Pytorch Basic in Colab

Machine Intelligence Lab
Handong Global University

MNIST example with Pytorch

adapted from https://github.com/pytorch/

assuming the fields of args have proper values.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim                                    # import torch modules
from torchvision import datasets, transforms
import os
```

```
train_loader = torch.utils.data.DataLoader(                    # data iterator
                datasets.MNIST('drive/My Drive/public/data', train=True,
                download=True, transform=transform),
                batch_size=batch_size, shuffle=True, **kwargs)
```

```
model = Net()
                                                               # model define (see the next slides)
```

```
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
                                                               # optimizer setting
```

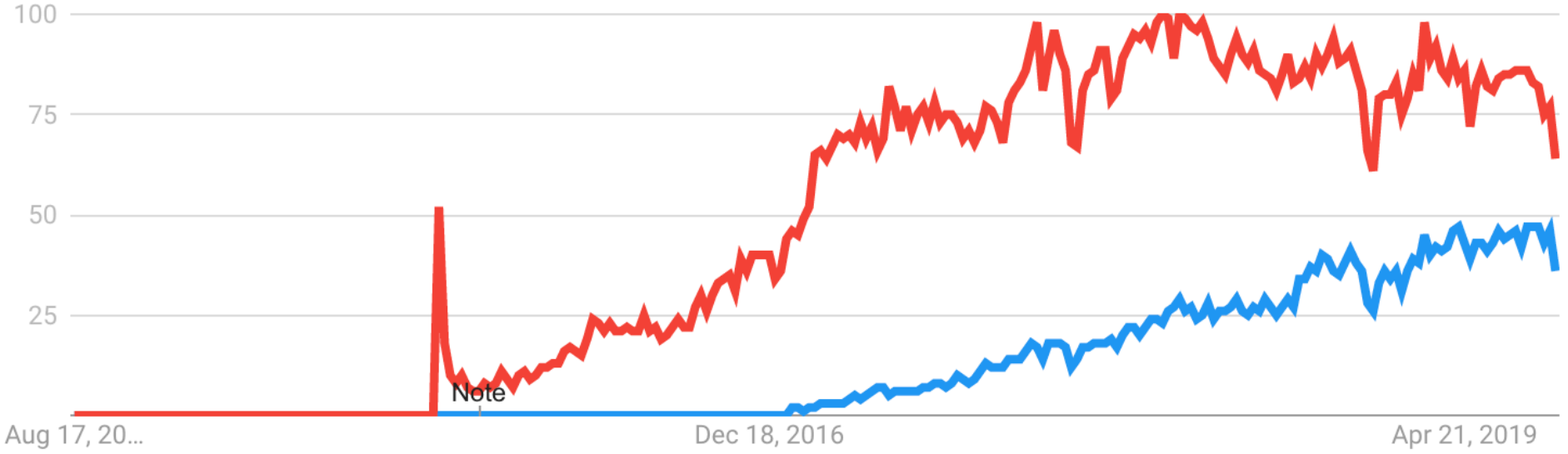
```
for epoch in range(1, epochs + 1):                             # training (see the next slide)
    train(model, device, train_loader, optimizer, epoch, log_interval)
    test(model, device, test_loader)
```

MNIST example

```
# model architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 500)
        self.fc2 = nn.Linear(500, 300)
        self.fc3 = nn.Linear(300, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        batch_size, c, h, w = x.data.size()
        x = x.view(batch_size, 784)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x, dim=1)
```

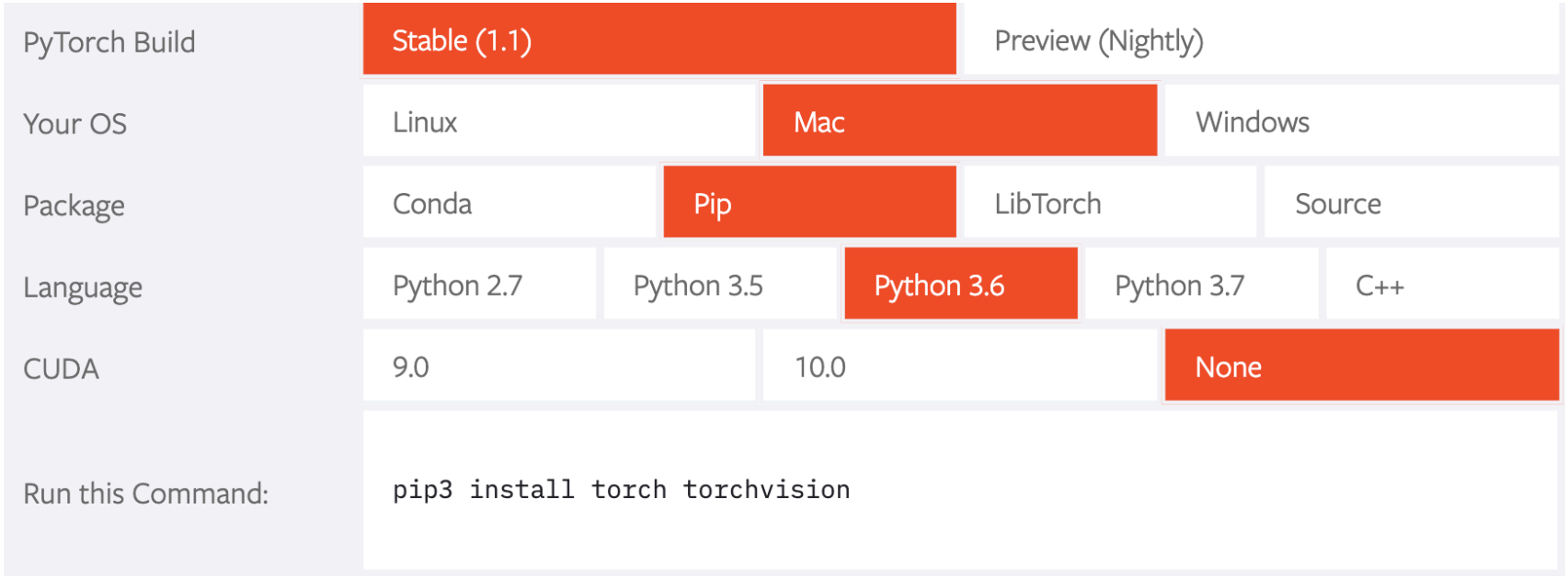
```
# training step
def train(model, device, train_loader, optimizer, epoch, log_interval):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

