Study of transformer-based TTS and its embedded implementation
基於變換器之文字語音轉換研究及嵌入式實現

Sian-Yi Chen

Advisors : Tay-Jyi Lin and Chingwei Yeh

Outline

- Introduction
  - TTS、Transformer、Transformer-based TTS
- Transformer-based TTS
  - 系統架構
  - 系統流程
  - TTS 合成結果
- Embedded system implementation
  - 演算法
  - 實作
  - Demo
- Conclusion

Outline

- Introduction
  - TTS、Transformer、 Transformer-based TTS
- Transformer-based TTS
  - 系統架構
  - 系統流程
  - TTS 合成結果
- Embedded system implementation
  - 演算法
  - 實作
  - Demo
- Conclusion

文字語音轉換（Text to Speech，TTS）

- 現今影音媒體盛行的時代，許多影音創作者選擇利用網路資源或Google語音，將文字稿輸入後，轉換為語音輸出，此種文字輸入轉換為語音輸出的技術，稱為文字語音轉換Text to Speech，簡稱TTS。

- TTS是一項需要同時結合許多領域應用的技術，例如語言學、深度學習、訊號處理領域等。利用語言學對語料集、文本，或是訓練資料做處理，再將處理好的文字輸入現有的神經網路，最後讓文字合成自然的語音。

- TTS追求的的目標具有許多面向，例如自然度、平滑度、相似度等。

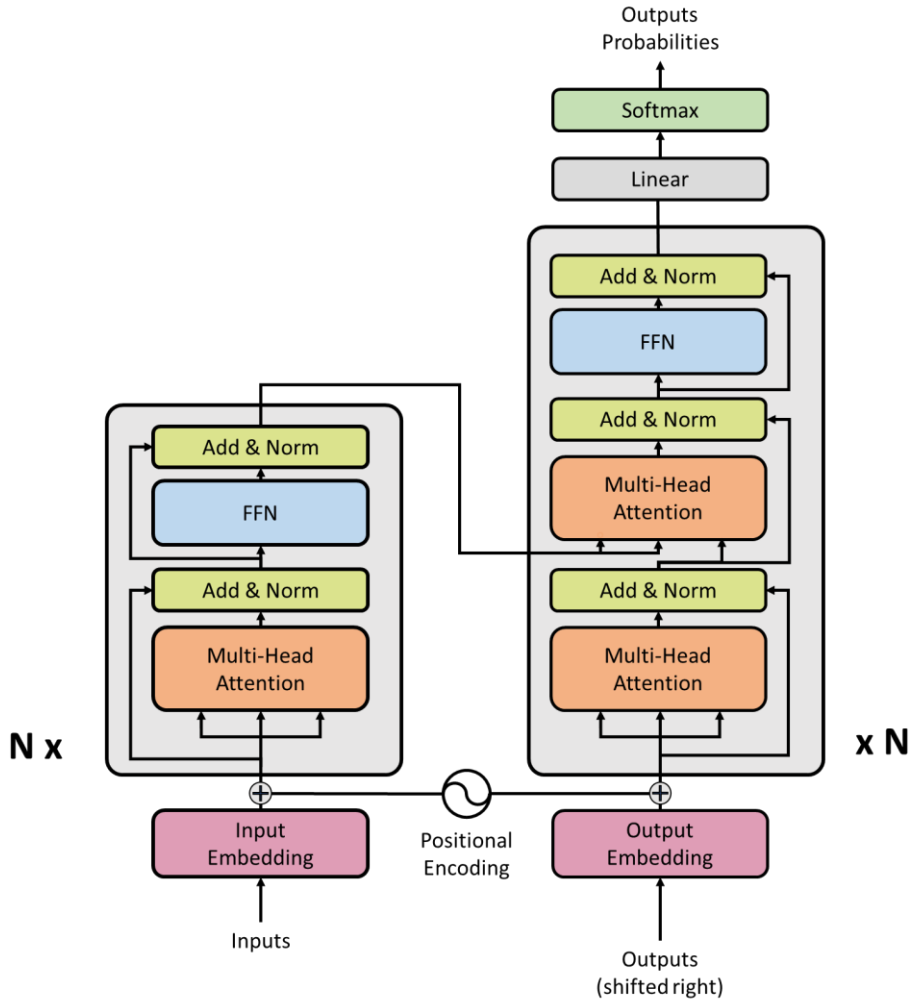
■ 語音轉換示意圖

變換器（Transformer）

- Transformer是一種採用自注意力機制（attention）的深度學習模型，這種機制不僅可以加速神經網路的速度，還可以根據輸入資料各部分重要性的不同而配置不同的權重。

