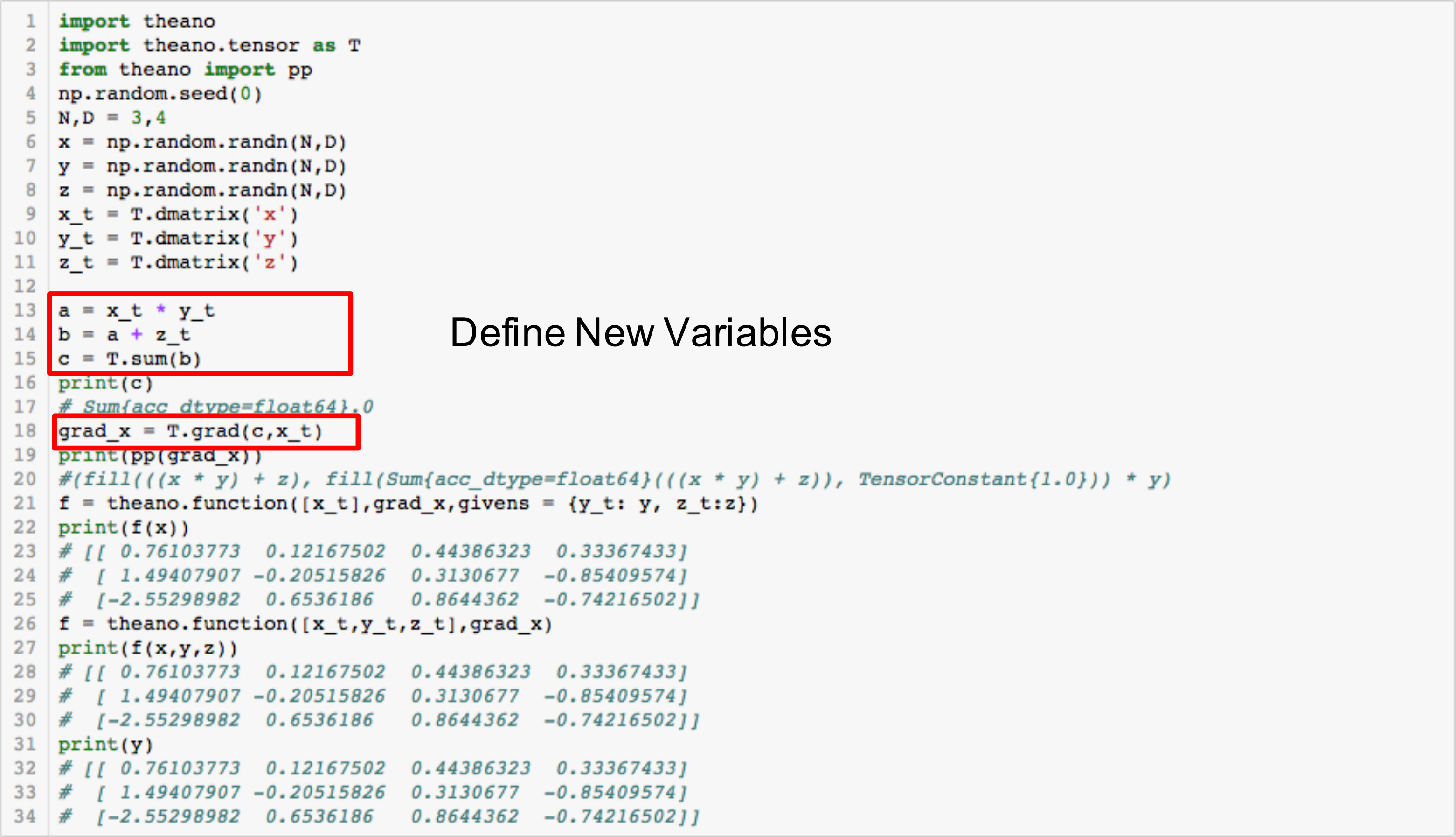


```
import theano
import theano.tensor as T
from theano import pp
np.random.seed(0)
N,D = 3,4
x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)
x_t = T.dmatrix('x')
y_t = T.dmatrix('y')
z_t = T.dmatrix('z')

a = x_t * y_t
b = a + z_t
c = T.sum(b)
print(c)
# Sum{acc_dtype=float64}.0
grad_x = T.grad(c,x_t)
print(pp(grad_x))
#(fill(((x * y) + z), fill(Sum{acc_dtype=float64}(((x * y) + z)), TensorConstant{1.0})) * y)
f = theano.function([x_t],grad_x,givens = {y_t: y, z_t:z})
print(f(x))
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
f = theano.function([x_t,y_t,z_t],grad_x)
print(f(x,y,z))
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
print(y)
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
```