pytorch or tensorflow in google trends



install

https://pytorch.org/

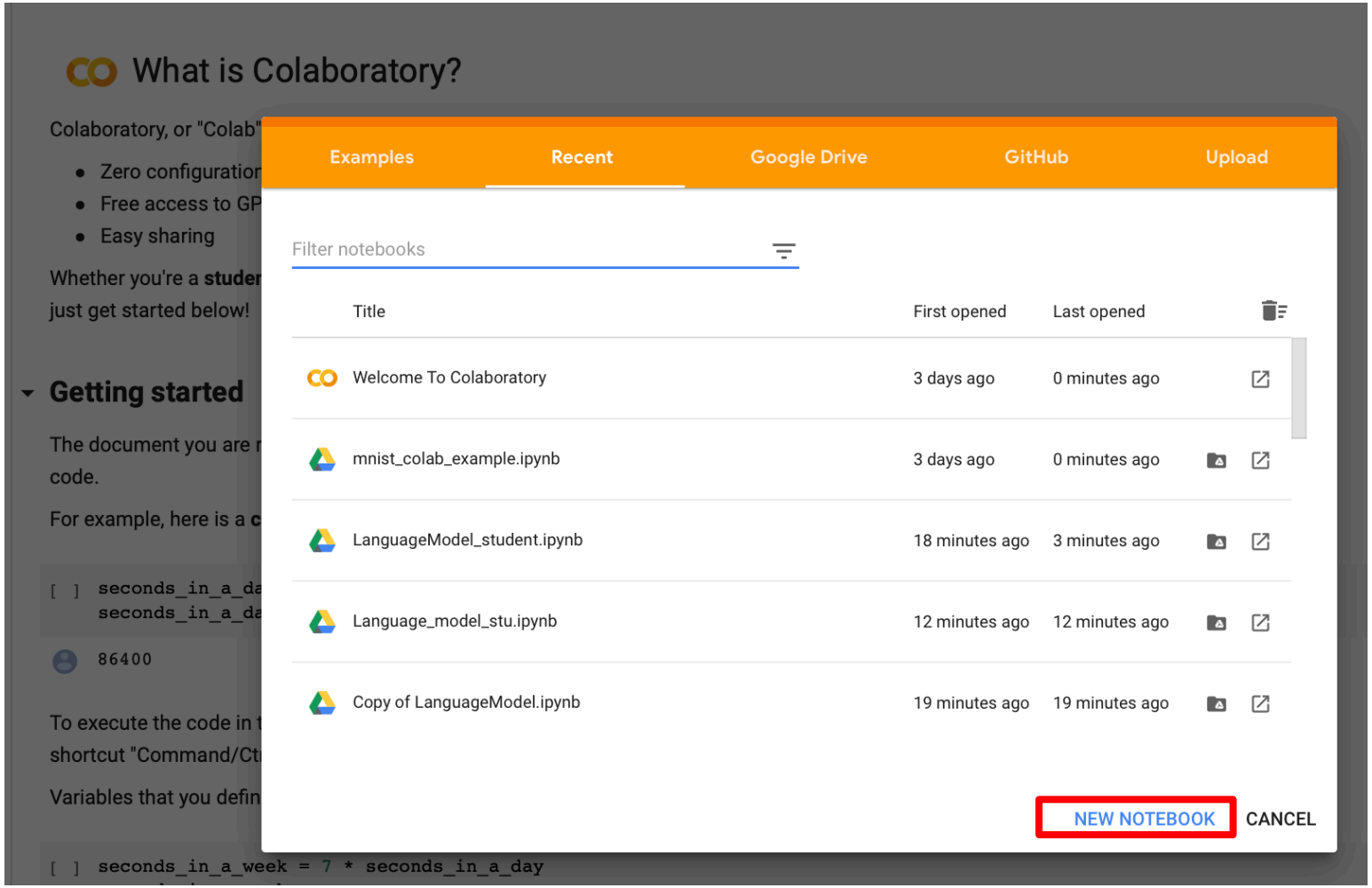


- Install Pytorch with your preference.
- Search about how to install Pytorch on your local preferences.

