DSP Final

A. 研究動機

我們想做的主題是 speech recognition，也希望嘗試上課教過的 GMM 相關模型，所以使用了 Kaldi Speech Recognition Toolkit 來建立 GMM、SGMM 等模型。除了比較傳統的 GMM 方法外，我們也找了應用深度學習的模型，選用的是今年九月由 OpenAI release 的 Whisper model，並直接透過 Hugging Face 使用。希望藉由這個 project 來學

習兩種模型的應用方法，作為研究語音辨識的入門，也期許之後能夠就此基礎繼續更深入的研究。

B. Kaldi

Reading Report

論文名稱: The Kaldi Speech Recognition Toolkit

摘要

1. Kaldi是一個是以C ++編寫，用於語音辨識的open-source toolkit。
2. Kaldi依賴OpenFst來實作finite-state transducers。
3. Kaldi支援任意長度的語音內容。
4. Kaldi的acoustic model有支援標準的GMM(Gaussian mixture models) 和 SGMM(subspace Gaussian mixture models)。

一、簡介

1. Kaldi在http://kaldi.sf.net/ 上供使用。
2. Kaldi的重要特性包括

   (1) 使用OpenFst來整合Finite State Transducers

   (2) 包含封裝了BLAS與LAPACK兩個線性代數庫的matrix library

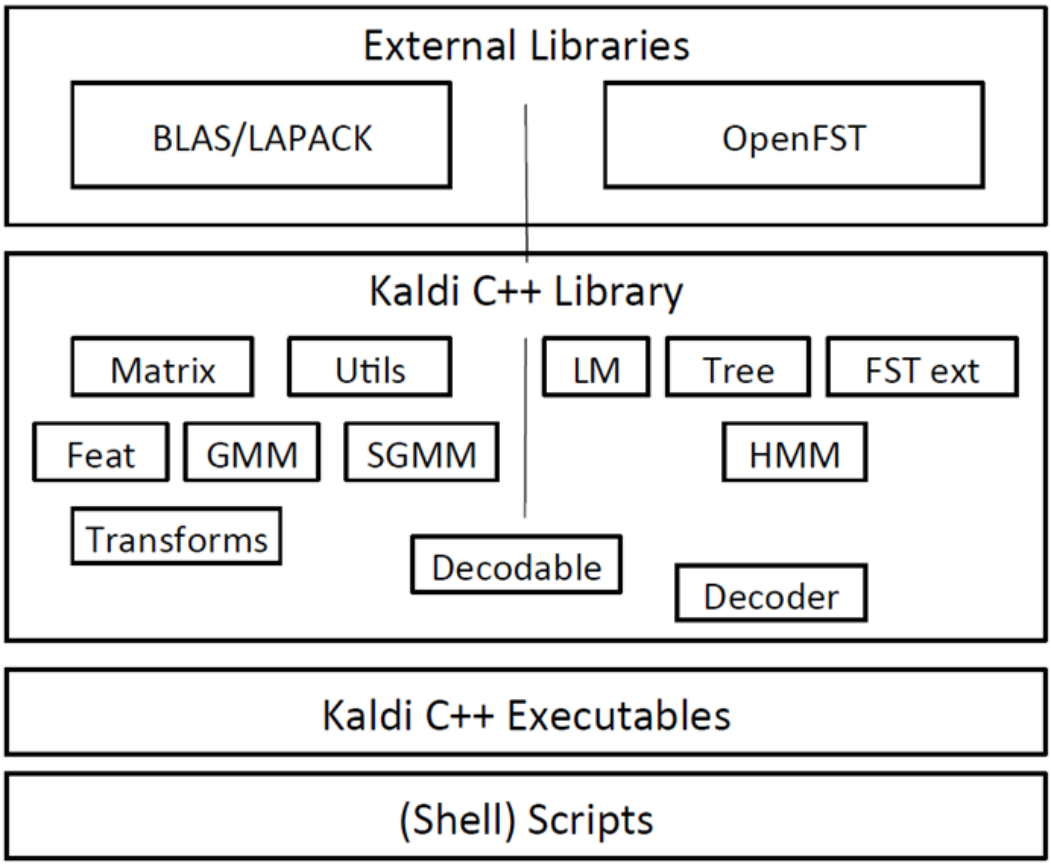
   (3) 通用的演算法

   (4) 使用最不受限制的Apache v2.0授權條款

   (5) 適用於Linguistic Data Consortium (LDC) 的數據集

   (6) 所有的程式碼都經過測試。
3. 比起HTK，Kaldi的優點是更現代、更彈性、更具結構性的程式碼和更好的WFST和數學的背景。

二、Kaldi概觀



1. External Libraries

   OpenFst(finite-state)、numerical algebra libraries(BLAS/LAPACK)。

2. Kaldi C++ Library

   分成兩部分，各對應到一個external library，透過DecodableInterface這個module將兩部分連結起來。

3. Kaldi C++ Executables

4. (Shell) Scripts

   若要使用Kaldi的功能，可以使用其以C++編寫的command-line tools。

三、特徵擷取

1. 創建標準的MFCC和PLP特徵，提供合理的預設參數，也供使用者自行調整一些參數，如mel bins的數量，頻率截斷值的最大值和最小值等。
2. 支援常見的特徵提取的方法，包含VTLN、cepstral mean and variance normalization、LDA、STC/MLLT與HLDA等。

四、聲學模型