- 在TTS領域中，資料通常都具有時間序列的特性，而對於擁有此種特性的資料較常會選擇使用RNN，但因為RNN以遞迴的方式處理資料而難以平行處理，因此在2017年Google Brain團隊A. Vaswani等人提出了一種僅使用attention的新穎架構Transformer。

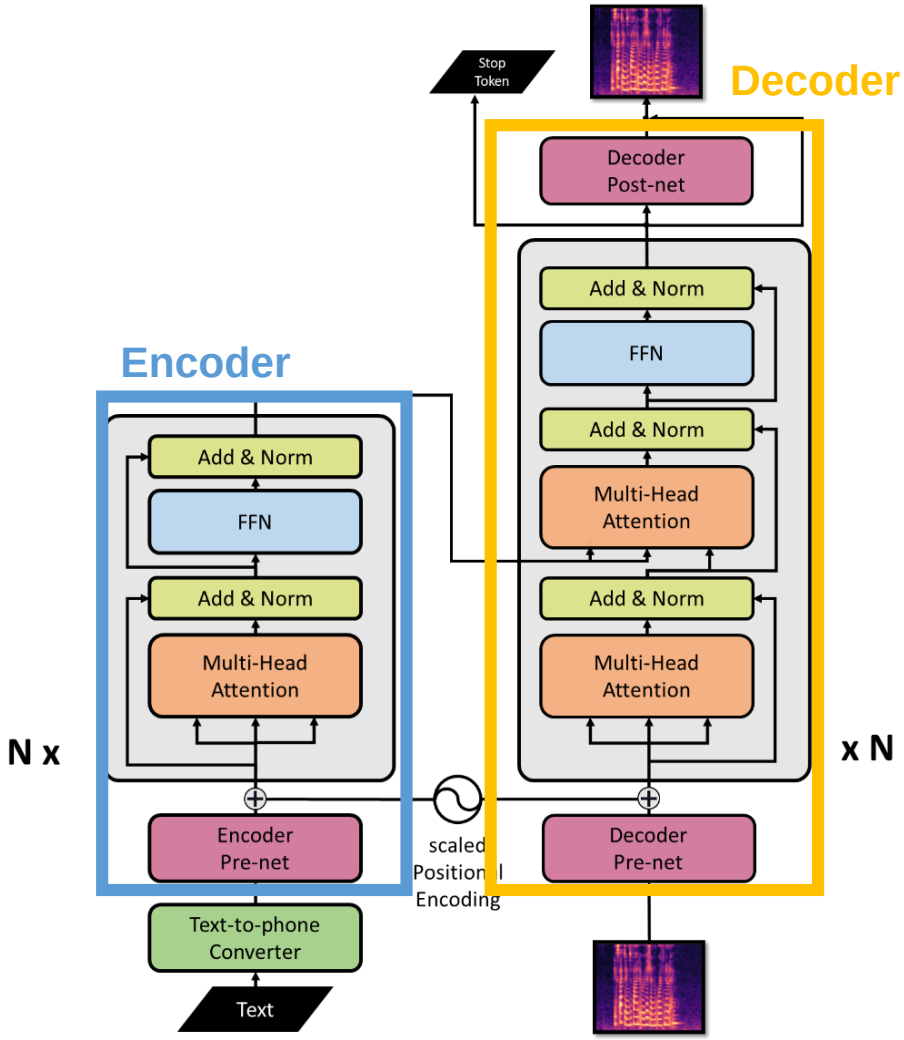
- Transformer神經網路可以一次性輸入所有資料，有效地達到平行化，並透過attention記憶包含時間序列資料的資訊，得以大幅降低訓練所需的時間。



■ Transformer

基於變換器的文字語音轉換（Transformer-based TTS）

- 基於RNN應用在TTS的例子，像是Tacotron2，雖然它生成的表現優異，但卻仍然存在訓練和推理的效率低下的問題。
- 因此將Transformer中間注意力機制的部分取代了Tacotron2 的結構，也就是右圖將Transformer應用到了TTS領域。
- Transformer-based TTS與Transformer主要的差別為Encoder與Decoder的輸入輸出； Encoder Pre-net、Decoder Pre-net以及Decoder Post-net 。



