Brief Introduction to Deep Learning and TensorFlow

Deep Learning in Earth Science
Lecture 1
By Xiao Zhuowei

For researchers interested in studying Earth science with deep learning.

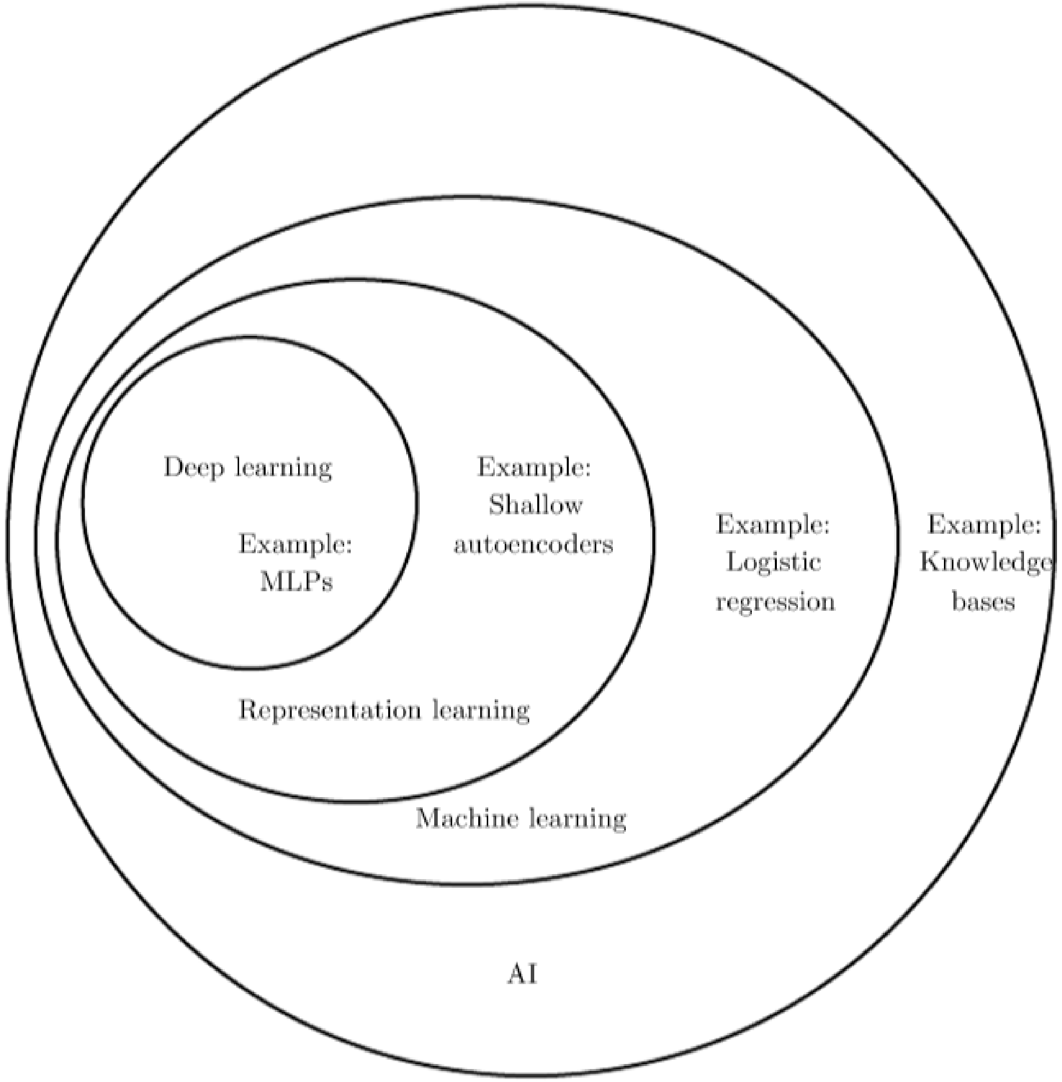
All resources in lectures are available at https://github.com/MrXiaoXiao/DLiES

OUTLINES

1. Brief Introduction to Deep Learning
2. TensorFlow Basics
3. Classifying Stability of Mantle with Neural Networks: An Example
4. Discussions

Brief Introduction to Deep Learning

Deep Learning is about automatically obtaining ***<u>representation of input</u>*** and ***<u>mapping (from representation) to output</u>*** with deep neural network architectures.



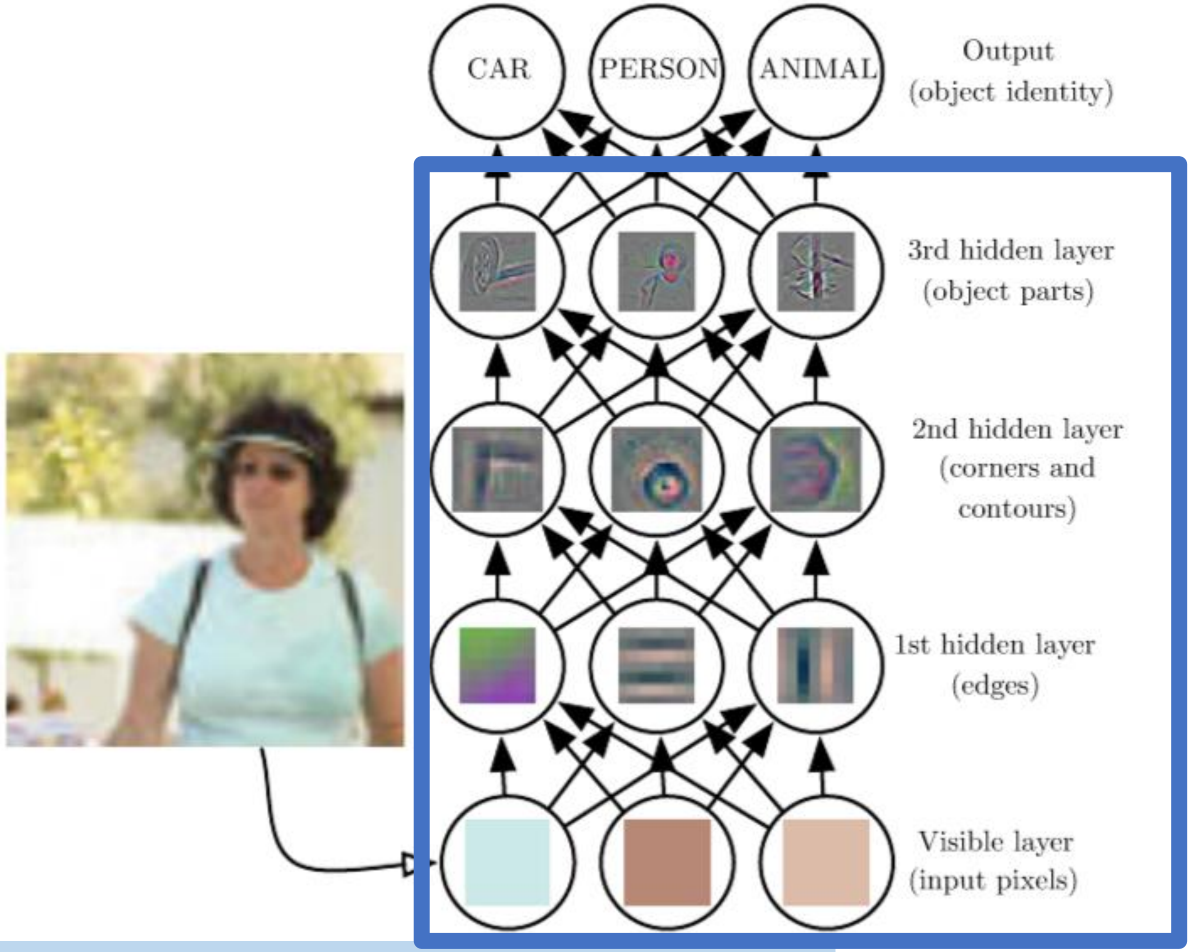
(Deep Learning, MIT Press, 2016)

Obtain the ***representation*** of input.

