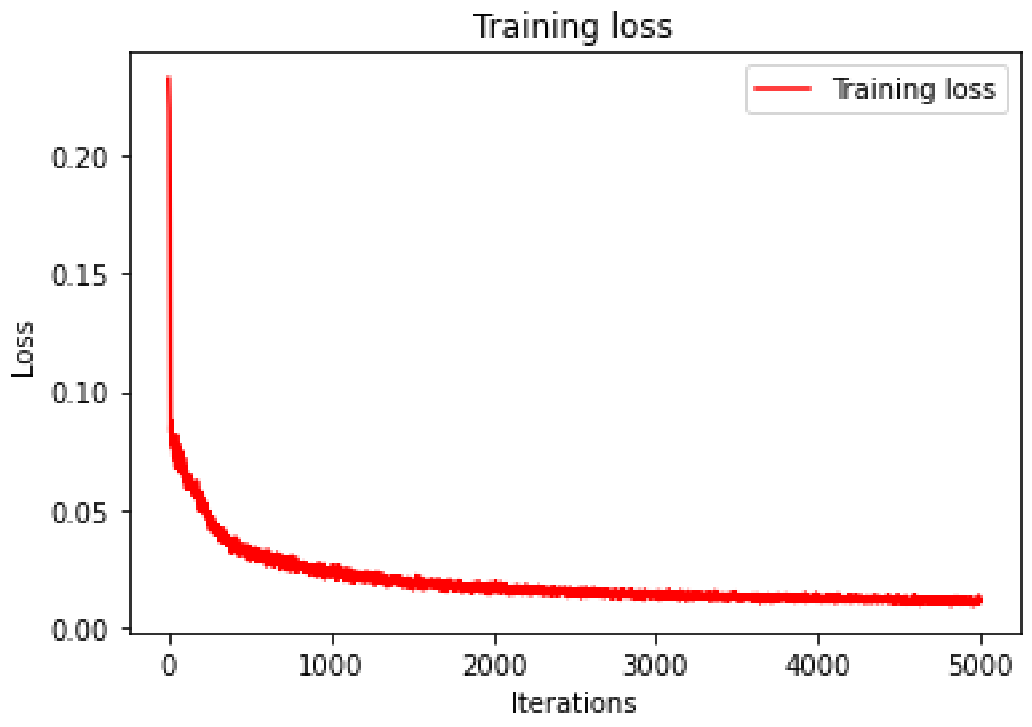
#1. 몇줄만 추가하면 잘 돈다. 코드와 결과는 다음과 같다. (cpu에서 돌렸음)

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets
import torch.optim as optim
from torchvision.transforms import transforms
from torchvision.utils import save_image

import numpy as np
import matplotlib.pyplot as plt

lr = 0.001
batch_size = 100
epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

'''
Step 1:
'''

# MNIST dataset
dataset = datasets.MNIST(root='./mnist_data/',
                              train=True,
                              transform=transforms.ToTensor(),
                              download=True)

train_dataset, validation_dataset = torch.utils.data.random_split(dataset, [50000, 10000])

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

# KMNIST dataset, only need test dataset
anomaly_dataset = datasets.KMNIST(root='./kmnist_data/',
                             train=False,
                             transform=transforms.ToTensor(),
                             download=True)

# print(len(train_dataset))  # 50000
# print(len(validation_dataset))  # 10000
# print(len(test_dataset))  # 10000
# print(len(anomaly_dataset))  # 10000

'''
Step 2: AutoEncoder
```

```python
'''
# Define Encoder
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 32)
    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        z = F.relu(self.fc3(x))
        return z

# Define Decoder
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(32, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 784)
    def forward(self, z):
        z = F.relu(self.fc1(z))
        z = F.relu(self.fc2(z))
        x = F.sigmoid(self.fc3(z))  # to make output's pixels are 0~1
        x = x.view(x.size(0), 1, 28, 28)
        return x

'''
Step 3: Instantiate model & define loss and optimizer
'''
enc = Encoder().to(device)
dec = Decoder().to(device)
loss_function = nn.MSELoss()
optimizer = optim.Adam(list(enc.parameters()) + list(dec.parameters()), lr=lr)