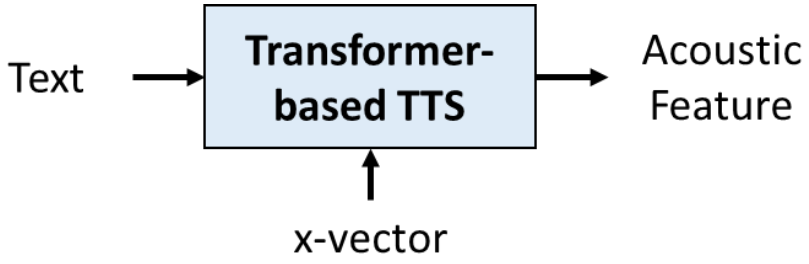
■ Transformer-based TTS

Outline

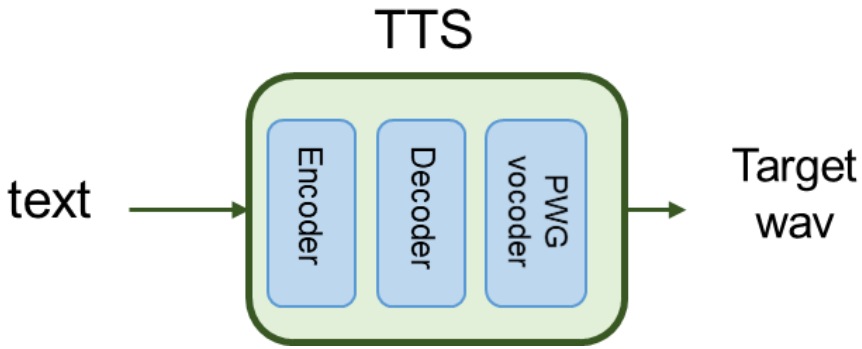
- Introduction
  - TTS、Transformer、 Transformer-based TTS
- Transformer-based TTS
  - 系統架構
  - 系統流程
  - TTS 合成結果
- Embedded system implementation
  - 演算法
  - 實作
  - Demo
- Conclusion

系統架構

- 在系統架構中，使用的Transformer-based TTS與1-3節所講述的Transformer-based TTS有些微差異。
- 其一，作者使用x-vector[6]作為Transformer解碼器的附加輸入，x-vector通過線性投影變換，並添加到編碼器輸出的每一幀。
- 在系統中單純使用Transformer-based TTS還不能生成語音，將文字處理成音素序列輸入，得到聲學特徵後，還要再使用聲碼器合成語音，系統中本論文是參考VCC2020（Voice Conversion Challenge 2020）大會提供。