What We See

What Computers See

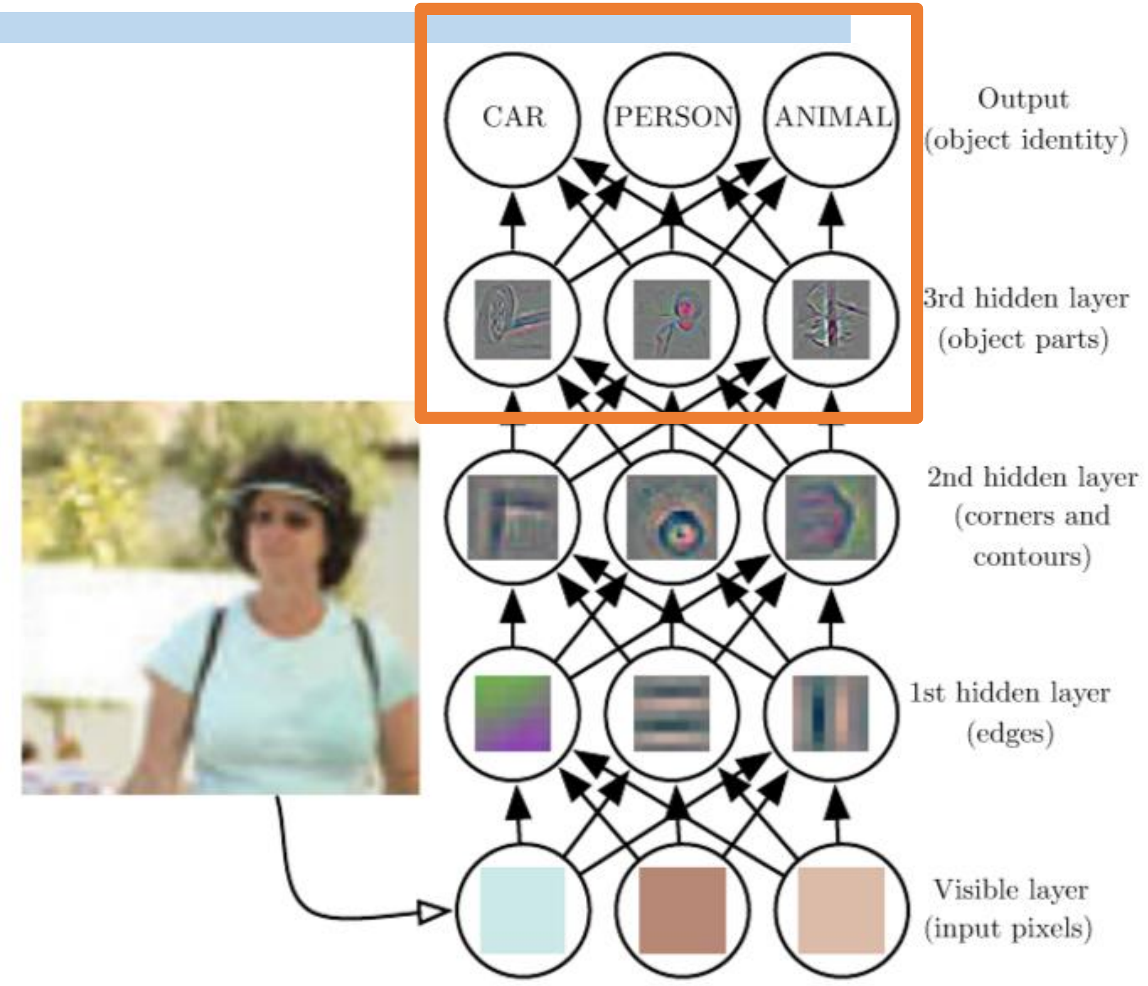
(https://adeshpande3.github.io)



(Deep Learning, MIT Press, 2016)

Brief Introduction to Deep Learning

Obtain ***mapping*** from representation to output.

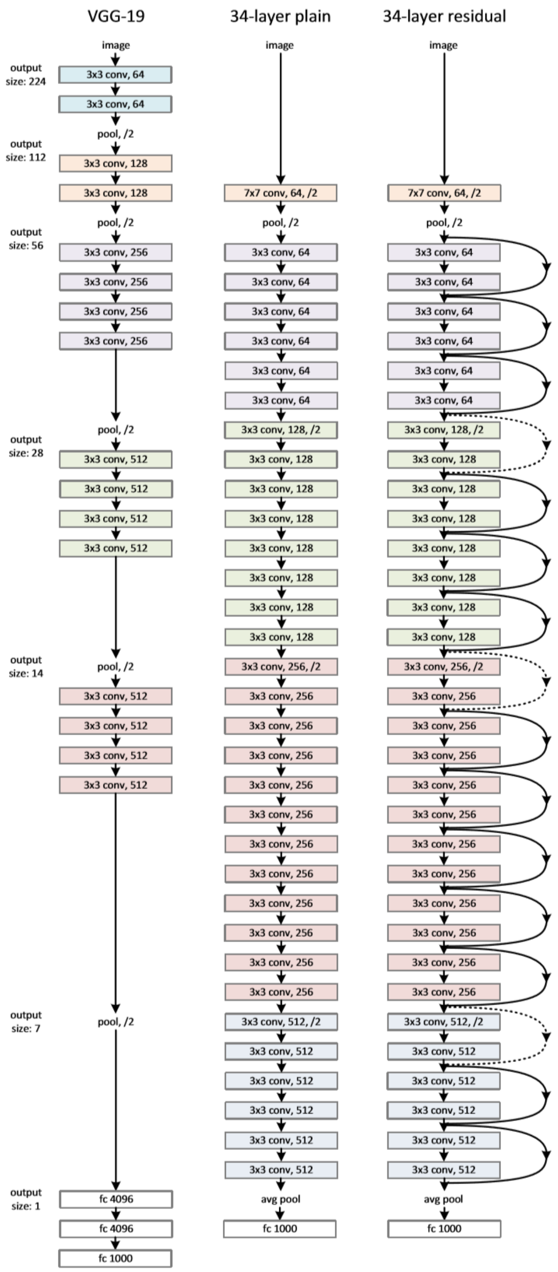


(Deep Learning, MIT Press, 2016)

Brief Introduction to Deep Learning

Complicated representations are built out of simpler ones.

The graph of deep learning architecture is deep, with many layers.



Example network architectures (He et al., 2015)

Brief Introduction to Deep Learning

Considering deep learning as algorithm for non-linear function approximation

$$Ideal\ Output = Ideal\ Function(Input + Noise)$$

$$Approximation\ of\ Ideal\ Output = DL\ Model(Input + Noise)$$

Brief Introduction to Deep Learning

What can deep learning do in Earth science?

Classification

Denoising

Forward Modeling

Inversion

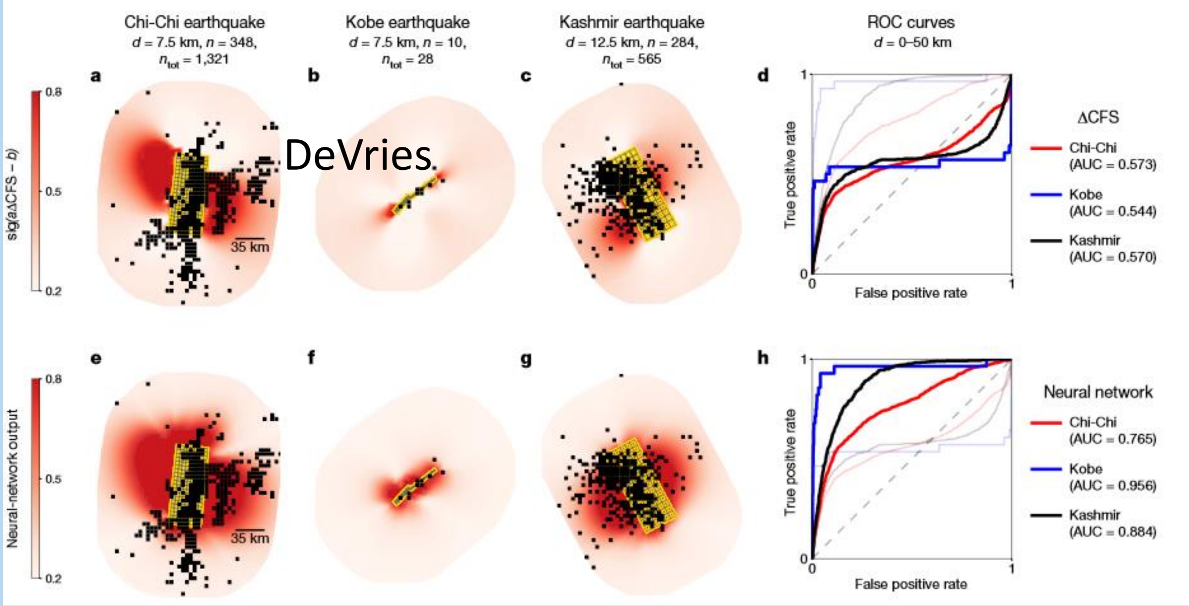
…

Brief Introduction to Deep Learning

What can deep learning do in Earth science?

