Meeting 3: conda, python, numpy, and pytorch intro

Speech tools minicourse
Summer 2020
Outline

• Installing anaconda and pytorch
• Reference works available online for python, numpy, and pytorch
• Training and testing a neural network using numpy, in jupyter
• Training and testing using pytorch tensors
• Copying your code to a standalone program
Installing anaconda
Installing anaconda
Installing anaconda

Your data science toolkit

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Download
...wait a while so it can install...
Installing pytorch
Installing pytorch

Choose “Stable”
Choose your OS
Choose “Conda”
Choose “Python”
Choose “None”
Installing pytorch

• Open a terminal
• Enter the appropriate command for your OS. It should be either:
• Windows and Linux:
  conda install pytorch torchvision cpuonly -c pytorch
• Mac:
  conda install pytorch torchvision -c pytorch
Outline

- Installing anaconda and pytorch
- Reference works available online for python, numpy, and pytorch
- Training and testing a neural network using numpy, in jupyter
- Training and testing using pytorch tensors
- Copying your code to a standalone program
Reference works available online for python, numpy, and pytorch

• When you use python, numpy, and pytorch, you want to keep the basic references open pretty much all the time:
  https://docs.python.org/3/tutorial/index.html
  https://www.tutorialspoint.com/numpy/index.htm
  https://pytorch.org/tutorials/

• You also want to make frequent use of the search engine, to search for answers to problems. For example, if I type “how can I change a numpy array from 4x3 to 2x2x3?” into ecosia, I get this answer, which almost solves the problem, but not quite:
Outline

• Installing anaconda and pytorch
• Reference works available online for python, numpy, and pytorch
• Training and testing a neural network using numpy, in jupyter
• Training and testing using pytorch tensors
• Copying your code to a standalone program
Training and testing a neural network using numpy

For the rest of today and next week, I’ll be mostly following this tutorial: https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
You can launch Jupyter from Anaconda navigator.
Create a new notebook

Navigate to the directory where you want to save your new notebook, by clicking here.

Then click the “notebook” icon, or else choose “New -> Notebook” to create a new one.
Save as ....

It creates one called “Untitled.ipynb”, which is probably not what you want to call it, so choose “File -> Save Notebook as..” to choose a new name.
import numpy as np
N, D_in, H, D_out = 90, 2, 100, 9
print('There are %d tokens, the network layer sizes are (%d,%d,%d)\%(N,D_in,H,D_out))
...
Create some class labels

```python
class_labels = np.array([int(n/10) for n in range(N)])
import matplotlib.pyplot as plt
plt.plot(class_labels)
plt.ylabel('class labels (y)')
plt.xlabel('token index (i)')
plt.savefig('class_labels.png')
```
Create one-hot label vectors for each training token

```python
y_onehot = np.zeros((N, D_out))
for i in range(N):
    y_onehot[i, class_labels[i]] = 1
plt.imshow(y_onehot)
plt.xlabel('class label, y')
plt.ylabel('token index, i')
plt.savefig('onehot.png')
```
Create some mean vectors

```
mean_vectors = np.array( [[int(y/3), y-3*int(y/3)] for y in class_labels ])
print(mean_vectors.shape)

ax=plt.subplot(111, xlim=[-1,3], ylim=[-1,3])
for i in range(N):
    plt.text(mean_vectors[i,0], mean_vectors[i,1], '%d'%(class_labels[i]))
plt.savefig('mean_vectors.png')
```
Create some data: normal with stdev=0.25, with mean that depends on the class label