■ Transformer-based TTS with x-vector

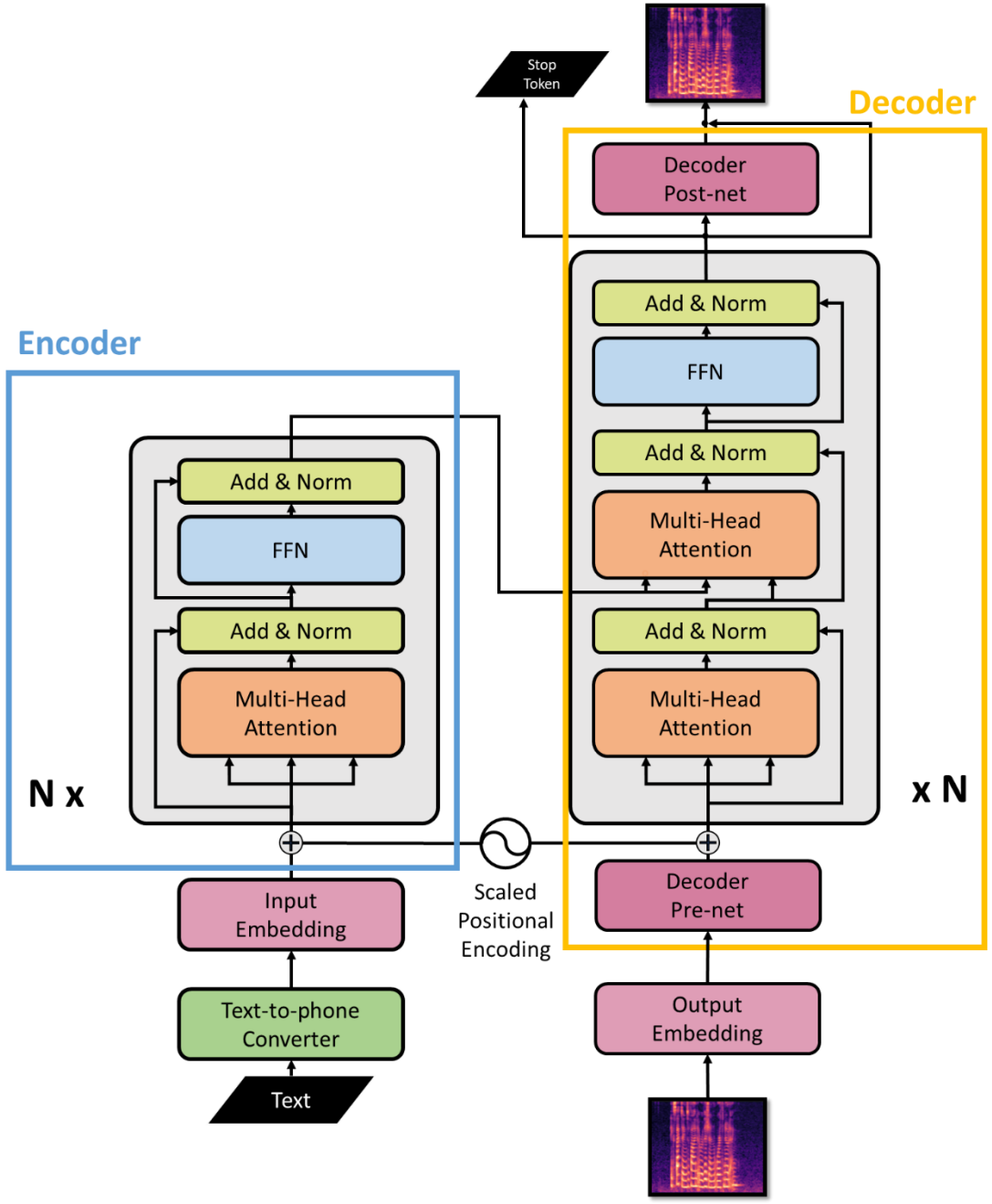


■ TTS 系統架構圖

Transformer-based TTS架構圖

- 其二，不使用Encoder Pre-net，因為目標訓練量通常在語音轉換任務中數量很小，因此如果沒有使用額外的語料集，容易受到過度擬合（overfitting）的影響。

- Transformer為左半邊編碼器（Encoder）以及右半邊解碼器（Decoder）組成，其中N為堆疊的層數，預設值為6。

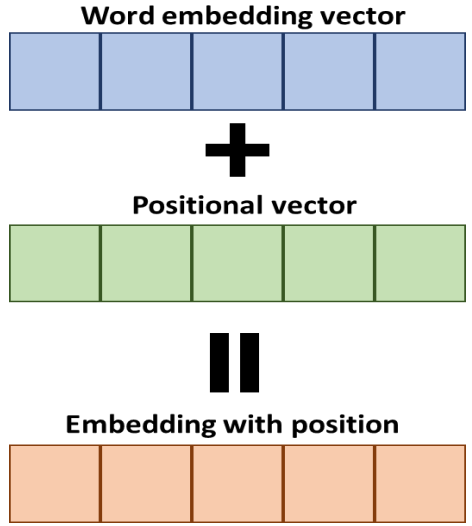


■ Transformer-based TTS架構圖

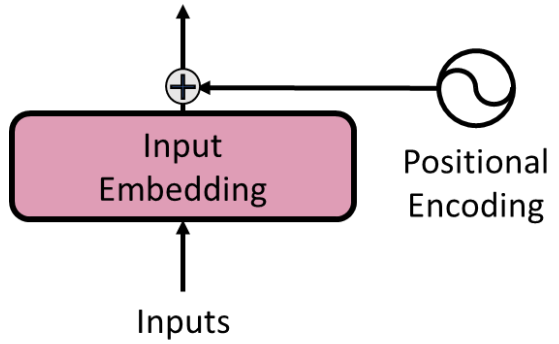
位置編碼（Scaled Positional Encoding）

- 位置編碼（Scaled Positional Encoding），從圖中可以看到輸入經過embedding後轉換成向量表示，接著再進去下一層之前須要先與positional encoding相加（input embedding與positional encoding維度必須相等）。

- embedding本身並不包含每個字在句子中的相對位置，在Transformer運算中，輸入文本後，所有字詞皆是平行運算的，因此此層目的就是為了讓神經網路可以學習詞語之間的位置資訊，否則若是沒有位置資訊，就算使用相同的字詞組成不同的句子，神經網路也無法判讀兩句話是否具有不同的意思。