'''
Step 4: Training
'''
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)


train_loss_list = []


import time
```

```python
start = time.time()
for epoch in range(epochs) :
    print("{}th epoch starting.".format(epoch))
    enc.train()
    dec.train()
    for batch, (images, _) in enumerate(train_loader) :
        images = images.to(device)
        z = enc(images)
        reconstructed_images = dec(z)

        optimizer.zero_grad()
        train_loss = loss_function(images, reconstructed_images)
        train_loss.backward()
        train_loss_list.append(train_loss.item())

        optimizer.step()

        print(f"[Epoch {epoch:3d}] Processing batch #{batch:3d} reconstruction loss:
{train_loss.item():.6f}", end='\r')
end = time.time()
print("Time ellapsed in training is: {}".format(end - start))

# plotting train loss
plt.plot(range(1,len(train_loss_list)+1), train_loss_list, 'r', label='Training loss')
plt.title('Training loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.savefig('loss.png')

enc.eval()
dec.eval()


'''
Step 5: Calculate standard deviation by using validation set
'''
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset, batch_size=batch_size)

score_list = []
for images, _ in validation_loader:
    scores = (images - dec(enc(images)))**2
    scores = scores.view(scores.size(0),-1)
    scores = torch.mean(scores, dim=1)
    for score in scores:
        score_list.append(score)
mean = torch.mean(torch.tensor(score_list))
std = torch.std(torch.tensor(score_list))
```

```python
threshold = mean + 3 * std
print("threshold: ", threshold)


'''
Step 6: Anomaly detection (mnist)
'''
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size)

error1 = 0
n_mnist = 0
for images, _ in test_loader:
    n_mnist += batch_size
    scores = (images - dec(enc(images)))**2
    scores = scores.view(scores.size(0),-1)
    scores = torch.mean(scores, dim=1)
    for score in scores:
        if score > threshold:
            error1 +=1
type_1_error_rate = error1/n_mnist
print("type 1 error rate : ", type_1_error_rate*100 , "%")

'''
Step 7: Anomaly detection (kmnist)
'''
anomaly_loader = torch.utils.data.DataLoader(dataset=anomaly_dataset, batch_size=batch_size)

error2 = 0
n_kmnist = 0
for images, _ in anomaly_loader:
    n_kmnist += batch_size
    scores = (images - dec(enc(images)))**2
    scores = scores.view(scores.size(0),-1)
    scores = torch.mean(scores, dim=1)
    for score in scores:
        if score <= threshold:
            error2 +=1
type_2_error_rate = error2/n_kmnist
print("type 2 error rate : ", type_2_error_rate*100, "%")
```




