DLHLP HW2 Voice Conversion Report

組長 github id: openopentw

組員：鄭淵仁

HW2-1 (Auto-Encoder) (2.5%)

(1) 請以 Auto-Encoder 之方法實做 Voice conversion。jo 如果同學不想重新刻一個 auto-encoder，可以試著利用這個 repo 的部分程式碼，達到實現出 auto-encoder。如果你是修改助教提供的 repo，請在 report 當中敘述你是如何更改原本程式碼，建議可以附上修改部分的截圖以利助教批閱；同時，果餓未有更動原本模型參數也請一併列出。如果你的 auto-encoder 是自己刻的，那也請你簡單敘述你的實作方法，並附上對應程式碼的截圖。(1%)

我是改動助教提供的 repo 來實作的，改動的主要方向是刪掉 classifier、generator 以及 discriminstor 的部分，只保留 encoder 跟 decoder。

細節上來說是讓 main.py 只執行 solver.py 的 'train'，並且在 solver.train 裡面刪掉除了 encoder 跟 decoder 以外的其他程式碼，最後在 convert.py 裡面把 generator 關掉。參數的部分我是直接用原本是 100000 的 iter，所以沒有改變。

下面是有改動的程式碼的截圖：

- solver.py：只跑 encoder、decoder。

```
elif mode == 'only_train':
    with trange(hps.iters) as iter_:
        for iteration in iter_:
            data = next(self.data_loader)
            c, x = self.permute_data(data)
            # encode
            enc = self.encode_step(x)
            # decode
            x_tilde = self.decode_step(enc, c)
            loss_rec = torch.mean(torch.abs(x_tilde - x))
            loss = loss_rec
            reset_grad([self.Encoder, self.Decoder])
            loss.backward()
            grad_clip([self.Encoder, self.Decoder], self.hps.max_grad_norm)
            self.ae_opt.step()
            info = {
                f'{flag}/loss_rec': loss_rec.item(),
            }
            iter_.set_postfix(loss_rec='%.3f' % list(info.values())[0])
            if iteration % 100 == 0:
                for tag, value in info.items():
                    self.logger.scalar_summary(tag, value, iteration + 1)
            if iteration % 1000 == 0 or iteration + 1 == hps.iters:
                self.save_model(model_path, iteration)
```

- main.py：只跑 train

```
if args.train:
        # solver.train(args.output_model_path, args.flag, mode='pretrain_G')
        # solver.train(args.output_model_path, args.flag, mode='pretrain_D')
        # solver.train(args.output_model_path, args.flag, mode='train')
        # solver.train(args.output_model_path, args.flag, mode='patchGAN')

        solver.train(args.output_model_path, args.flag, mode='only_train')
```

- convert.py：不跑 generator，以及產生 interpolation。

```
def convert_one_sp(h5_path, src_speaker, tar_speaker, utt_id, solver, dir_path,
                                    dset = 'test', gen = False, tar_speaker_2=None,
                                    speaker_used_path = './hps/en_speaker_used.txt'):
        # read speaker id file
        with open(speaker_used_path) as f:
                speakers = [line.strip() for line in f]
                speaker2id = {speaker:i for i, speaker in enumerate(speakers)}

        with h5py.File(h5_path, 'r') as f_h5:
                sp = f_h5[f'{dset}/{src_speaker}/{utt_id}/lin'][()]
                c2 = speaker2id[tar_speaker_2] if tar_speaker_2 else None
                converted_sp = convert_sp(sp, speaker2id[tar_speaker], solver, gen=gen, c2=c2)
                wav_data = sp2wav(converted_sp)

                if tar_speaker_2 is None:
                        fn = f'{src_speaker}_{tar_speaker}_{utt_id}.wav'
                else:
                        fn = f'{src_speaker}_{tar_speaker}_{utt_id}_inter.wav'

                wav_path = os.path.join(dir_path, fn)
                wavfile.write(wav_path, 16000, wav_data)
```

