DLHLP HW2 Voice Conversion Report

組長GitHub id : lou-tun-chieh

組員：B05902111 婁敦傑 B05902010 張頌平

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

1. 敘述你是如何更改原本程式碼，附上修改部分的截圖，有更動原本模型參數也請一併列出。
    - 我們使用助教提供的 repo，原先這部分包含 Encoder, Decode, Classifier, Generator, PatchDiscriminator，我們只需使用 Encoder 和 Decoder 即可，更改程式碼的部分為我們只需訓練 solver.train(mode = pretrain_G) 的部分，另外 convert.py 因為原先是預設有使用 Generator，需將 convert_sp 中的 gen 參數改為 False。
    - 模型的 iteration 調整為 150000 次
    - 繳交音檔為 1_2_334.wav 和 2_1_338.wav
    -

main.py 只訓練 pretraio_G	solver.py 參數 gen = False

`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')`
	`def test_step(self, x, c, gen=False):`
	`  self.set_eval()`
	`  x = to_var(x).permute(0, 2, 1)`
	`  enc = self.Encoder(x)`
	`  x_tilde = self.Decoder(enc, c)`
	`  if gen:`
	`  x_tilde += self.Generator(enc, c)`
	`  return x_tilde.data.cpu().numpy()`

2. 將助教要求轉換的音檔轉成 source speaker 和 target speaker 的 interpolation，並比較分析 interpolated 的聲音和 p1 以及 p2 的關係。可從頻率高低、口音、語調等面向進行觀察。
    - 我們轉換成 interpolation 的方式是改 Decoder 中 embedding ，原先的 embedding 的部分只有 target，interpolation 則是改成將 target 和 source embedding 後取平均，Decoder 內有五個 embedding，我們都有換成 target 和 source embedding 的平均。
    - 我們覺得 interpolation 的聲音，在頻率的部分會介於 p1 和 p2 之間，語調的部分，我們認為會比較接近source(p1)，在整體聲音過程中前半部語調較高而後半部較低，target(p2)聲音的語調則是語調起伏較不明顯，口音也是更接近 source(p1)，說話的捲舌音相較於target(p2)輕微許多 。

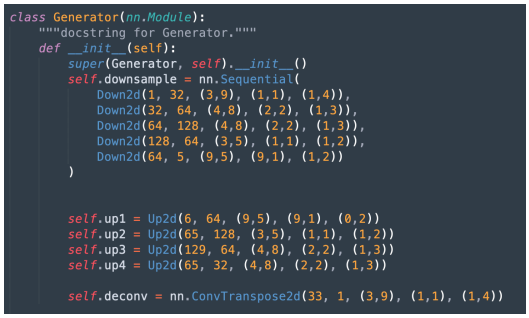
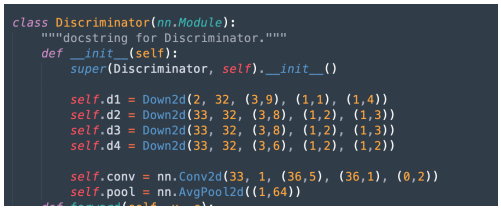
model.py 內 Decoder 的 embedding 改成平均

```
def interpolation(self, x, c, c2):
    # conv layer
    out = self.conv_block(x, [self.conv1, self.conv2], self.ins_norm1, (self.emb1(c) + self.emb1(c2)) / 2 , res=True )
    out = self.conv_block(out, [self.conv3, self.conv4], self.ins_norm2, (self.emb2(c) + self.emb2(c2)) / 2, res=True)
    out = self.conv_block(out, [self.conv5, self.conv6], self.ins_norm3, (self.emb3(c) + self.emb3(c2)) / 2, res=True)
    # dense layer
    out = self.dense_block(out, (self.emb4(c) + self.emb4(c2)) / 2, [self.dense1, self.dense2], self.ins_norm4, res=True)
    out = self.dense_block(out, (self.emb4(c) + self.emb4(c2)) / 2, [self.dense3, self.dense4], self.ins_norm5, res=True)
    emb = (self.emb5(c) + self.emb5(c2)) / 2
    out_add = out + emb.view(emb.size(0), emb.size(1), 1)
    # rnn layer
    out_rnn = RNN(out_add, self.RNN)
    out = torch.cat([out, out_rnn], dim=1)
    out = append_emb( (self.emb5(c) + self.emb5(c2)) / 2, out.size(2), out)
    out = linear(out, self.dense5)
    out = F.leaky_relu(out, negative_slope=self.ns)
    out = linear(out, self.linear)
    #out = torch.tanh(out)
    return out
```

    - 繳交音檔 1_2_334_inter.wav 和 2_1_338_inter.wav

HW2-2 (GAN) (2.5%)