```
threshold:  tensor(0.0321)
type 1 error rate :  0.8699999999999999 %
type 2 error rate :  3.16 %
```

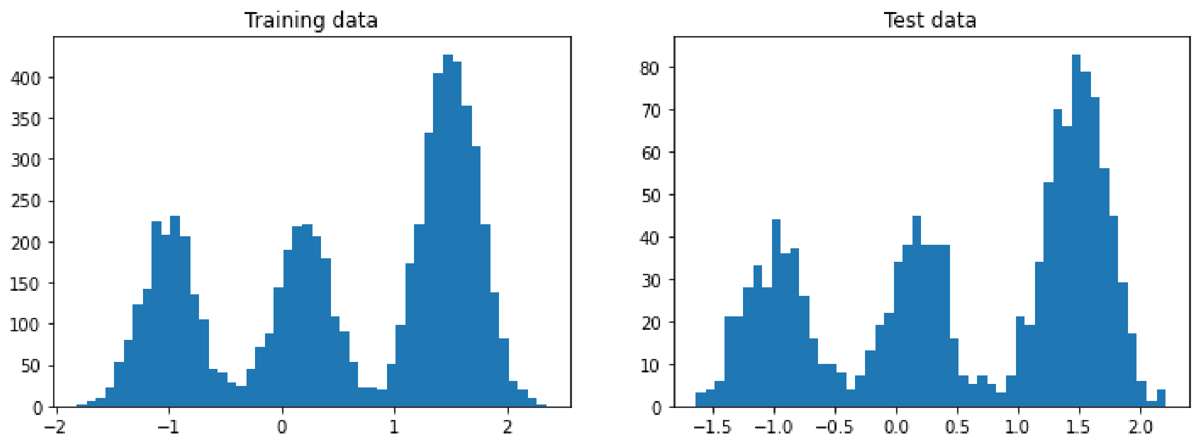
#2. 수업 코드를 가져와서 조금만 수정하면 그만이다.

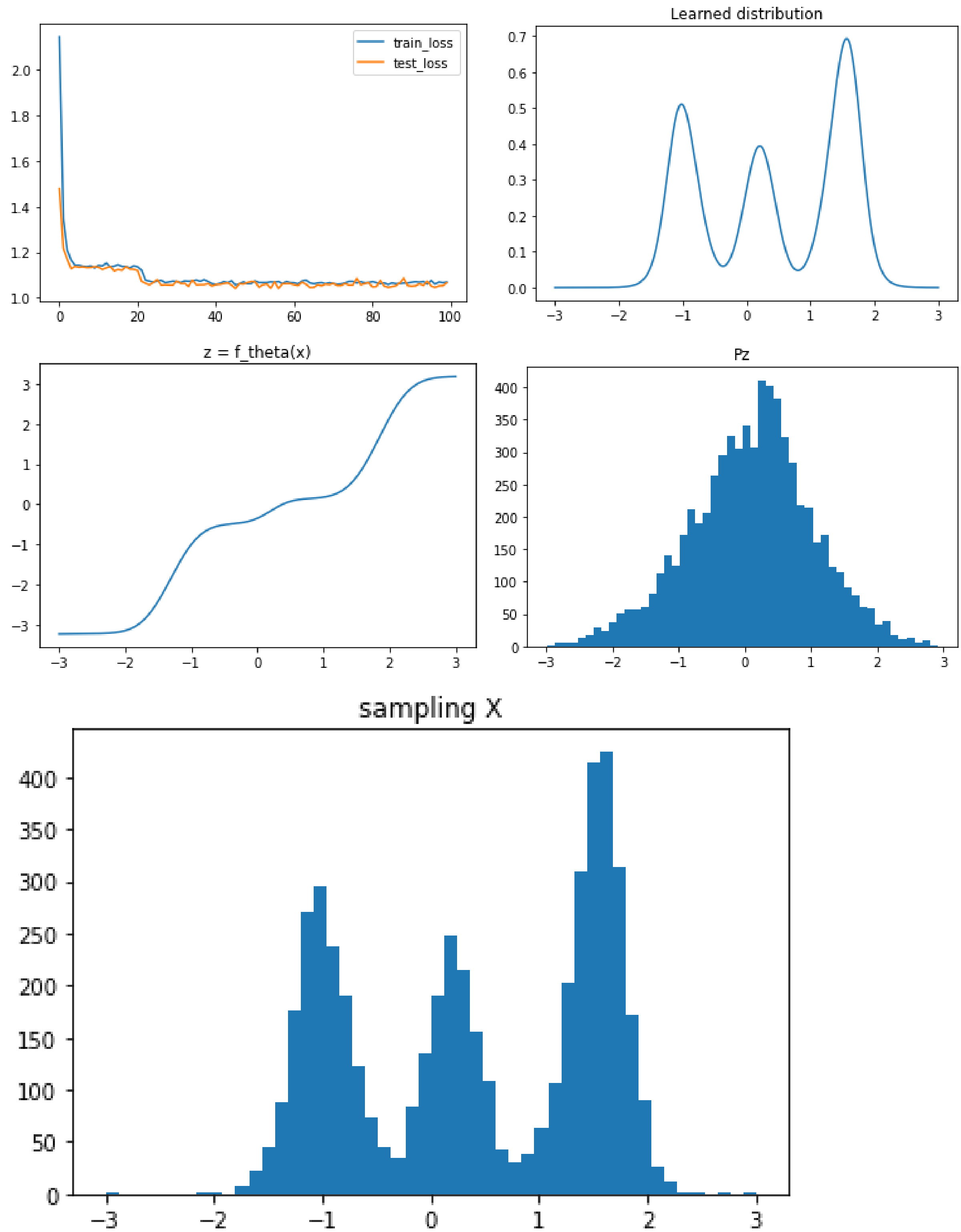
```
import torch
import torch.utils.data as data
import torch.nn as nn
from torch.distributions.normal import Normal
from torch.distributions.uniform import Uniform
import numpy as np
import matplotlib.pyplot as plt



