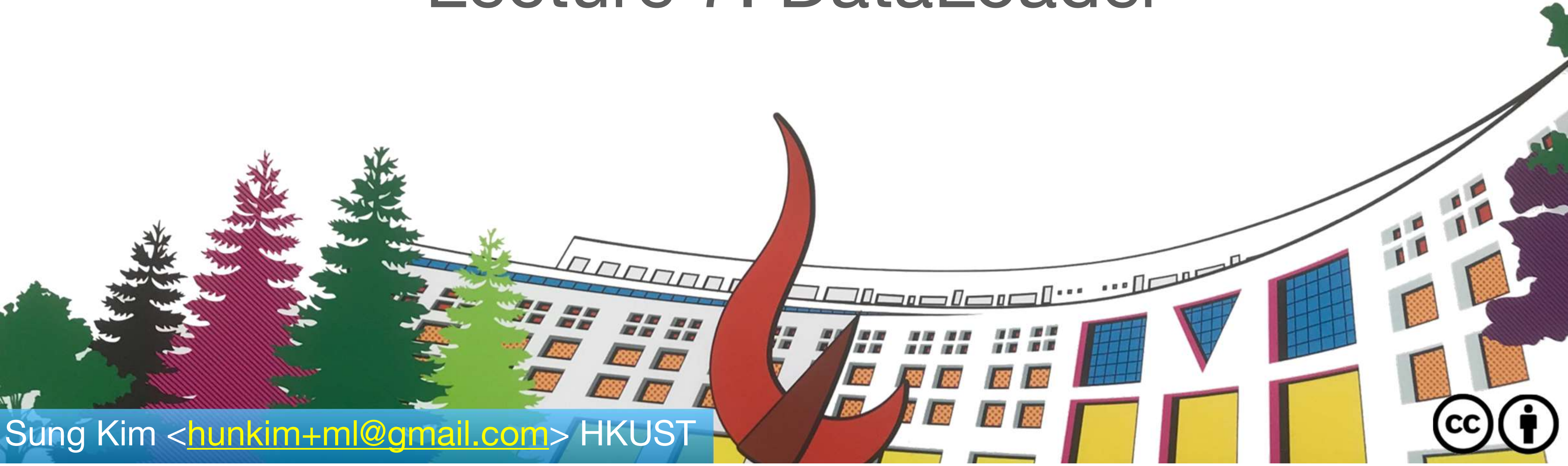
Machine Learning

Lecture 7: DataLoader



Sung Kim <hunkim+ml@gmail.com> HKUST

Recall

```
import torch
from torch.autograd import Variable

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = Variable(torch.Tensor([1.0]),  requires_grad=True)  # Any random value

# our model forward pass
def forward(x):
   return x * w

# Loss function
def loss(x, y):
   y_pred = forward(x)
   return (y_pred - y) * (y_pred - y)

# Before training
print("predict (before training)",  4, forward(4).data[0])
```

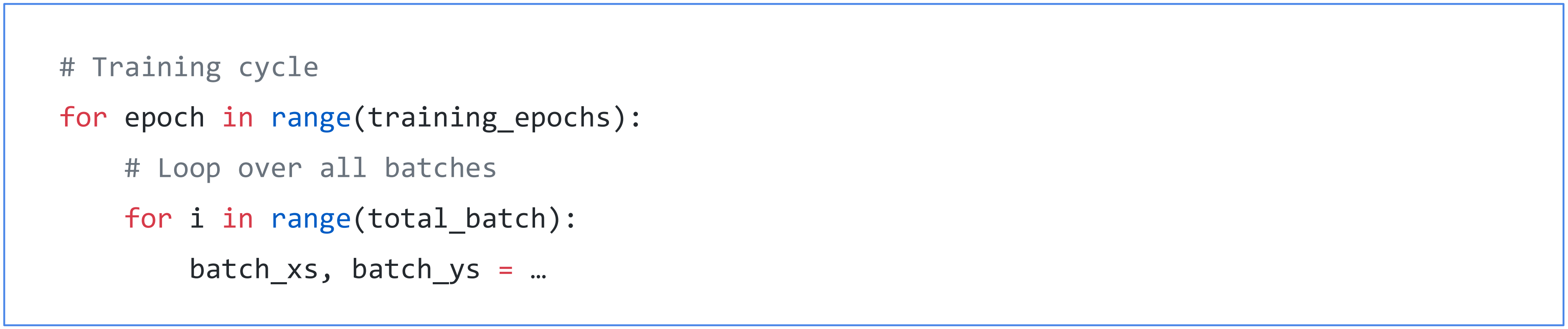
Recall

```
# Training loop
for epoch in range(10):
   for x_val, y_val in zip(x_data, y_data):
       l = loss(x_val, y_val)
       l.backward()
       print("\tgrad: ", x_val, y_val, w.grad.data[0])
       w.data = w.data - 0.01 * w.grad.data

       # Manually zero the gradients after updating weights
       w.grad.data.zero_()

   print("progress:", epoch, l.data[0])

# After training
print("predict (after training)",  4, forward(4).data[0])
```