Kaldi的目標是可以支援一些像diagonal GMMs這種傳統的模型和SGMM，同時也可以很容易地延伸到新的模型。

a. Gaussian mixture models

1. 支援diagonal與full covariance structures的GMM。

b. GMM-based acoustic model

1. AmDiagGmm這個class代表DiagGmm object的集合，由對應於其context-dependent之HMM states的 “pdf-ids”為其索引，這個class不代表任何HMM結構，只是density的集合，也就是GMM。

c. HMM Topology

1. Topology結構允許非放射的state，且允許使用者預先指定不同的HMM狀態下pdf的綁定

d. Speaker adaptation

1. 支援MLLR與fMLLR，兩者都可以使用迴歸樹來估計多個變換。
2. 也支援對VTLN的線性估計以進行speaker的normalization，或是傳統的特徵級VTLN，或一種更通用的性別normalization方法，稱為exponential transform。
3. MLLR與VTLN都可以用在acoustic models的speaker adaptive training (SAT) 上。

e. Subspace Gaussian Mixture Models

1. AmSgmm這個class代表所有pdf的集合，沒有代表SGMM裡單獨pdf的class。

五、語音決策樹

1. 建立語音決策樹的目標是有效率地處理任意長度的上下文，也就是避免列舉所有上下文，並使其能夠支援廣泛的方法。
2. 傳統的方法是，在每個monophone的每個HMM state下，都有一個決策樹來詢問有關左邊跟右邊的phone的問題。
3. 在Kaldi的框架中，決策樹根可以在phone之間跟phone state之間共享，可以詢問關於context window中的任何phone和關於HMM狀態的問題。

六、語言模型

1. 因為Kaldi使用基於FST的框架，所以原則上它應該要可以使用任何可以表示為FST的語言模型。
2. Kaldi使用IRSTLM toolkit來做LM pruning。

七、產生解碼圖

1. Kaldi的訓練和解碼算法是WFSTs。
2. 在傳統方法中，解碼圖上的輸入符號pdf-id對應到與上下文有關的state。但因為Kaldi允許不同的phone共享相同的pdf-id，所以使用傳統方法會遇到很多問題，包含沒辦法確定FST、和沒有從Viterbi path到FST的足夠資訊以求出phone sequence或訓練其transition probability。
3. 為了解決這些問題，Kaldi在FST的輸入上放了一個更fine-grained的整數identifier，稱為transition-id。它對pdf-id和那個phone的topology裡的transition進行編碼。
4. Transition-id和模型中的transition probability參數是一對一mapping的關係，作者決定在不增加解碼圖大小的情況下盡可能去細化transitions。