■ 詞嵌入加入位置資訊且維度相等示意圖

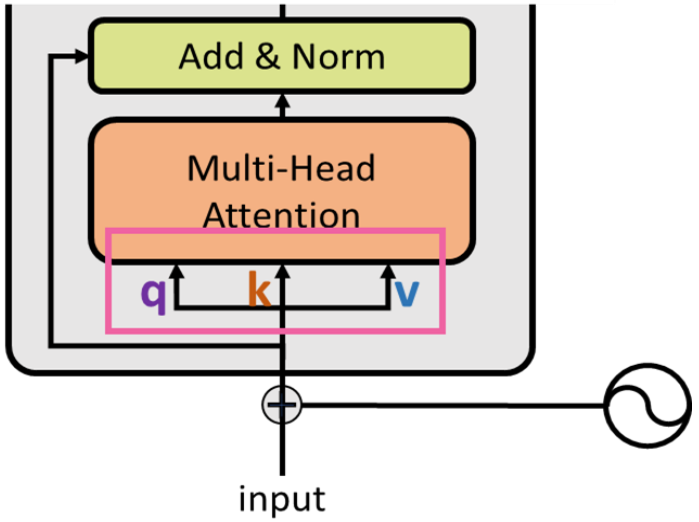


■ 輸入經過詞嵌入再經過位置編碼

自注意力機制（Self-attention）

- 自注意力機制有3個重要的參數q、k、v，分別是query、key和value，query指的是當前的詞向量、key指的是輸入的文本序列中所有的詞向量，而value則是指實際上的文本序列內容。

- 這些參數由輸入經過embedding和positional encoding後再分別乘上屬於q、k、v的矩陣得到，在attention機制中，q會與k做內積得到所有文本詞向量與當前文本詞向量的匹配程度。



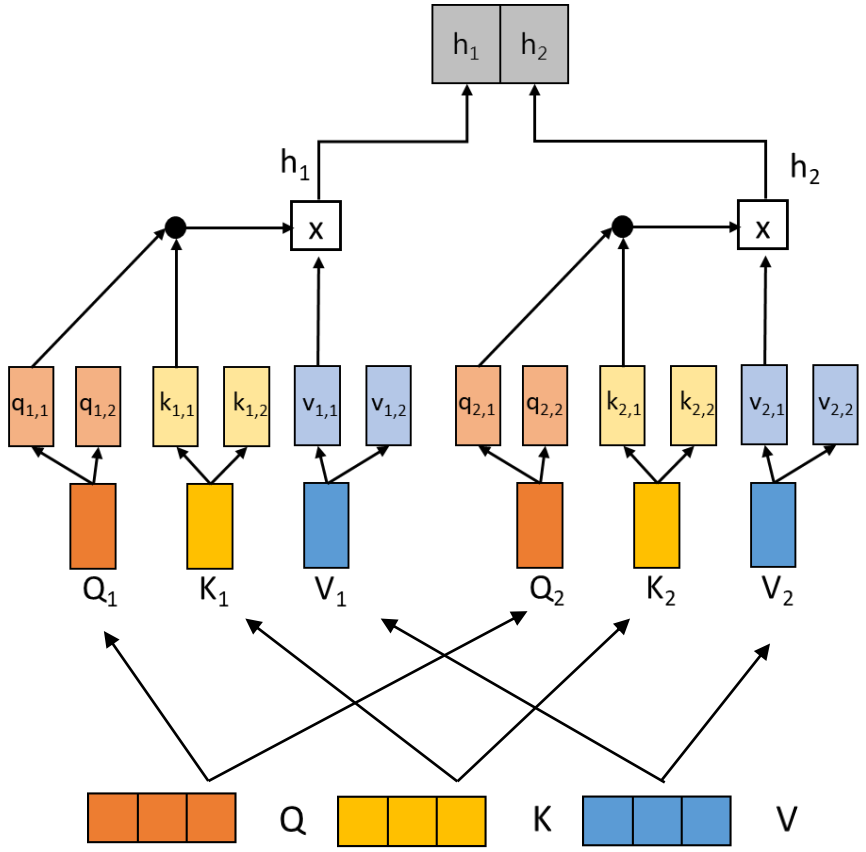
■ 自注意力機制（Self-attention Mechanism）

多頭注意力機制（Multi-head Self-attention）

- 多頭注意力機制就代表著我們不只有一組，其運算方式與self-attention mechanism相同，差別在於會將原先的q、k、v拆分成多組較低為度的向量qi、ki、vi，最後運算完再將他們全部連接（concat）起來運算。