Manual data feed

```
xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = Variable(torch.from_numpy(xy[:, 0:-1]))
y_data = Variable(torch.from_numpy(xy[:, [-1]]))

...

# Training loop
for epoch in range(100):
        # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

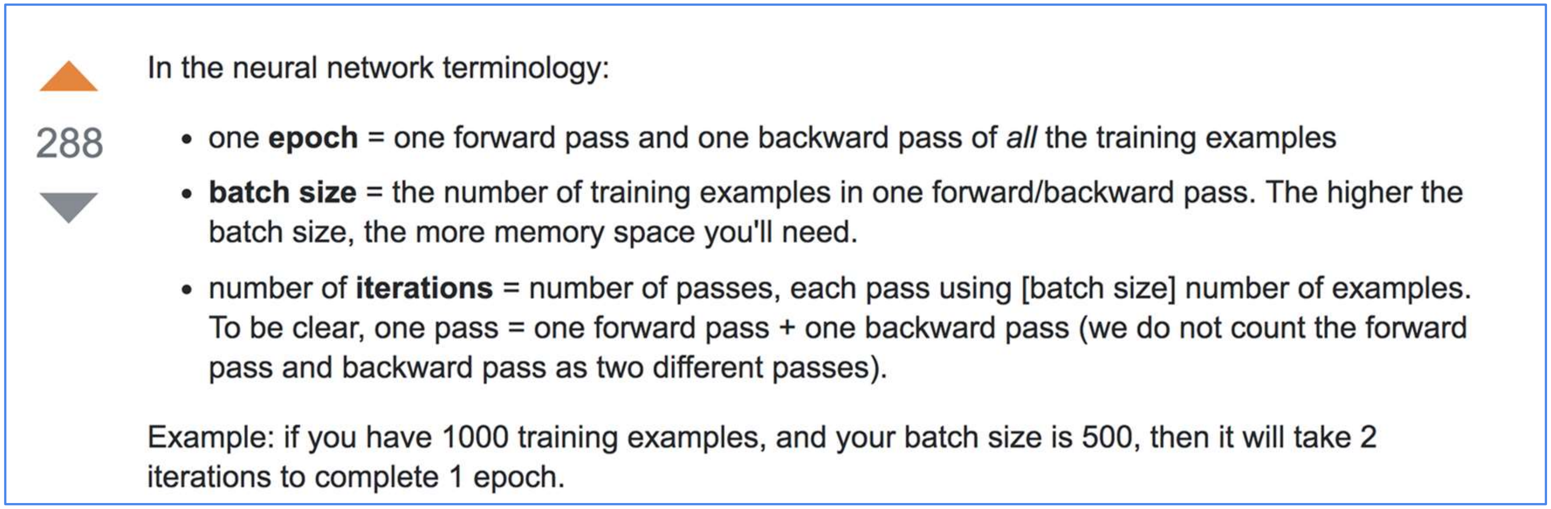
Batch (batch size)



```
# Training cycle
for epoch in range(training_epochs):
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = …
```

Batch (batch size)