epochs = 100
learning_rate = 5e-2
batch_size = 128
n_components=5 # the number of kernel
target_distribution = Normal(0.0, 1.0)



####################################
# STEP 1: Implement 1-d Flow model #
# Model is misture of Gaussian CDFs
#
####################################
class Flow1d(nn.Module):
    def __init__(self, n_components):
        super(Flow1d, self).__init__()
        self.mus = nn.Parameter(torch.randn(n_components), requires_grad=True)
        self.log_sigmas = nn.Parameter(torch.zeros(n_components),
requires_grad=True)
        self.weight_logits = nn.Parameter(torch.ones(n_components),
requires_grad=True)
    def forward(self, x):
        x = x.view(-1,1)
        weights = self.weight_logits.exp()
        distribution = Normal(self.mus, self.log_sigmas.exp())
        z = ( (distribution.cdf(x)-0.5) * weights).sum(dim=1)
        dz_by_dx = (distribution.log_prob(x).exp() * weights).sum(dim=1)
        return z, dz_by_dx

################################################
# STEP 2: Create Dataset and Create Dataloader #
################################################

def mixture_of_gaussians(num, mu_var=(-1,0.25, 0.2,0.25, 1.5,0.25)):
    n = num // 3
    m1,s1,m2,s2,m3,s3 = mu_var
```

```python
    gaussian1 = np.random.normal(loc=m1, scale=s1, size=(n,))
    gaussian2 = np.random.normal(loc=m2, scale=s2, size=(n,))
    gaussian3 = np.random.normal(loc=m3, scale=s3, size=(num-n,))
    return np.concatenate([gaussian1, gaussian2, gaussian3])

class MyDataset(data.Dataset):
    def __init__(self, array):
        super().__init__()
        self.array = array

    def __len__(self):
        return len(self.array)

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



################################
# STEP 3: Define Loss Function #
################################
def loss_function(target_distribution, z, dz_by_dx):
    # log(p_Z(z)) = target_distribution.log_prob(z)
    # log(dz/dx) = dz_by_dx.log() (flow is defined so that dz/dx>0)
    log_likelihood = target_distribution.log_prob(z) + dz_by_dx.log()
    return -log_likelihood.mean()  #flip sign, and sum of data X_1,...X_N

###########################
# STEP 4: Train the model #
###########################

# create dataloader
n_train, n_test = 5000, 1000
train_data = mixture_of_gaussians(n_train)
test_data = mixture_of_gaussians(n_test)

train_loader = data.DataLoader(MyDataset(train_data), batch_size=batch_size,
shuffle=True)
test_loader = data.DataLoader(MyDataset(test_data), batch_size=batch_size,
shuffle=True)

# create model
flow = Flow1d(n_components)
optimizer = torch.optim.Adam(flow.parameters(), lr=learning_rate)

train_losses, test_losses = [], []

for epoch in range(epochs):
```

```python
    # train
#     flow.train()
    mean_loss = 0
    for i, x in enumerate(train_loader):
        z, dz_by_dx = flow(x)
        loss = loss_function(target_distribution, z, dz_by_dx)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        mean_loss += loss.item()
    train_losses.append(mean_loss/(i+1))

    # test
    flow.eval()
    mean_loss = 0
    for i, x in enumerate(test_loader):
        z, dz_by_dx = flow(x)
        loss = loss_function(target_distribution, z, dz_by_dx)

        mean_loss += loss.item()
    test_losses.append(mean_loss/(i+1))