八、解碼器

1. “Decoder”是指實現核心解碼算法的C++ class。
2. Decoder不需要特定類型的聲學模型，只需要一個物件，其有提供某種聲學模型分數的函數之簡單介面。
3. Command-line的解碼十分簡單，只進行一次解碼，而且都專用於一種解碼器和一種聲學模型。

九、實驗

用egs/rm/s1和egs/wsj/s1的script來做實驗。

1. Basic Triphone System on Resource Management

	Test set				
	Feb’89	Oct’89	Feb’91	Sep’92	Avg
HTK	2.77	4.02	3.30	6.29	4.10
Kaldi	3.20	4.21	3.50	5.86	4.06

→ HTK和Kaldi的WER本質上是相同的。

2. Basic Triphone System, WSJ, 20k Open Vocabulary, Bigram LM, SI-284 Train

	Test set	
	Nov'92	Nov'93
Bell	11.9	15.4
HTK (+GD)	11.1	14.5
KALDI	11.8	15.0

→ 跟上一個實驗的結果很相似，但這裡HTK的表現稍微好一些，因為它是gender-dependent的。

3. Result on RM and on WSJ, 20k Open Vocabulary, Bigram LM, Trained on Half on SI-84

	RM (Avg)	WSJ Nov'92	WSJ Nov'93
Triphone	3.97	12.5	18.3
+ fMLLR	3.59	11.4	15.5
+ LVTLN	3.30	11.1	16.4
Splice-9 + LDA + MLLT	3.88	12.2	17.7
+ SAT (fMLLR)	2.70	9.6	13.7
+ SGMM + spk-vecs	2.45	10.0	13.4
+ fMLLR	2.31	9.8	12.9
+ ET	2.15	9.0	12.3

→ 比較Kaldi支援的不同功能使用起來的表現，發現Splice-9 + LDA + MLLT + SGMM + spk-vecs + ET是當中表現最好的。

十、結論

1. Kaldi是一個免費且開源的語音辨識toolkit。
2. Kaldi支援對任意上下文長度的context-dependent phone進行建模。
3. Kaldi支援所有可以用maximum likelihood估計的常見技術。
4. Kaldi支援SGMM。

Computer Project

使用Kaldi的情境大致分成兩塊，HMM-GMM與神經網路。這邊使用TIMIT這個dataset來進行實驗，由於自身的電腦沒有GPU，於是就不執行範例shell script(run.sh) 中DNN的部分。

以下為成功執行kaldi/egs/timit/s5/run.sh所需指令:

(1) docker run -it kaldiasr/kaldi:latest bash

(2) 下載TIMIT dataset至欲存放的路徑
(3) 將run.sh中的timit=/mnt/matylda2/data/TIMIT/timit改為dataset存放的路徑
(4) chmod +x ./run.sh
(5) tools/extras/install_irstlm.sh
(6) apt-get install bc
(7) 註解掉run.sh中的exit 0
(8) chmod +x ./run.sh
(9) 將cmd.sh中的queue.pl皆改為run.pl
(10) cd src/sgmm2
make
cd src/sgmm2bin
make
(11) ./run.sh (總執行時間: 1小時45分鐘11秒)

結果:

Training & Decoding	dev (%WER)	test (%WER)
MonoPhone	31.6	31.7
tri1 : Deltas + Delta-Deltas	24.8	26.3
tri2 : LDA + MLLT	22.7	23.7
tri3 : LDA + MLLT + SAT	20.4	22.3
SGMM2	17.6	19.6
MMI + SGMM2	18.4	19.7

C. Whisper

論文名稱: Robust Speech Recognition via Large-Scale Weak Supervision

Reading Report

一、簡介

Whisper 是一個 OpenAI 推出的 automatic speech recognition (ASR) model，號稱語音辨識的能力接近人類水平，也有公開 source code，供大家使用。

目前的 speech recognition 使用的多為 self-supervision 跟 self-training 的技巧，而這篇 Whisper 是藉由 supervised 的在多個龐大的資料集上訓練，來做到只要用 zero-shot transfer 就可以讓語音辨識的 robustness 提升的效果。

