Term Project

2019028313 / Kim Hyewon

MNIST dataset

28 * 28 크기의 흑백 이미지로 이루어진 dataset 으로 총 60000 만개의 data 가 있다. 미니배치의 사이즈를 12 로 설정하여 5000 개의 mini-batch 로 나누어 학습에 사용하였습니다.

- 5000 * 12 * (28 * 28 * 1)

Deep Neural Network

```
# Using ModuleList

class DeepNN_ML(nn.Module):
    def __init__(self):
        super(DeepNN_ML, self).__init__()

        self.in_dim = 28 * 28 # MNIST
        self.out_dim = 10 # 0 ~ 9
        self.epochs = epoch_num
        self.az = [ [] for i in range(10)] #a0,z1,a1,z2,a2,z3,a3,z4,z4.z5

        self.list = nn.ModuleList([
            nn.Linear(in_features = self.in_dim, out_features = 512),
            nn.ReLU(),
            nn.Linear(in_features = 512, out_features = 256),
            nn.ReLU(),
            nn.Linear(in_features = 256, out_features = 128),
            nn.ReLU(),
            nn.Linear(in_features = 128, out_features = 64),
            nn.ReLU(),
            nn.Linear(in_features = 64, out_features = self.out_dim)
        ])


    def forward(self, x):
        fw = x.view(-1, self.in_dim)
        if self.epochs > 1:
            for i, module in enumerate(self.list):
                fw = module(fw)
        else: # 마지막 epoch 에서 a[0].z[1]~z[5]를 self.az 에 저장한다
            self.az[0].append(fw.tolist())
            for i, module in enumerate(self.list):
                fw = module(fw)
                self.az[i+1].append(fw.tolist())
        return fw
```

DNN은 ModuleList로 이루어 forward과정에서 moduleList에 속한 module들을 하나씩 실행한다.

moduleList를 사용하여 sequential일 때와 달리 module의 실행 사이에 추가적인 작업이 가능하기 때문에 hidden layer의 정보를 저장할 수 있다. 마지막 epoch에서 (즉, 학습할 mini-batch가 5000개 이하로 남은 경우) tensor로 이루어진 input image(a[0])와 hidden vector(z[1], a[1], z[2], a[2])의 정보를 list로 전환하여 self.az 배열에 담는다. (tensor를 유지하고 cat을 사용하여 데이터들을 이어 붙이는 방법은 느리기 때문에 list로 변환하여 append로 data들을 모은 후 reshape, numpy를 통해 원하는 data형식으로 맞추어 주었습니다.)

```
label_list = np.array(label_list).reshape(-1,60000)[0] # (60000,784)
for i in range(10):
    model_ML.az[i] = np.array(model_ML.az[i])
    model_ML.az[i] = model_ML.az[i].reshape(60000,-1) # (60000,784),(60000,512) ~ (60000, 10)
```

label_list	a[0]	z[1]	a[1]	z[2]	a[2]
(60000,)	(60000, 784)	(60000, 512)	(60000, 512)	(60000, 256)	(60000, 256)

