Term Project

2017012251 윤영훈

기본적으로 강의시간에 제공된 DNN 코드를 기반으로 일부 수정하여 프로젝트 코드를 완성했다.

- **변경사항 : model**

Model

In [5]:
```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()

        self.in_dim = 28*28 # MNIST
        self.out_dim = 10

        self.fc1 = nn.Linear(self.in_dim, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, self.out_dim)

        self.relu = nn.ReLU()
        self.log_softmax = nn.LogSoftmax()

    def forward(self, x):
        z1 = self.fc1(x.view(-1, self.in_dim))
        a1 = self.relu(z1)
        z2 = self.fc2(a1)
        a2 = self.relu(z2)
        a3 = self.relu(self.fc3(a2))
        a4 = self.relu(self.fc4(a3))
        logit = self.fc5(a4)
        return x, z1, a1, z2, a2, logit
```

- forward part에서 z1, z2를 사용하기 위해 a1, a2에서 코드를 분리했으며 최종적으로 model이 x, z1, a1, z2, a2, logit을 모두 retrun하도록 코드를 변경했다.

- **변경사항 : train**

In [8]:
```
a0 = []
z1 = []
a1 = []
z2 = []
a2 = []
l = []

for epoch in range(10):
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        model.zero_grad()

        #forward + backward + optimize
        outputs = model(inputs)

        if (epoch == 9):
            for j in range(batch_size):
                a0.append(outputs[0][j].detach().numpy().ravel())
                z1.append(outputs[1][j].detach().numpy().ravel())
                a1.append(outputs[2][j].detach().numpy().ravel())
                z2.append(outputs[3][j].detach().numpy().ravel())
                a2.append(outputs[4][j].detach().numpy().ravel())
                l.append(labels[j].detach().numpy())

        loss = criterion(outputs[5], labels)
        loss.backward()
        optimizer.step()

        #print statistics
        running_loss += loss.item()
        if (i+1) % 2000 == 0:
            print('[%d, %5d] loss : %.3f' % (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Traning')
```

- 마지막 epoch에서 데이터를 추출한다.
- 이때 a0 (= x), z1, a1, z2, a2를 각각 return받고 저장하기위해 empty list 를 미리 선언해둔다.
- labels도 저장하기 위해 empty list l 또한 선언한다.
- 추출한 데이터를 numpy화 한다. 이때 (60000, 784), (60000, 512) 등 지정된 양식에 맞추기 위해 각 데이터를 ravel()을 통해 flatten한 뒤 list에 append한다.

In [9]:
```
a0 = np.array(a0)
z1 = np.array(z1)
a1 = np.array(a1)
z2 = np.array(z2)
a2 = np.array(a2)
l = np.array(l)
```

In [10]:
```
df_a0 = pd.DataFrame(a0)
df_a0['y'] = l

df_z1 = pd.DataFrame(z1)
df_z1['y'] = l

df_a1 = pd.DataFrame(a1)
df_a1['y'] = l

df_z2 = pd.DataFrame(z2)
df_z2['y'] = l

df_a2 = pd.DataFrame(a2)
df_a2['y'] = l
```

- 완성된 list를 numpy화 한다.
- 이후 각 layer에 해당하는 DataFrame을 선언한다.

In [11]:
```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(df_a0[range(784)].values)

df_a0['pca-one'] = pca_result[:,0]
df_a0['pca-two'] = pca_result[:,1]

plt.figure(figsize=(13,9))
sns.scatterplot(
    x="pca-one", y="pca-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_a0,
    legend="full",
    alpha=0.3
)
plt.show()
```

PCA for a0

- PCA 분석을 실행하고 scatterplot을 이용해 plot graph를 출력한다.
- layer에 따라 반복적으로 실행해준다.
- 이때 pca_result = pca.fit_transform(df_a0[range(784)].values) 에서 각 layer에 맞춰 DataFrame을 변경한다.
- scatterplot에서도 DataFrame을 각 layer에 맞춰 변경한다.