```
# Training cycle
for epoch in range(training_epochs):
    # Loop over all batches
    for i in range(total_batch):
```



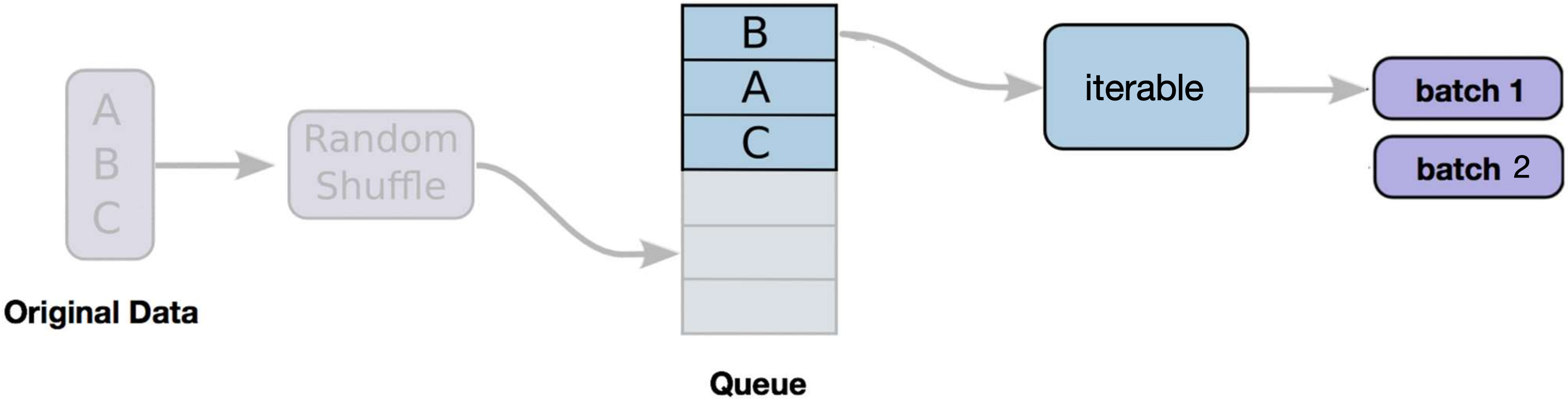
In the neural network terminology:

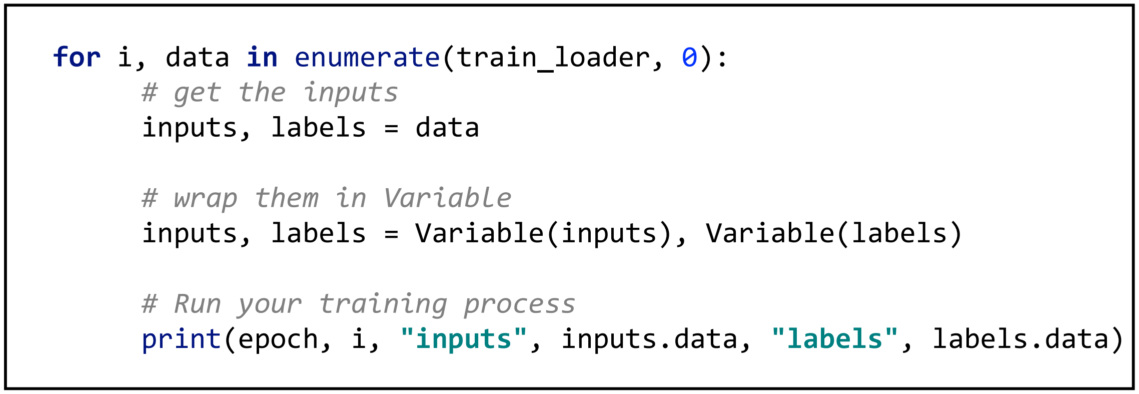
- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

https://stackoverflow.com/questions/4752626

DataLoader





```
for i, data in enumerate(train_loader, 0):
      # get the inputs
      inputs, labels = data

      # wrap them in Variable
      inputs, labels = Variable(inputs), Variable(labels)

      # Run your training process
      print(epoch, i, "inputs", inputs.data, "labels", labels.data)
```

Custom DataLoader

```
class DiabetesDataset(Dataset):
    """ Diabetes dataset."""

    # Initialize your data, download, etc.
    def __init__(self):



    def __getitem__(self, index):
        return

    def __len__(self):
        return


dataset = DiabetesDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=32,
                          shuffle=True,
                          num_workers=2)
```

1 download, read data, etc.

2 return one item on the index

3 return the data length

https://github.com/yunjey/pytorch-tutorial

Custom DataLoader

```python
class DiabetesDataset(Dataset):
    """ Diabetes dataset."""