Compression & Visualization

```
class visualization():
    def __init__(self, Y, N):
        self.Y = Y
        self.N = N
        self.pca = PCA(n_components=2)
        self.tsne = TSNE(n_components=2, verbose=1, perplexity = 40, n_iter=300)
        self.df = [[] for i in range(10)]
        self.df_subset = [[] for i in range(10)]

        np.random.seed(42)
        self.rndperm = [[] for i in range(10)]

    def Pca(self,i,fc):
        pca_result = self.pca.fit_transform(self.df_subset[i][fc].values)
        self.df_subset[i]['pca-one'] = pca_result[:,0]
        self.df_subset[i]['pca-two'] = pca_result[:,1]

    def Tsne(self,i,fc):
        time_start = time.time()
        tsne_result = self.tsne.fit_transform(self.df_subset[i][fc].values)
        self.df_subset[i]['tsne-2d-one'] = tsne_result[:,0]
        self.df_subset[i]['tsne-2d-two'] = tsne_result[:,1]
        print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_start))

    def compression(self, X, i):
        feat_cols = [f'pixel{i}' for i in range (X.shape[1])]
        self.df[i] = pd.DataFrame(X, columns = feat_cols)
        self.df[i]["y"] = self.Y
        self.rndperm[i] = np.random.permutation(self.df[i].shape[0])
        self.df_subset[i] = self.df[i].loc[self.rndperm[i][:self.N],:].copy() # make subset
        self.Pca(i,feat_cols)
        self.Tsne(i,feat_cols)

    def draw(self, i, name):
        plt.figure(figsize=(26,9))
        plt.subplot(1,2,1)
        plt.title("PCA:" + name)
        sns.scatterplot(
            x="pca-one", y="pca-two",
            hue="y",
            palette=sns.color_palette("hls",10), # hue(=y)의 가지수
            data= self.df_subset[i],
            alpha=0.3
        )
        plt.subplot(1, 2, 2)
        plt.title("t-SNE:" + name)
        sns.scatterplot(
            x="tsne-2d-one", y="tsne-2d-two",
            hue="y",
            palette=sns.color_palette("hls",10),
            data= self.df_subset[i],
            alpha=0.3
        )
        plt.show()
```

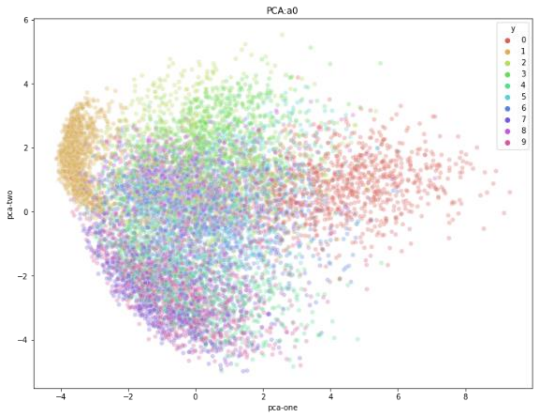
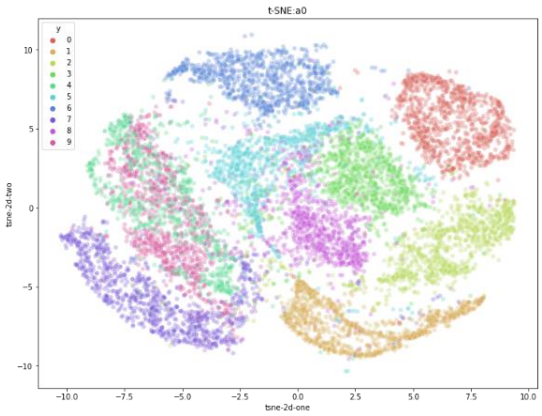
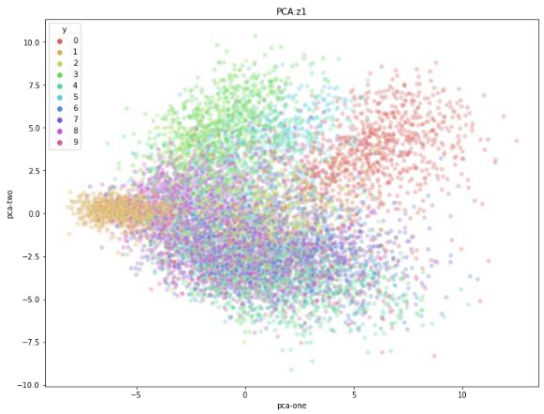
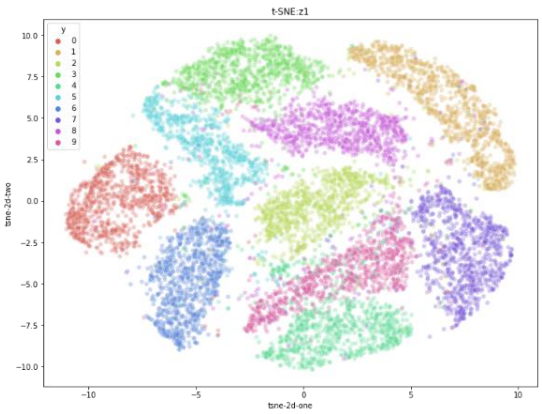
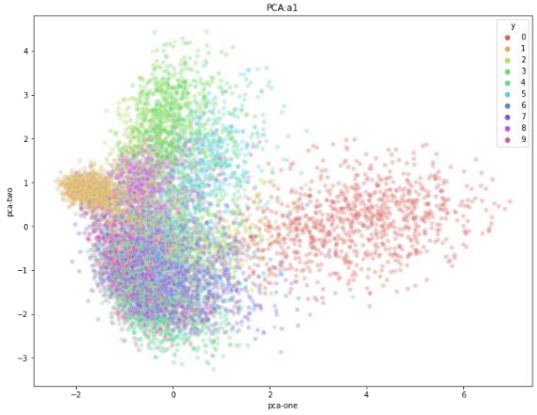
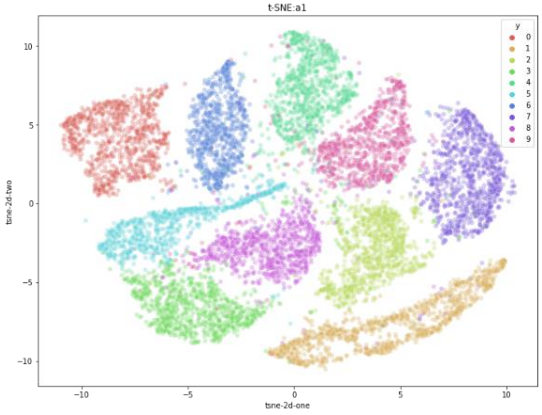
PCA, t-SNE를 통한 data compression과 compression결과를 plt를 통해 시각화 하는 과정을 visualization class로 구현하였습니다.