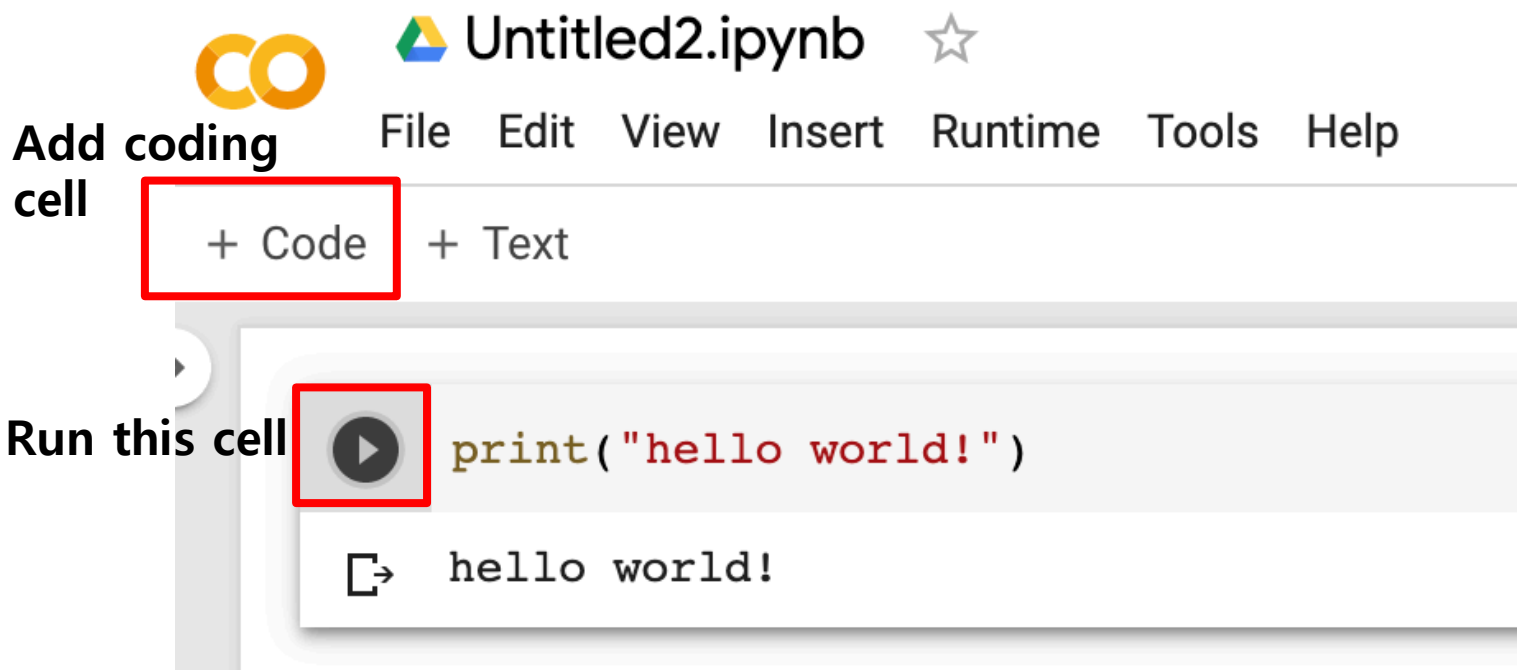
everything is set in Google Colab!

https://colab.research.google.com/

free 12 hours at a time



“hello world” in Google Colab



Pytorch: Basic

PYTORCH

- Python-based scientific computing package
  - to use the power of GPUs instead of Numpy
  - to provide maximum flexibility and speed for deep learning

```
import torch
x = torch.Tensor(5,3)
y = torch.Tensor(5,3)
print(x)
print(y)
```

```
tensor([[9.9759e-36, 0.0000e+00, 3.7835e-44],
        [0.0000e+00,        nan, 0.0000e+00],
        [1.3733e-14, 6.4069e+02, 4.3066e+21],
        [1.1824e+22, 4.3066e+21, 6.3828e+28],
        [3.8016e-39, 1.6115e-43, 0.0000e+00]])
tensor([[9.5391e-36, 0.0000e+00, 3.9236e-44],
        [0.0000e+00,        nan, 0.0000e+00],
        [1.3733e-14, 1.2102e+25, 1.6992e-07],
        [2.9599e+21, 2.8175e+20, 1.7566e+25],
        [1.7748e+28, 0.0000e+00, 9.9884e-36]])
```

Pytorch: Basic

Converting a Torch Tensor to a Numpy array and vice versa is breeze
(Torch Tensor ←→ Numpy)

```
a = torch.ones(5)
print(a)
b = a.numpy()
print(b)
c = torch.from_numpy(b)
print(c)
```