    # Initialize your data, download, etc.
    def __init__(self):
        xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
        self.len = xy.shape[0]
        self.x_data = torch.from_numpy(xy[:, 0:-1])
        self.y_data = torch.from_numpy(xy[:, [-1]])

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.len


dataset = DiabetesDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=32,
                          shuffle=True,
                          num_workers=2)
```

Using DataLoader

```python
dataset = DiabetesDataset()
train_loader = DataLoader(dataset=dataset, batch_size=32, shuffle=True, num_workers=2)


# Training loop
for epoch in range(2):
    for i, data in enumerate(train_loader, 0):
        # get the inputs
        inputs, labels = data

        # wrap them in Variable
        inputs, labels = Variable(inputs), Variable(labels)

        # Forward pass: Compute predicted y by passing x to the model
        y_pred = model(inputs)

        # Compute and print loss
        loss = criterion(y_pred, labels)
        print(epoch, i, loss.data[0])

        # Zero gradients, perform a backward pass, and update the weights.
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Classifying Diabetes

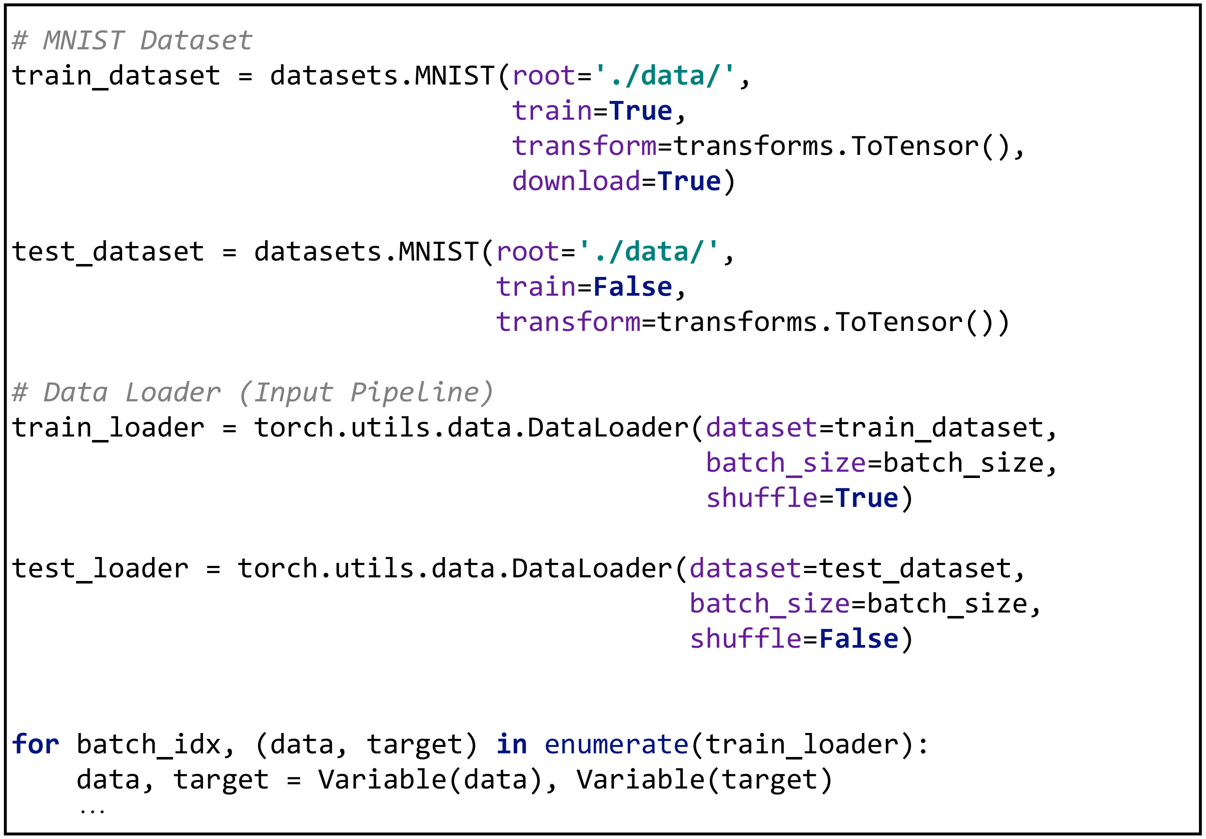
```python
# References
# https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/01-basics/pytorch_basics/main.py
# http://pytorch.org/tutorials/beginner/data_loading_tutorial.html#dataset-class
import torch
import numpy as np
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader


class DiabetesDataset(Dataset):
    """ Diabetes dataset."""

    # Initialize your data, download, etc.
    def __init__(self):
        xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
        self.len = xy.shape[0]
        self.x_data = torch.from_numpy(xy[:, 0:-1])
        self.y_data = torch.from_numpy(xy[:, [-1]])

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.len


dataset = DiabetesDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=32,
                          shuffle=True,
                          num_workers=2)
```

```python
class Model(torch.nn.Module):

    def __init__(self):
        """
        In the constructor we instantiate two nn.Linear module
        """
        super(Model, self).__init__()