# visualizing px
_, axes = plt.subplots(1,2, figsize=(12,4))
_ = axes[0].hist(train_loader.dataset.array, bins=50)
_ = axes[1].hist(test_loader.dataset.array, bins=50)
_ = axes[0].set_title('Training data')
_ = axes[1].set_title('Test data')
plt.show()
# visualizing loss (as training progresses)
plt.plot(train_losses, label='train_loss')
plt.plot(test_losses, label='test_loss')
plt.legend()
plt.show()
#visualizing learned distribution and z=f(theta)
x = np.linspace(-3,3,1000)
with torch.no_grad():
    z, dz_by_dx = flow(torch.FloatTensor(x))
    px = (target_distribution.log_prob(z) +
dz_by_dx.log()).exp().cpu().numpy()
plt.plot(x, px)
plt.title("Learned distribution")
plt.show()
plt.plot(x,z)
plt.title("z = f_theta(x)")
plt.show()
```

```
#visualizing pz
with torch.no_grad():
    z, _ = flow(torch.FloatTensor(train_loader.dataset.array))

plt.hist(np.array(z), bins=50)
plt.title("Pz")
plt.show()


# sampling

N = 5000
z = torch.normal(torch.zeros(N), torch.ones(N))
x_low = torch.full((N,), -3.)
x_high = torch.full((N,), 3.)

#Perform bisection
with torch.no_grad():
    for _ in range(30):
        m = (x_low+x_high)/2
        f,_ = flow(m)
        x_high[f>=z] = m[f>=z]
        x_low[f<z] = m[f<z]
    x = (x_low+x_high)/2

