[ALTBI.py]

- base, data, datasets, networks 파일 odim과 동일

- unsupervised의 경우 epoch : tot_filter_model_n_epoch = 90

- semi-supervised의 경우 epoch : tot_filter_model_n_epoch = 100

- optimizer, lr, weight decay: odim과 동일

- adaptive_batch_size

- epoch_qt = 10 -> 10epoch까지는 m_0(=128)로 batch_size 유지

: gamma(minibatch size 증가할때 사용되는 parameter) -> <ALTBI_tr.py>에서 1.03로 설정

: 각 에폭 당 adaptive_batch_size = min(int(m_0 * (gamma ** (epoch-epoch_qt))), train_n, 20000)

-> 일정 mb size 이상이 되면 gpu 문제 발생하여 20000으로 max 설정

- 20000으로 max 설정하여도 사이즈가 큰 data는 그 전에 gpu error 발생하여 따로 max 설정 (CIFAR10, MNIST-C, InternetAds, SVHN, mnist)

```
if epoch<epoch_qt:
    adaptive_batch_size = m_0
else:
    if 'CIFAR'in dataset_name:
        if gamma_ > 1.01:
            adaptive_batch_size = min(int(m_0 * (gamma ** (epoch-epoch_qt))), train_n, 3000)

    elif 'MNIST-C'in dataset_name:
        if gamma_ > 1.01:
            adaptive_batch_size = min(int(m_0 * (gamma ** (epoch-epoch_qt))), train_n, 1000)
    elif 'InternetAds' in dataset_name:
        if gamma_ > 1.01:
            adaptive_batch_size = min(int(m_0 * (gamma ** (epoch-epoch_qt))), train_n, 1000)
    elif 'SVHN' in dataset_name:
        if gamma_ > 1.01:
            adaptive_batch_size = min(int(m_0 * (gamma ** (epoch-epoch_qt))), train_n, 1000)
    elif dataset_name=='mnist':
        if gamma_ > 1.01:
            adaptive_batch_size = min(int(m_0 * (gamma ** (epoch-epoch_qt))), train_n, 5000)
    else:
        adaptive_batch_size = min(int(m_0 * (gamma ** (epoch-epoch_qt))), train_n, 20000)
```

- epoch마다 minibatch size가 증가함에 따라 모든 epoch마다 iter 횟수를 5번으로 통일

- train_iteration = 5 -

:

```
### Train VAE
for batch_idx in range(train_iteration):
    #break
    try:
        inputs, targets, idx = next(train_iter)

    except StopIteration:
        train_iter = iter(train_loader)
        inputs, targets, idx = next(train_iter)
```

- 목적함수 odim과 동일

```
# Update network parameters via backpropagation: forward + backward + optimize
if 'pixel' in filter_net_name:
    loss,loss_vec = VAE_IWAE_loss_gaussian_mean_var_pixelcnn(inputs , filter_model , num_sam , num_aggr,alpha)
elif 'gaussian' in filter_net_name:
    loss,loss_vec = VAE_IWAE_loss_gaussian_mean_var(inputs , filter_model , num_sam , num_aggr,alpha)
elif 'gaussian' in train_option:
    loss,loss_vec = VAE_IWAE_loss_gaussian(inputs , filter_model , num_sam , num_aggr,alpha)
else:
    loss,loss_vec = VAE_IWAE_loss(inputs , filter_model , num_sam , num_aggr,alpha)
```

- qt(loss truncate 때 사용되는 parameter) : <ALTBI_tr.py>에서 0.92로 설정

- m_1 = qt * m_0

- loss_quantile_prob: 10에폭 이전이면 모든 loss 사용, 이후면 92%의 loss만 사용

- trimmed_loss: threshold 이하인 loss들의 평균

```
loss_quantile_prob = 1-(np.clip((epoch-epoch_qt),0,1)*(1-m_1/m_0))
loss_quantile = torch.quantile(loss_vec.detach(), loss_quantile_prob)
trimming_ind = (loss_vec.detach()<=loss_quantile)
trimmed_loss = (loss_vec*trimming_ind).sum()/trimming_ind.sum()
filtered_targets = targets[trimming_ind]

trimmed_loss.backward()
filter_optimizer.step()
```

- Ensemble

- UOD에서 71epoch ~ 90epoch에서 loss들(train_losses) 에폭마다 저장

- loss들 minmax 후 hstack하여(best_loss_matrix) 평균(mean_loss) 저장

- idx_mean_loss = pd.DataFrame({'idx' : best_idx, 'loss' :mean_loss.numpy()}) 데이터 프레임을 return 하여 <ALTBI_tr> 에서 AUC,PRAUC 계산

- loss들 평균 구하기 위해 shuffule=False인 loader (train_loader_noshuf) 사용

```
else:
    if epoch > 70:
        patience_idx = 0
        running_time += (time.time() - start_time)

        train_loss_list_eval = []
        train_targets_list_eval = []
        train_idx_list_eval = []

        filter_model.eval()
        for data in train_loader_noshuf:
            inputs, targets ,idx = data
            inputs = inputs.to(device)
            inputs = inputs.view(inputs.size(0), -1)
            if 'binarize' in train_option:
                inputs = binarize(inputs)
            # Update network parameters via backpropagation: forward + backward + optimize
            if 'pixel' in filter_net_name:
                loss,loss_vec = VAE_IWAE_loss_gaussian_mean_var_pixelcnn(inputs , filter_model , num_sam , num_aggr,1.)
            elif 'gaussian' in filter_net_name:
                loss,loss_vec = VAE_IWAE_loss_gaussian_mean_var(inputs , filter_model , num_sam , num_aggr,1.)
            elif 'gaussian' in train_option:
                loss,loss_vec = VAE_IWAE_loss_gaussian(inputs , filter_model , num_sam , num_aggr,1.)
            else:
                loss,loss_vec = VAE_IWAE_loss(inputs , filter_model , num_sam , num_aggr,1.)

            train_loss_list_eval.append(loss_vec.data.cpu())
            train_targets_list_eval.append(targets.cpu().numpy())
            train_idx_list_eval += list(idx.numpy())


        train_losses = torch.cat(train_loss_list_eval,0).numpy().reshape(-1,1)
        best_idx = train_idx_list_eval
        #Epoch = epoch
        start_time = time.time()

        minmax_scaler = MinMaxScaler()
        if best_loss_matrix.size == 0 :
            best_loss_matrix = minmax_scaler.fit_transform(train_losses)
        else:
            best_loss_matrix = np.hstack((best_loss_matrix, minmax_scaler.fit_transform(train_losses)))
```

<ALTBI_tr> : odim과 동일합니다.