Classification

Predicting aftershocks following large earthquakes



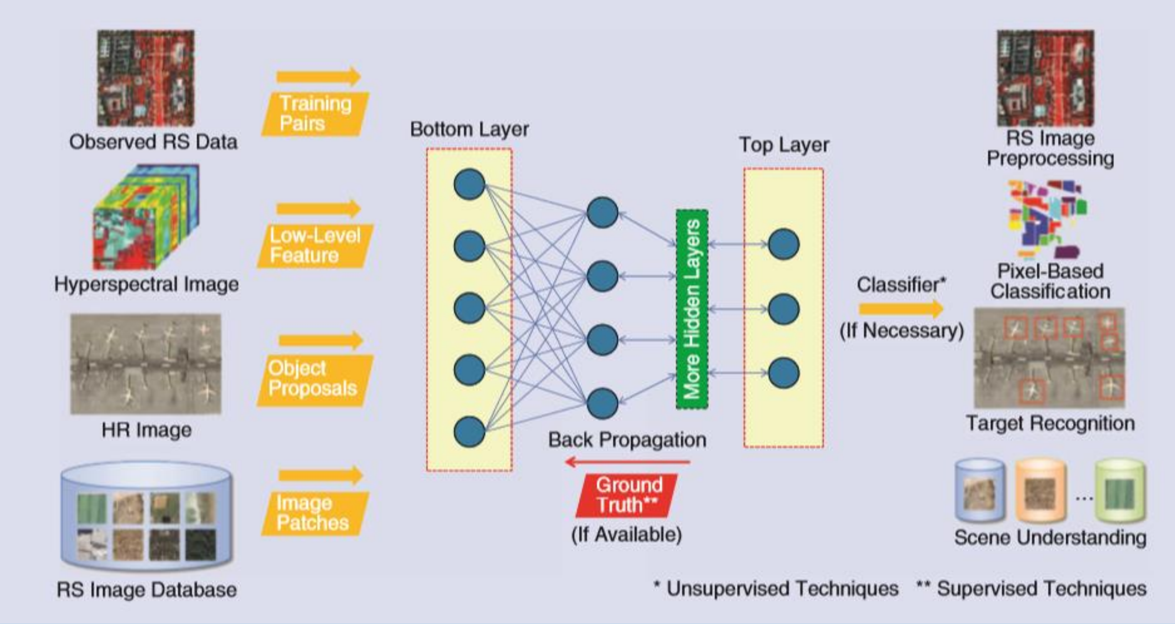
(DeVries et al., 2018)

Brief Introduction to Deep Learning

What can deep learning do in Earth science?

Classification

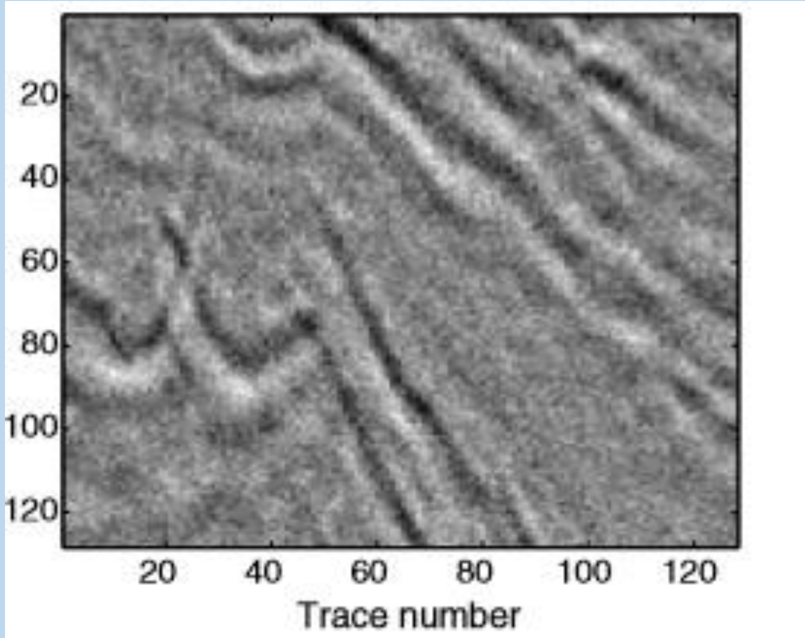
Processing remote sensing data



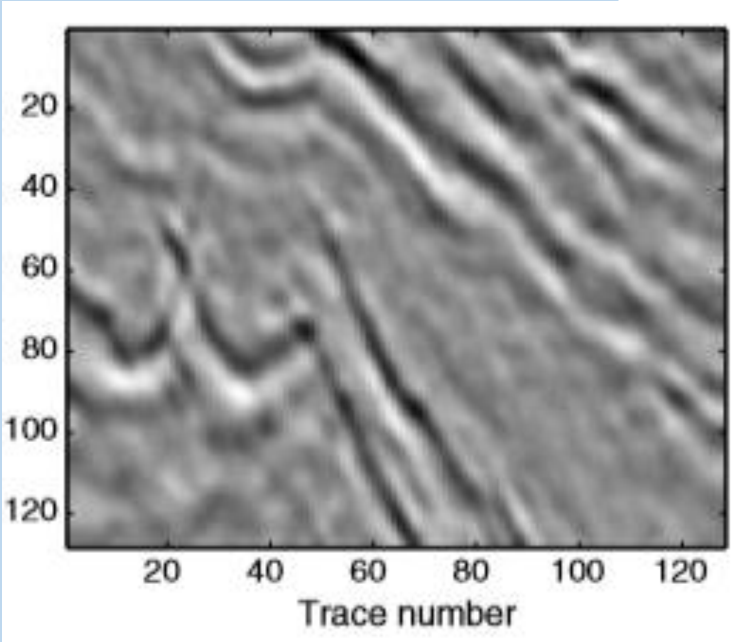
(Zhang et al., 2016)

What can deep learning do in Earth science?

Denoising



Noisy input



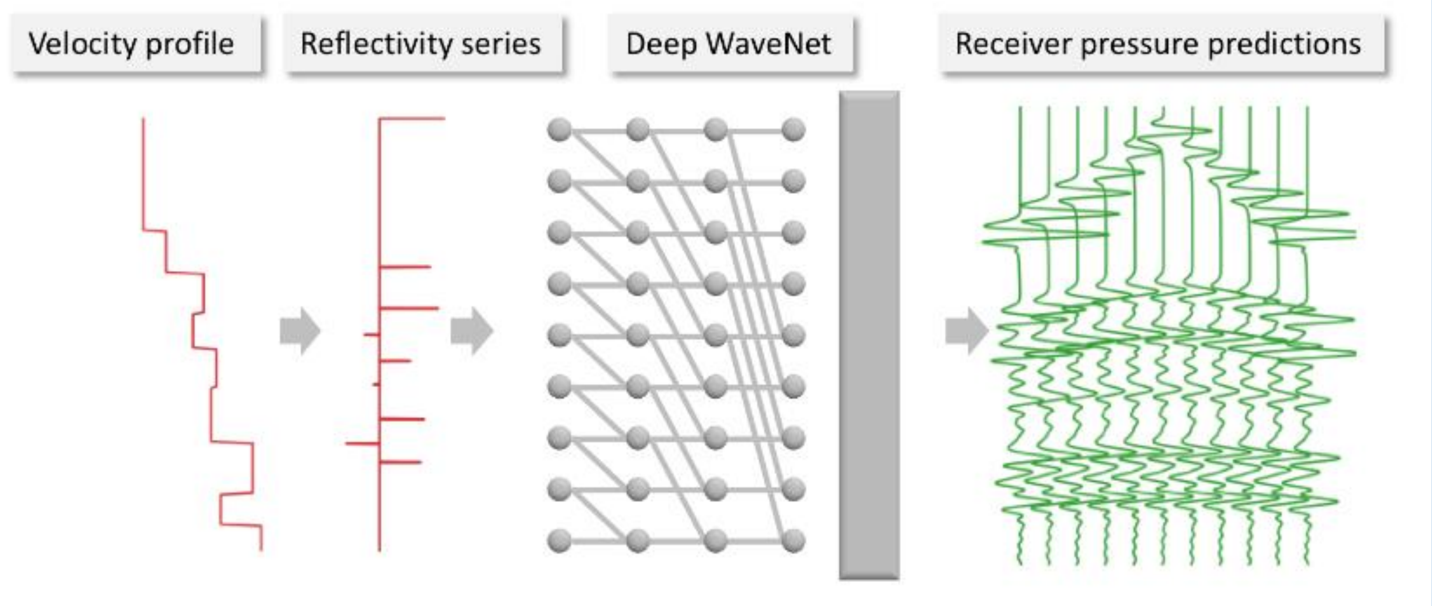
DL output

(Beckouche and Ma, 2014)

What can deep learning do in Earth science?