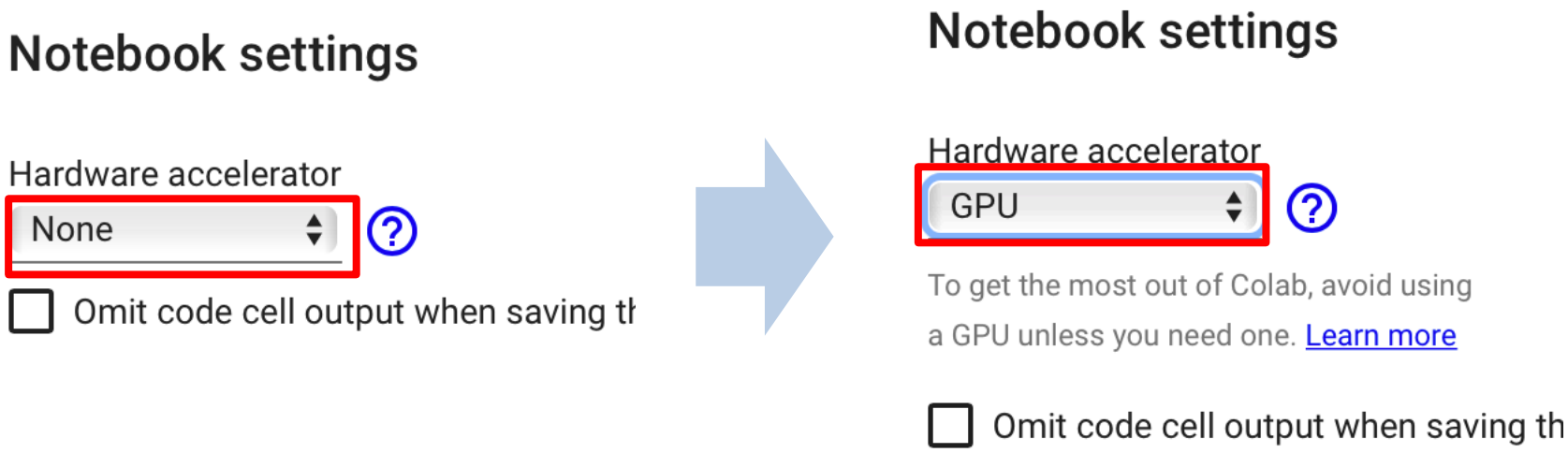
```
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
tensor([1., 1., 1., 1., 1.])
```

torch.ones : create n dim. tensor full of 1.

Pytorch: cuda

in order to use CUDA (i.e., GPU)
you have to set the Hardware accelerator to GPU

Edit tab -> Notebook settings



torch.cuda.is_available() -> True

Pytorch: cuda

CUDA Tensors

Tensors can be moved onto any device using the .to method

```
[10] import torch
     a = torch.ones(5)
     b=a.cuda()
     c=a.to(torch.device("cuda"))
```

```
[11] a
```

```
tensor([1., 1., 1., 1., 1.])
```

```
[12] b
```

```
tensor([1., 1., 1., 1., 1.], device='cuda:0')
```

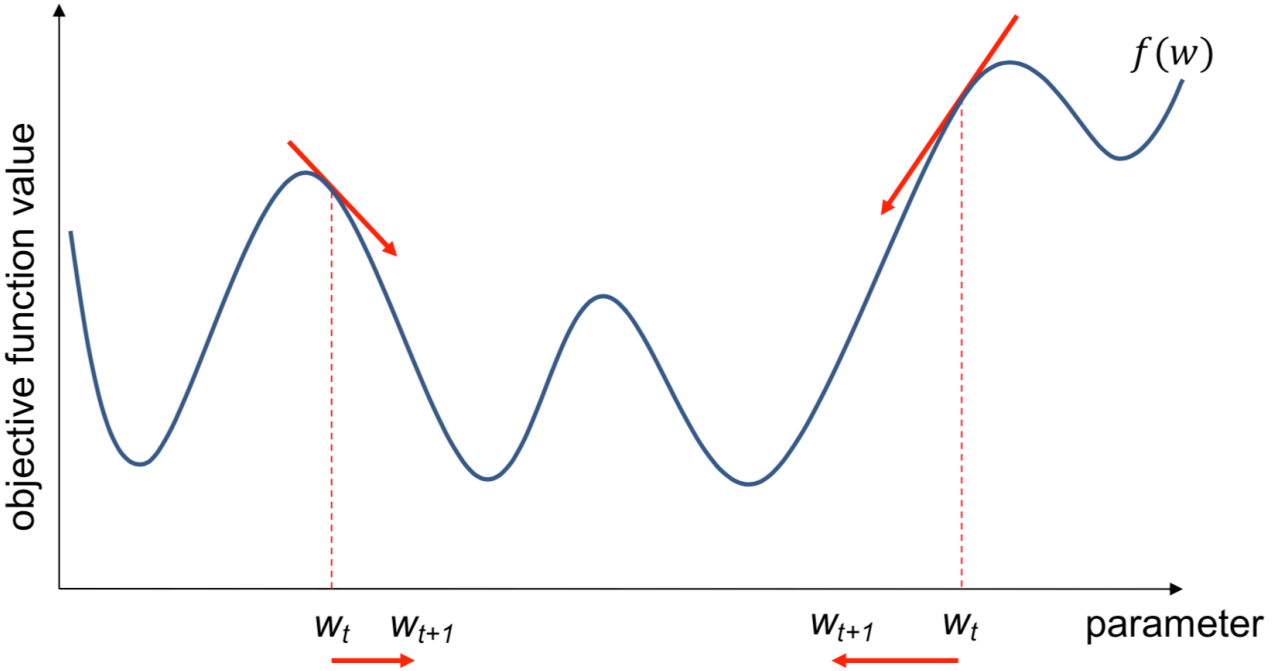
```
[13] c
```

```
tensor([1., 1., 1., 1., 1.], device='cuda:0')
```

CUDA (**Compute Unified Device Architecture**)

a parallel computing platform and API model created by Nvidia.