- 好處是可以讓每一個head都關注不一樣的事情，像是學習字詞在序列中不同位置不同表示空間中的資訊，部分學習區域資訊，部分學習廣域資訊等。

- 此種方法還有一種好處，那就是可以有效的增加model capacity。



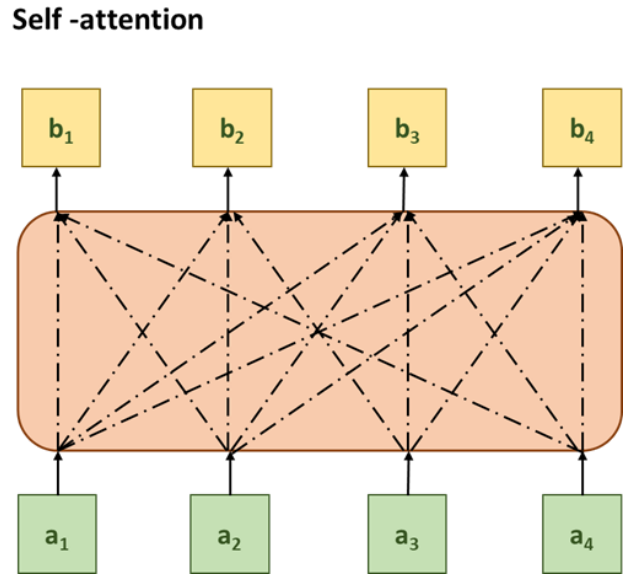
■ Multi-head self-attention計算示意圖

具遮罩的自注意力機制（Masked Self-attention）

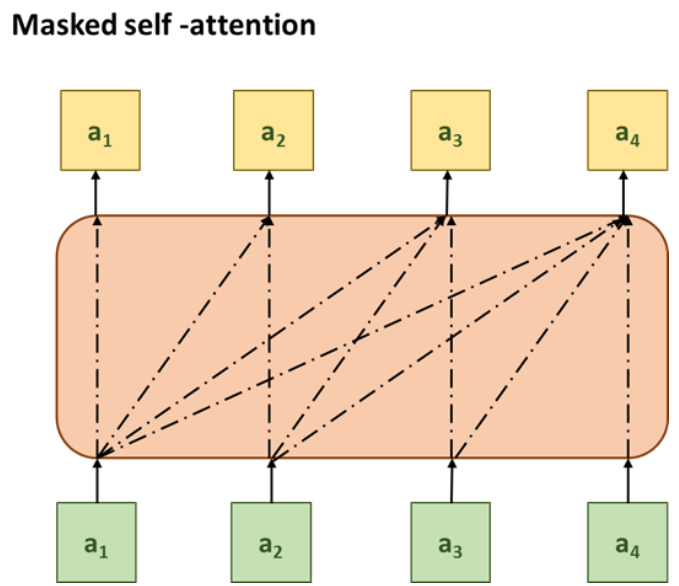
- 從右上圖可見，在沒有遮罩的版本中，a 為輸入，b 為輸出，自注意力機制可以得知整段資料的資訊，但在右下圖中，第一筆輸出的資料僅有第一筆輸入資料的資訊。

- Mask機制僅存在decoder中，且只有在第一個multi-head attention中使用，這種機制是在模擬神經網路在測試階段僅知道當前的輸入的資訊，並無法得知未來的資訊。