Forward Modeling

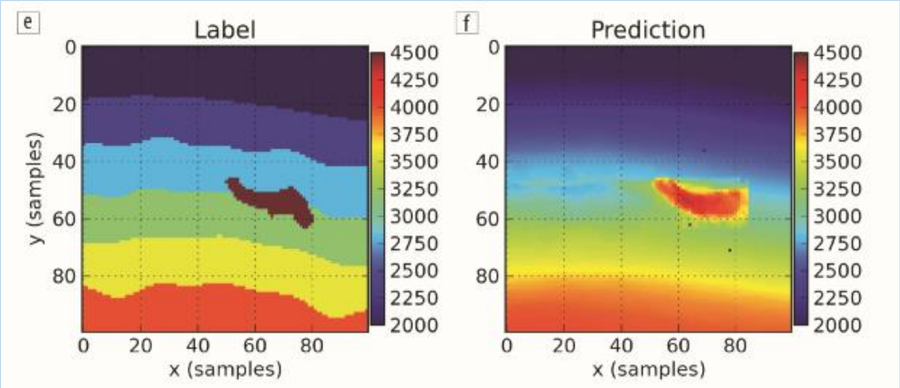
Fast approximate simulation of seismic waves with deep learning



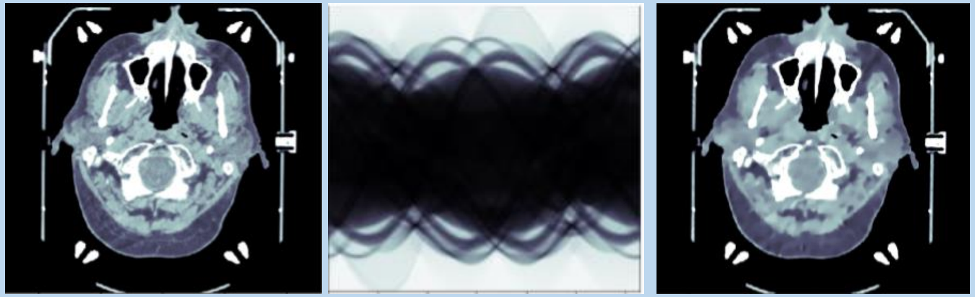
(Moseley et al., 2018)

What can deep learning do in Earth science?

Inversion



(Araya-Polo et al., 2018)



Model Observation Inversion by DL

(Adler and Öktem, 2017)

OUTLINES

1. Brief Introduction to Deep Learning
2. TensorFlow Basics
3. Classifying Stability of Mantle with Neural Networks: An Example
4. Discussions

TensorFlow Basics

TensorFlow™ is an open source software library for high performance numerical computation.

https://www.tensorflow.org/

or

https://tensorflow.google.cn/

Install TensorFlow via Anaconda

Anaconda Distribution is a free, easy-to-install package manager, environment manager and Python distribution with a collection of 1,000+ open source packages with free community support.

Anaconda Download (https://www.anaconda.com/download/)

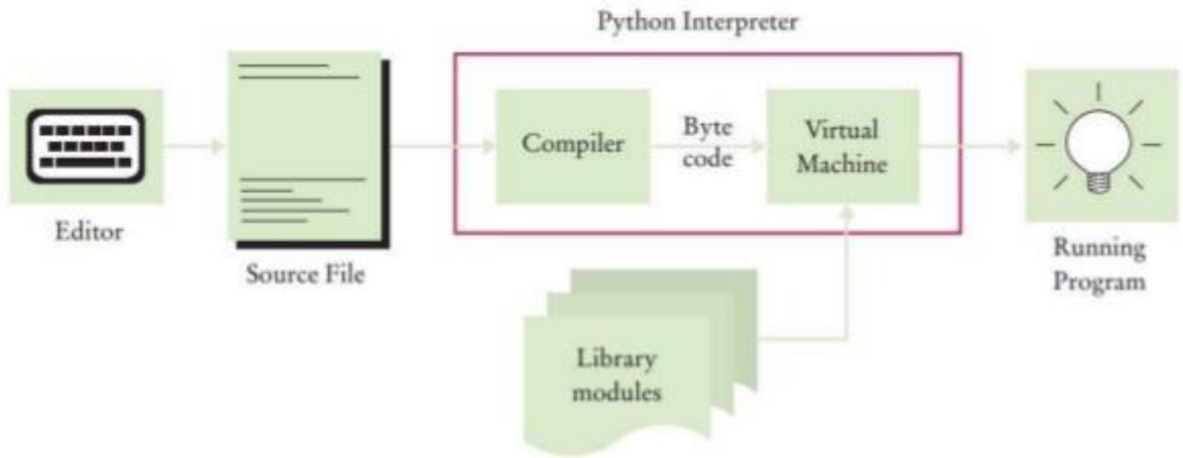
Tensorflow-in-Anaconda
(https://www.anaconda.com/blog/developer-blog/tensorflow-in-anaconda/)

Python is an interpreted high-level programming language for general-purpose programming.

python™

(https://www.python.org/)

How The Python Interpreter Works



(http://opensourceforgeeks.blogspot.com/2015/10/how-python-works.html)

TensorFlow Basics

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

https://jupyter.org/

jupyter

Image Manipulation with skimage

This example builds a simple UI for performing basic image manipulation with scikit-image.

In [21]:
```
from ipywidgets import interact, interactive, fixed
from IPython.display import display
```

In [22]:
```
import skimage
from skimage import data, filter, io
```

In [23]:
```
i = data.coffee()
```

In [24]:
```
io.Image(i)
```

Out[24]:

In [25]:
```
def edit_image(image, sigma=0.1, r=1.0, g=1.0, b=1.0):
    new_image = filter.gaussian_filter(image, sigma=sigma, multichannel=True)
    new_image[:,:,0] = r*new_image[:,:,0]
    new_image[:,:,1] = g*new_image[:,:,1]
    new_image[:,:,2] = b*new_image[:,:,2]
    new_image = io.Image(new_image)
    display(new_image)
    return new_image
```

In [26]:
```
lims = (0.0,1.0,0.01)
w = interactive(edit_image, image=fixed(i), sigma=(0.0,10.0,0.1), r=lims, g=lims, b=lims)
display(w)
```

TensorFlow Basics

TensorFlow Hello World

TensorFlow Hello World

Modified from https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/1_Introduction/helloworld.py

In [1]:
```
import tensorflow as tf
```

In [2]:
```
# Simple hello world using TensorFlow

# Create a Constant op

# The op is added as a node to the default graph.

#

# The value returned by the constructor represents the output

# of the Constant op.

hello = tf.constant('Hello, TensorFlow!')
```

In [3]:
```
print(hello)
```

```
Tensor("Const:0", shape=(), dtype=string)
```

In [4]:
```
# Start tf session

sess = tf.Session()
```

In [5]:
```
# Run the op

print(sess.run(hello))
```

```
b'Hello, TensorFlow!'
```

TensorFlow Multiply Matrices

TensorFlow Multiply Matrices Example

Modified from https://github.com/vahidk/EffectiveTensorflow

In [1]:
```
import tensorflow as tf
```

In [2]:
```
x = tf.random_normal([3,3])
y = tf.random_normal([3,3])
z = tf.matmul(x,y)
```

In [3]:
```
print('{}\n{}\n{}'.format(x,y,z))
```

```
Tensor("random_normal:0", shape=(3, 3), dtype=float32)
Tensor("random_normal_1:0", shape=(3, 3), dtype=float32)
Tensor("MatMul:0", shape=(3, 3), dtype=float32)
```

In [4]:
```
sess = tf.Session()
z_val = sess.run(z)
```

In [5]:
```
print(z_val)
```

```
[[-1.8857789   0.02845232  2.23009   ]
 [ 0.20160252  0.49441913  0.37605742]
 [ 3.5984905   1.7590961  -0.84973013]]
```

