Kaldi 快速入門教學文件

國立臺灣大學資訊工程學系

多媒體資訊檢索實驗室

National Taiwan University

Multimedia Information Retrieval LAB.

C. Y. Adler Au (歐政鷹)

December, 2020

目錄

1. Kaldi 安裝 1

1.1 Kaldi 介紹 1

1.2 Kaldi 環境架設 1

1.3 Kaldi 環境測試 2

2. 範例架構 2

2.1 範例版本 2

2.2 內部架構 3

3. 建立個人範例 4

3.1 基本檔案準備 4

3.2 資料準備 4

3.3 發音詞典準備 6

3.4 語言模型準備 7

3.5 特徵抽取 8

3.6 模型訓練 9

3.6.1 GMM 模型訓練 9

3.6.2 DNN 模型訓練 10

3.7 模型測試 12

4. 參考資料 14

1. Kaldi 安裝

GitHub page: https://github.com/kaldi-asr/kaldi
官網：http://kaldi-asr.org/

1.1 Kaldi 介紹

Kaldi 是一套語音辨識相關用途的 open source 工具套件，底層程式主要由 C++ 寫成，由 shell script 作控制，並且其 I/O 大量依賴 pipeline ，因此最佳的執行環境為 UNIX 類系統。

Kaldi 的 GPU 計算部份需要使用 CUDA 技術，Kaldi 通常能支援最新的 CUDA 版本，在安裝 Kaldi 前也請確認系統已經有安裝好 CUDA。

1.2 Kaldi 環境架設

先將 Kaldi 從 GitHub clone 到本機端上

```
git clone https://github.com/kaldi-asr/kaldi.git
```

安裝流程可參考 INSTALL 檔案，主要分為兩個步驟，第一步為進入 tools/ 資料夾，依照 INSTALL 檔案中的流程安裝 Kaldi 所需要用到的函式庫。以下安裝指令可能需根據系統的 C++ compiler 版本作出調整，詳細可參閱 INSTALL 檔案的說明。

```
cd tools
extras/check_dependencies.sh
make -j 4
```

第二步移動到 src/ 資料夾，同樣依照 INSTALL 檔案中的流程設定 Kaldi 的編譯環境，一般本說只要根據以下設定來編譯即可，如有特殊需求則可調整 configure 設定進行編譯。

```
cd src
./configure --shared
make depend -j 4
make -j 4
```

以上步驟中 make -j 後的數字代表執行緒的數量，若 CPU 的運算能力較好，可調高執行緒的數量，能有效減少安裝的所需時間。

1.3 Kaldi 環境測試

在 egs/ 資料夾下放的是由不同開發者貢獻的範例程式，每個資料夾均對應一個 Corpus 及其範例程式，在這當中 YesNo 範例是一個簡單的範例可以讓你快速執行並測試 Kaldi 的環境。

Kaldi 的範例都習慣以 run.sh 作為主程式，範例中的其他指令及程式都是由 run.sh 直接或間接呼叫，只要執行 run.sh 就能重現範例結果。YesNo 範例無需修改就可以直接執行，包含資料的下載、整理、模型訓練、辨識及測試。我們只要進入 yseno 資料夾，執行 run.sh 即可

```
cd egs/yesno/s5
./run.sh
```

在執行期間程式會印出不同的訊息，分別代表從資料下載到辨識的過程中所執行的各個動作，在 YesNo 範例中，最後一行的輸出資訊應如下：

```
%WER 0.00 [ 0 / 232, 0 ins, 0 del, 0 sub ]
```

這行資訊是指訓練好的模型對測試資料集的測試結果，在 232 個詞中辨識錯誤的詞有 0 個，全部都辨識正確。所以 WER 為 0，插入錯誤 (ins)、刪除錯誤 (del) 及替換錯誤 (sub)均為 0。只要在 YesNo 範例中能取得上述結果，基本上代表你的 Kaldi 環境已經安裝成功。

2. 範例架構

2.1 範例版本

在任一個 Kaldi 範例中通常會看到如下的架構：

```
./
├── s5/
└── README.txt
```

其中包含一個簡單的 readme 檔案描述該範例的 corpus、訓練流程、開發者的測試結果等資訊。以及一個由英數字組合而成的資料夾，該資料夾代表範例的版本號，通常以 s 開頭後接數字，數字越大代表版本越新，一般當新版本被上傳後，舊有的版本就不會再更新或被刪除。