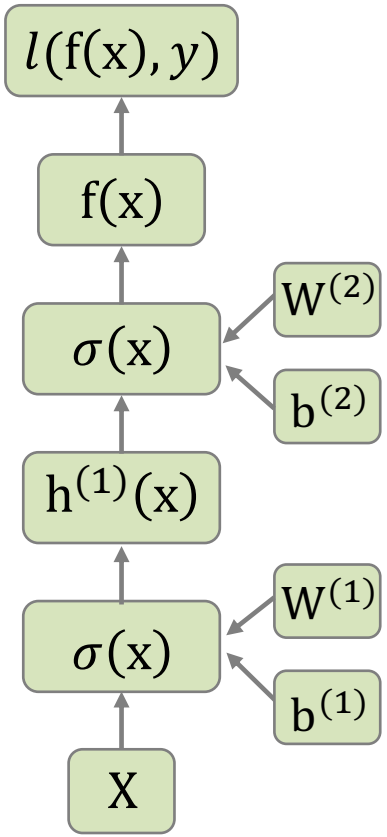
Autograd



- The autograd provides automatic differentiation for all operations on Tensors.
- Once you finish your computation, you can call .backward() for autograd

- **Autograd allows you to automatically compute gradients.**

Autograd



- each object has an fprop/bprop method,
- forward propagation:
  - calling fprop of each box in the right order
- backpropagation:
  - calling bprop in the reverse order

- **a large portion of deep learning research is based on Theano, PyTorch or TensorFlow**

computational graph

Static vs Dynamic Graph

- We define a computational graph, and use automatic differentiation to compute GRADIENTS.

- Tensorflow : Static Graph
  - The graph is defined once and then executed over and over again.
  - Graph is optimized upfront, before the execution.
  - Loops requires specific operations(tf.scan)

- Pytorch : Dynamic Graph
  - Each forward pass defines a new computational graph.
  - Easy control flow
  - Easy to perform different operations for different data points.

torch.nn

Torch.nn?

Neural Network Module.

