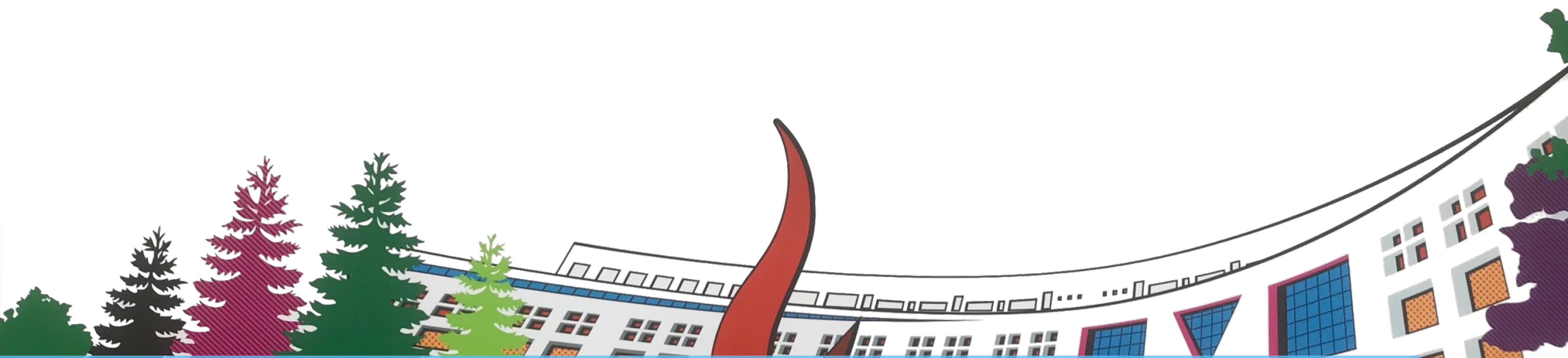
ML/DL for Everyone Season2

MNIST Introduction



Code: https://github.com/deeplearningzerotoall/PyTorch

Slides: http://bit.ly/2VrZcWM

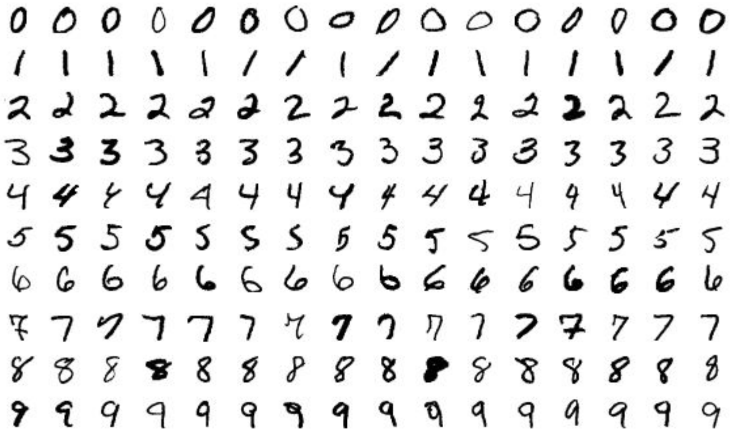
Lecturer: 강현우 znxlwm@gmail.com

MNIST Introduction

- What is MNIST?
- Code: mnist_introduction

What is MNIST?

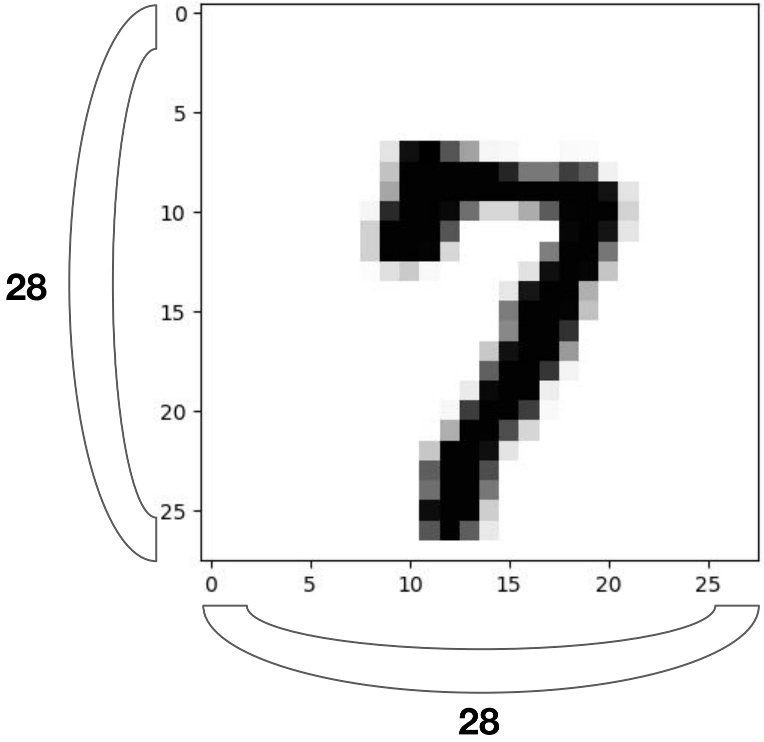
MNIST: handwritten digits dataset



train-images-idx3-ubyte.gz: training set images (9912422 bytes; 60,000 samples)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
T10k-images-idx3-ubyte.gz : test set images (1648877 bytes; 10,000 samples)
T10k-labels-idx1-ubyte.gz : test set labels (4542 bytes)

https://ko.wikipedia.org/wiki/파일:MnistExamples.png
http://yann.lecun.com/exdb/mnist/