在語者辨識及影像的範例中，會以 v 取代 s 來命名，命名規則與語音辨識範例的命名規則一致。

2.2 內部架構

實際進入到任一個語音辨識的範例裡，裡面會有不同的資料夾與檔案，當中可能包括開發者定義的特殊檔案或資料夾，但大部份範例均會包含以下的檔案：

```
s5/
├── cmd.sh
├── path.sh
├── conf/
├── local/     // specific code and resource of the project
├── run.sh
├── RESULT
├── steps/     // -> ../../wsj/s5/steps/
└── utils/     // -> ../../wsj/s5/utils/
```

- cmd.sh 定義不同工作的執行環境，用於設定程式是在單機上還是在多台機器的 cluster system 上執行。不同範例的 cmd.sh 內容大同小異，其基本設定如下：

  ```
  export train_cmd="queue.pl"
  export decode_cmd="queue.pl -mem 2G"
  ```

  以上兩行分別指定在訓練及辨識時的執行環境，queue.pl 是供 GridEngine cluster system 所使用設定，若需在單機上執行並沒有特定的記憶體限制等需求，即可以把設定都改成 run.pl。

  ```
  export train_cmd="run.pl"
  export decode_cmd="run.pl"
  ```

- path.sh 定義了各種環境變數的位置，包括各函式庫的位置、Kaldi 程式編譯的位置等。一般各範例的 path.sh 都是相同，無需作出修改。
- conf 資料夾下是不同的設定檔，包括特徵抽取、辨識解碼等設定。
- local 資料夾下是該範例專用的指令程式或檔案，通常包含資料準備與處、NN 訓練流程、NN 訓練架構等指令。
- run.sh 是主執行程式，從資料準備到測試的所有流程都整合在 run.sh 下。
- RESULT 紀錄了開發者在各訓練模型中的測試結果。
- steps 及 utils 是兩個軟連結檔案，除了 wsj 範例自身外其他所有範例均會將以上兩個檔案指向 wsj 範例下的對應資料夾。steps 中包含了訓練相關的所有指令程式。utils 中包含資料處理、驗證，檔案處理或其他的各種指令程式。

3. 建立個人範例

3.1 基本檔案準備

參考現有的範例，我們可以對自己擁有的語料進行訓練及測試。首先需要建立一個範例資料夾供這次的訓練所使用。

在 egs/ 資料夾下，根據語料的名稱建立一個版本號 s0 的範例，並根據上一章節的介紹準備以下的檔案：

```
cd egs/
mkdir -p $corpus/s0
cd corpus/s0                # 進入範例資料夾
mkdir local/ conf/          # 建立 local & conf 資料夾
ln -s  ../../wsj/s5/steps  # symlink steps 資料夾
ln -s ../../wsj/s5/utils   # symlink utils 資料夾
cp ../../wsj/s5/{path.sh,cmd.sh} ./   # 複製 path.sh & cmd.sh
```

其中要注意是否需要對 cmd.sh 作出修改，如參照上述的架構，無需對 path.sh 作出修改。但當範例資料夾並非依照以上架構建立時，需注意 path.sh 中是以相對位置來設置環境變數，當架構改變時會導致環境變數的設定錯誤，此時需要將 path.sh 中第一行所指定的路徑重新指向到正確的 Kaldi 資料夾位置。

```
export KALDI_ROOT=`pwd`/../../..  # 需重新指向到 Kaldi 根目錄
```

3.2 資料準備

在準備語音辨識模型環境時，通常會將資料分為訓練 (train)、開發 (dev)及測試資料 (test) 三個 subset，並視擁有的資料量及需求來進行劃分。

Kaldi 習慣將資料整理成所指定格式的檔案後放在 data/ 資料夾下，整體架構如下：

```
./data/
└── {train,test,dev}
    ├── wav.scp
    ├── text
    ├── utt2spk
    └── spk2utt
```