Easy to make neural network
such as Linear, CNN, RNN, and so on…

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 500)
        self.fc2 = nn.Linear(500, 300)
        self.fc3 = nn.Linear(300, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x, dim=1)
```

loss and optimizer

Loss function

How to define the loss? (MSE, RMSE, CrossEntropy, L1, NLL, etc..)

Optimizer

How to update weights ? (SGD, Adam, RMSProp, AdaDelta, etc…)

Practice

1. Prepare the data and preprocess it
    - implement your data loader

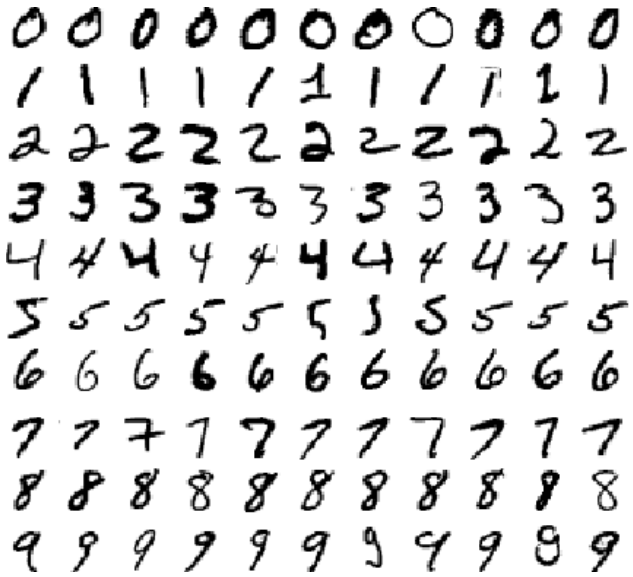
do:

2. Set the model architecture
3. Choose and set an optimizer and an objective function
4. Train the model
5. Validate

while(!good_performance on Validation)

6. Done

Practice – MNIST

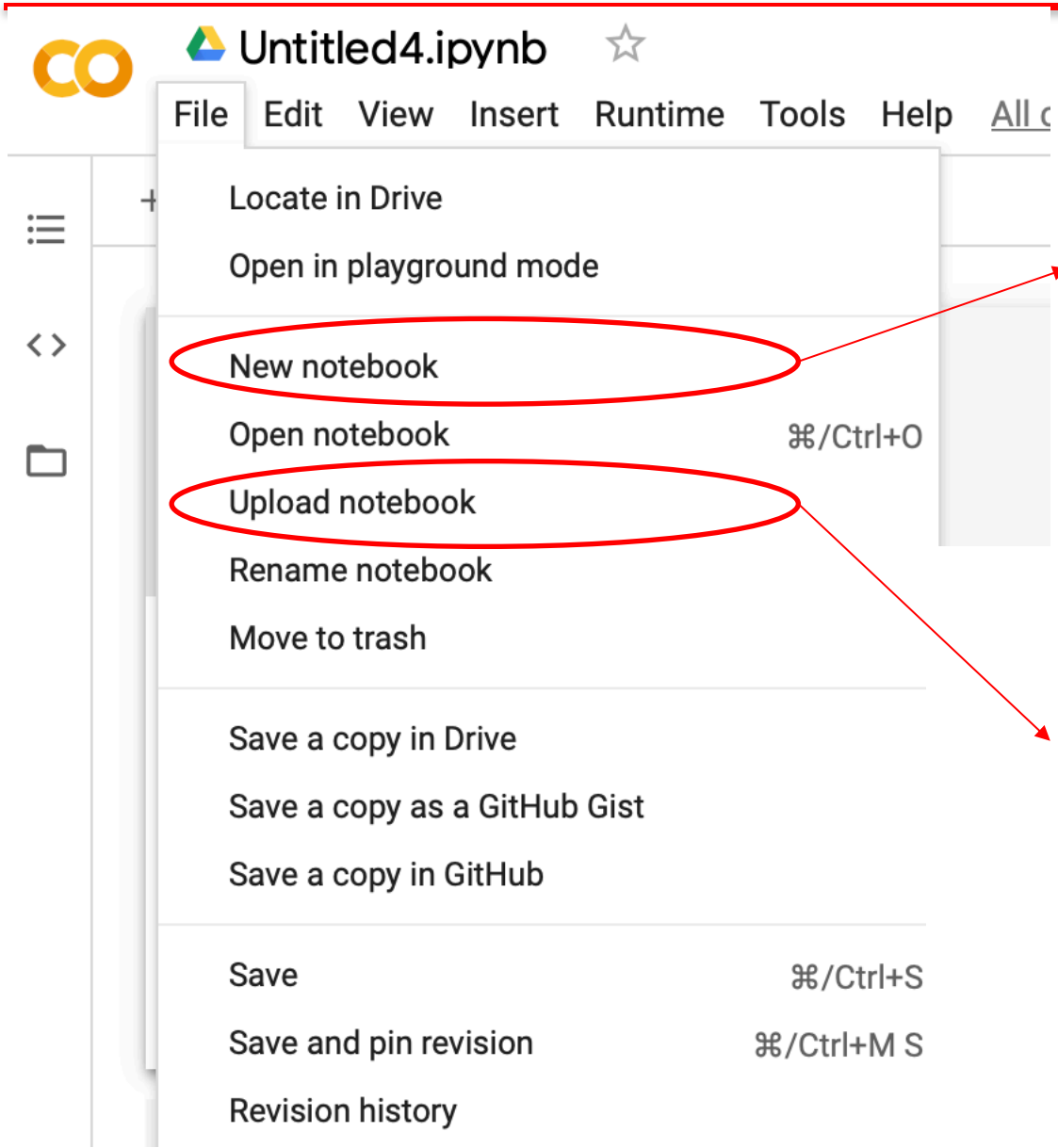


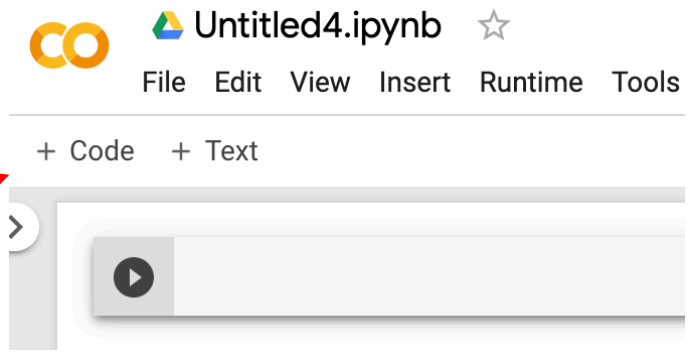
MNIST dataset [28X28] = [1 X 784]

Hand Written Digit Number from 0 to 9
Data includes data and label.

Pytorch Dataset(torchvision) provides
50,000 images to train,
10,000 images to test.

getting started



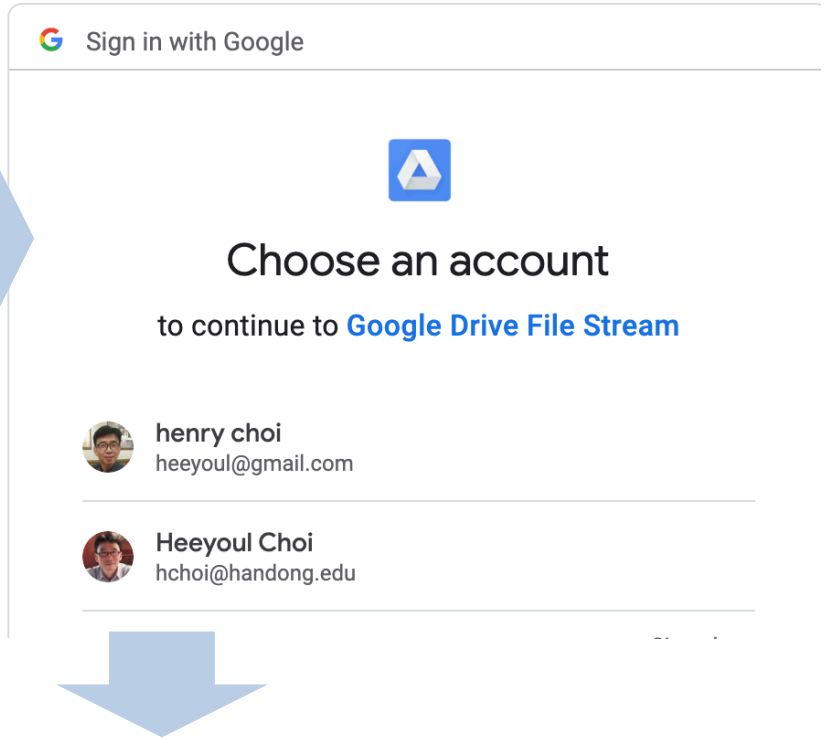


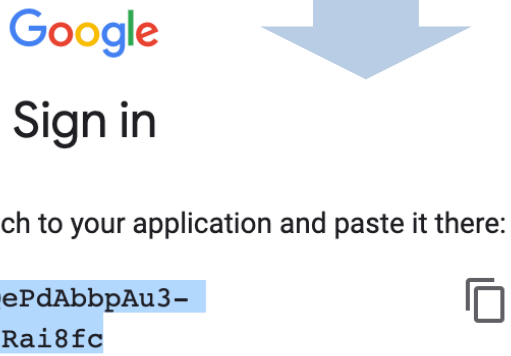
upload 'mnist_colab_example.ipynb'

code: mount google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.g

Enter your authorization code:
```