Approximate Quadratic Function With TensorFlow

```
In [1]: import numpy as np
        import tensorflow as tf
        #Remeber to install matplotlib in your environment
        import matplotlib.pyplot as plt
```

```
In [2]: #When using Jupyter notebook make sure to call tf.reset_default_graph()
        # at the beginning to clear the symbolic graph before defining new nodes.
        tf.reset_default_graph()
```

```
In [3]: # Placeholders are used to feed values from python to TensorFlow ops. We define
        # two placeholders, one for input feature x, and one for output y.
        x = tf.placeholder(tf.float32)
        y = tf.placeholder(tf.float32)
```

```
In [4]: # Assuming we know that the desired function is a polynomial of 2nd degree, we
        # allocate a vector of size 3 to hold the coefficients. The variable will be
        # automatically initialized with random noise.
        w = tf.get_variable("w", shape=[3, 1])
```

```
In [5]: # We define yhat to be our estimate of y.
        f = tf.stack([tf.square(x), x, tf.ones_like(x)], 1)
        yhat = tf.squeeze(tf.matmul(f, w), 1)
```

```
In [6]: # The loss is defined to be the l2 distance between our estimate of y and its
        # true value. We also added a shrinkage term, to ensure the resulting weights
        # would be small.
        loss = tf.nn.l2_loss(yhat - y)
```

```
In [7]: # We use the Adam optimizer with learning rate set to 0.1 to minimize the loss.
        train_op = tf.train.AdamOptimizer(0.001).minimize(loss)
```

```
In [8]: def generate_data(size = 100):
            x_val = np.random.uniform(-10.0, 10.0, size=size)
            y_val = 5 * np.square(x_val) + 14.3 * x_val + 8.9
            return x_val, y_val
```

TensorFlow Basics

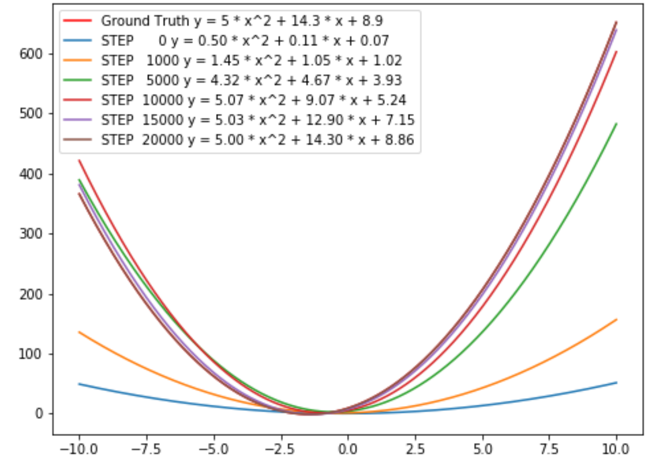
Approximate Quadratic Function With TensorFlow

```
In [9]: inspect_step = [0,1000,5000,10000,15000,20000]
        inspect_list = list()
```

```
In [10]: sess = tf.Session()
         # Since we are using variables we first need to initialize them.
         sess.run(tf.global_variables_initializer())
         for step in range(20001):
             x_val, y_val = generate_data()
             _, loss_val = sess.run([train_op, loss], {x: x_val, y: y_val})
             if step in inspect_step:
                 print('STEP {:5d}: loss_val {:}'.format(step,loss_val))
                 inspect_data = dict()
                 inspect_data['w'] = w.eval(sess)
                 inspect_data['step'] = step
                 inspect_list.append(inspect_data)
```

```
STEP     0: loss_val 2060689.375
STEP  1000: loss_val 1881029.25
STEP  5000: loss_val 201219.46875
STEP 10000: loss_val 47986.7421875
STEP 15000: loss_val 4134.412109375
STEP 20000: loss_val 0.029617290943861008
```

```
In [11]: plt.figure(figsize=(8,6))

         x_axis = np.arange(-10.0,10.0,0.0001)
         y_gt = 5.0 * np.square(x_axis) + 14.3 * x_axis + 8.9
         plt.plot(x_axis,y_gt,color='r',label='Ground Truth y = 5 * x^2 + 14.3 * x + 8.9')
         for data in inspect_list:
             yhat = data['w'][0][0] * np.square(x_axis) + data['w'][1][0] * x_axis + data['w'][2][0]
             plt.plot(x_axis,yhat,label='STEP  {:5d} y = {:.2f} * x^2 + {:.2f} * x + {:.2f}'.format(
                 data['step'],data['w'][0][0],data['w'][1][0],data['w'][2][0]))
         #plt.xlim([-10.0,10.0])
         plt.legend()
         plt.show()
```



Recommended TensorFlow tutorials

Effective TensorFlow

https://github.com/vahidk/EffectiveTensorflow

TensorFlow Official Tutorial

https://www.tensorflow.org/tutorials/

Simple and ready-to-use tutorials for TensorFlow

https://github.com/astorfi/TensorFlow-World

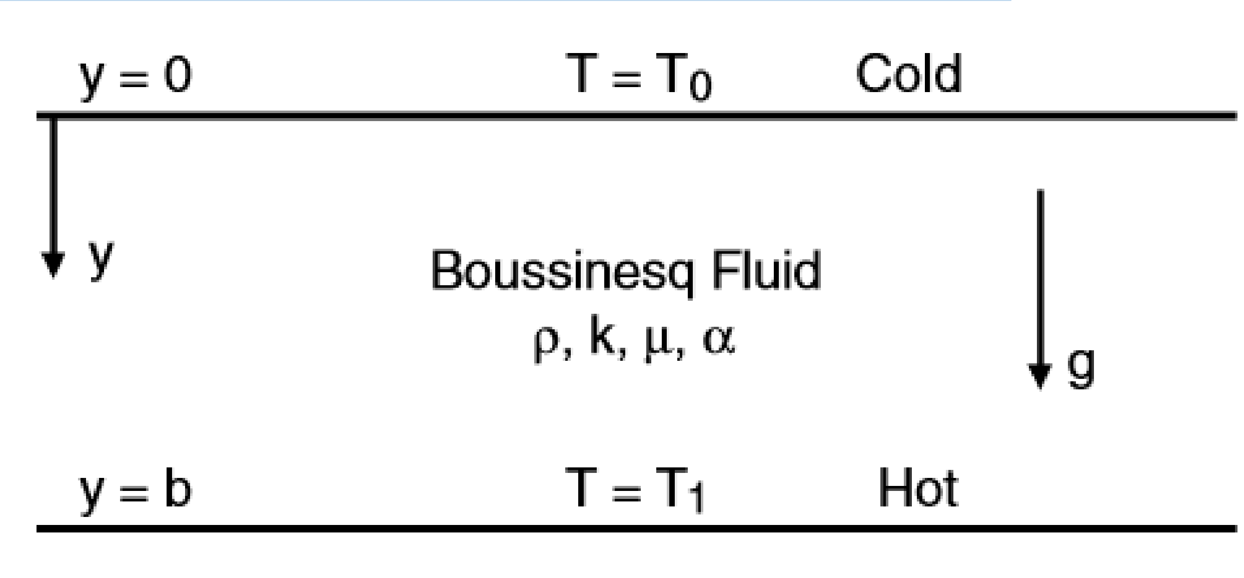
OUTLINES

1. Brief Introduction to Deep Learning
2. TensorFlow Basics
3. Classifying Stability of Mantle with Neural Networks: An Example
4. Discussions

Plane Layer Heated from Below

Factors determine stability of Mantle

(a) Gravitational acceleration
(b) Volume expansion coefficient
(c) Kinematic viscosity coefficient
(d) Thermal diffusivity
(e) Depth
(f) Thickness
(g) λ
(h) dT: (T0 - T1)



(Schuber et al., 2001)