- wav.scp：紀錄語句資訊 (音訊)檔案路徑位置的 list，第一欄是語句 ID，第二欄是對應檔案所在位置，以空格隔開。語句 ID 由用戶自行定義，除不要重複外沒特別規範。

```
#<utterance_id> <file_path>
utt_1 ./dataset/wav1.wav
...
```

  wav.scp 同時支援以 pipeline 方式傳輸檔案資料，由於 Kaldi 只支援 wav 格式音檔，當語句音檔並非 wav 格式時，則可透過 pipeline 將檔案轉換為 wav 格式的指令直接記錄在 wav.scp 上，節省直接輸出轉換後檔案的儲存空間。

```
#<utterance_id> <pipeline>
utt_1 flac -c -d -s ./test.flac |
...
```

- text：紀錄每一個語句的文本內容，第一欄是語句 ID，第二欄以後是文本內容，以空格隔開。在此文本應已經完成 normalize 處理，如英文大小寫統一、中文分詞等，每一句標記文本透過語句 ID 對應到所屬的語句。

```
#<utterance_id> <text>
utt_1 ALL WORD ARE UTTERANCE TEXT AFTER THE FIRST COLUMN
...
```

- utt2spk：紀錄每一個語句與語者的對應關係，第一欄是語句 ID，第二欄語者 ID，以空格隔開。若每一個語句均對應到一名獨立語者時可以將語者 ID 設定為語句 ID。

```
#<utterance_id> <speaker_id>
utt_1 spk_1
utt_2 spk_1
utt_3 spk_2
...
```

  在建立 utt2spk 時，要注意 speaker_id 應設定成 utterance_id 的 prefix，這跟 Kaldi 的處理方式有關，各檔案的排序結果會影響到 Kaldi 在實作上的運作，如果 utt2spk 中語者的排序方式與 spk2utt 中語者的排序方式不一樣，就會導致在根據語者 ID 或語句 ID 的索引出錯，因此 Kaldi 建議以 speaker_id 作為 utterance_id 的 prefix，確保兩者的索引排序是正確的。
- spk2utt：紀錄每一個語者與語句的對應關係，與 utt2spk 互相對應，第一欄是語者 ID，第二欄以後是與該語者相關的語句 ID，以空格隔開。若每一個語句均對應到一名獨立語者時檔案內容會與 utt2spk 相同。

```
#<speaker_id> <utterance_id>
spk_1 utt_1 utt_3 ...
spk_2 utt_2 ...
...
```

在上述架構中，wav.scp, text 及 utt2spk 都需要自行產生，而 spk2utt 可以透過 Kaldi 提供的轉換程式由 utt2spk 轉換獲得。

```
perl ./utils/spk2utt_to_utt2spk.pl ./data/utt2spk > ./data/spk2utt
```

在所有檔案準備完成後，Kaldi 提供了一支檢查檔案程式，它會為原有的 data 資料夾作一個備份，刪除沒有同時出現在各檔案中的語句資訊，並修正語句排序。

```
./utils/fix_data_dir.sh ./data
```

3.3 發音詞典準備

在資料準備階段，我們在 text 檔案中是以文字文本作標記，而聲學模型實際上需要以文本的發音作訓練，因此我們需要提供以下檔案以描述文本與發音的對應關系。

```
./data/dict/
├── lexicon.txt
├── lexiconp.txt
├── extra_questions.txt
├── nonsilence_phones.txt
├── optional_silence.txt
└── silence_phones.txt
```

- lexicon.txt：紀錄每個詞對應的發音序列，發音序列一般以 phone 為單位，並視需求結合 tone, stress 等資訊使用。除此之外通常還會加入“!SIL”用來表示靜音的詞、其發音是靜音音素，及“<SPOKEN_NOISE>”&“<UNK>”分別表示雜訊及未登錄詞，其發音都是 SPN。

```
#<word> <phones>
!SIL SIL
<SPOKEN_NOISE> SPN
<UNK> SPN
WAS W AA1 Z # sample of word ‘was’
...
```

- lexicon.txt：對於一詞多音的情況，Kaldi 支援定義發音機率，對不同的發音加以區分，此檔案則是帶發音機率的發音詞典名稱，lexicon.txt 及 lexicon.txt 至少要有一個存在，如果同時存在，將優先使用 lexiconp.txt。

```
#<word> <pron-prob> <phones> (0.0 < pron-prob <= 1.0)
!SIL 1.0 SIL
<SPOKEN_NOISE> 1.0 SPN
<UNK> 1.0 SPN
WAS 0.3 W AA1 Z # sample of word ‘was’
WAS 0.7 W AH0 Z # sample of word ‘was’
...
```