作者發現比起使用 ImageNet 那種 gold-standard (有人工標記、確認過的) dataset，使用 weakly supervised 但資料量更大的 dataset 更能提升模型的 robustness 跟 generalization。然而這些 weakly supervised 的 dataset 資料量仍然比不上 unsupervised 的 dataset，所以這篇 Whisper 使用了高達 680,000 小時且涵蓋了 96 種英文以外的語言的 labeled audio data 來訓練。

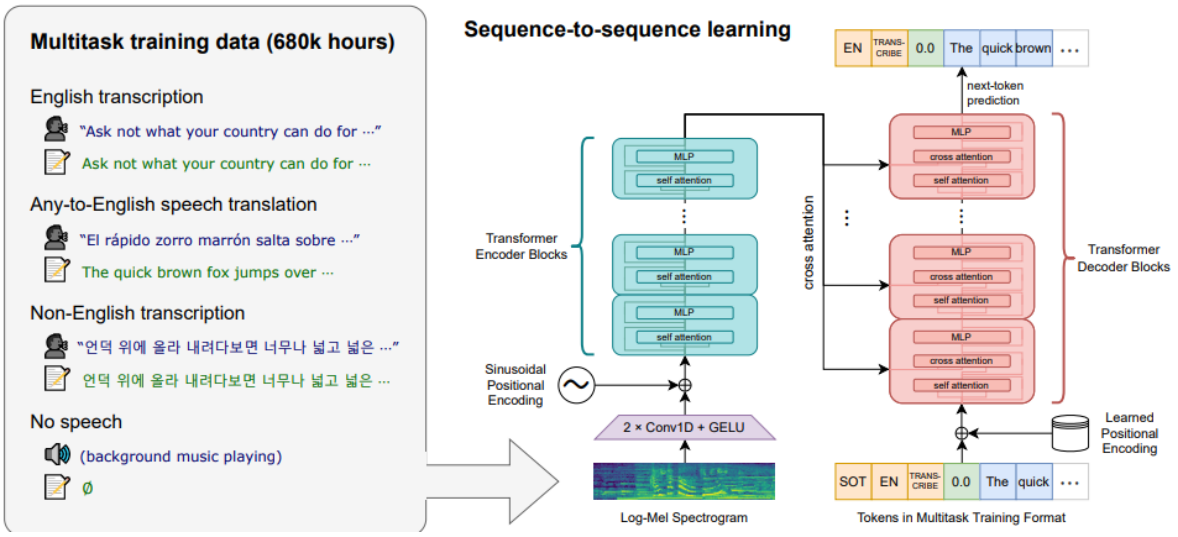
二、方法

- 資料處理

我們只希望 Whisper 預測出 without any significant standardization 的 raw text of transcript 即可，也就是不用特別再為了產生自然的transcription做處理，像是inverse text normalization。
資料集的部分包含聲音 (audio) 跟文本 (transcription)。聲音的環境、錄製設定、語言等等都有不同，這種 audio diversity 對訓練是有幫助的。然而 transcript 的部分則是需要在設定上越一致越好，所以作者也提出了一些 automated filtering method 來提升 transcript quality。比如 ASR 生成的文字通常不會包含較複雜的標點符號（驚嘆號、逗號、問號…）、段落符號、正確的大小寫等，作者便是透過這些觀察來處理掉機器生成的 transcript。此外，作者也使用 CLD2 (Compact Language Detector 2) 來判斷文本的語言，並確認文本使用的語言跟聲音的語言是相同的，才將 (audio, transcript) 的 pair 加入dataset。
audio files 會被分成以三十秒為單位的 segment，用來做 voice activity detection。
之後，再人工檢查同時有 high error rate 跟 data source size 的 data source，如此一來便可以把只有部分抄錄 (transcribe) 或 misaligned 的 transcript 去除。最後也要把 training dataset 跟 evaluation dataset 裡 transcript 有重複的資料去除，也就是做 de-duplication。

