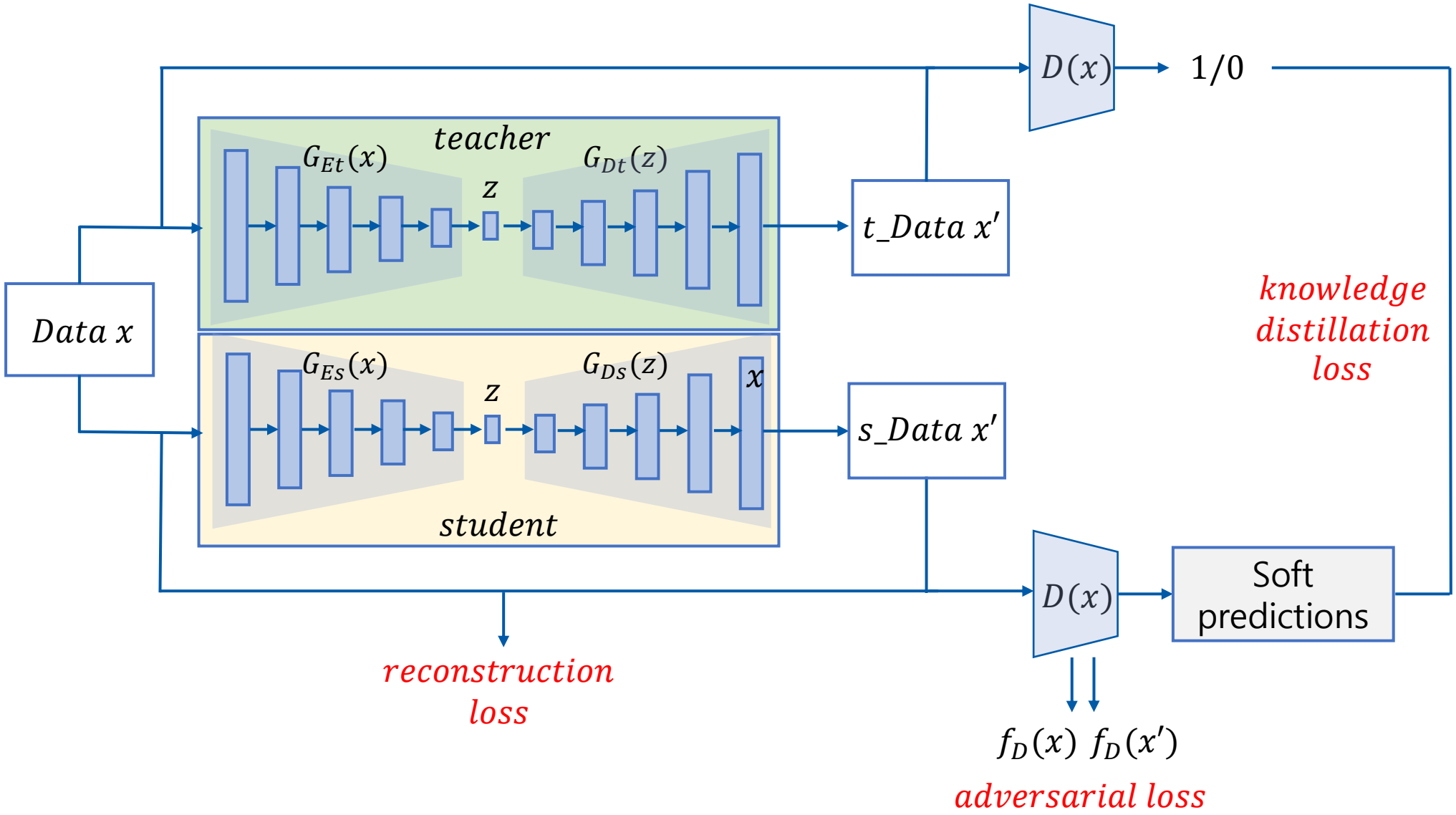
1. network 구조 (2p)
2. student model 변수 선언 코드 (3~4p)
3. generator 손실 계산 코드 (5p)
4. KD loss 함수 코드 (6p)
5. y_와 y_pred 변수 (7p~8p)
6. github 주소 (9p)