- nonsilence_phones.txt：紀錄所有非靜音的 phone。
- silence_phones.txt：紀錄所有可以用來表示無效語音 (non-speech) 的 phone，如 SIL 及 SPN。
- optional_silence.txt：定義了用於填充詞間靜音的 phone，如 SIL。
- extra_question.txt：列出了建置 phone 的聲學上下文決策樹時會遇到的分群問題，每行對應一個分群問題。如所有表示無效語音的 phone 應為一群，具有不同 tone 的 phone 表應各為一群等。

發音詞典的準備可基於現有的發音詞典來完成，如英文有 CMUDict，中文有 CEDict 等，不同的範例中也有提供各自的發音詞典準備程式可供參考或使用。Kaldi 提供了一個用於檢查發音詞典資料夾內容的和格式是否符合要求的程式，可透過以下方式使用，如有檔案內容不符合要求，則按照錯誤訊息修改即可。

```
perl ./utils/validate_dict_dir.pl ./data/dict
```

3.4 語言模型準備

聲學模型中儲存的資訊僅是各 HMM 狀態的觀察機機率分佈，只透過以上的機率分佈難以在聲學辨識中取得好的結果，在實作上還需要透過語言學模型的輔助來更準確地辨識語音。

在不同的範例中，使用 N-gram 語言模型即可在語音辨識上取得不錯的效果，N 通常設定為 3 或 4。我們可以利用網絡上已有的 N-gram 模型或自行利用不同的文本資料作訓練，Kaldi 有提供 3-gram/4-gram 模型的訓練程式 train_lm.sh 作使用，我們也能透過 SRILM Toolkit 訓練 N-gram 模型。

當準備好發音詞典及 N-gram 模型後，就可以透過 Kaldi 提供的程式產生語言模型 FST 與其他相關檔案。

```
./utils/prepare_lang.sh ./data/dict "<UNK>" ./data/lang/tmp ./data/lang
./utils/format_lm.sh ./data/lang local/n-gram.mdl ./data/dict/lexicon.txt \
   ./data/lang/lm_model
```

完成後應得到如下方的架構。

```
./data/lang/
├── lm_model/
│   ├── G.fst
│   ├── L_disambig.fst
│   ├── L.fst
│   ├── oov.int
│   ├── oov.txt
│   ├── topo
│   ├── words.txt
│   ├── phones.txt
│   └── phones/
├── L_disambig.fst
├── L.fst
├── oov.int
├── oov.txt
├── words.txt
├── phones.txt
├── phones/
└── tmp/
```

3.5 特徵抽取

我們需要先從語音訊號中抽取有用的資訊作為訓練時使用的聲學特徵，Kaldi 主要使用的聲學特徵為 MFCC 及 ivector，並透過 config 檔設定特徵抽取的參數，Kaldi 預設 config 檔均放在 conf 資料夾下，下方列出幾個可能會使用到的 config 檔，各 config 檔的參數設定可參考各範例及 Kaldi 文件的說明，特徵抽取的程式將在下一章節連同訓練程式一同展示。

```
./conf
├── mfcc.conf
├── pitch.conf
└── mfcc_hires.conf
```

以下為 Kaldi 特徵抽取的步驟，第一步從語音檔案中分析聲學特徵，三個參數分別為資料目錄、執行目錄，用於儲存執行紀錄及特徵目錄，用於儲存特徵檔案。第二步為 CMVS 係數的計算，三個參數的意義與第一步相同。在以上兩步執行結束後，data 資料目錄會新增 cmvn.scp, feats.scp 等檔案用於記錄各語句與其特徵檔案位置的對應關系。

```
# training, dev & test data
for subset in train test dev; do
  echo "$subset: making mfccs"
  steps/make_mfcc.sh --mfcc-config conf/mfcc.conf data/$subset mfcc/$
mfcc/$subset/log mfcc/$subset
  steps/compute_cmvn_stats.sh data/$subset $mfcc_dir/$subset/log $mfcc_dir/$subset
done
```