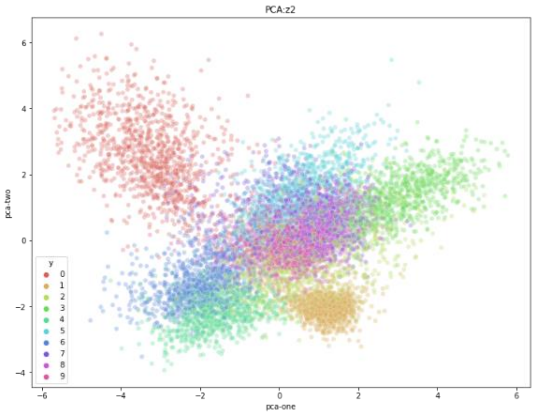
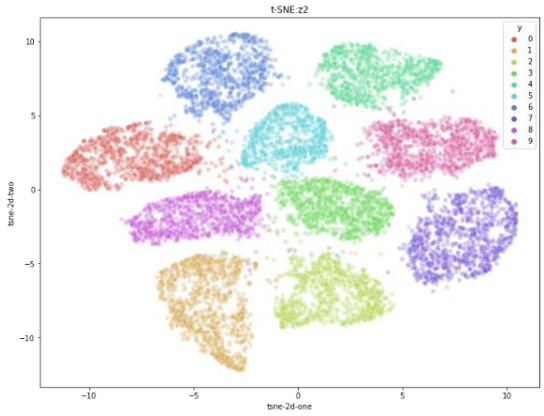
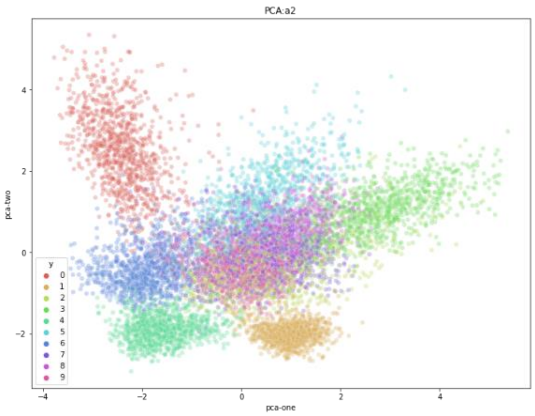
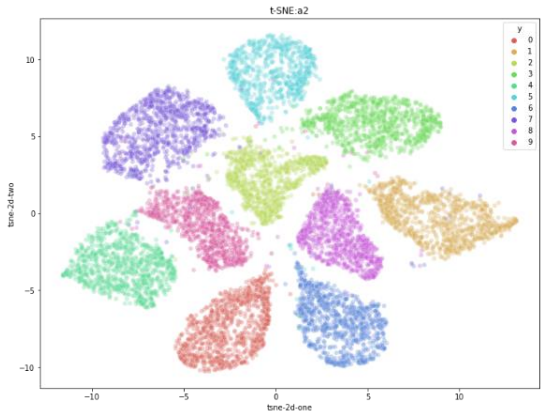
1. init과 compression은 학습 과정에 저장한 input image, hidden vector data와 label정보를 받아 pandas를 통해 변형하여 self.df에 저장합니다. 이중 10000개의 행을 random으로 추출하여 subset을 만들고 Pca, Tsne함수를 실행합니다.