Define Variables

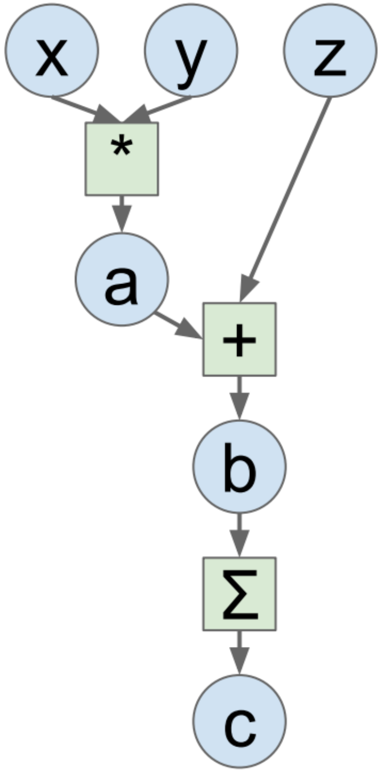


Computational Graph-Theano

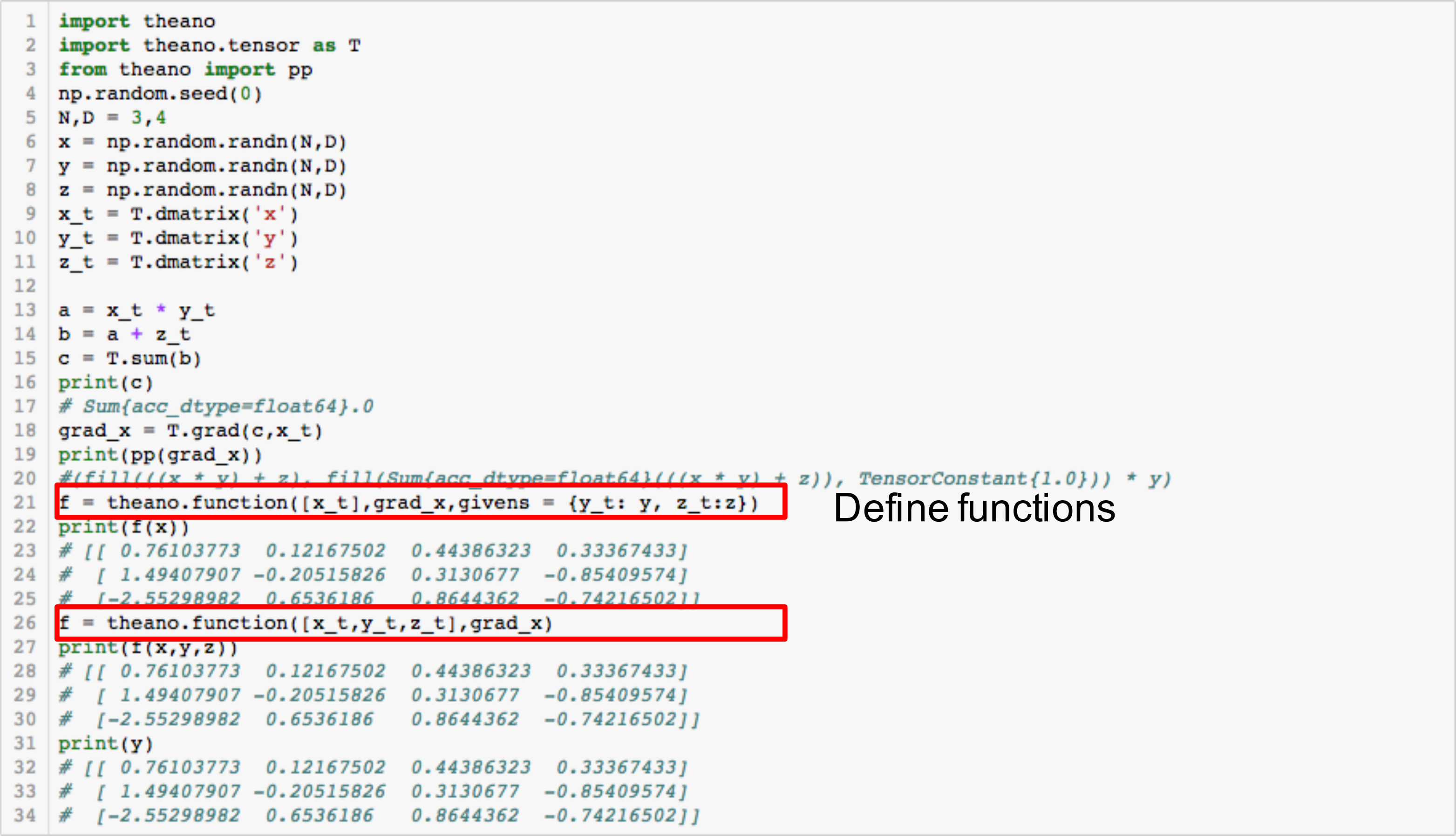


```
import theano
import theano.tensor as T
from theano import pp
np.random.seed(0)
N,D = 3,4
x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)
x_t = T.dmatrix('x')
y_t = T.dmatrix('y')
z_t = T.dmatrix('z')

a = x_t * y_t
b = a + z_t
c = T.sum(b)
print(c)
# Sum{acc_dtype=float64}.0
grad_x = T.grad(c,x_t)
print(pp(grad_x))
#(fill(((x * y) + z), fill(Sum{acc_dtype=float64}(((x * y) + z)), TensorConstant{1.0})) * y)
f = theano.function([x_t],grad_x,givens = {y_t: y, z_t:z})
print(f(x))
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
f = theano.function([x_t,y_t,z_t],grad_x)
print(f(x,y,z))
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
print(y)
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
```