3.6 模型訓練

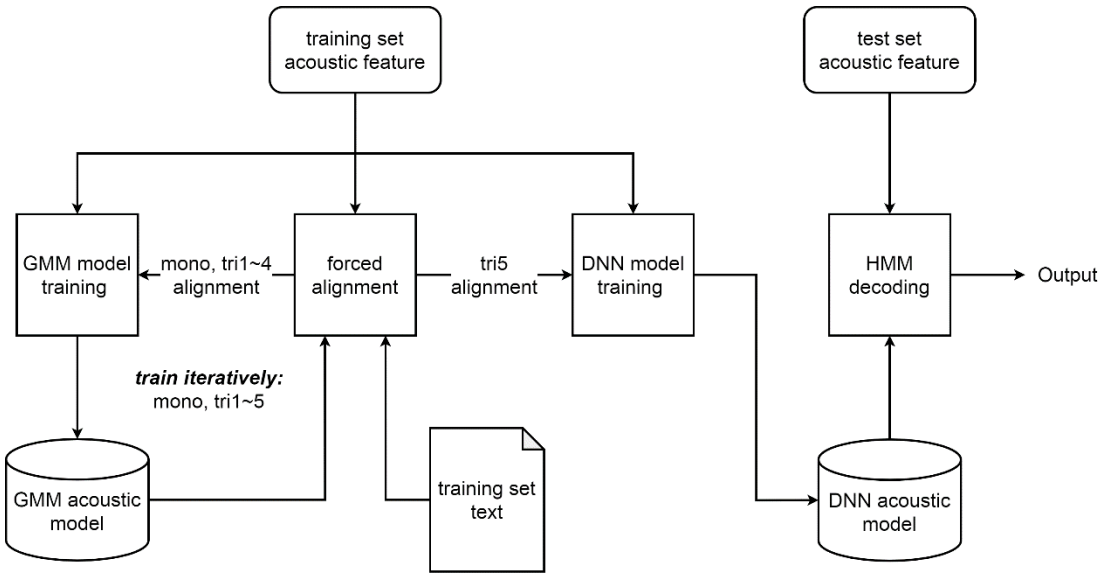

圖 3.1. Kaldi 聲學模型訓練流程圖

上圖為 Kaldi 的聲學模型訓練流程，主要分為 GMM 模型及 DNN 模型兩部分，我們會先訓練出 GMM 模型，透過 GMM 模型強制對位的結果進行 DNN 模型的訓練。

首先，以 mono-phone 訓練一個初始的 GMM 模型，並利用此模型對訓練資料文本作強制對位，強制對位的結果將會用於訓練一個 tri-phone GMM 模型。tri-phone 模型會重複上述 mono-phone 模型的訓練步驟 4 次，過程中會訓練出另外 4 個 tri-phone 模型，使用最後被訓練出來的 tri-phone 模型作為實驗中的 GMM 聲學模型。

在圖中，mono 代表初始的 mono-phone GMM 模型，tri1-tri5 分別依順序代表迭代訓練過程中 5 個 tri-phone 模型，當中 tri5 亦是最終的 GMM 聲學模型。

在 GMM 模型訓練完成後 (tri5 GMM model)，如同在 GMM 訓練流程中的步驟，GMM 模型的強制對位結果將用來訓練一個 DNN 聲學模型。在訓練開始前一般會先透過資料擴充 (data augmentation) 方法，依照訓練資料語音檔案的速度，分別作 0.9 倍及 1.1 倍的調整後與原有資料合併取得 3 倍數量的訓練資料，最後以鏈模型 (chain model) 來訓練 DNN 聲學模型。在不同情況下，我們可以加入其他的資料擴充方式以取得更多的訓練資料作訓練。