2. Pca, Tsne함수는 subset에 속한 value들을 PCA, t-SNE를 통해 compression하여 결과를 self.df_subset에 저장합니다.

3. draw 함수는 self.df_subset에 담긴 정보를 통해 PCA, t-SNE를 통한 compression 결과를 시각화합니다.

RESULT

	PCA	t-SNE
a[0]		
z[1]		
a[1]		

z[2]	 
a[2]	 

z[3]~a[4]의 visualization 결과는 보고서의 양을 고려하여 이미지를 생략하였습니다.

DNN의 학습은 batch_size: 12/ learning_rate: 0.01/ epoch_num: 10으로 설정했을 때 loss값이 0.02까지 떨어지는 것을 확인할 수 있었습니다. (즉, DNN을 통한 class 분류의 정확도가 매우 높습니다.)

input image와 hidden vector를 t-SNE로 compression한 결과를 확인해 보면 정차적으로 클래스의 분류가 분명해지는 것을 확인할 수 있습니다. 이를 통해 DNN이 잘 학습된 것을 확인할 수 있을 뿐 아니라 input data가 hidden layer를 거치며 점점 classification되는 과정을 확인할 수 있습니다.

PCA를 통한 compression은 t-SNE와 같은 data를 입력으로 받지만 시각화결과 class분류가 제대로 보이지 않는 것을 확인할 수 있습니다. (input data뿐 아니라 a[2]에서도 명확한 class의 분류를 확인할 수 없다.) 이는 PCA가 분산이 최대인 축으로 데이터를 투영하는 방법으로 원 데이터의 분산을 보존하는 목적으로 compression하기 때문에 이 과정에서 군집화 된 데이터들이 뭉개지기 때문입니다.

이와 달리 t-SNE는 고차원 데이터의 유사성을 유지하여 compression하는 방법으로 기준점과 다른 데이터 사이의 거리와 t distibution을 이용하여 유사한 데이터를 묶어주는 방법이다. 때문에 위의 결과에서 clustering을 확인할 수 있다. 그러나 t-SNE의 실행시간을 확인해보면 시간이 오래 걸리며 돌릴 때마다 다른 시각화 결과가 반환되는 것 또한 확인할 수 있습니다.

PCA : 고차원의 data를 compression하는 경우 분산을 보존되나 cluster들이 겹치거나 뭉개지는 현상이 확인된다.

t-SNE : 고차원 데이터를 compression하여도 cluster를 확인할 수있다. 그러나, PCA에 비해 시간이 매우 오래 걸리며 돌릴 때마다 시각화의 결과가 달라지는 것을 확인할 수 있다.