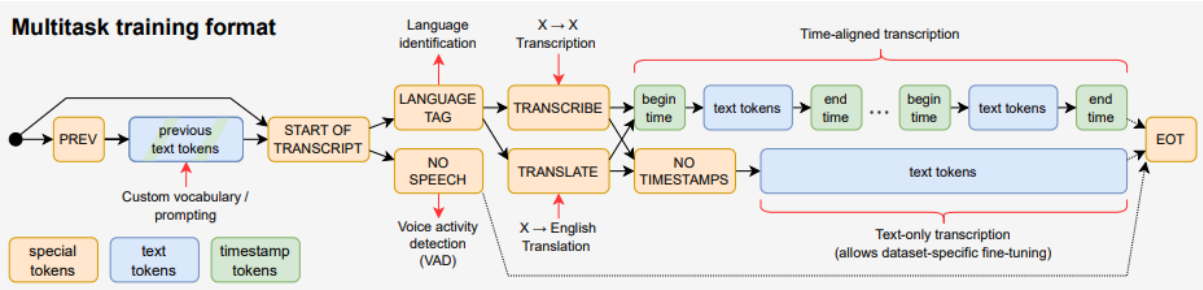
- 模型架構

由於這篇的重點在於 large-scale supervised pre-training for speech recognition，所以為了方便 scale up，模型架構僅採用最基本的 encoder-decoder Transformer。輸入是 16000 Hz 的 80-channel MFCC ； window size 為 25ms；stride 為 10 ms。輸入會先經過兩層 filter width = 3 的 convolutional layer，並使用 GELU 作為激發函數。之後會加入 sinusoidal positional embedding，才接上原本 transformer 的 encoder block。



tokenizer 部分，純英文模型使用的是 GPT-2 的 byte-level BPE text tokenizer，其餘的多語言模型則在字典大小不變的情況下 refit the vocabulary (跟 lemmatization 相反)。

而在 decoder 部分，是一個 audio-conditional language model。為了在這個比較 general 的 model 上實現多種不同的 task，所以需要加入一些特殊的 token，讓輸入有一定的格式。首先是在句子一開始會用 <|startoftranscript|> 作為標記，接著是要預測講出來的語言，若是不含有文字的語音訊號（如背景音樂），則以<|nospeech|>來表示。接著會用 <|transcribe|>或<|translate|>來判斷要執行的任務是抄寫或翻譯。之後再用 <|notimestamp|> 來決定是否要預測 timestamp，而 timestamp 以 20ms 作為單位。此外，預測時也會將產生過的 history transcript 拿來當成 input，讓模型可以學到更長時間的資訊，以解決 ambiguous audio 的問題。



Whisper model 系列有不同的 layer 數目跟參數量，共分為 tiny, base, small, medium, large 五個模型。

Model	Layers	Width	Heads	Parameters
Tiny	4	384	6	39M
Base	6	512	8	74M
Small	12	768	12	244M
Medium	24	1024	16	769M
Large	32	1280	20	1550M

Computer Project

資料集

這次使用 Hugging Face 上 Mozilla 提供的資料集 Common Voice 11.0 裡面中文的語音資料。這是一個公開 Crowdsource 的語音資料集，同時他也公開徵求世界上任何人的貢獻，也因此涵蓋了很多冷門語言。也就是說，每個人都可以選擇自己的語言、上傳自己的語音、基本資料等作為 data，之後會再有人負責驗證，每三個月他們會發布一次新的資料集。目前 Common Voice 已有超過 60 種語言、超過 9283 小時的資料。而在華語（台灣）的部分目前最新的版本是 2022/12/15 釋出的 Common Voice Corpus 12.0，有來自 2099 人錄製的 116 小時語音資料。