3.6.1 GMM 模型訓練

以下為 Kaldi GMM 訓練流程的實作程式範例，exp/ 是用於存放實驗數據的

資料夾。steps/train_*.sh 為訓練程式，各程式的後綴分別代表在訓練過程中所加入的額外處理方式，如 sat 代表 speaker adaptive training，lat 代表 linear discriminant analysis 等。steps/align_*.sh 為強制對位程式，針對不同訓練方式會需要用到對應的強制對位程式。

```
# mono phone (mono)
steps/train_mono.sh data/train data/lang exp/mono
# tri-phone 1 (tri1)
steps/align_si.sh data/train data/lang exp/mono exp/mono_ali
steps/train_deltas.sh 2500 20000 data/train data/lang exp/mono_ali exp/tri1
# tri2
steps/align_si.sh data/train data/lang exp/tri1 exp/tri1_ali
steps/train_deltas.sh 2500 20000 data/train data/lang exp/tri1_ali exp/tri2
# tri3
steps/align_si.sh data/train data/lang exp/tri2 exp/tri2_ali
steps/train_lda_mllt.sh 3500 55000 data/train data/lang exp/tri2_ali exp/tri3
# tri4
steps/align_fmllr.sh data/train data/lang exp/tri3 exp/tri3_ali
steps/train_sat.sh 3500 55000 data/train data/lang exp/tri3_ali exp/tri4
# tri5
steps/align_fmllr.sh data/train data/lang exp/tri4 exp/tri4_ali
steps/train_sat.sh 3500 100000 data/train data/lang exp/tri4_ali exp/tri5
```

3.6.2 DNN 模型訓練

完成 GMM 模型的訓練後，我們需要準備 DNN 模型的訓練，首先需視需求進行資料擴充，以下程式碼為速度調整及音量調整的範例。第一行將原始音檔分別依 0.9 倍及 1.1 倍速度作調整後連同原始音檔將資訊儲存在 data/train_sp 裡，第二行再對該目錄下的音檔作隨機的音量調整。

Kaldi 在資料擴充的實作上大多以 pipeline 的方式完成，並不會產生大量檔案佔用儲存空間，從輸出資料夾的 wav.scp 中可以觀看出 Kaldi 對不同資料擴充的實作方式。

在完成資料擴充後，僅記需要對新的資料再次抽取聲學特徵，才能在後續步驟中完成訓練，同時要注意在之後的步驟中所指的訓練資料均是已經完成資料擴充後的資料。

```
# do speed-perturbation on the training data
utils/data/perturb_data_dir_speed_3way.sh data/train data/train_sp
# do volume-perturbation on the training data
utils/data/perturb_data_dir_volume.sh data/train_sp
```

在 DNN 模型訓練中，除了使用 MFCC 作為聲學特徵外，很多實驗範例同時也會加入具有語者資訊的 i-vetcor 一同作為神經網絡輸入特徵的一部分進行訓練中，在實作中能夠有效地提升語音辨識率。以下為從資料中抽取的 i-vector 特徵的流程：

```
echo "$0: computing a PCA transform from the hires data."
steps/online/nnet2/get_pca_transform.sh --max-utts 10000 --subsample 2 \
    --splice-opts "--left-context=3 --right-context=3" \
    data/train_sp_hires exp/nnet3/pca_transform

# training the diagonal UBM, use 512 Gaussians in the UBM.
steps/online/nnet2/train_diag_ubm.sh --nj 30 --num-frames 700000 \
  --num-threads $num_threads_ubm \
  data/train_sp_hires 512 exp/nnet3/pca_transform exp/nnet3/diag_ubm

# training the iVector extractor
steps/online/nnet2/train_ivector_extractor.sh --nj 10 \
  --num-processes 4 data/train_sp_hires exp/nnet3/diag_ubm exp/nnet3/extractor

# extracting iVectors for train/test data
steps/online/nnet2/extract_ivectors_online.sh --nj 60 \
  data/train_sp_hires exp/nnet3/extractor exp/nnet3/ivectors_train_sp_hires
steps/online/nnet2/extract_ivectors_online.sh --nj 10 \
  data/test_hires exp/nnet3/extractor exp/nnet3/ivectors_test_hires
```

在完成資料擴充及特徵抽取後，需要對 chain 模型訓練先作準備，準備工作主要為產生一個用於 chain 訓練的語言目錄、對訓練資料作強制對位、產生訓練所需的決策樹，準備工作程式流程如下：

```
cp -r data/lang data/lang_chain
silphones=$(cat $lang/phones/silence.csl)
nonsilphones=$(cat $lang/phones/nonsilence.csl)
steps/nnet3/chain/gen_topo.py $nonsilphones $silphones > data/lang_chain/topo
```

```
steps/align_fmllr_lats.sh data/train_sp data/lang_chain \
  exp/tri5 exp/chain/tri5_ali_lat


steps/nnet3/chain/build_tree.sh --frame-subsampling-factor 3 \
  --context-opts "--context-width=2 --central-position=1"
  9000 data/train_sp data/lang_chain exp/tri5_ali_sp exp/chain/tree
```