data = np.random.normal(loc=mean_vectors, scale=0.25, size=(90,2))
ax=plt.subplot(111,xlim=[-1,3],ylim=[-1,3])
for i in range(N):
    plt.text(data[i,0],data[i,1],'%d'%class_labels[i])
plt.title('Data with ground-truth labels')
plt.savefig('data.png')
Randomly initialize the weights

```python
w1 = np.random.normal(loc=0.0, scale=0.1, size=(D_in, H))
b1 = np.random.normal(loc=0.0, scale=0.1, size=(1,H))
w2 = np.random.normal(loc=0.0, scale=0.1, size=(H, D_out))
b2 = np.random.normal(loc=0.0, scale=0.1, size=(1, D_out))
learning_rate = 0.001
print('w1 has shape %s, w2 has shape %s'%(w1.shape, w2.shape))
print('learning rate is %g'%learning_rate)
```
How good is the initialization?

```python
def forward_pass(x, w1, b1, w2, b2):
    N, D_in = x.shape
    h_excitation = np.dot(x, w1) + b1
    h_activation = np.maximum(0, h_excitation)
    y_excitation = np.dot(h_activation, w2) + b2
    y_numerator = np.exp(y_excitation)
    y_activation = y_numerator / np.sum(y_numerator, axis=1).reshape((N, 1))
    return(y_activation, y_excitation, h_activation, h_excitation)

(ya, ye, ha, he) = forward_pass(data, w1, b1, w2, b2)
ax = plt.subplot(111, xlim=[-1, 3], ylim=[-1, 3])
for i in range(N):
    plt.text(data[i, 0], data[i, 1], '%d' % np.argmax(ya[i, :]))

loss = -np.average(np.log(np.choose(class_labels, np.transpose(ya))))
plt.title('Data labeled by Initial Weights; average loss=%g nats' % (loss))
plt.savefig('network_output_initial.png')
```
Let’s train it by one iteration...

def training_iter(x,y_onehot,w1,b1,w2,b2,learning_rate):
    (ya,ye,ha,he)=forward_pass(x,w1,b1,w2,b2)
    ye_deriv = ya - y_onehot
    ha_deriv = np.dot(ye_deriv, np.transpose(w2))
    he_deriv = ha_deriv * np.heaviside(he,0.5)
    db2 = np.sum(ye_deriv, axis=0)
    dw2 = np.dot(np.transpose(ha),ye_deriv)
    db1 = np.sum(he_deriv, axis=0)
    dw1 = np.dot(np.transpose(x),he_deriv)
    new_b2 = b2 - learning_rate*db2
    new_w2 = w2 - learning_rate*dw2
    new_b1 = b1 - learning_rate*db1
    new_w1 = w1 - learning_rate*dw1
    return(new_w1,new_b1,new_w2,new_b2)

nw1,nb1,nw2,nb2 = training_iter(data, y_onehot, w1, b1,w2, b2,learning_rate)
print('w1 average change is %g'%(np.average(np.absolute(w1-nw1))))
print('b1 average change is %g'%(np.average(np.absolute(b1-nb1))))
print('w2 average change is %g'%(np.average(np.absolute(w2-nw2))))
print('b2 average change is %g'%(np.average(np.absolute(b2-nb2))))
...and see if it got any better

def visualize_and_score(x,y,class_labels,iternum):
    N,D = x.shape
    ax=plt.subplot(111,xlim=[-1,3],ylim=[-1,3])
    for i in range(N):
        plt.text(x[i,0],x[i,1],'%d' % np.argmax(y[i,:]))
    loss = -np.average(np.log(np.choose(class_labels, np.transpose(y))))
    plt.title('Data after %d iters; average loss=%g nats' % (iternum,loss))

(ya,ye,ha,he)=forward_pass(data,nw1,nb1,nw2,nb2)
visualize_and_score(data,ya,class_labels,1)
plt.savefig('iter1.png')
Let’s train it by 500 iterations...

for iternum in range(500):
    (w1,b1,w2,b2)=training_iter(data,y_onehot,w1,b1,w2,b2,learning_rate)
    (ya,ye,ha,he)=forward_pass(data,w1,b1,w2,b2)
    loss = -np.average(np.log(np.choose(class_labels, np.transpose(ya))))
    print('Iter %d: average loss=%g nats'%(iternum,loss))
...and see if it got any better