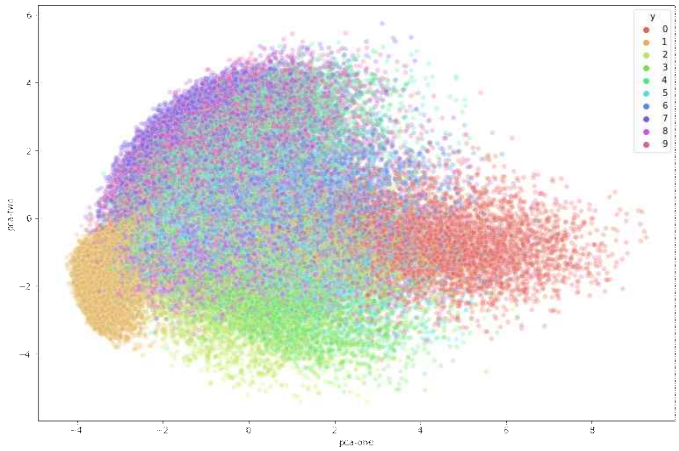
In [16]:
```
time_start = time.time()
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(df_a0[range(784)].values)
df_a0['tsne-2d-one'] = tsne_results[:,0]
df_a0['tsne-2d-two'] = tsne_results[:,1]
print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_start))

plt.figure(figsize=(21,21))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_a0,
    legend="full",
    alpha=0.3
)
plt.show()
```

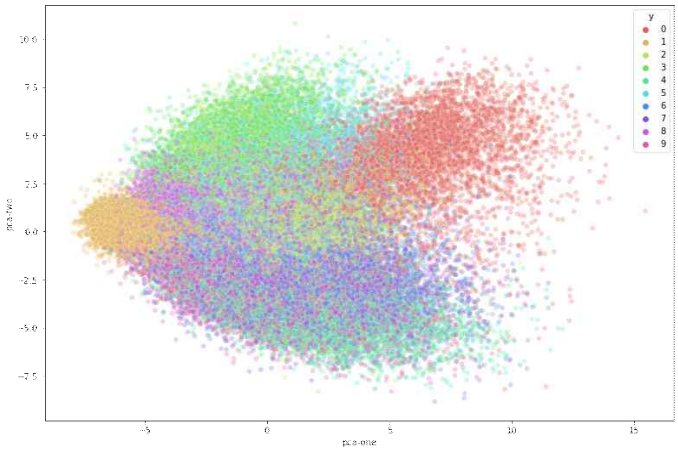
T-SNE for a0

- T-SNE 분석을 실행하고 scatterplot을 이용해 plot graph를 출력한다.
- 전체 데이터 (N=60000) 에 대해 분석을 실행하므로 subnet 처리 하지 않음.
- layer에 따라 반복적으로 실행해준다.
- 이때 tsne_results = tsne.fit_transform(df_a0[range(784)].values) 에서 각 layer에 맞춰 DataFrame을 변경한다.
- scatterplot에서도 DataFrame을 각 layer에 맞춰 변경한다.