Define New Variables

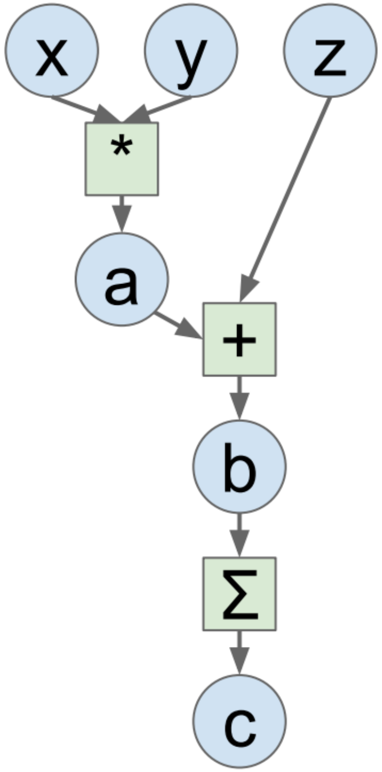


Computational Graph-Theano



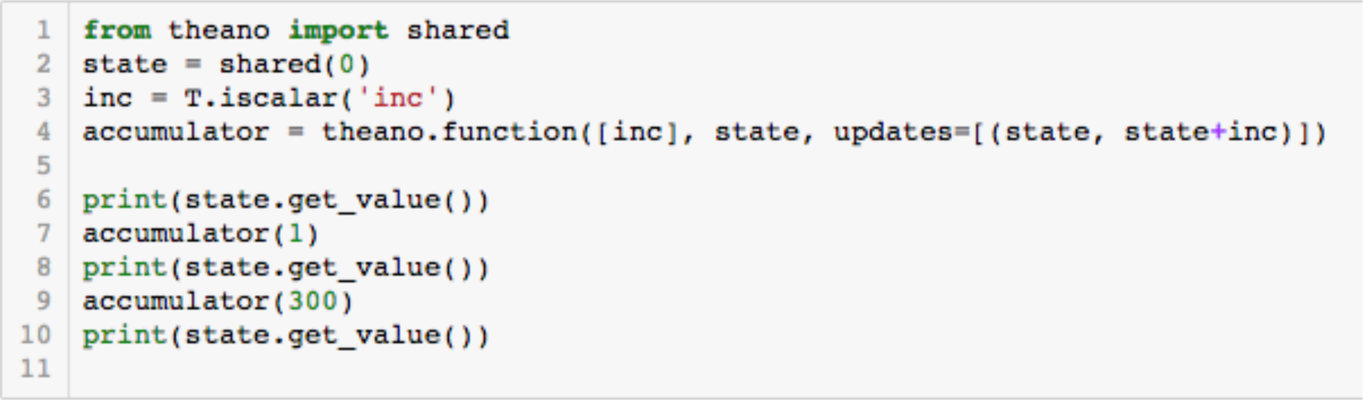
```
import theano
import theano.tensor as T
from theano import pp
np.random.seed(0)
N,D = 3,4
x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)
x_t = T.dmatrix('x')
y_t = T.dmatrix('y')
z_t = T.dmatrix('z')

a = x_t * y_t
b = a + z_t
c = T.sum(b)
print(c)
# Sum{acc_dtype=float64}.0
grad_x = T.grad(c,x_t)
print(pp(grad_x))
#(fill(((x * y) + z), fill(Sum{acc_dtype=float64}(((x * y) + z)), TensorConstant{1.0})) * y)
f = theano.function([x_t],grad_x,givens = {y_t: y, z_t:z})
print(f(x))
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
f = theano.function([x_t,y_t,z_t],grad_x)