(2) 在訓練完成後，試著將助教要求轉換的音檔轉成 source speaker 和 target speaker 的 interpolation，也就是在 testing 的時候，除了將指定的音檔轉成 p1 和 p2 的聲音之外，請嘗試轉成 p1 和 p2 interpolation 的聲音。並比較分析 interpolated 的聲音和 p1 以及 p2 的關係。你可以從聲音頻率的高低、口音、語調等面向進行觀察。只要有合理分析助教就會給分。請同時將這題的音檔放在 github 的 hw2-1 資料夾中，檔名格式請參考投影片。(1.5%)

我使用的 interpolation 的方法是把 5 個 embedding 都使用 p1 和 p2 的平均，如下圖：

```
def forward(self, x, c, c2=None):
    w_c = 0.5

    emb1 = self.emb1(c) if c2 is None else self.emb1(c) * w_c + self.emb1(c2) * (1 - w_c)
    emb2 = self.emb2(c) if c2 is None else self.emb2(c) * w_c + self.emb2(c2) * (1 - w_c)
    emb3 = self.emb3(c) if c2 is None else self.emb3(c) * w_c + self.emb3(c2) * (1 - w_c)
    emb4 = self.emb4(c) if c2 is None else self.emb4(c) * w_c + self.emb4(c2) * (1 - w_c)
    emb5 = self.emb5(c) if c2 is None else self.emb5(c) * w_c + self.emb5(c2) * (1 - w_c)
```

Interpolate 之後的聲音很像是 p1 和 p2 的聲音混在一起講的感覺。也不太好區分是男或女的聲音，很像是新聞媒體在對聲音馬賽克之後的聲音。另外聲音的品質下降很多，更像是機器合成出來的聲音，可能是因為 speaker embedding 是兩個 embedding 合成出來的，而不是原本 train 出來的 embedding。

HW2-2 (GAN) (2.5%)

(1) 請使用助教在投影片中提到的連結，進行 voice conversion。請描述在這個程式碼中，語者資訊是如何被嵌入模型中的？請問這樣的方式有什麼優缺點？有沒有其他的作法可以將 speaker information 放入 generator 裡呢？(1%)

- **語者資訊是如何被嵌入模型中的？**

  語者的表示法是 0-1 vector，放入的方法是把這個 vector expand 成 input feature 的 size 再 concatenate 到 feature 的旁邊。接下來就跟著 feature 一起經過剩下的 layers。

- **請問這樣的方式有什麼優缺點？**

  這樣的方式的優點在於不太需要更改原始 model 的架構就可以把語者資訊加入到 feature 之中。缺點是 train 好之後，無法直接 test 在沒 train 過的語者上；以及要變更語者數量時，model 參數還要再改動。

(2) 請描述你如何將原本的程式碼改成訓練兩個語者的 voice conversion 程式。(0.5%)

主要是把 model 內的 down2d、up2d 及 convolutional layers 的 in channel 都減 2（因為少了兩個語者）。

另外因為 LabelBinarizer 在兩個語者的 output 會只有一維，所以我在 LabelBinarizer 的 output 後方再加一維相反的 label。

最後也把 data 資料夾內的 p3 及 p4 刪掉。

(3) 請問這個程式碼中，input acoustic feature 以及 generator output 分別是什麼呢？ (1%) Hint: 請研究一下 preprocess 時做了哪些事情。

Input acoustic feature 與 generator output 都是 mcep (Mel-Cepstral Coefficients)，其中 input 是 wav 再 normalize 過的。

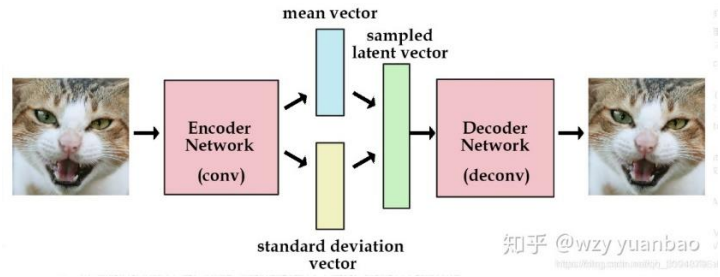
HW2-3 (1) 和 (2) 擇一回答 (4%)