plt.hist(np.array(x), bins=50)
plt.title("sampling X")
plt.show()
```





train_loss
test_loss
Learned distribution
z = f_theta(x)
Pz
sampling X

#3. Let $\Omega = (\Omega_1 < \Omega_2 < \dots < \Omega_{|\Omega|})$, $\Omega^c = (\Omega^c_1 < \dots < \Omega^c_{|\Omega^c|})$ and

$\tau = (\tau_1, \dots, \tau_n) = (\Omega_1, \Omega_2, \dots, \Omega_{|\Omega|}, \Omega^c_1, \dots, \Omega^c_{|\Omega^c|})$. Recall $P_\tau = \begin{bmatrix} e^T_{\tau(1)} \\ e^T_{\tau(2)} \\ \vdots \\ e^T_{\tau(n)} \end{bmatrix}$.

Then
$$\left(P_\tau \frac{\partial z}{\partial x_j}\right)_i = \left(\frac{\partial z}{\partial x_j}\right)_{\tau(i)} = \frac{\partial z_{\tau(i)}}{\partial x_j} = \begin{cases} 1 & \text{if } \tau(i) \in \Omega \text{ and } j = \tau(i) \\ 0 & \text{if } \tau(i) \in \Omega \text{ and } j \neq \tau(i) \\ \left(e^{s_\theta(x_\Omega)}\right)_{i-|\Omega|} & \text{if } \tau(i) \in \Omega^c \text{ and } j = \tau(i) \\ \text{something weird} & \text{else (denote as } *) \end{cases}$$

Then
$$P_\tau \frac{\partial z}{\partial x_j} = \begin{pmatrix} \frac{\partial z_{\tau(1)}}{\partial x_j} \\ \vdots \\ \frac{\partial z_{\tau(n)}}{\partial x_j} \end{pmatrix} = \begin{bmatrix} I & 0 \\ * & \mathrm{diag}\left(e^{s_\theta(x_\Omega)}\right) \end{bmatrix} e_{\tau^{-1}(j)}$$

$$P_\tau = P_{\tau^{-1}}^T = \left[e_{\tau^{-1}(1)} \middle| e_{\tau^{-1}(2)} \middle| \cdots \middle| e_{\tau^{-1}(n)} \right]$$

Thus
$$P_\tau \frac{\partial z}{\partial x} = \begin{bmatrix} I & 0 \\ * & \mathrm{diag}\left(e^{s_\theta(x_\Omega)}\right) \end{bmatrix} P_\tau$$

Then
$$\frac{\partial z}{\partial x} = P_\tau^{-1} \begin{bmatrix} I & 0 \\ * & \mathrm{diag}\left(e^{s_\theta(x_\Omega)}\right) \end{bmatrix} P_\tau$$

$$\log \left| \frac{\partial z}{\partial x} \right| = \log 1 + \log\left(\prod_{i=1}^{n-|\Omega|} \left(e^{s_\theta(x_\Omega)}\right)_i \right) + \log 1$$

$$= 1^T_{n-|\Omega|} \, s_\theta(x_\Omega) \quad //$$

#4 (a) $D_{KL}(X\|Y) = \int_{\mathbb{R}^d} f(x)\log\left(\frac{f(x)}{g(x)}\right)dx = \mathop{\mathbb{E}}_{x\sim f(x)}\left[-\log\left(\frac{g(x)}{f(x)}\right)\right] \geq -\log \mathop{\mathbb{E}}_{x\sim f(x)}\left[\frac{g(x)}{f(x)}\right] = -\log 1 = 0$

$\uparrow$

$\because$ Jensen's inequality ($-\log$ is convex)

(b) $D_{KL}(X\|Y) = \int_{\mathbb{R}^d} f(x)\log\left(\frac{f(x)}{g(x)}\right)dx = \int_{\mathbb{R}^d} f(x)\log\left(\frac{f_{X_1}(x_1)f_{X_2}(x_2)\cdots f_{X_d}(x_d)}{g_{Y_1}(x_1)\cdots g_{Y_d}(x_d)}\right)dx$ (by independence)

$= \sum_{i=1}^{d}\int_{\mathbb{R}^d} f(x)\log\left(\frac{f_{X_i}(x_i)}{g_{Y_i}(x_i)}\right)dx = \sum_{i=1}^{d}\left(\int_{\mathbb{R}} f(x_i)\log\left(\frac{f_{X_i}(x_i)}{g_{Y_i}(x_i)}\right)dx_i\right)\times\underbrace{1\times 1\times 1\cdots\times 1}_{d-1}$

$= \sum_{i=1}^{d} D_{KL}(X_i|Y_i) = D_{KL}(X_1|Y_1) + \cdots + D_{KL}(X_d|Y_d)$

#15 $D_{KL}(N(\mu_0,\Sigma_0) \| N(\mu_1,\Sigma_1)) = \underset{x\sim f}{\mathbb{E}}\left[\log f(x) - \log g(x)\right]$

$$\left(\begin{array}{l} f(x) = \dfrac{1}{(2\pi)^{d/2}|\Sigma_0|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\right) \\ g(x) = \dfrac{1}{(2\pi)^{d/2}|\Sigma_1|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right) \end{array} \right.$$

$$= \underset{x\sim f}{\mathbb{E}}\left[\frac{1}{2}\log\frac{|\Sigma_1|}{|\Sigma_0|} - \frac{1}{2}(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0) + \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right]$$

$$= \frac{1}{2}\log\frac{|\Sigma_1|}{|\Sigma_0|} - \frac{1}{2}\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\right] + \frac{1}{2}\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right]$$