Classifying Stability of Mantle with Neural Networks: An Example

Prepare Dataset

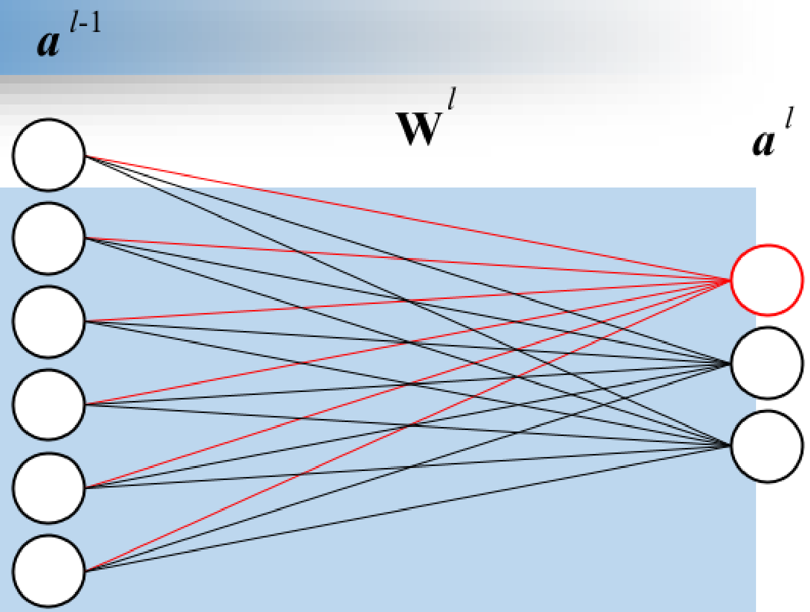
```
In [3]: #Define a function to generate data set
def generate_data_set(instance_num = 10000, split_rate = 0.6):
    #instance[0] Gravitational acceleration
    #instance[1] Volume expansion coefficient
    #instance[2] Kinematic viscosity coefficient
    #instance[3] Thermal diffusivity
    #instance[4] Depth
    #instance[5] b
    #instance[6] λ
    #instance[7] (T0 - T1)/1000

    #label (1,0) for stable (0,1) for unstable

    #instance[8] stability 0 is unstable and 1 is stable
    counter_for_stable = 0
    counter_for_unstable = 0

    data_set = {'input':np.zeros([instance_num, 8]),'label':np.zeros([instance_num,2])}

    #simulate gravitational accelerations
    data_set['input'][:,0] = np.random.uniform(0.8,1.0,size=instance_num)

    #simulate Volume expansion coefficient
    data_set['input'][:,1] = np.random.uniform(1e-4,1e-2,size=instance_num)

    #simulate Kinematic viscosity coefficient
    data_set['input'][:,2] = np.random.uniform(1e-2,1.0,size=instance_num)

    #simulate Thermal diffusivity
    data_set['input'][:,3] = np.random.uniform(0.1,1.0,size=instance_num)

    #simulate Depth 10000km
    data_set['input'][:,4] = np.random.uniform(0,0.35,size=instance_num)

    #simulate b 10000km
    for idx in range(instance_num):
        data_set['input'][idx,5] = np.random.uniform(max([0.25,data_set['input'][idx,4]]),0.35)

    #simulate λ
    data_set['input'][:,6] = np.random.uniform(0.0,0.39,size=instance_num)

    #simulate T0 - T1
    data_set['input'][:,7] = np.random.uniform(0.0,0.5,size=instance_num)

    for idx in range(instance_num):
        #
        g = data_set['input'][idx,0]*10
        a = data_set['input'][idx,1]
        v = data_set['input'][idx,2]*1000
        k = data_set['input'][idx,3]*10000
        d = data_set['input'][idx,4]*10000
        b = data_set['input'][idx,5]*10000
        lam = data_set['input'][idx,6]*10000
        dT = data_set['input'][idx,7]*1000

        Ra = (a*g*dT*(d**3))/(v*k)

        Racr = (np.pi**4*((4+(lam/b)**2)**3))/(4*((lam/b)**4))

        if Ra > Racr:
            data_set['label'][idx,0] = 0
            data_set['label'][idx,1] = 1
            counter_for_unstable += 1
        else:
            data_set['label'][idx,0] = 1
            data_set['label'][idx,1] = 0
            counter_for_stable += 1
    split_index = int(instance_num*split_rate)

    train_set = {'input':data_set['input'][0:split_index,:],
                 'label':data_set['label'][0:split_index,:]}
    test_set = {'input':data_set['input'][split_index:,:],
                'label':data_set['label'][split_index:,:]}

    print('Stable:{} UnStable:{}'.format(counter_for_stable, counter_for_unstable))

    return train_set,test_set
```

Build Inference

```
#define full connection layer
def full_connection_layer(input_tensor, n_out,
                          w_init=tf.truncated_normal_initializer(stddev=0.1),
                          b_init=tf.constant_initializer(0.1),
                          activation=tf.nn.sigmoid, name=None):
    n_in = input_tensor.get_shape().as_list()[1]
    with tf.variable_scope(name):
        weight = tf.get_variable('weight', [n_in, n_out], initializer=w_init)
        bias = tf.get_variable('bias', [n_out], initializer=b_init)
    output_tensor = activation(tf.matmul(input_tensor, weight)+bias, name=name+'_output')
    return output_tensor
```

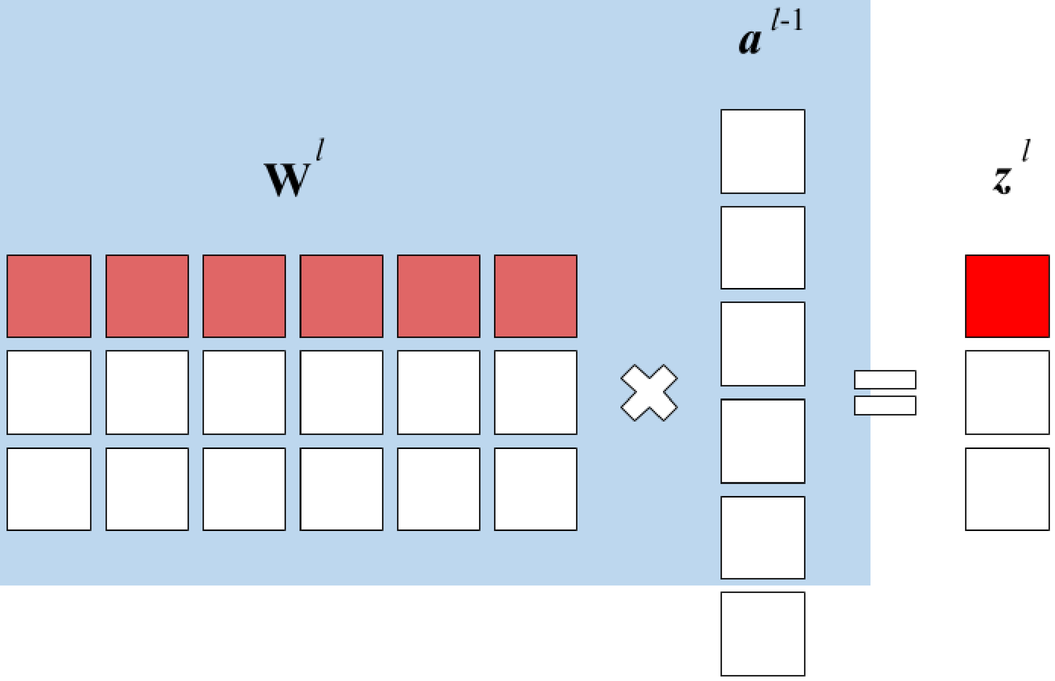
```
def inference(input_tensor):
    hidden_layer_1 = full_connection_layer(input_tensor=input_tensor, n_out=4, name='fc_layer_1')
    hidden_layer_2 = full_connection_layer(input_tensor=hidden_layer_1, n_out=4, name='fc_layer_2')
    hidden_layer_3 = full_connection_layer(input_tensor=hidden_layer_2, n_out=4, name='fc_layer_3')
    pred = full_connection_layer(input_tensor=hidden_layer_3, n_out=2, name='pred')
    return pred
```

Classifying Stability of Mantle with Neural Networks: An Example



Fully Connected layer

$$\mathbf{a}^{l} = \sigma(\mathbf{W}^{l}\mathbf{a}^{l-1} + \mathbf{b}^{l})$$



https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d

Classifying Stability of Mantle with Neural Networks: An Example

Training

In [7]:
```
#set param for training
step_num = 20001
batch_size = 1000
data_length = 8
learning_rate = 0.01
#setup training
input_tensor = tf.placeholder(tf.float32, [None, data_length], name='input')
label = tf.placeholder(tf.float32, [None, 2], name='label')
pred = inference(input_tensor=input_tensor)
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=label))
train_op = tf.train.AdamOptimizer(learning_rate = learning_rate).minimize(loss)
```

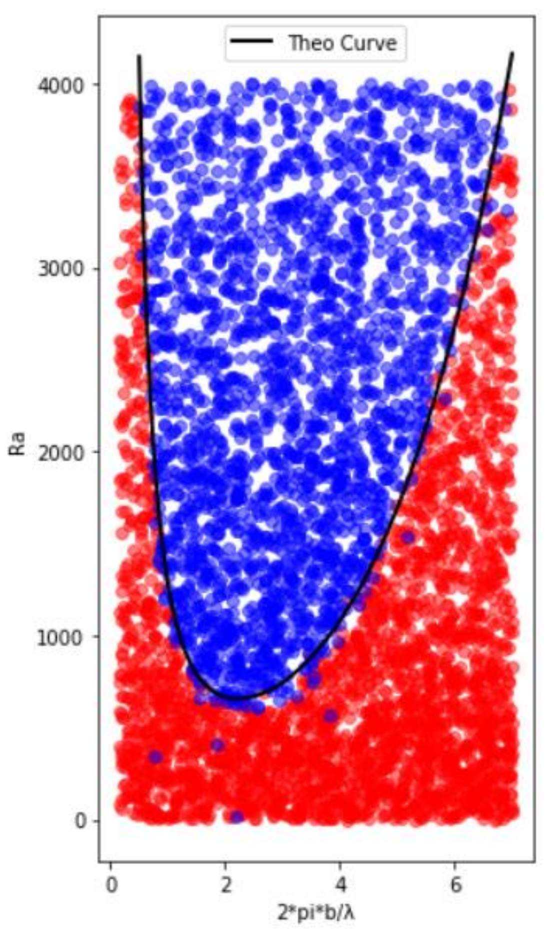
In [8]:
```
def get_traning_batch(train_set, batch_size, data_length):
    batch_ids = np.random.choice(len(train_set['input']), batch_size)
    input_batch = np.zeros([batch_size, data_length])
    label_batch = np.zeros([batch_size, 2])
    for idx in range(batch_size):
        input_batch[idx][:] = train_set['input'][batch_ids[idx]][:]
        label_batch[idx][:] = train_set['label'][batch_ids[idx]][:]
    return input_batch, label_batch
```