10/29 임에딘

Network 구조


D(x)
1/0
teacher
G_Et(x)
z
G_Dt(z)
t_Data x'
Data x
knowledge distillation loss
G_Es(x)
z
G_Ds(z)
x
s_Data x'
student
D(x)
Soft predictions
reconstruction loss
f_D(x) f_D(x')
adversarial loss

```python
class BeatGAN_s(AD_MODEL):
    def __init__(self, opt, dataloader, device):
        super(BeatGAN_s, self).__init__(opt, dataloader, device)

        self.teacher=BeatGAN_t(opt, dataloader, device) #teacher객체 생성
        #self.teacher.copy() #load teacher G model and save to student path

        self.dataloader = dataloader
        self.device = device
        self.opt=opt

        self.batchsize = opt.batchsize
        self.nz = opt.nz
        self.niter = opt.niter

        self.G = Generator(opt).to(device)
        self.G.apply(weights_init)
        if not self.opt.istest:
            print_network(self.G)

        self.D = Discriminator(opt).to(device)
        self.D.apply(weights_init)
        if not self.opt.istest:
            print_network(self.D)


        self.bce_criterion = nn.BCELoss()
        self.mse_criterion=nn.MSELoss()
        self.kd_criterion=DistillKL(opt)
        #self.att_criterion=Attention(opt) ##
```

1

2

* student model 코드 (다음 페이지까지)

(1) teacher 객체 생성
(2) Knowledge Distillation 손실함수

```
self.optimizerD = optim.Adam(self.D.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
self.optimizerG = optim.Adam(self.G.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))


self.total_steps = 0
self.cur_epoch=0


self.input = torch.empty(size=(self.opt.batchsize, self.opt.nc, self.opt.isize), dtype=torch.float32, device=self.device)
self.label = torch.empty(size=(self.opt.batchsize,), dtype=torch.float32, device=self.device)
self.gt    = torch.empty(size=(opt.batchsize,), dtype=torch.long, device=self.device)
self.fixed_input = torch.empty(size=(self.opt.batchsize, self.opt.nc, self.opt.isize), dtype=torch.float32, device=self.device)
self.real_label = 1
self.fake_label= 0


self.out_d_real = None
self.feat_real = None

self.fake = None
self.latent_i = None
self.out_d_fake = None
self.feat_fake = None

self.err_d_real = None
self.err_d_fake = None
self.err_d = None

self.out_g = None
self.err_g_adv = None
self.err_g_rec = None
self.err_g = None
```

1

2

1

2

3

* student model 코드 (이어서)

(1) input, fake : x, x'
(2) feat_real, feat_fake : f(x), f(x')
(3) generator 손실 (update_netg에서 자세히 설명)

* Generator의 손실

```
def update_netg(self):
    self.G.zero_grad()
    self.label.data.resize_(self.opt.batchsize).fill_(self.real_label)
    self.fake, self.latent_i = self.G(self.input)
    self.out_g, self.feat_fake = self.D(self.fake)
    _, self.feat_real = self.D(self.input)



    # self.err_g_adv = self.bce_criterion(self.out_g, self.label)   # loss for ce
    self.err_g_adv=self.mse_criterion(self.feat_fake,self.feat_real)  # loss for feature matching
    self.err_g_rec = self.mse_criterion(self.fake, self.input)  # constrain x' to look like x


    #self.err_g =  self.err_g_rec + self.err_g_adv * self.opt.w_adv
    self.err_g = self.err_g_rec + self.err_g_adv * self.opt.w_adv
    self.err_g+= self.kd_criterion(self.y_pred,self.teacher.y_t)
    self.err_g.backward()
    self.optimizerG.step()
```

(1)err_g : Generator의 손실

(2)err_g_adv=f(x)와 f(x')의 손실
(3)err_g_rec=x와 x'의 손실
(4)kd_criterion=student의 soft prediction과
teacher의 ground truth label의 손실

* Knowledge Distillation 손실함수

```python
import torch.nn as nn
import torch
import torch.nn.functional as F
from options import Options ##

class DistillKL(nn.Module):
    def __init__(self, opt):
        super(DistillKL, self).__init__()
        opt = Options().parse()
        self.T = opt.temperature
        self.alpha= opt.alpha

    def forward(self, y_s, y_t):
        #B, C, H, W = y_s.size()
        y_s=torch.from_numpy(y_s)
        y_t=torch.from_numpy(y_t)
        y_s=y_s.reshape(1,len(y_s))
        y_t=y_t.reshape(1,len(y_t))
        #print("y_s reshape: ", y_s.shape)
        #print("y_t reshape: ",y_t.shape)
        p_s = F.log_softmax(y_s/self.T, dim=1)
        p_t = F.softmax(y_t/self.T, dim=1)
        loss = self.alpha*F.kl_div(p_s, p_t.detach(), reduction='sum') * (self.T**2) / y_s.shape[0]
        return loss
```

* student model의 train 함수

```
def train(self):
    self.train_hist = {}
    self.train_hist['D_loss'] = []
    self.train_hist['G_loss'] = []
    self.train_hist['per_epoch_time'] = []
    self.train_hist['total_time'] = []

    self.teacher.copy() #load teacher G model and save to student path
    self.y_,self.y_pred=self.predict(self.dataloader["train"]) ##
    self.teacher.y_t,self.teacher.y_pred_t=self.teacher.predict(self.dataloader["train"]) ##

    print("Train model.")
    start_time = time.time()
    best_auc=0
    best_auc_epoch=0

    with open(os.path.join(self.outf, self.model, self.dataset, "val_info.txt"), "w") as f:
        for epoch in range(self.niter):
            self.cur_epoch+=1
            self.train_epoch()
            auc,th,f1=self.validate()
            if auc > best_auc:
                best_auc = auc
                best_auc_epoch=self.cur_epoch
                self.save_weight_GD()
            f.write("[{}] auc:{:.4f} \t best_auc:{:.4f} in epoch[{}]\n".format(self.cur_epoch,auc,best_auc,best_auc_epoch ))
            print("[{}] auc:{:.4f} th:{:.4f} f1:{:.4f} \t best_auc:{:.4f} in epoch[{}]\n".format(self.cur_epoch,auc,th,f1,best_auc,best_auc_epoch ))



    self.train_hist['total_time'].append(time.time() - start_time)
    print("Avg one epoch time: %.2f, total %d epochs time: %.2f" % (np.mean(self.train_hist['per_epoch_time']),
                                                                    self.niter,
                                                                    self.train_hist['total_time'][0]))

    self.save(self.train_hist)

    self.save_loss(self.train_hist)
```

predict함수에서 (뒷 페이지)
y_와 y_pred가 리턴됩니다.

teacher model의 y_ 변수와
student model의 y_pred 변수
두개를 받아 kd loss에 넣었습니다.

이를 위해
teacher model로부터 리턴값을 받아옵니다.
(y_t, y_pred_t)

*predict 함수

리턴되는 값:
y_
y_pred

(1) y_
gt_labels를
받아옵니다.
(ground truth
label)

(2) y_pred
an_scores를
받아옵니다.
(error)

```python
def predict(self,dataloader_,scale=True):
    with torch.no_grad():

        self.an_scores = torch.zeros(size=(len(dataloader_.dataset),), dtype=torch.float32, device=self.device)
        self.gt_labels = torch.zeros(size=(len(dataloader_.dataset),), dtype=torch.long,    device=self.device)
        self.latent_i  = torch.zeros(size=(len(dataloader_.dataset), self.opt.nz), dtype=torch.float32, device=self.device)
        self.dis_feat = torch.zeros(size=(len(dataloader_.dataset), self.opt.ndf*16*10), dtype=torch.float32,
                                    device=self.device)


        for i, data in enumerate(dataloader_, 0):

            self.set_input(data)
            self.fake, latent_i = self.G(self.input)


            # error = torch.mean(torch.pow((d_feat.view(self.input.shape[0],-1)-d_gen_feat.view(self.input.shape[0],-1)), 2), dim=1)
            #
            error = torch.mean(
                torch.pow((self.input.view(self.input.shape[0], -1) - self.fake.view(self.fake.shape[0], -1)), 2),
                dim=1)

            self.an_scores[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = error.reshape(error.size(0))
            self.gt_labels[i*self.opt.batchsize : i*self.opt.batchsize+error.size(0)] = self.gt.reshape(error.size(0))
            self.latent_i [i*self.opt.batchsize : i*self.opt.batchsize+error.size(0), :] = latent_i.reshape(error.size(0), self.opt.nz)


        # Scale error vector between [0, 1]
        if scale:
            self.an_scores = (self.an_scores - torch.min(self.an_scores)) / (torch.max(self.an_scores) - torch.min(self.an_scores))

        y_=self.gt_labels.cpu().numpy()
        y_pred=self.an_scores.cpu().numpy()

        return y_,y_pred
```

github 주소

https://github.com/edinide/Self-Improving-BeatGAN-via-Knowledge-Distillation