1. 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？
    - 會將語者的 label 做 one-hot encoding ，再經過 Generator downsample，也就是 encoder 後，在upsample 的每一層將 one-hot vector 和 acoustic feature 串接在一起。
    - one-hot encoding 是一個比較簡單能夠提供語者資訊的方法，缺點為想要轉換成沒有聽過的語者必須重新訓練。
    - 可以用一些 pretrained model 將聲音轉換成語者資訊的 vector，像是 d-vector 或 i-vector，這樣的方法在遇到沒有聽過的語者，不需重新訓練即可轉換成功。
    - 模型的 iteration 為 100000 次
    - 繳交音檔 p1_to_p2.wav 和 p2_to_p1.wav
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。
    - 主要改動的部分為 data 資料夾的位置，將 p3 和 p4 拿出，並改 Generator 和 Discriminator 某些 layers 的維度。
    -

model.py 內 Generator 的維度	model.py 內 Discriminator 的維度
	

3. 請問這個程式碼中，input acoustic feature 以及 generator output 分別是什麼呢？
    - input acoustic feature 為 MCEP feature，在 preprocess 時會將 wav 轉換成 MCEP
    - generator output 為轉換後的 acoustic feature，也就是根據內容和語者重新製造的 acoustic feature

HW2-3

1. 想辦法 improve HW2-1或是 HW2-2 的 model (或是改一些有趣的東西)。
    1. 我們是 improve 2-1的 model，一開始我們有先將 L1 loss 改成 L2 loss，調整過後雜音有變少， 整體也比較清楚。接著我們試著加入 RNN 架構的 speaker encoder，但都無法成功轉換聲音，因此最後再加入 2-1內的classifier 幫助訓練，成功達到 Voice conversion。
    2. Speaker encoder 我們是放入 Decoder 內訓練，並和 2-1 一樣有五個 Speaker encoder，另外 content encoder 和 speaker encoder 的輸入為同一語者但不同內容，最後轉換聲音時會將 source 的 acoustic feature 放入 content encoder，target 的 acoustic feature 放入 Speaker encoder。

SpeakerEncoder 的架構

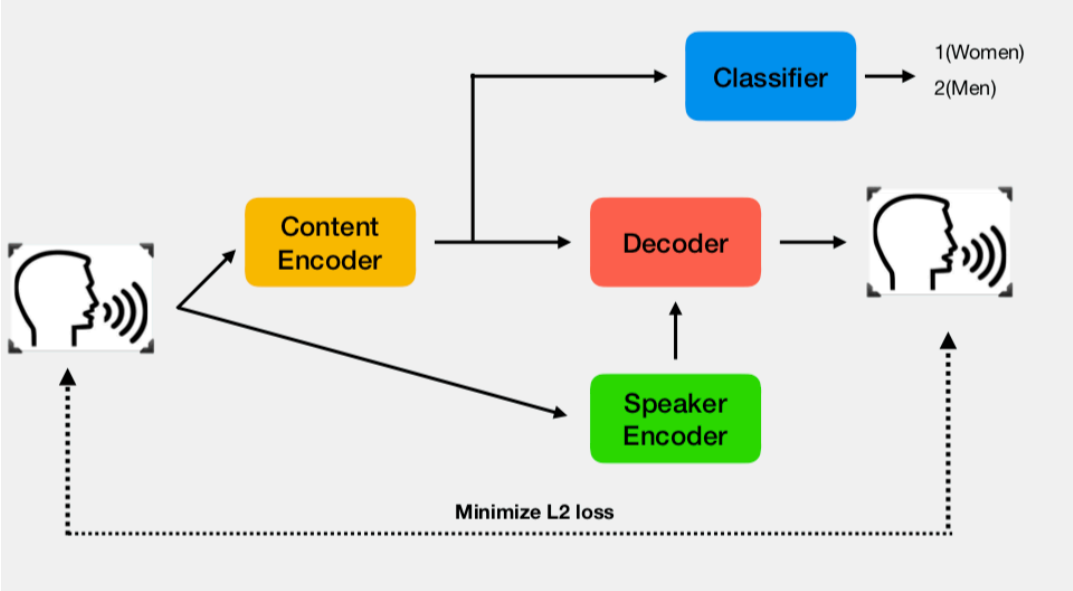
```python
class SpeakerEncoder(nn.Module):
    def __init__(self, input_dim=513, proj_dim=512, lstm_dim=384, num_lstm_layers=3):
        super().__init__()
        layers = []
        layers.append(LSTMWithProjection(input_dim, lstm_dim, proj_dim))
        for _ in range(num_lstm_layers - 1):
            layers.append(LSTMWithProjection(proj_dim, lstm_dim, proj_dim))
        self.layers = nn.Sequential(*layers)
```

每一層為LSTM與Linear的組成

```python
class LSTMWithProjection(nn.Module):
    def __init__(self, input_size, hidden_size, proj_size):
        super().__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.proj_size = proj_size
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.linear = nn.Linear(hidden_size, proj_size, bias=False)

    def forward(self, x):
        self.lstm.flatten_parameters()
        o, (_, _) = self.lstm(x)
        return self.linear(o)
```

整體的架構



3. 我們有比較 improvement 和 2-1 的結果，感覺並沒有太大的差異，僅在重音的部分 improvement 的結果稍微大聲一點，透過加入 SpeakerEncoder 後續我們可以試著加入未聽過的聲音嘗試 voice conversion，以解決可能某些data過少的情況。
4. 模型的 iteration 調整為 80000 次
5. 繳交音檔為 1_2_334.wav 和 2_1_338.wav
6. Reference
    - [Li et, al., ICASSP 2018] Generalized End-to-End Loss for Speaker Verification
    - [Zhang et, al., 2019 IEEE] Non-Parallel Sequence-to-Sequence Voice Conversion with Disentangled Linguistic and Speaker Representations