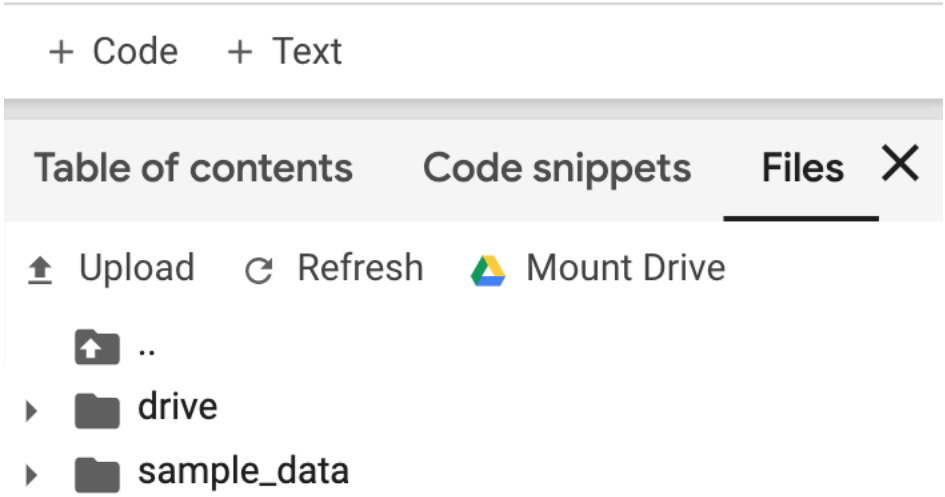




check: mount

```
!ls
```

```
drive  sample_data
```



code: import

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import os
```

code: models

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 500)
        self.fc2 = nn.Linear(500, 300)
        self.fc3 = nn.Linear(300, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        batch_size, c, h, w = x.data.size()
        x = x.view(batch_size, 784)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x, dim=1)
```

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 20, 5, 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(20, 50, 5, 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        conv_size = self.get_conv_size((1, 28, 28))
        self.fc = nn.Sequential(
            nn.Linear(conv_size, 500), # conv_size = 4*4*50
            nn.Linear(500, 10)
        )

    def get_conv_size(self, shape):
        o = self.conv(torch.zeros(1, *shape))
        return int(np.prod(o.size()))

    def forward(self, x):
        batch_size, c, h, w = x.data.size() # 32*1*28*28
        x = self.conv(x)
        x = x.view(batch_size, -1) # conv_size = 4*4*50
        x = self.fc(x)
        return F.log_softmax(x, dim=1)
```

Other network architectures can be used

code: train function

```
def train(model, device, train_loader, optimizer, epoch, log_interval):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

code: test function

```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)

            # sum up batch loss
            test_loss += F.nll_loss(output, target, reduction='sum').item()

            # get the index of the max log-probability
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

code: configuration

```
seed = 1
epochs = 2
batch_size = 32
test_batch_size = 1000
lr = 0.001
momentum = 0.9
log_interval = 100
save_model = True

torch.manual_seed(seed)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

kwargs = {'num_workers': 1, 'pin_memory': True} if torch.cuda.is_available() else {}
```

code: data

```
transform=transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize((0.1307,), (0.3081,)) ])
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('drive/My Drive/public/data', train=True,
                   download=True, transform=transform),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('drive/My Drive/public/data', train=False,
                   transform=transform),
    batch_size=test_batch_size, shuffle=True, **kwargs)
```

check: data loader

```
for batch, (data, target) in enumerate(train_loader):
  print(data)
  print(target)
  break
```