(ya, ye, ha, he) = forward_pass(data, w1, b1, w2, b2)
visualize_and_score(data, ya, class_labels, 500)
plt.savefig('iter500.png')
Outline

• Installing anaconda and pytorch
• Reference works available online for python, numpy, and pytorch
• Training and testing a neural network using numpy, in jupyter
• Training and testing using pytorch tensors
• Copying your code to a standalone program
Pytorch tensors

• Pytorch tensors are basically the same thing as numpy arrays, with two important differences:
  1. Dot products are done on the GPU, if you have one
  2. The “autograd” feature will automatically calculate the derivatives for you, so that you don’t have to write that extra code. More about this feature next time.

• Pytorch also provides a higher-level API called “nn” (neural net), with a lot more features. More about this next time.

• How to use pytorch: write everything at the nn level, then debug by extracting torch.tensors and np.arrays, so you can plot them.
Pytorch tensors step 1: import torch

import torch
dtype = torch.float
device = torch.device("cpu")
N, D_in, H, D_out = 90, 2, 100, 9
print('There are %d tokens, the network layer sizes are (%d,%d,%d)'%(N,D_in,H,D_out))
Pytorch tensors step 2: define class labels

class_labels = torch.tensor([ int(n/10) for n in range(N)], dtype=torch.int, device=device)

import matplotlib.pyplot as plt
plt.plot(class_labels)
plt.ylabel('class labels (y)')
plt.xlabel('token index (i)')
plt.savefig('class_labels.png')
Pytorch tensors step 3: create onehots

```python
y_onehot = torch.zeros((N, D_out), dtype=dtype, device=device)
for i in range(N):
    y_onehot[i,class_labels[i]]=1
plt.imshow(y_onehot)
plt.xlabel('class label, y')
plt.ylabel('token index, i')
plt.savefig('onehot.png')
```
Pytorch tensors step 4: create mean vectors

```python
mean_vectors = torch.tensor( [[int(y//3), y-3*int(y//3)] for y in class_labels ], dtype=dtype,
device=device)
print(mean_vectors.shape)

ax=plt.subplot(111, xlim=[-1,3], ylim=[-1,3])
for i in range(N):
    plt.text(mean_vectors[i,0], mean_vectors[i,1], '%d'%%(class_labels[i]))
plt.savefig('mean_vectors.png')
```
Pytorch tensors step 5: create data

```python
stdev = torch.ones((N,D_in),dtype=dtype,device=device)*0.25
data = torch.distributions.Normal(loc=mean_vectors,scale=stdev).sample()
ax=plt.subplot(111,xlim=[-1,3],ylim=[-1,3])
for i in range(N):
    plt.text(data[i,0],data[i,1],'%d'%class_labels[i])
plt.title('Data with ground-truth labels')
plt.savefig('data.png')
```
Pytorch tensors step 6: initialize weights

```python
w1 = torch.randn(D_in, H, dtype=dtype, device=device)*0.1
b1 = torch.randn(1,H, dtype=dtype, device=device)*0.1
w2 = torch.randn(H, D_out, dtype=dtype, device=device)*0.1
b2 = torch.randn(1, D_out, dtype=dtype, device=device)*0.1
learning_rate = 0.001
print('w1 has shape %s, w2 has shape %s'%(w1.shape, w2.shape))
print('learning rate is %g'%(learning_rate))
```
def forward_pass(x, w1, b1, w2, b2):
    N, D_in = x.shape
    h_excitation = x.mm(w1) + b1
    h_activation = h_excitation.clamp(min=0)
    y_excitation = h_activation.mm(w2) + b2
    y_numerator = torch.exp(y_excitation)
    y_activation = y_numerator / torch.sum(y_numerator, dim=1)
    return (y_activation, y_excitation, h_activation, h_excitation)