print(f(x,y,z))
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
print(y)
# [[ 0.76103773  0.12167502  0.44386323  0.33367433]
#  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
#  [-2.55298982  0.6536186   0.8644362  -0.74216502]]
```

Define functions



Theano: Shared Variable and Update

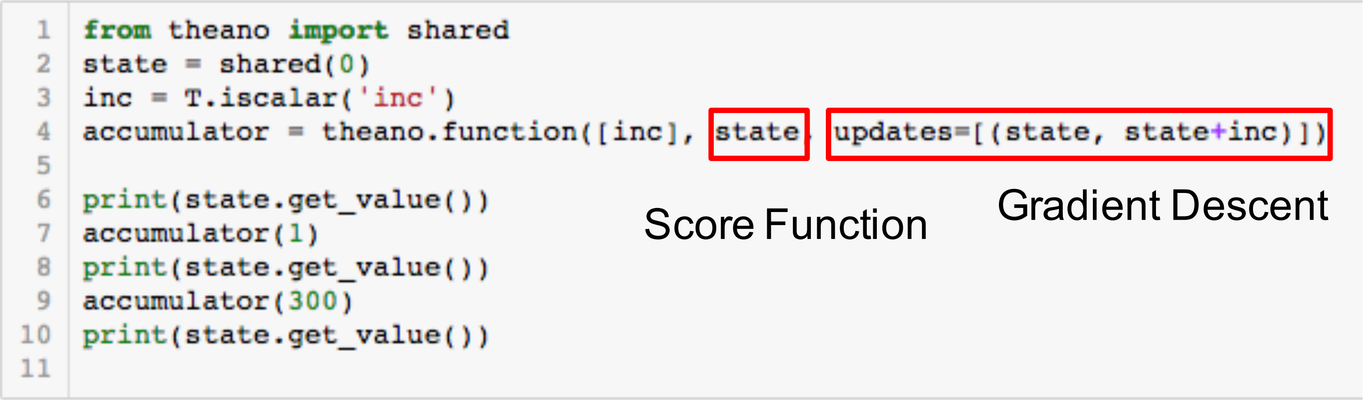
- Hybrid symbolic and non-symbolic variables
- Shared between multiple functions



```python
from theano import shared
state = shared(0)
inc = T.iscalar('inc')
accumulator = theano.function([inc], state, updates=[(state, state+inc)])

print(state.get_value())
accumulator(1)
print(state.get_value())
accumulator(300)
print(state.get_value())

```

Theano: Shared Variable and Update

- Hybrid symbolic and non-symbolic variables
- Shared between multiple functions

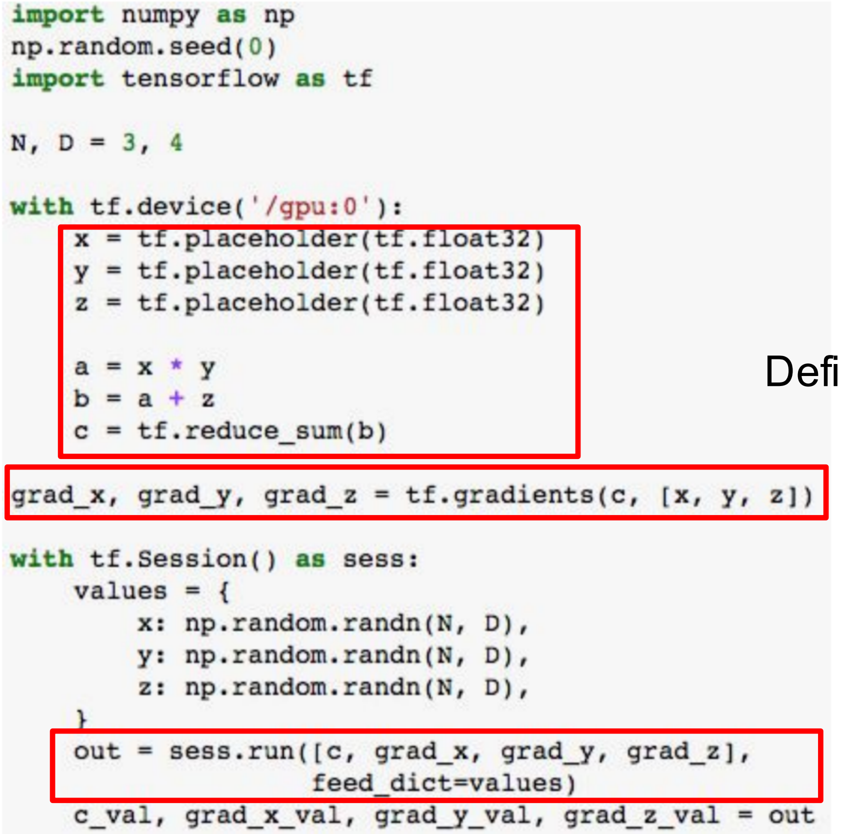


```
from theano import shared
state = shared(0)
inc = T.iscalar('inc')
accumulator = theano.function([inc], state, updates=[(state, state+inc)])

print(state.get_value())
accumulator(1)
print(state.get_value())
accumulator(300)
print(state.get_value())