In [9]:
```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

In [10]:
```
#start traning
for idx in range(step_num):
    input_batch, label_batch = get_traning_batch(train_set, batch_size, data_length)
    _, loss_val = sess.run([train_op, loss], {input_tensor: input_batch, label: label_batch})
    if idx%2000 == 0:
        print(loss_val)
```

Classifying Stability of Mantle with Neural Networks: An Example

Testing

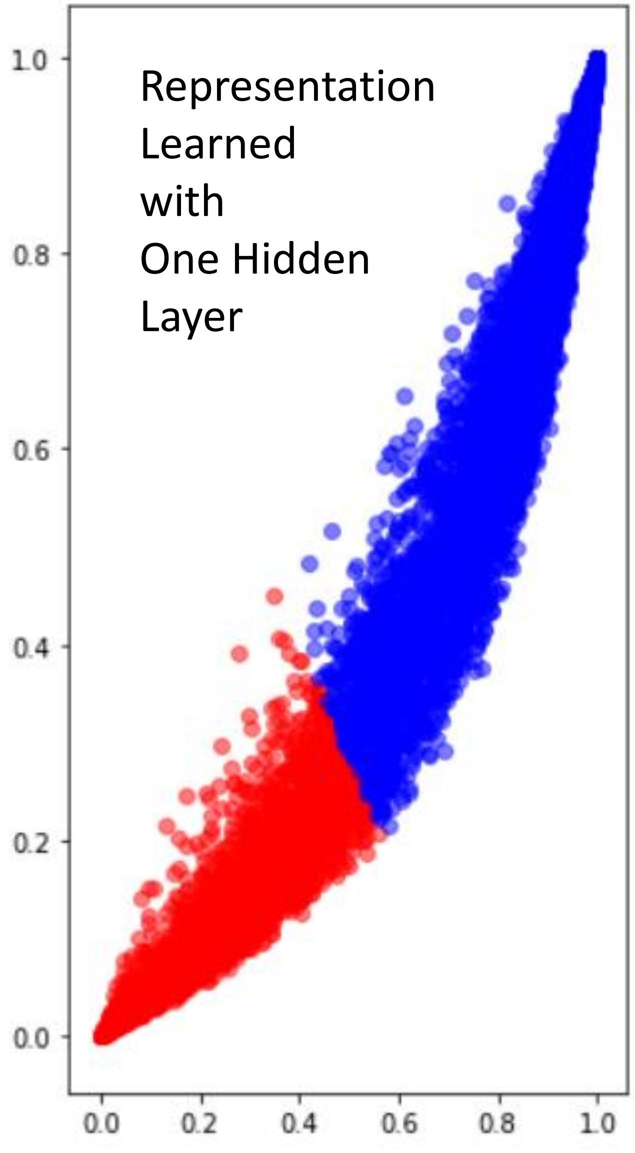


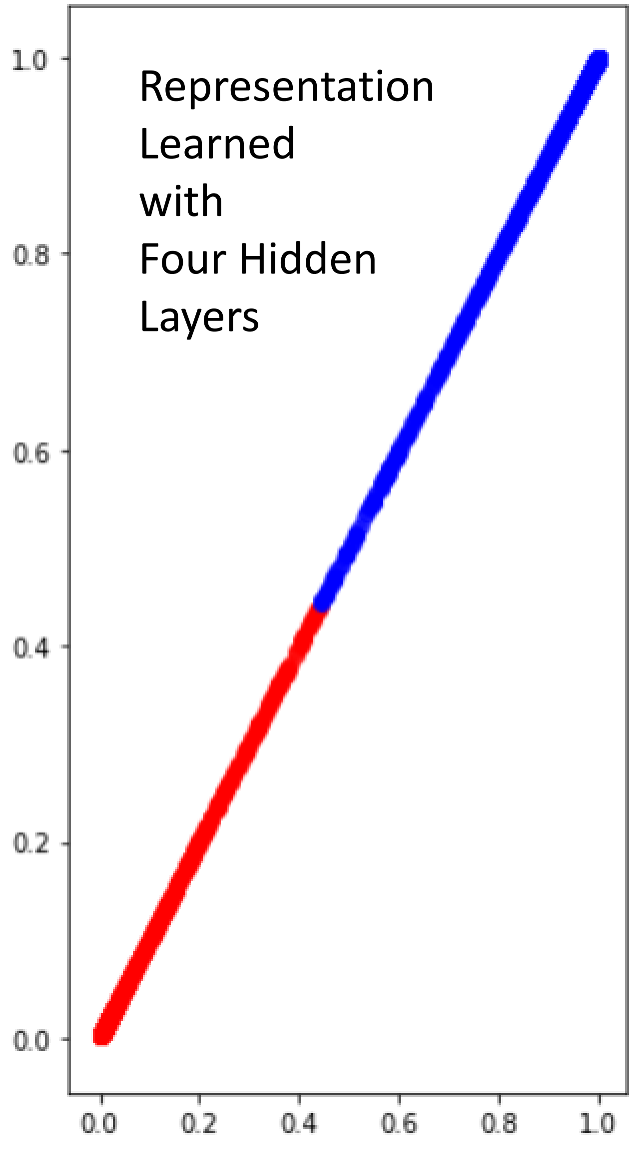
In [11]:
```
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(label, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
test_pred, test_accuracy = sess.run([pred, accuracy], {input_tensor: test_set['input'], label: test_set['label']})
print('Accuracy: {}'.format(test_accuracy))
```

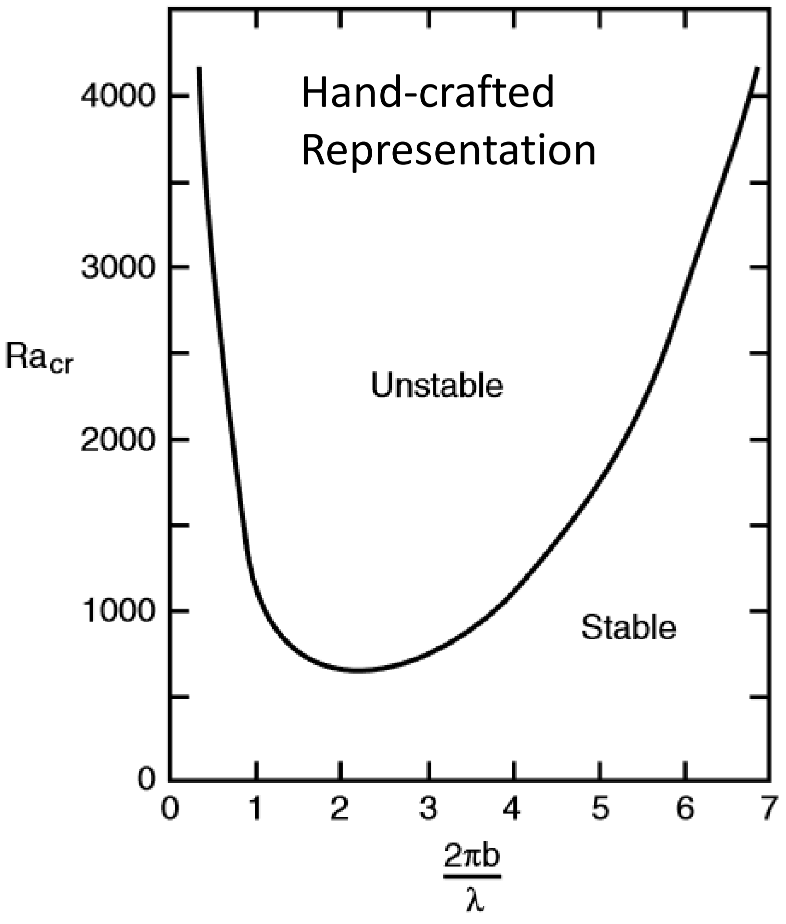
```
Accuracy: 0.9589059948921204
```

In [24]:
```
#compare test results with theoretical curve
plt.figure(figsize=(4,8))