        self.l1 = torch.nn.Linear(8, 6)
        self.l2 = torch.nn.Linear(6, 4)
        self.l3 = torch.nn.Linear(4, 1)

        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Variables.
        """
        out1 = self.sigmoid(self.l1(x))
        out2 = self.sigmoid(self.l2(out1))
        y_pred = self.sigmoid(self.l3(out2))
        return y_pred

# our model
model = Model()

# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

# Training loop
for epoch in range(2):
    for i, data in enumerate(train_loader, 0):
        # get the inputs
        inputs, labels = data

        # wrap them in Variable
        inputs, labels = Variable(inputs), Variable(labels)

        # Forward pass: Compute predicted y by passing x to the model
        y_pred = model(inputs)

        # Compute and print loss
        loss = criterion(y_pred, labels)
        print(epoch, i, loss.data[0])

        # Zero gradients, perform a backward pass, and update the weights.
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

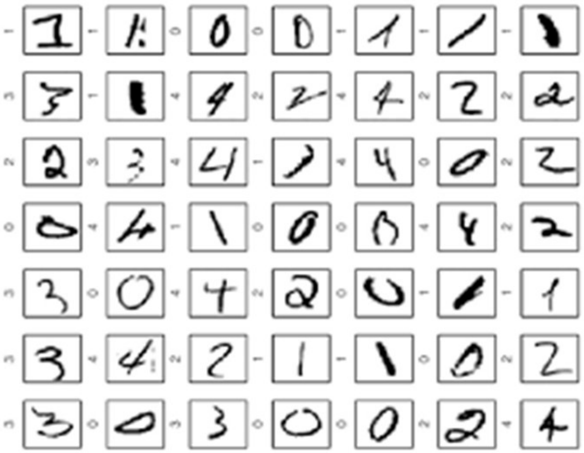
The following dataset loaders are available

- MNIST and FashionMNIST
- COCO (Captioning and Detection)
- LSUN Classification
- ImageFolder
- Imagenet-12
- CIFAR10 and CIFAR100
- STL10
- SVHN
- PhotoTour

MNIST dataset loading



```python
# MNIST Dataset
train_dataset = datasets.MNIST(root='./data/',
                               train=True,
                               transform=transforms.ToTensor(),
                               download=True)

test_dataset = datasets.MNIST(root='./data/',
                              train=False,
                              transform=transforms.ToTensor())

# Data Loader (Input Pipeline)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)


for batch_idx, (data, target) in enumerate(train_loader):
    data, target = Variable(data), Variable(target)
    ...
```



https://github.com/yunjey/pytorch-tutorial

More dataset loaders are available

- MNIST and FashionMNIST
- COCO (Captioning and Detection)
- LSUN Classification
- ImageFolder
- Imagenet-12
- CIFAR10 and CIFAR100
- STL10
- SVHN
- PhotoTour

Exercise 7-1:

- Check out existing data sets (torch.vision)

- Build DataLoader for
  - Titanic dataset: https://www.kaggle.com/c/titanic/download/train.csv
- Build a classifier using the DataLoader

Lecture 8:
Basic CNN