(ya, ye, ha, he) = forward_pass(data, w1, b1, w2, b2)
ax = plt.subplot(111, xlim=[-1,3], ylim=[-1,3])
for i in range(N):
    plt.text(data[i, 0], data[i, 1], '%d' % torch.argmax(ya[:, i]))

loss = -np.average(np.log(np.choose(class_labels.numpy(), ya.numpy().T)))
plt.title('Data labeled by Initial Weights; average loss=%g nats' % loss)
plt.savefig('network_output_initial.png')
def training_iter(x,y_onehot,w1,b1,w2,b2,learning_rate):
    (ya,ye,ha,he)=forward_pass(x,w1,b1,w2,b2)
    ye_deriv = ya - y_onehot
    ha_deriv = ye_deriv.mm(w2.t())
    he_deriv = ha_deriv.clone()
    he_deriv[he < 0] = 0
    db2 = torch.sum(ye_deriv, dim=0)
    dw2 = ha.t().mm(ye_deriv)
    db1 = torch.sum(he_deriv, dim=0)
    dw1 = x.t().mm(he_deriv)
    new_b2 = b2 - learning_rate*db2
    new_w2 = w2 - learning_rate*dw2
    new_b1 = b1 - learning_rate*db1
    new_w1 = w1 - learning_rate*dw1
    return(new_w1,new_b1,new_w2,new_b2)

nw1,nb1,nw2,nb2 = training_iter(data, y_onehot, w1, b1,w2, b2,learning_rate)
print('w1 average change is %g'*(torch.mean(torch.abs(w1-nw1))))
print('b1 average change is %g'*(torch.mean(torch.abs(b1-nb1))))
print('w2 average change is %g'*(torch.mean(torch.abs(w2-nw2))))
print('b2 average change is %g'*(torch.mean(torch.abs(b2-nb2))))
Pytorch tensors step 9: visualize the result

def visualize_and_score(x,y,class_labels,iternum):
    N,D = x.shape
    ax=plt.subplot(111,xlim=[-1,3],ylim=[-1,3])
    for i in range(N):
        plt.text(x[i,0],x[i,1],'%d'%torch.argmax(y[i,:]))
    loss = -np.average(np.log(np.choose(class_labels.numpy(), y.numpy().T)))
    plt.title('Data after %d iters; average loss=%g nats'%((iternum,loss))

(ya,ye,ha,he)=forward_pass(data,nw1,nb1,nw2,nb2)
visualize_and_score(data,ya,class_labels,1)
plt.savefig('iter1.png')
Pytorch tensors step 10: train 500 iterations

for iternum in range(500):
    (w1,b1,w2,b2)=training_iter(data,y_onehot,w1,b1,w2,b2,learning_rate)
    (ya,ye,ha,he)=forward_pass(data,w1,b1,w2,b2)
    loss = -np.average(np.log(np.choose(class_labels.numpy(), ya.numpy().T)))
    print('Iter %d: average loss=%g nats'%(iternum,loss))

    (ya,ye,ha,he)=forward_pass(data,w1,b1,w2,b2)
    visualize_and_score(data,ya,class_labels,500)
plt.savefig('iter500.png')
Outline

• Installing anaconda and pytorch
• Reference works available online for python, numpy, and pytorch
• Training and testing a neural network using numpy, in jupyter
• Training and testing using pytorch tensors
• Copying your code to a standalone program
Copying your code to a standalone program

• Choose “Export notebook to executable script”
• This will save it, by default, in your Downloads directory. You’ll have to rename it to somewhere useful.
Copying your code to a standalone program

more pytorch_tutorial.py:

```python
#!/usr/bin/env python
# coding: utf-8

# In[133]:

import numpy as np

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 90, 2, 100, 9
print('There are %d tokens, the network layer sizes are (%d,%d,%d)'%(N,D_in,H,D_out))

# In[134]:

class_labels = np.array([ int(n/10) for n in range(N) ], dtype='int')
import matplotlib.pyplot as plt
plt.plot(class_labels)
plt.ylabel('class labels (y)')
plt.xlabel('token index (i)')
plt.savefig('class_labels.png')
```
Copying your code to a standalone program

conda activate
python pytorch_tutorial.py: