8번째 미팅발표

2021-11-24

임에딘

코멘트

1. Teacher model로부터 Feature 정보를 받아와서 Distillation
2. Generator <-> Discriminator 수정

knowledge distillation(KL), feature distillation(FT)

```
self.bce_criterion = nn.BCELoss()
self.mse_criterion=nn.MSELoss()
self.kd_criterion=DistillKL(opt)
self.ft_criterion=DistillFT(opt)
```

```
#knowledge distillation
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))
        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]
        #loss= (-1)*p_s*p_t.detach()
        return loss
```

knowledge distillation(KL), feature distillation(FT)

```
self.bce_criterion = nn.BCELoss()
self.mse_criterion=nn.MSELoss()
self.kd_criterion=DistillKL(opt)
self.ft_criterion=DistillFT(opt)
```

```python
#feature distillation
class DistillFT(nn.Module):
    def __init__(self, opt):
        super(DistillFT, self).__init__()
        opt = Options().parse()
        self.p = 2

    def forward(self, g_s, g_t):
        loss = sum([self.at_loss(f_s, f_t.detach()) for f_s, f_t in zip(g_s, g_t)])

        return loss

    def at_loss(self, f_s, f_t):
        return (self.at(f_s) - self.at(f_t)).pow(2).mean()

    def at(self, f):
        return F.normalize(f.pow(self.p).mean(1).view(f.size(0), -1))
```

Feature Distillation

* 저번 미팅시간에서 제기된 문제 :

Teacher-> Student로 feature distillation이 이뤄지지 않음
(teacher model로부터 feature 정보를 받아오자)

→ Discriminator이 아닌 Generator에서 feature list를 받아오는 것으로 변경
→ Teacher model로부터 feature list 정보를 받아오자

```python
class Encoder(nn.Module):
    def __init__(self, ngpu,opt,out_z):
        super(Encoder, self).__init__()
        self.ngpu = ngpu
        self.conv1=nn.Sequential(
            # input is (nc) x 320
            nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # state size. (ndf) x 160
        self.conv2=nn.Sequential(
            nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 2),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # state size. (ndf*2) x 80
        self.conv3=nn.Sequential(
            nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 4),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # state size. (ndf*4) x 40
        self.conv4=nn.Sequential(
            nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 8),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # state size. (ndf*8) x 20
        self.conv5=nn.Sequential(
            nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 16),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # state size. (ndf*16) x 10
        self.conv6=nn.Sequential(
            nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False)
        )
        # state size. (nz) x 1
```

```python
class Decoder(nn.Module):
    def __init__(self, ngpu,opt):
        super(Decoder, self).__init__()
        self.ngpu = ngpu
        self.conv1=nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose1d(opt.nz,opt.ngf*16,10,1,0,bias=False),
            nn.BatchNorm1d(opt.ngf*16),
            nn.ReLU(True)
        )
        # state size. (ngf*16) x10
        self.conv2=nn.Sequential(
            nn.ConvTranspose1d(opt.ngf * 16, opt.ngf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 8),
            nn.ReLU(True)
        )
        # state size. (ngf*8) x 20
        self.conv3=nn.Sequential(
            nn.ConvTranspose1d(opt.ngf * 8, opt.ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 4),
            nn.ReLU(True)
        )
        # state size. (ngf*4) x 40
        self.conv4=nn.Sequential(
            nn.ConvTranspose1d(opt.ngf * 4, opt.ngf*2, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf*2),
            nn.ReLU(True)
        )
        # state size. (ngf*2) x 80
        self.conv5=nn.Sequential(
            nn.ConvTranspose1d(opt.ngf * 2, opt.ngf , 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf ),
            nn.ReLU(True)
        )
        # state size. (ngf) x 160
        self.conv6=nn.Sequential(
            nn.ConvTranspose1d(opt.ngf , opt.nc, 4, 2, 1, bias=False),
            nn.Tanh()
        )
        # state size. (nc) x 320
```

저번 시간의 코드 Discriminator

```python
class Discriminator(nn.Module):

    def __init__(self, opt):
        super(Discriminator, self).__init__()
        model = Encoder(opt.ngpu,opt,1)
        layers = list(model.main.children())

        self.features = nn.Sequential(*layers[:-1])
        self.classifier = nn.Sequential(layers[-1])
        self.classifier.add_module('Sigmoid', nn.Sigmoid())

    def forward(self, x):
        features = self.features(x)
        features = features
        classifier = self.classifier(features)
        classifier = classifier.view(-1, 1).squeeze(1)

        return classifier, features
```

```python
def forward(self, x):
    #feature 1~6 (6: features)
    feat1=self.feat1(x)
    feat1=feat1
    feat2=self.feat2(x)
    feat2=feat2
    feat3=self.feat3(x)
    feat3=feat3
    feat4=self.feat4(x)
    feat4=feat4
    feat5=self.feat5(x)
    feat5=feat5

    features = self.features(x)
    features = features

    feat_list=[feat1,feat2,feat3,feat4,feat5,features]

    classifier = self.classifier(features)
    classifier = classifier.view(-1, 1).squeeze(1)

    return classifier, features ,feat_list
```

Discriminator 수정

저번주: feature도 차례로 받아왔고 리스트 feat_list를 정의
변경: feat_list 제거

```python
class Discriminator(nn.Module):

    def __init__(self, opt):
        super(Discriminator, self).__init__()
        model = Encoder(opt.ngpu,opt,1)
        layers = list(model.conv1.children())
        layers.extend(list(model.conv2.children()))
        layers.extend(list(model.conv3.children()))
        layers.extend(list(model.conv4.children()))
        layers.extend(list(model.conv5.children()))
        layers.extend(list(model.conv6.children()))

        self.features = nn.Sequential(*layers[:-1])
        self.classifier = nn.Sequential(layers[-1])
        self.classifier.add_module('Sigmoid', nn.Sigmoid())

    def forward(self, x):
        features = self.features(x)
        features = features
        classifier = self.classifier(features)
        classifier = classifier.view(-1, 1).squeeze(1)

        return classifier, features
```

저번 시간의 코드 Generator

```python
class Generator(nn.Module):

    def __init__(self, opt):
        super(Generator, self).__init__()
        self.encoder1 = Encoder(opt.ngpu,opt,opt.nz)
        self.decoder = Decoder(opt.ngpu,opt)

    def forward(self, x):
        latent_i = self.encoder1(x)
        gen_x = self.decoder(latent_i)
        return gen_x, latent_i
```

Generator 수정

기존: layer을 한번에 받아와서 features 정의
변경: layer 차례로 받아오면서 feature도 차례로 받아왔고 리스트 feat_list를 정의

```python
class Generator(nn.Module):

    def __init__(self, opt):
        super(Generator, self).__init__()
        self.encoder1 = Encoder(opt.ngpu,opt,opt.nz)
        self.decoder = Decoder(opt.ngpu,opt)

        layers=list(self.encoder1.conv1.children())
        self.feat1=nn.Sequential(*layers[:-1])
        layers.extend(list(self.encoder1.conv2.children()))
        self.feat2=nn.Sequential(*layers[:-1])
        layers.extend(list(self.encoder1.conv3.children()))
        self.feat3=nn.Sequential(*layers[:-1])
        layers.extend(list(self.encoder1.conv4.children()))
        self.feat4=nn.Sequential(*layers[:-1])
        layers.extend(list(self.encoder1.conv5.children()))
        self.feat5=nn.Sequential(*layers[:-1])
        layers.extend(list(self.encoder1.conv6.children()))
        self.feat6=nn.Sequential(*layers[:-1])
```

```python
def forward(self, x):
    latent_i = self.encoder1(x)
    gen_x = self.decoder(latent_i)
    #feature 1~6 (6: features)
    feat1=self.feat1(x)
    feat1=feat1
    feat2=self.feat2(x)
    feat2=feat2
    feat3=self.feat3(x)
    feat3=feat3
    feat4=self.feat4(x)
    feat4=feat4
    feat5=self.feat5(x)
    feat5=feat5
    feat6=self.feat6(x)
    feat6=feat6
    feat_list=[feat1,feat2,feat3,feat4,feat5,feat6]
    return gen_x, latent_i, feat_list
```

손실 함수 통합

```
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.feat_list = 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
1   self.err_g_adv=self.mse_criterion(self.feat_fake,self.feat_real)  # loss for feature matching
2   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
3   self.err_g+= self.kd_criterion(self.y_,self.teacher.y_pred_t) #knowledge distillation
4   self.err_g+= self.ft_criterion(self.feat_list,self.teacher.feat_list)# feature distillation

    self.err_g.backward()
    self.optimizerG.step()
```

① f(x)와 f(x')의 손실
② x와 x'의 손실
③ knowledge distillation 손실
④ features 손실

Train함수 일부 (teacher model로부터 정보 받아오는 코드)

```
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.y_feat_list=self.predict(self.dataloader["train"]) ##
    self.teacher.y_t,self.teacher.y_pred_t,self.teacher_feat_list=self.teacher.predict(self.dataloader["train"]) ##
```

update_netd 함수

```
def update_netd(self):
    ##

    self.D.zero_grad()
    # --
    # Train with real
    self.label.data.resize_(self.opt.batchsize).fill_(self.real_label)
    self.out_d_real, self.feat_real = self.D(self.input)
    # --
    # Train with fake
    self.label.data.resize_(self.opt.batchsize).fill_(self.fake_label)
    self.fake, self.latent_i, self.feat_list = self.G(self.input)
    self.out_d_fake, self.feat_fake = self.D(self.fake)
    # --


    self.err_d_real = self.bce_criterion(self.out_d_real, torch.full((self.batchsize,), self.real_label, device=self.device,dtype=torch.float))
    self.err_d_fake = self.bce_criterion(self.out_d_fake, torch.full((self.batchsize,), self.fake_label, device=self.device,dtype=torch.float))


    self.err_d=self.err_d_real+self.err_d_fake
    self.err_d.backward()
    self.optimizerD.step()
```

train/eval teacher model & train/eval student model

```
################################
########  Result  ############
ap:0.9107557515425027
auc:0.9436909619756461
best th:0.0053040790371596 81 --> best f1:0.8191078654821154
(py37)
edin@DESKTOP-FF4AVFF MINGW64 ~/Desktop/4-2/캡디1/코드/BeatGAN-master (master)
$
```

```
################################
########  Result  ############
ap:0.8606605804375185
auc:0.8769271269704162
best th:0.0151094635948538 78 --> best f1:0.7760926803580831
(py37)
edin@DESKTOP-FF4AVFF MINGW64 ~/Desktop/4-2/캡디1/코드/BeatGAN-master (master)
$
```

predict 함수 (y_, y_pred, y_feat_list 리턴)

```
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)
        self.feat_list = 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.feat_list = 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)
```

predict 함수 (y_, y_pred, y_feat_list 리턴)

```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)
        self.feat_list = torch.zeros(size=(len(dataloader_.dataset), self.opt.ndf*16*10), dtype=torch.float32,
                                    device=self.device)


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


        # 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()
        y_feat_list=self.feat_list


        return y_,y_pred,y_feat_list
```