Deep Learning

```
import pandas as pd
import os
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Data import

```
traindata_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/889649d1bc273bf53967
testdata_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/889649d1bc273bf53967c
train_data = pd.read_csv(traindata_url)
test_data = pd.read_csv(testdata_url)
```

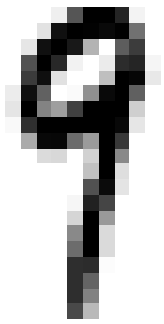
데이터 확인

```
def plot_digit(data):
    image = data.reshape(28, 28) # 1d vector를 28*28 형태로 변경
    plt.imshow(image, cmap = matplotlib.cm.binary,
               interpolation="nearest")
    plt.axis("off")
```

```
train_data
```

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	1x10	1x11	1x12	1x13	1
0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	9	0	0	0	0	0	0	0	0	0	0	0	0	0	

```
index = 600
plot_digit(train_data.values[index, 1:])
plt.show()
print('label: ', train_data.values[index, 0])
```



```
label:  9
```

```

def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = matplotlib.cm.binary,
               interpolation="nearest")
    plt.axis("off")
```

```
# 숫자 그림을 위한 추가 함수
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    size,size) for instance in instances]
    1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = matplotlib.cm.binary, **options)
    plt.axis("off")
```

저장 중...

```
plt.figure(figsize=(9,9))
example_images = train_data.values[:60000:600, 1:]
plot_digits(example_images, images_per_row=10)
plt.show()
```

Convert to 05 data

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

저장 중...

```
train_data.label = le.transform(train_data.label == 5)
test_data.label = le.transform(test_data.label == 5)
```

Deep learning - classification 모델

Pytorch 모델에 입력하기 위한 데이터 변환

```
train_data = torch.from_numpy(train_data.values).float()
test_data = torch.from_numpy(test_data.values).float()
```

```
BATCH_SIZE = 15
epochs = 2
learning_rate = 0.001
```

```

data_loader = torch.utils.data.DataLoader(train_data,
                                          batch_size=BATCH_SIZE,
                                          shuffle=True,
                                          num_workers=0)
```

Deep learning 모델 정의

```
class DNNModel(nn.Module):
    def __init__(self):
        super(DNNModel, self).__init__()
        self.layer1 = nn.Linear(28 *28, 300)
        self.layer2 = nn.Linear(300, 2)
        self.relu = nn.ReLU()


    def forward(self, x):

        layers = nn.Sequential(self.layer1,
                               self.relu,
                               self.layer2,
                               self.relu
                               ).to(device)
        out = layers(x)
        return out

model = DNNModel()
model
```



```
, out_features=300, bias=True)
  (layer2): Linear(in_features=300, out_features=2, bias=True)
  (relu): ReLU()
)
```

```
class DNNModel(nn.Module):
    def __init__(self):
        super(DNNModel, self).__init__()
        self.layer1 = nn.Linear(28 *28, 300)
        self.layer2 = nn.Linear(300, 100)
        self.layer3 = nn.Linear(100, 2)
```

```
        self.relu = nn.ReLU()


    def forward(self, x):

        layers = nn.Sequential(self.layer1,
                               self.relu,
                               self.layer2,
                               self.relu,
                               self.layer3).to(device)
        out = layers(x)
        return out

model = DNNModel()
model
```

```
DNNModel(
  (layer1): Linear(in_features=784, out_features=300, bias=True)
  (layer2): Linear(in_features=300, out_features=100, bias=True)
  (layer3): Linear(in_features=100, out_features=2, bias=True)
  (relu): ReLU()
)
```

학습 시작

GPU로 넘겨야 하는것

- model의 layer
- cost(criterion)
- data

torch.nn.CrossEntropyLoss() 함수에 대해

원래 cross entropy는

```
from IPython.display import Image
image_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/99785e9d01523e8bb6bf78d1
Image(image_url)
```


$$D(S,L) = \boxed{-\sum_i L_i \log(S_i)}$$

pytorh.nn.CrossEntropyLoss() 는

```
from IPython.display import Image
image_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/99785e9d01523e8bb6bf78d1
Image(image_url)
```

$$loss(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j]$$

softmax와 label의 elemental wise 곱을하면 label이 0인 확률들은 모두 없어지게 되어 아래 수식이 가능

따라서 class에는 one-hot encoding 되지 않은 값이 들어가야 함

x에는 soft max를 거치지 않은 벡터가 들어감(CrossEntropyLoss()에 softmax가 포함)

```
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.Adam(model.parameters(), lr = learning_rate)
```

```
for epoch in range(epochs):
    running_cost = 0.0

    for step, (batch_data) in enumerate(data_loader):
        batch_x = batch_data[:, 1:].view(-1, 28*28).to(device)
        batch_y = batch_data[:, 0].to(device).long()

        optimizer.zero_grad()

        outputs = model(batch_x)
        cost = criterion(outputs, batch_y)

        cost.backward()
        optimizer.step()

        running_cost += cost.item()
        if step % 200 == 199:
            print('[%d, %5d] cost: %.3f' % (epoch + 1, step + 1, running_cost
            running_cost = 0.0

```

저장 중...

```
[1,   600] cost: 0.070
[1,   800] cost: 0.058
[1,  1000] cost: 0.061
[1,  1200] cost: 0.056
[1,  1400] cost: 0.076
[1,  1600] cost: 0.050
[1,  1800] cost: 0.054
[1,  2000] cost: 0.038
[1,  2200] cost: 0.047
[1,  2400] cost: 0.047
[1,  2600] cost: 0.046
```

```
[1,  2800] cost: 0.052
[1,  3000] cost: 0.042
[1,  3200] cost: 0.050
[1,  3400] cost: 0.035
[1,  3600] cost: 0.051
[1,  3800] cost: 0.044
[1,  4000] cost: 0.025
[2,   200] cost: 0.044
[2,   400] cost: 0.028
[2,   600] cost: 0.041
[2,   800] cost: 0.037
[2,  1000] cost: 0.035
[2,  1200] cost: 0.021
[2,  1400] cost: 0.033
[2,  1600] cost: 0.035
[2,  1800] cost: 0.068
[2,  2000] cost: 0.032
[2,  2200] cost: 0.033
[2,  2400] cost: 0.030
[2,  2600] cost: 0.035
[2,  2800] cost: 0.024
[2,  3000] cost: 0.031
[2,  3200] cost: 0.033
[2,  3400] cost: 0.028
[2,  3600] cost: 0.031
[2,  3800] cost: 0.034
[2,  4000] cost: 0.019
```

정확도 판단

Confusion matrix

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
```

```
with torch.no_grad():
    X_test = test_data[:, 1:].view(-1, 28 * 28).float().to(device)
    y_test = test_data[:, 0].float()

    prediction = model(X_test).cpu()
                                 st, torch.argmax(prediction, 1)))

    print(precision_score(y_test, torch.argmax(prediction, 1), average=None))
    print(precision_score(y_test, torch.argmax(prediction, 1), average='weight
    print("Recall")
    print(recall_score(y_test, torch.argmax(prediction, 1), average=None))
    print(recall_score(y_test, torch.argmax(prediction, 1), average='weighted'
```

저장 중...

```
[[9050   58]
 [  32  860]]
==Precision==
[0.99647655 0.93681917]
```

```
0.9911551091747473
Recall
[0.99363197 0.96412556]
0.991
```

```
1
```

✓ 0초 오전 11:41에 완료됨

저장 중...