(1) 請自己找一個不是 StarGAN-VC，也不是 HW2-1 的 model，實際 train 看看。請詳細描述 model 得架構， training objective，訓練時是否需要 paired data 等等。(4%) Hint: useful link

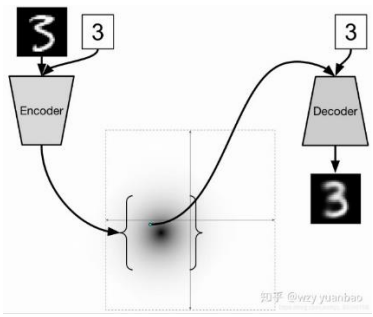
我使用 cVAE 來 train VC。

- **model 的架構與 training objective**

cVAE 是由 VAE 改良而來，而 VAE 是把原本 AE 的 encoder 的 output 改由 mean 跟 standard deviation 來合成出來，並在計算 loss 時，除了考慮 reconstruction 以外，也考慮 mean 跟 standard deviation 是否跟正態分布接近。架構如下圖：



而 cVAE 則是讓 VAE 的 encoder 與 decoder 在 input feature 的同時也知道這個 input 的 label 是什麼。架構如下圖：



- **實作的內容**

我基於 hw2-1 來改動。

在 train 的時候，讓 encoder 產生 mean 跟 standard deviation，並使用這兩個值來合成出 decoder 的 input：

```
# encode
means, log_var = self.encode_step(x, c)
# vae
std = torch.exp(0.5 * log_var)
eps = torch.randn_like(std)
z = eps * std + means
# decode
x_tilde = self.decode_step(z, c)
```

而 loss 除了原始的 reconstruction loss 之外，也計算 mean 跟 standard deviation 與正態分布的 KL diversions：

```
# loss
loss_rec = torch.mean(torch.abs(x_tilde - x))
loss_kl = torch.mean(- 0.5 * (1 + log_var - means * means - torch.exp(log_var)))
loss = loss_rec + loss_kl
```

另外，為了讓 encoder 可以知道 input feature 的 label，我也在 encoder 裡面開了 5 個 embedding，並在每次 convolution 的 block 中都把 embedding 加進去，跟原本 decoder 的作法類似：

```
def conv_block_emb(self, x, conv_layers, norm_layers, emb, res=True):
    # first layer
    x_add = x + emb.view(emb.size(0), emb.size(1), 1)
    out = pad_layer(x_add, conv_layers[0])
    out = F.leaky_relu(out, negative_slope=self.ns)
    # second layer
    out = out + emb.view(emb.size(0), emb.size(1), 1)
    out = pad_layer(out, conv_layers[1])
    out = F.leaky_relu(out, negative_slope=self.ns)
    # norm & drop
    for layer in norm_layers:
        out = layer(out)
    if res:
        x_pad = F.pad(x, pad=(0, x.size(2) % 2), mode='reflect')
        x_down = F.avg_pool1d(x_pad, kernel_size=2)
        out = x_down + out
    return out
```

- **model 的差別**

使用 cVAE 在 train 的時候，需要比較多的 iteration 才能讓 reconstruction 的 loss 下降。我認為這是因為 cVAE 還要考慮 KL diversion 的緣故。因為時間關係，我使用了 200000 個 iteration，結果聲音品質比 AE（hw2-1）差一些些，之後可能可以再加大 iteration 數。

而 cVAE predict 出來的結果，除了品質比較差以外，聽起來跟 AE 的語調及語速快慢也有些細微的不太一樣。

以 p2_338 轉為 p1 的檔案為例（2_1_338.wav）。AE 的語調及語速比較接近原本的 p2，而 cVAE 的語調比較平一點點，語速變化比較小一點點，跟 p2 細微的不太一樣，但也說不準是不是比較像 p1。