- 其餘與Self-attention一模一樣。



■ 自注意力機制可得知整段資訊

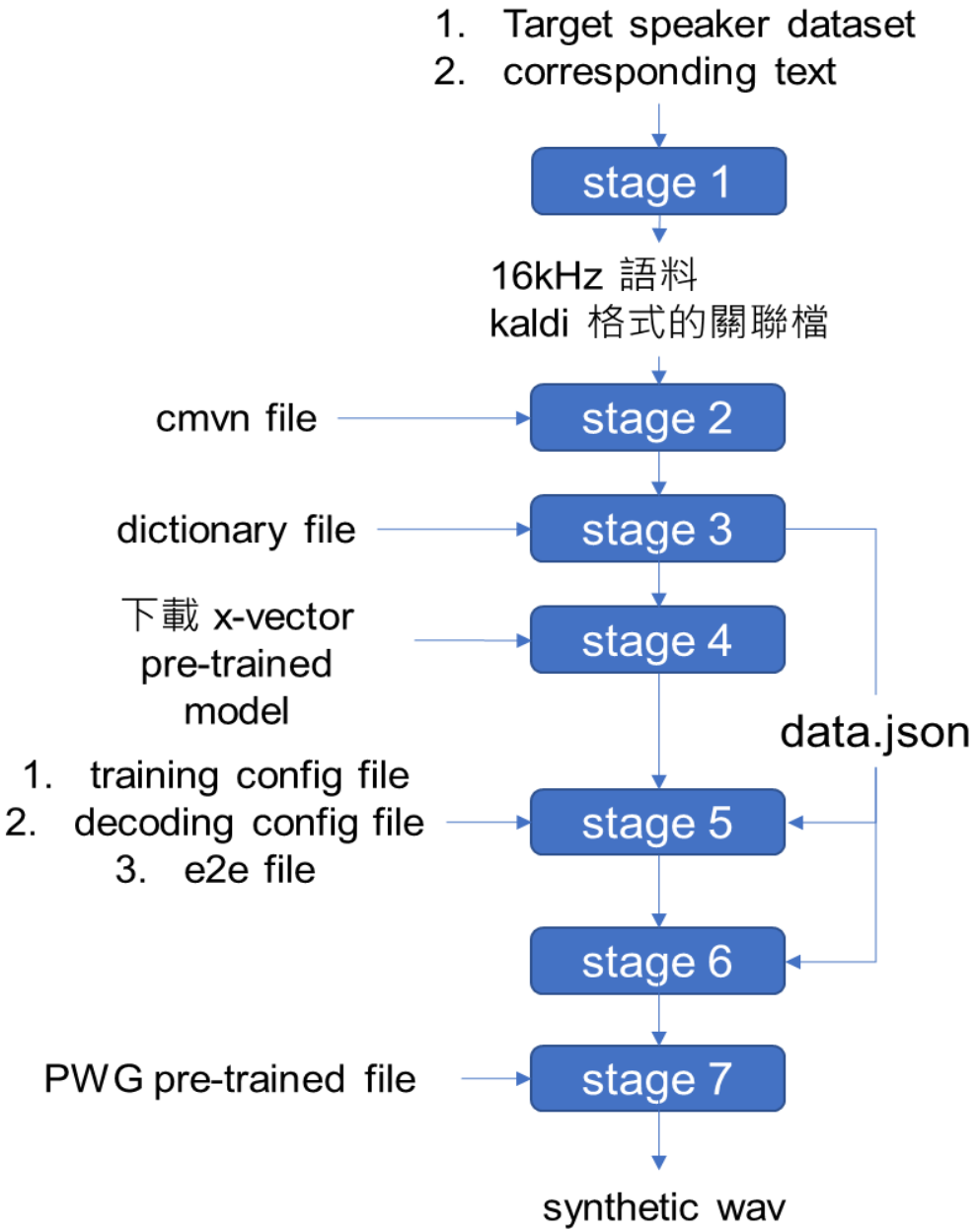


■ 遮罩自注意力機制僅可得當前輸入資訊

系統流程

- 系統流程總共有7個步驟，其中前4個步驟是資料前處理，第5步驟為微調（fine-tune）為主要訓練，第6步驟為推論（inference），最後則是使用聲碼器合成語音。

- TTS 訓練流程
  1. Data preparation
  2. Feature Generation
  3. Dictionary and Json Data Preparation
  4. x-vector extraction
  5. Fine-tuning
  6. Decoding
  7. Synthesis



■ 系統流程圖

TTS 合成結果
（猜猜這是誰的聲音？）

這是我真心推薦

我竟然發現在我的書桌上

結果非常好

Outline

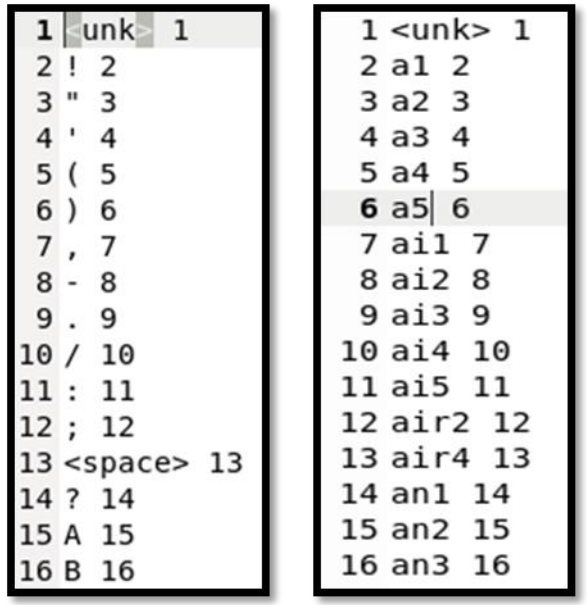
- Introduction
  - TTS、Transformer、 Transformer-based TTS
- Transformer-based TTS
  - 系統架構
  - 系統流程
  - TTS 合成結果
- Embedded system implementation
  - 演算法
  - 實作
  - Demo
- Conclusion