```

Score Function

Gradient Descent

Tensorflow

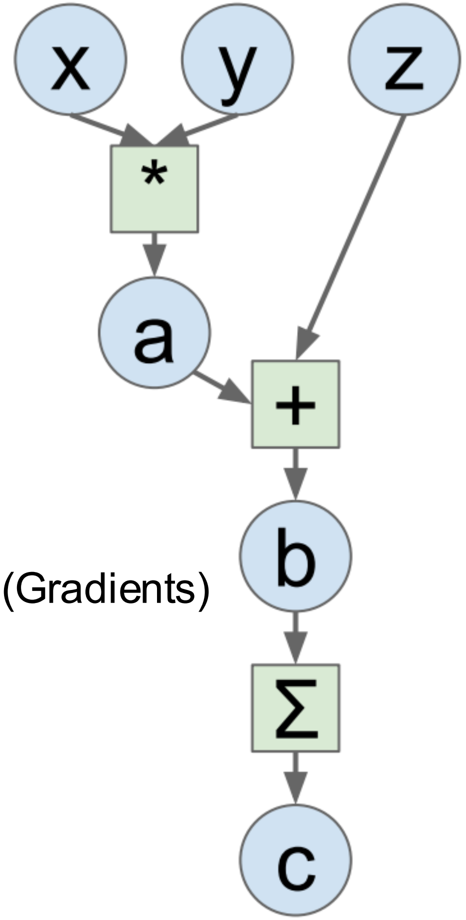


```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

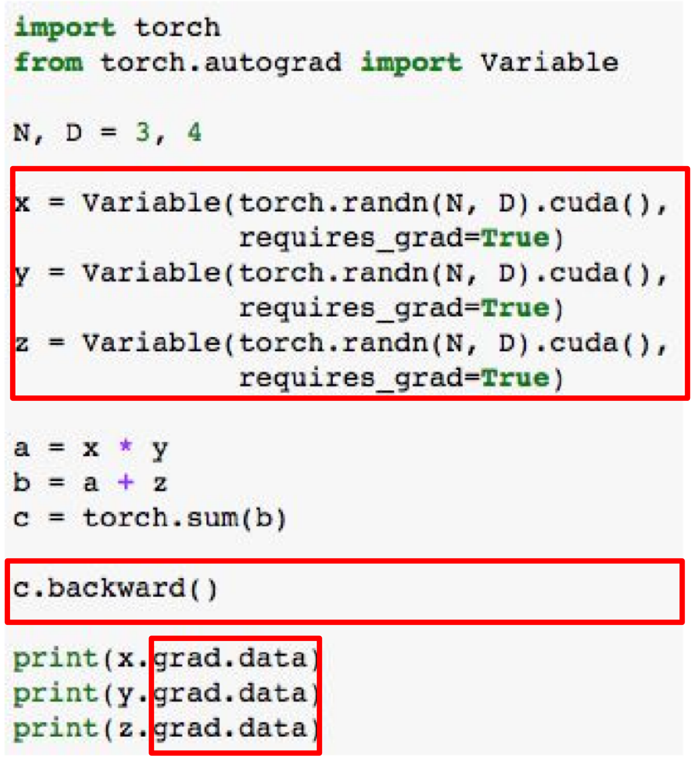
Define Variables

Define New Variables(Gradients)

Define Functions



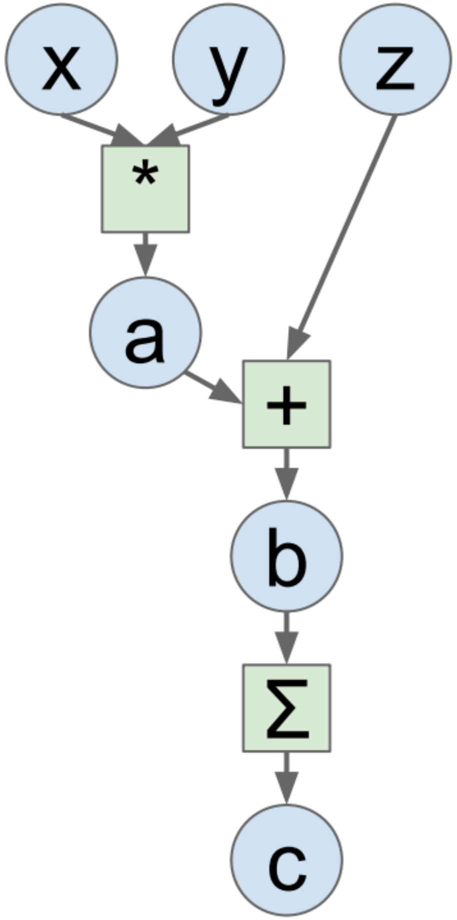
Pytorch



```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Define Variables

Define Functions



Comparison

Framework	Pros/Cons
Theano	Development completed, No development in progress, Static
Tensorflow	Actively developed, big community, Static
PyTorch	Better for Research, relatively new, Dynamic
Keras	High-level, Easy, Not flexible

Resources

- Great Documentation on all of the DL software
- Deeplearning.ai
- State-of-art result for machine learning problems
  - https://github.com/RedditSota/state-of-the-art-result-for-machine-learning-problems