Example of MNIST



- 28 x 28 image
- 1 channel gray image
- 0 ~ 9 digits

```python
for X, Y in data_loader:
    # reshape input image into [batch_size by 784]
    # label is not one-hot encoded
    X = X.view(-1, 28 * 28)
```

torchvision

The <u>torchvision</u> package consists of <u>popular datasets</u>, <u>model architectures</u>, and <u>common image transformations</u> for computer vision.

torchvision.datasets
- MNIST
- Fashion-MNIST
- EMNIST
- COCO
- LSUN
- ImageFolder
- DatasetFolder
- Imagenet-12
- CIFAR
- STL10
- SVHN
- PhotoTour
- SBU
- Flickr
- VOC

torchvision.models
- Alexnet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3

torchvision.transforms
- Transforms on PIL Image
- Transforms on torch.*Tensor
- Conversion Transforms
- Generic Transforms
- Functional Transforms

torchvision.utils

https://pytorch.org/docs/stable/torchvision/index.html

Reading data

```
import torchvision.datasets as dsets
...
mnist_train = dsets.MNIST(root="MNIST_data/", train=True, transform=transforms.ToTensor(),
download=True)

mnist_test = dsets.MNIST(root="MNIST_data/", train=False, transform=transforms.ToTensor(),
download=True)

data_loader = torch.utils.DataLoader(DataLoader=mnist_train, batch_size=batch_size,
shuffle=True, drop_last=True)
...
for epoch in range(training_epochs):
...
    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
```

Epoch / Batch size / Iteration

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/epoch-vs-iteration-when-training-neural-networks

Softmax

```python
# MNIST data image of shape 28 * 28 = 784
linear = torch.nn.Linear(784, 10, bias=True).to(device)
# initialization
torch.nn.init.normal_(linear.weight)
# parameters
training_epochs = 15
batch_size = 100
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally
optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)

for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(data_loader)
    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
        optimier.zero_grad()
        hypothesis = linear(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        avg_cost += cost / total_batch
    print("Epoch: ", "%04d" % (epoch+1), "cost =", "{:.9f}".format(avg_cost))
```

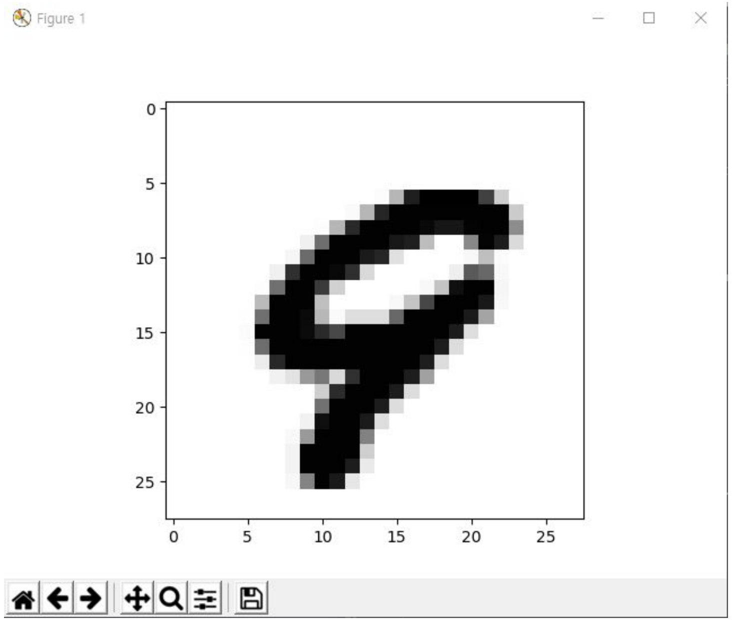
```
Epoch: 0001 cost = 2.511683702
Epoch: 0002 cost = 0.977319956
Epoch: 0003 cost = 0.797017217
Epoch: 0004 cost = 0.710427940
Epoch: 0005 cost = 0.655205429
Epoch: 0006 cost = 0.615207732
Epoch: 0007 cost = 0.584421575
Epoch: 0008 cost = 0.559486568
Epoch: 0009 cost = 0.538655698
Epoch: 0010 cost = 0.520880997
Epoch: 0011 cost = 0.505315244
Epoch: 0012 cost = 0.491431117
Epoch: 0013 cost = 0.479477882
Epoch: 0014 cost = 0.468681127
Epoch: 0015 cost = 0.458788306
Learning finished
Accuracy: 0.8718999624252319
```

Test

```
# Test the model using test sets
With torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = linear(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print("Accuracy: ", accuracy.item())
```

Visualization




```python
import matplotlib.pyplot as plt
import random
...
r = random.randint(0, len(mnist_test) - 1)
X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 *
28).float().to(device)
Y_single_data = mnist_test.test_labels[r:r + 1].to(device)

print("Label: ", Y_single_data.item())
single_prediction = linear(X_single_data)
print("Prediction: ", torch.argmax(single_prediction,
1).item())

plt.imshow(mnist_test.test_data[r:r + 1].view(28, 28),
cmap="Greys", interpolation="nearest")
plt.show()
```

Label: 8
Prediction: 8

What’s Next?

- Perceptron