最後只需要定義好 network 的架構即可開始 chain 模型的訓練，由於架構的建立及訓練涉及到各種不同的參數設定，建議可透過參考已有範例的架構及 Kaldi 文檔的說明進行調整後再進行訓練。

Kaldi 透過 steps/nnet3/chain/train.py 呼叫不同的 C++ 執行檔與 Bash 指令來執行整個訓練流程，在訓練前確保已經設定使用 GPU 來進行訓練，加速訓練的進行。

3.7 模型測試

在模型訓練完成後，則可以準備對測試資料進行測試，測試主要分為兩個步驟，第一步為建立解碼所需要的解碼圖，第二步為對資料進行解碼。針對 GMM 模型及 DNN 模型在製圖及解碼上需要用到不一樣的參數設定，而在解碼程式中 Kaldi 預設會對解碼結果進行 scoring 運算，Kaldi 假設開發者在 local/ 資料夾下放了一支名為 local/score.sh 的程式，並執行該程式進行 CER 或 WER 的運算，因此在執行解碼要前視個案需求將 steps/scoring/ 目錄下的 score_kaldi_wer.sh 或 score_kaldi_cer.sh 其中一支程式 symlink 到 local/ 目錄下並命名為 score.sh，才能在解碼後取得測試結果的 CER 或 WER。

製圖及解碼的程式碼範例如下，要注意所提供的都是已經完成特徵抽取的測試資料，並假設將 chain 模型儲存在 exp/chain/tdnn/ 資料夾下。

```
# GMM model
utils/mkgraph.sh data/lang/lm_model exp/tri5 exp/tri5/graph
steps/decode.sh exp/tri5/graph data/test exp/tri5/decode_test


# DNN model
utils/mkgraph.sh --self-loop-scale 1.0 data/lang/lm_model \
  exp/chain/tdnn exp/chain/tdnn/graph
steps/nnet3/decode.sh --acwt 1.0 --post-decode-acwt 10.0 \
  --online-ivector-dir exp/nnet3/ivectors_test_hires \
  exp/chain/tdnn/graph data/test_hires exp/chain/tdnn/decode_test
```

在完成 scoring 後，Kaldi 會將結果儲存在 scoring_kaldi/ 資料夾下，資料夾架構如下，當中 {cer,wer} 代表該檔案會根據 scoring 所使用的方式而命名，${wip} 及 ${lmwt} 則是根據 scoring 的參數設定命名。

```
./exp/…/scoring_kaldi/
├── {cer,wer}_details/
│   ├── {cer,wer}_bootci
│   ├── lmwt
│   ├── per_utt
│   ├── per_spk
│   ├── ops
│   └── wip
├── best_{cer,wer}
├── test_file.txt
├── test_file.chars.txt
├── penalty_${wip}/
│   └── ${lmwt}.txt/
└── log/
```

- best_{cer,wer}：檔案代表在不同的參數組合中的最佳測試結果，檔案內容實際上如同 YesNo 範例中最後一行的輸出資訊，當中最後一列代表測試結果檔案的實際位置，測試結果檔案除了 CER 或 WER 外還額外提供 SER 結果供開發者參考。
- test_file.txt：為測試資料的文本內容，與測試資料的 text 檔案內容相同，只在計算 wer 時輸出。
- test_file.chars.txt：為測試資料的文本內容，與測試資料的 text 檔案內容接近，但所有文字內容都以 char 為一個單位，只在計算 cer 時輸出。
- penalty_${wip}：資料夾下每個檔案為依照各參數組合進行解碼的結果，格式與 test_file.txt 或 test_files.chars.txt 相同。
- {cer,wer}_details：資料下裡檔案為最佳測試結果的分析檔案。當中 lmwt 及 wip 紀錄了測試參數設定，per_utt 列出了每一測試語句中標注內容與解碼內容的差異，per_spk 列出了以語者為單位的錯誤情況，ops 以 word 或 char 為單位列出有出現的錯誤情況。

4. 參考資料

[1] 陳果果, 都家宇, 那興宇, 張俊博, “AI 語音辨識：用 Kaldi 實作應用全集”, 2020

[2] Kaldi documentation, “https://kaldi-asr.org/doc/”

[3] Kaldi GitHub, “https://github.com/kaldi-asr/kaldi”