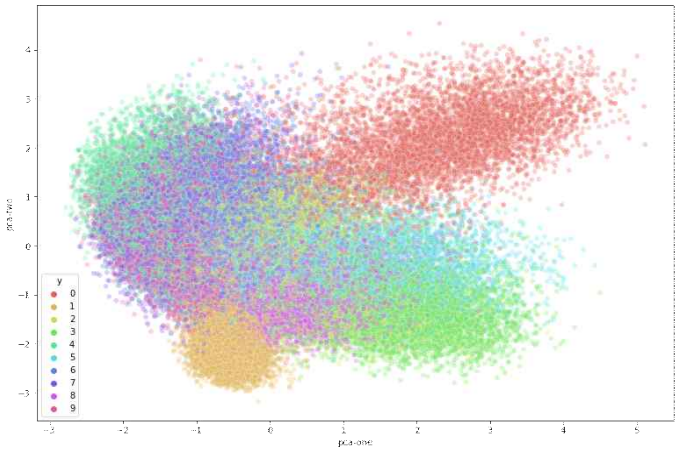
- PCA Plot Graph for a0, z1, a1, z2, a2



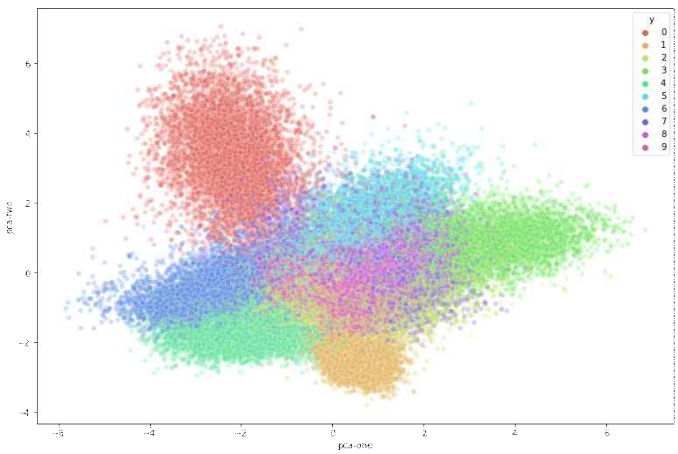
a0



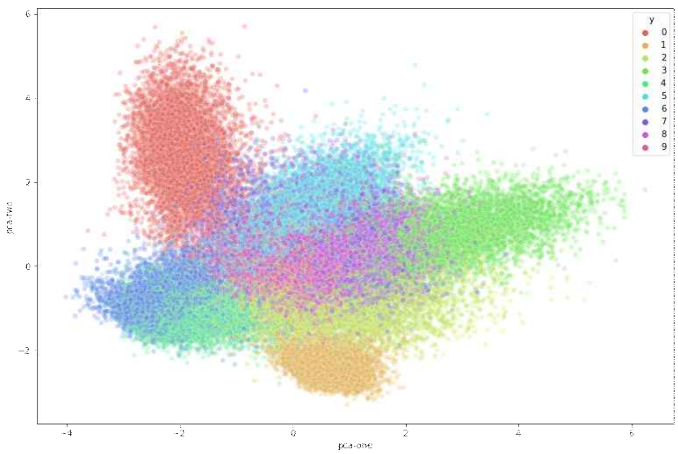
z1



a1



z2

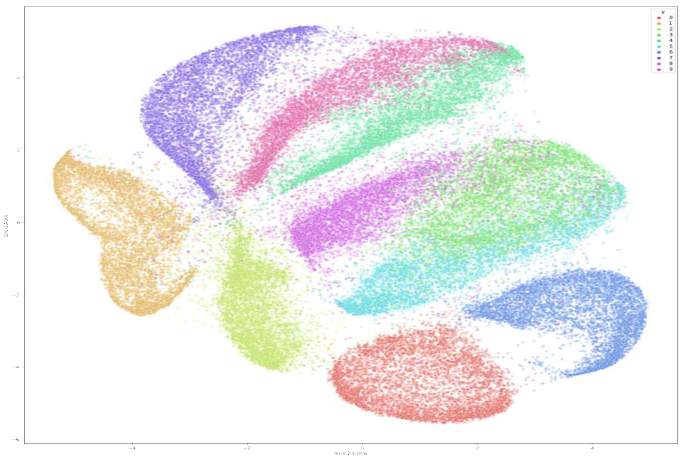


a2

- PCA 결과분석

- T-SNE에 비해 매우 짧은 시간에 분석 완료. (각 분석 당 평균 5초 소모 / T-SNE의 경우 각 분석 당 평균 180초 소모)
- first layer (z1, a1)에서는 전체적으로 데이터가 서로 섞여 구분하기에 어려움이 있음.
- second layer의 z1부터 다수의 데이터가 육안으로 구분할 수 있을정도로 구분됨.
- 그러나 일부 데이터의 경우 여전히 서로 섞여있어 육안으로 구분이 불가능.

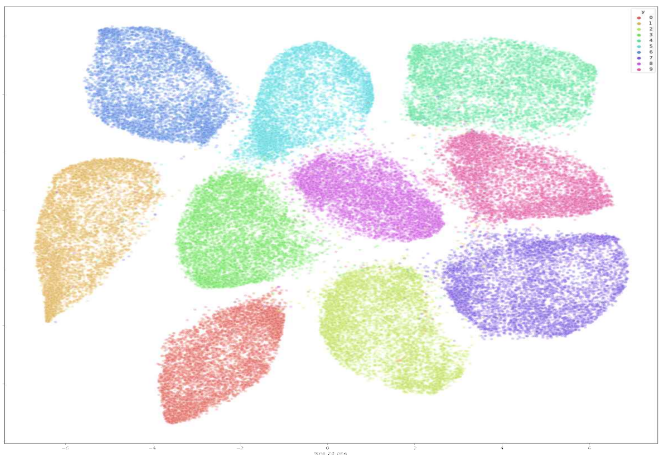
- T-SNE Plot Graph for a0, z1, a1, z2, a2



a0

z1

a1



z2

a2

- T-SNE 결과분석

- first layer (z1, a1)부터 대부분의 데이터가 육안으로 쉽게 구분 가능함.
- second layer (z2, a2)는 겹치는 데이터가 거의 없을 정도로 확연히 구분됨.
- 그러나 PCA에 비해 분석 시간이 매우 오래걸림. (각 분석 당 평균 180초 소모 / PCA의 경우 각 분석 당 평균 5초 소모)