for idx in range(max(len(test_set['input']), 10000)):
    g = test_set['input'][idx,0]*10
    a = test_set['input'][idx,1]*1e-5
    v = test_set['input'][idx,2]
    k = test_set['input'][idx,3]
    d = test_set['input'][idx,4]
    b = test_set['input'][idx,5]*10000
    lam = test_set['input'][idx,6]*10000
    dT = test_set['input'][idx,7]*1000
    #map_x = ((np.pi**4)*((4+(lam/b)**2)**3))/(4*((lam/b)**4))
    map_x = (2.0*np.pi*b)/lam
    map_y = ((a*g*dT*(d**3))/(v*k))*1e7
    if map_y > 4000:
        continue

    if np.argmax(test_pred[idx])==0:
        map_color = 'r'
    else:
        map_color = 'b'
    plt.plot([map_x], [map_y], color=map_color, marker='o')
#plot theo curve
plot_x = np.arange(0.5,7,0.001)
plot_y = ((np.pi**4)*((4+(2*np.pi/plot_x)**2)**3))/(4*((2*np.pi/plot_x)**4))
plt.plot(plot_x, plot_y, color='k', linewidth=2, label='Theo Curve')
plt.xlabel('2*pi*b/λ')
plt.ylabel('Ra')
plt.legend()
plt.show()
```

Classifying Stability of Mantle with Neural Networks: An Example





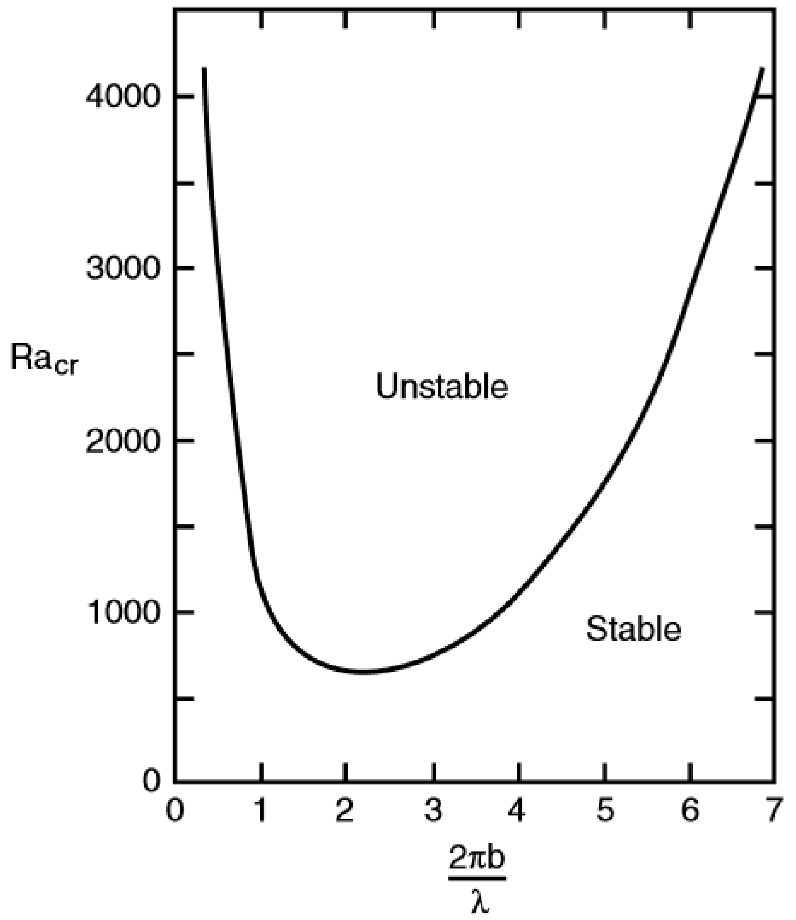


Classifying Stability of Mantle with Neural Networks: An Example

$$Ra = Ra_{cr} = \frac{(\pi^2 + 4\pi^2/\lambda^{*2})^3}{4\pi^2/\lambda^{*2}} = \frac{\pi^4}{4\lambda^{*4}}(4 + \lambda^{*2})^3$$

$$Ra = \frac{\alpha g (T_1 - T_0) b^3}{\nu\kappa}$$

$$\lambda^* = \lambda / b$$



(Schuber et al., 2001)

OUTLINES

1 Brief Introduction to Deep Learning

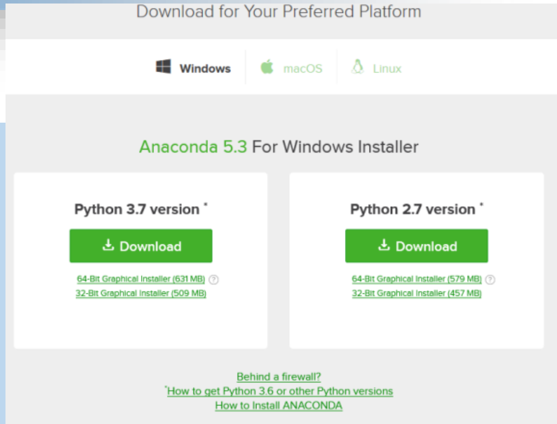
2 TensorFlow Basics

3 Classifying Stability of Mantle with Neural Networks: An Example

4 Discussions

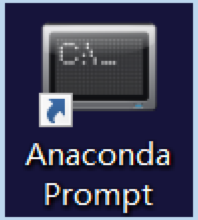
Discussions

TensorFlow Installation via Anaconda

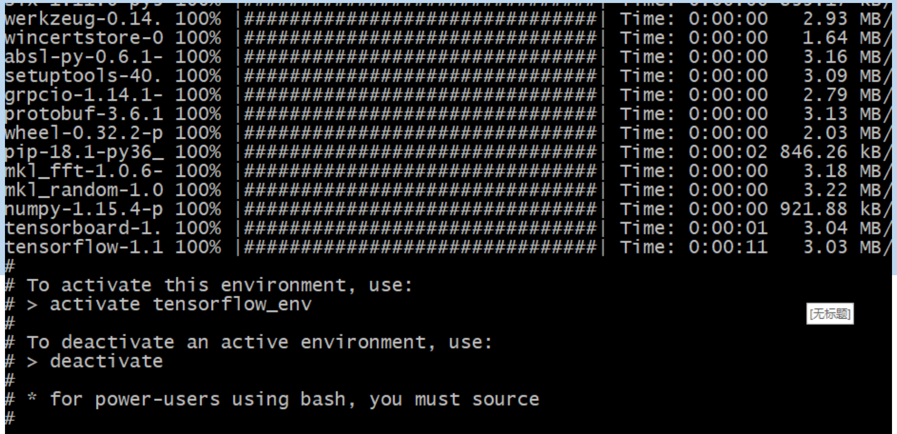


Step 1. Install Anaconda from (https://www.anaconda.com/download/)

Step 2. Create a new conda environment containing TensorFlow.
Open Anaconda Prompt and run
'conda create -n tensorflow_env tensorflow python=3.6'
or
'conda create -n tensorflow_gpuenv tensorflow-gpu python=3.6'
for GPU version



Congratulations...



References

[1] Ian Goodfellow Yoshua Bengio and A. Courville, “Deep Learning,” 2016.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015.

[3] P. M. R. DeVries, F. Viégas, M. Wattenberg, and B. J. Meade, “Deep learning of aftershock patterns following large earthquakes,” *Nature*, vol. 560, no. 7720, pp. 632–634, Aug. 2018.

[4] L. Zhang, L. Zhang, and B. Du, “Deep Learning for Remote Sensing Data: A Technical Tutorial on the State of the Art,” *IEEE Geoscience and Remote Sensing Magazine*, vol. 4, no. 2, pp. 22–40, Jun. 2016.

[5] S. Beckouche and J. Ma, “Simultaneous dictionary learning and denoising for seismic data,” *GEOPHYSICS*, vol. 79, no. 3, pp. A27–A31, May 2014.

[6] B. Moseley, A. Markham, and T. Nissen-Meyer, “Fast approximate simulation of seismic waves with deep learning,” *arXiv:1807.06873 [physics]*, Jul. 2018.

[7] M. Araya-Polo, J. Jennings, A. Adler, and T. Dahlke, “Deep-learning tomography,” *The Leading Edge*, vol. 37, no. 1, pp. 58–66, Jan. 2018.

[8] J. Adler and O. Öktem, “Solving ill-posed inverse problems using iterative deep neural networks,” *Inverse Problems*, vol. 33, no. 12, p. 124007, Dec. 2017.