```
tensor([[[[-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          ...,
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242]]],


        [[[-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          ...,
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242]]],


        [[[-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          ...,
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242],
          [-0.4242, -0.4242, -0.4242,  ..., -0.4242, -0.4242, -0.4242]]],


        ...,

tensor([9, 3, 0, 1, 2, 4, 6, 8, 1, 1, 2, 3, 0, 7, 0, 0, 1, 3, 9, 4, 1, 1, 2, 9,
        0, 0, 1, 4, 6, 2, 1, 3])
```

code: main

```
model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)

for epoch in range(1, epochs + 1):
    train(model, device, train_loader, optimizer, epoch, log_interval)
    test(model, device, test_loader)

if (save_model):
    if not os.path.exists('drive/My Drive/public/results'):
        os.mkdir('drive/My Drive/public/results')
    torch.save(model,"drive/My Drive/public/results/mnist_nn.pth")
```

results

```
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.301313
Train Epoch: 1 [3200/60000 (5%)]        Loss: 1.915514
Train Epoch: 1 [6400/60000 (11%)]       Loss: 1.429157
Train Epoch: 1 [9600/60000 (16%)]       Loss: 0.774661
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.816357
Train Epoch: 1 [16000/60000 (27%)]      Loss: 0.586290
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.472406
Train Epoch: 1 [22400/60000 (37%)]      Loss: 0.457613
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.525044
Train Epoch: 1 [28800/60000 (48%)]      Loss: 0.478475
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.265142
Train Epoch: 1 [35200/60000 (59%)]      Loss: 0.181658
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.292922
Train Epoch: 1 [41600/60000 (69%)]      Loss: 0.211006
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.447125
Train Epoch: 1 [48000/60000 (80%)]      Loss: 0.295472
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.416876
Train Epoch: 1 [54400/60000 (91%)]      Loss: 0.369973
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.389900

Test: Average loss: 0.3018, Accuracy: 9148/10000 (91%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 0.404442
Train Epoch: 2 [3200/60000 (5%)]        Loss: 0.237755
Train Epoch: 2 [6400/60000 (11%)]       Loss: 0.178738
Train Epoch: 2 [9600/60000 (16%)]       Loss: 0.338589
Train Epoch: 2 [12800/60000 (21%)]      Loss: 0.660104
```

check: save/load model

```
torch.save(model,"drive/My Drive/public/results/mnist_nn.pth")
```

```
!ls "drive/My Drive/public/results"
```

```
cifar10_model.pth  mnist_nn.pth  ptb_trained_model_best.pth
```

```
load_model = torch.load("drive/My Drive/public/results/mnist_nn.pth")
```

practice: with real images

Put image files to test in your google drive

drive/My Drive/public/data/mnist_test_images/

```
!ls 'drive/My Drive/public/data/mnist_test_images/'
```

```
test0.jpg   test13.jpg  test17.jpg  test20.jpg  test5.jpg  test9.jpg
test10.jpg  test14.jpg  test18.jpg  test2.jpg   test6.jpg
test11.jpg  test15.jpg  test19.jpg  test3.jpg   test7.jpg
test12.jpg  test16.jpg  test1.jpg   test4.jpg   test8.jpg
```

practice: with one image

```
[7]  load_model = torch.load("drive/My Drive/public/results/mnist_cnn.pth")
```

```
from skimage import io

img_name = 'drive/My Drive/public/data/mnist_test_images/test0.jpg'
test_img = io.imread(img_name).reshape(28,28)
test_data = transform(test_img).view(1,1,28,28).to(device)
with torch.no_grad():
  output=load_model(test_data)
print(img_name, output.argmax(dim=1).cpu().numpy()[0])
```

```
drive/My Drive/public/data/mnist_test_images/test0.jpg 5
```

practice: with a directory

```
load_model = torch.load("drive/My Drive/public/results/mnist_cnn.pth")
```

```
from skimage import io
import glob
file_list = glob.glob("drive/My Drive/public/data/mnist_test_images/*.jpg")
for img_name in file_list:
  #img_name = 'drive/My Drive/public/data/mnist_test_images/test0.jpg'
  test_img = io.imread(img_name).reshape(28,28)
  test_data = transform(test_img).view(1,1,28,28).to(device)
  with torch.no_grad():
    output=load_model(test_data)
  print(img_name, output.argmax(dim=1).cpu().numpy()[0])
```

```
drive/My Drive/public/data/mnist_test_images/test3.jpg 1
drive/My Drive/public/data/mnist_test_images/test8.jpg 1
drive/My Drive/public/data/mnist_test_images/test9.jpg 4
drive/My Drive/public/data/mnist_test_images/test2.jpg 4
drive/My Drive/public/data/mnist_test_images/test0.jpg 5
drive/My Drive/public/data/mnist_test_images/test6.jpg 1
drive/My Drive/public/data/mnist_test_images/test4.jpg 9
drive/My Drive/public/data/mnist_test_images/test7.jpg 3
drive/My Drive/public/data/mnist_test_images/test5.jpg 2
drive/My Drive/public/data/mnist_test_images/test1.jpg 0
drive/My Drive/public/data/mnist_test_images/test10.jpg 3
```

Machine Intelligence Lab
https://milab.handong.edu/

Handong Global University