$*$ $\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\right] = \underset{x\sim f}{\mathbb{E}}\left[tr\left((x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\right)\right] = \underset{x\sim f}{\mathbb{E}}\left[tr\left((x-\mu_0)(x-\mu_0)^T\Sigma_0^{-1}\right)\right]$

$$= tr\left(\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_0)(x-\mu_0)^T\Sigma_0^{-1}\right]\right) = tr\left(\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_0)(x-\mu_0)^T\right]\Sigma_0^{-1}\right) = tr(\Sigma_0\Sigma_0^{-1}) = d.$$

$*$ $\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right] = tr\left(\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_1)(x-\mu_1)^T\right]\Sigma_1^{-1}\right)$

$$= tr\left(\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_0+\mu_0-\mu_1)(x-\mu_0+\mu_0-\mu_1)^T\right]\Sigma_1^{-1}\right) = tr\left(\underset{x\sim f}{\mathbb{E}}\left[(x-\mu_0)(x-\mu_0)^T + (x-\mu_0)(\mu_0-\mu_1)^T + (\mu_0-\mu_1)(x-\mu_0)^T + (\mu_0-\mu_1)(\mu_0-\mu_1)^T\right]\Sigma_1^{-1}\right)$$

$$= tr\left(\Sigma_1^{-1}\left(\Sigma_0 + (\mu_0-\mu_1)(\mu_0-\mu_1)^T\right)\right)$$

$$= tr(\Sigma_1^{-1}\Sigma_0) + tr\left(\Sigma_1^{-1}(\mu_0-\mu_1)(\mu_0-\mu_1)^T\right)$$

$$= tr(\Sigma_1^{-1}\Sigma_0) + tr\left((\mu_0-\mu_1)^T\Sigma_1^{-1}(\mu_0-\mu_1)\right) = tr(\Sigma_1^{-1}\Sigma_0) + (\mu_0-\mu_1)^T\Sigma_1^{-1}(\mu_0-\mu_1)$$

Thus, we have

$$D_{KL}(N(\mu_0,\Sigma_0)\|N(\mu_1,\Sigma_1)) = \frac{1}{2}\left(\log\frac{|\Sigma_1|}{|\Sigma_0|} - d + tr(\Sigma_1^{-1}\Sigma_0) + (\mu_0-\mu_1)^T\Sigma_1^{-1}(\mu_0-\mu_1)\right)$$

$$= \frac{1}{2}\left(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1-\mu_0)^T\Sigma_1^{-1}(\mu_1-\mu_0) - d + \log\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)\right)$$

used identities $*$
- $tr(AB) = tr(BA)$
- $tr(ABC) = tr(BCA) = tr(CAB)$
- $tr(x) = x$ (when $x \in \mathbb{R}$)

#6 $\underset{\theta \in \Theta}{\text{maximize}}\ f(\theta) \iff \underset{\theta\in\Theta,\ \phi\in\Phi}{\text{maximize}}\ g(\theta,\phi)$.

pf) First show $\operatorname{argmax} f \subseteq \{\theta \mid (\theta,\phi) \in \operatorname{argmax} g\}$.

Consider $\theta^* \in \operatorname{argmax} f$

$g(\theta^*,\phi^*) = f(\theta^*) - h(\theta^*,\phi^*)$ $\left(\exists \phi^* \text{ s.t. } h(\theta^*,\phi^*)=0\right)$

$\geq f(\theta) - h(\theta,\phi)$ $\left(\because f(\theta^*) \geq f(\theta) \text{ and } h(\theta^*,\phi^*)=0 \text{ while } h(\theta,\phi) \geq 0\right)$

Thus $g(\theta^*,\phi^*) \geq g(\theta,\phi)$ $(\forall\theta, \forall\phi)$. i.e. $\theta^* \in \{\theta \mid (\theta,\phi) \in \operatorname{argmax} g\}$.

Now, we show $\{\theta \mid (\theta,\phi) \in \operatorname{argmax} g\} \subseteq \operatorname{argmax} f$.

Consider $\theta^* \in \{\theta \mid (\theta,\phi) \in \operatorname{argmax} g\}$. Then $\exists \phi^*$ s.t. $(\theta^*,\phi^*) \in \operatorname{argmax} g$.

$f(\theta^*) = g(\theta^*,\phi^*) + h(\theta^*,\phi^*)$

$\geq g(\theta,\phi(\theta)) + h(\theta,\phi(\theta))$

$= f(\theta)$

$\left(\begin{array}{l} \phi(\theta) \text{ is } \phi \text{ s.t. } h(\theta,\phi(\theta))=0 \\ \text{Since } g(\theta^*,\phi^*) \geq g(\theta,\phi(\theta)) \text{ and} \\ h(\theta^*,\phi^*) \geq 0 = h(\theta,\phi(\theta)) \\ \text{the equality holds} \end{array}\right)$

Thus $f(\theta^*) \geq f(\theta)$ $(\forall\theta\in\Theta)$

i.e. $\theta^* \in \operatorname{argmax} f$.

Thus we showed that $\operatorname{argmax} f = \{\theta \mid (\theta,\phi) \in \operatorname{argmax} g\}$

//