模型

使用 whisper-small

參數

使用 Hugging Face 提供的 sample code，但因為想要做的是中文語音辨識，所以就是將 dataset 選成 “zh-TW”，並將 language 選成 chinese。iteration數目使用預設的 5000 個 step，最終是跑 14.2 個 epoch。

```
--model_name_or_path="openai/whisper-small" \
--dataset_name="mozilla-foundation/common_voice_11_0" \
--dataset_config_name="zh-TW" \
--language="chinese" \
--train_split_name="train+validation" \
--eval_split_name="test" \
--max_steps="5000" \
--output_dir="./whisper-small-zh-TW" \
--per_device_train_batch_size="8" \
--gradient_accumulation_steps="4" \
--per_device_eval_batch_size="8" \
--logging_steps="1000" \
--learning_rate="1e-5" \
--warmup_steps="500" \
--evaluation_strategy="steps" \
--eval_steps="1000" \
```

```
--save_strategy="steps" \
--save_steps="1000" \
--generation_max_length="225" \
--preprocessing_num_workers="1" \
--length_column_name="input_length" \
--max_duration_in_seconds="30" \
--text_column_name="sentence" \
--freeze_feature_encoder="False" \
--gradient_checkpointing \
--group_by_length \
--fp16 \
--overwrite_output_dir \
--do_train \
--do_eval \
--predict_with_generate \
```

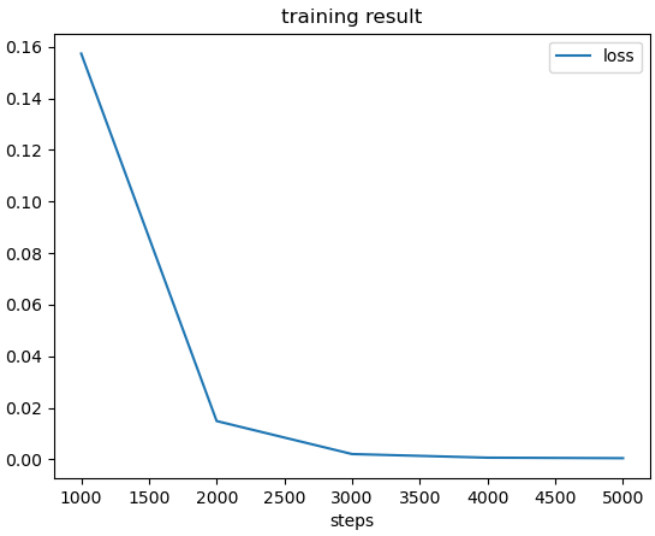
結果

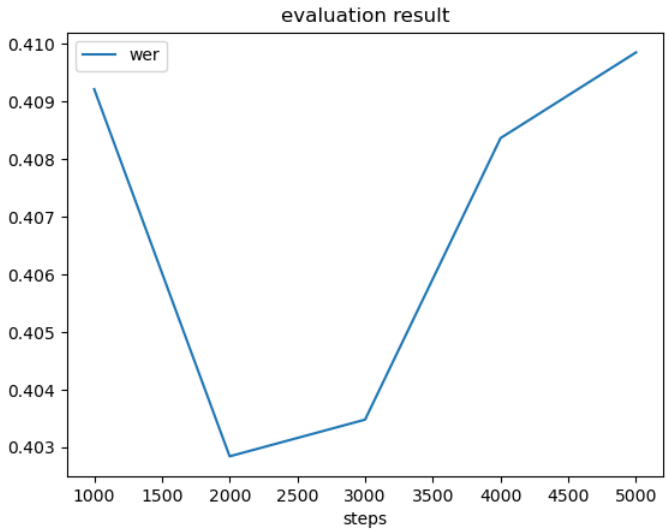
train time: 47671 秒 (約 13.2 小時)

evaluation time: 1220 秒 (約 20.3 分鐘)

train loss: 0.035

evaluation wer (word error rate): 0.409





觀察 evaluation 的結果可以很明顯的發現模型在 2000 ~ 3000 個 step 之間開始就有點 overfitting，推測可能因為原先 steps = 5000 的設定是應用在英文的 dataset 上的，但我們這次是拿中文的資料來訓練，資料量差非常多，所以 gradient 也較快就達到 local minimum了。之後應該將 step 設在這個區間就好，或者再對 learning rate 做調整。

Team Work

Kaldi: R11944002 葉映彤

Whisper: R11944026 柯婷文