演算法 - 詞嵌入（embedding）

- 在Transformer的embedding層中，它的輸入為文字，但在embedding之前，還需要先將文字編碼，最後才將編碼完的文字（數字）轉換成向量。

- 以英文模型舉例，如左圖，模型會內置一張字典，而英文的A到Z，若A從15開始，就會依序編碼至40；而中文因為詞彙量太過龐大，若是將其編成字典，編碼會變得過於龐大，因此中文會先轉換成具有固定表示方式的拼音。

- 實際舉例來說，本論文的embedding的weight維度為 [337, 384]，假設第一筆資料長度為19，embedding完輸出結果維度則為 [19, 384]。



■ 英文模型embedding字典（左）、中文模型embedding字典（右）

演算法 - 位置編碼（Scaled Positional Encoding）

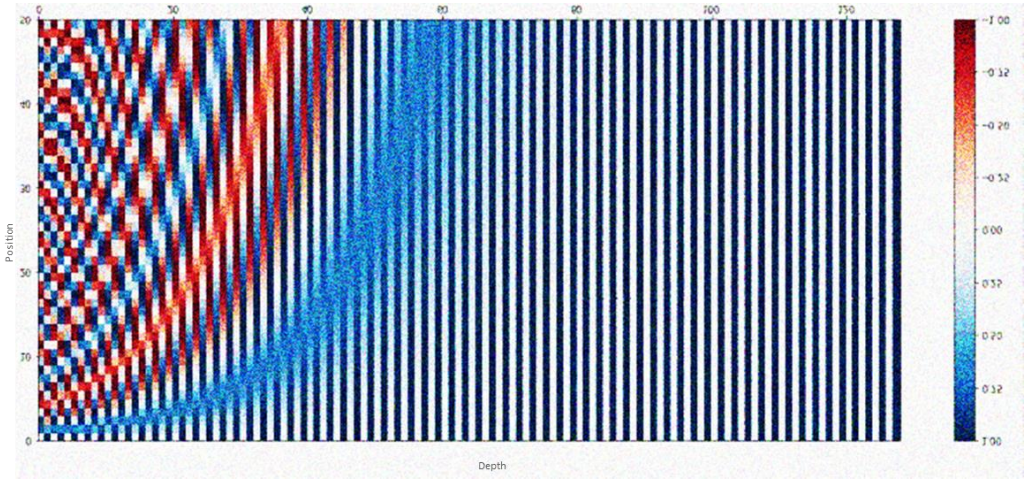
- positional encoding目的是為了將embedding完的資料加入位置資訊。
  - pos為positional的縮寫，也就是該詞語在序列中的位置
  - i是該詞語positional encoding中排列的位置
  - dmodel則是positional encoding 的維度與embedding相同維度

$$PE_{(pos,\ 2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,\ 2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

■ positional encoding（PE）的公式

- 看公式可以理解，它的算法就按照偶數位置使用正弦函數（sin）來編碼，而奇數位置則使用餘弦（cos）函數處理。

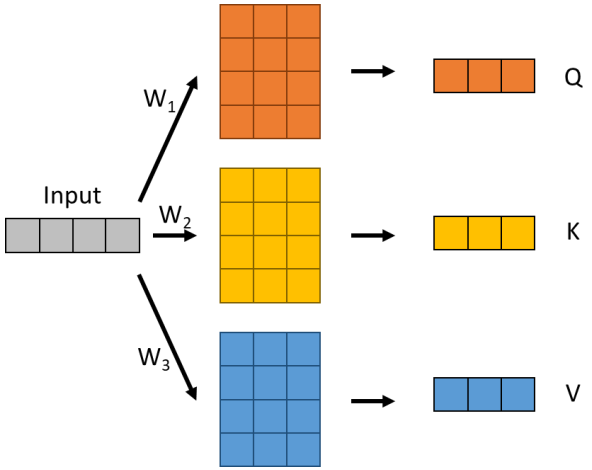


■ 位置編碼可視化結果

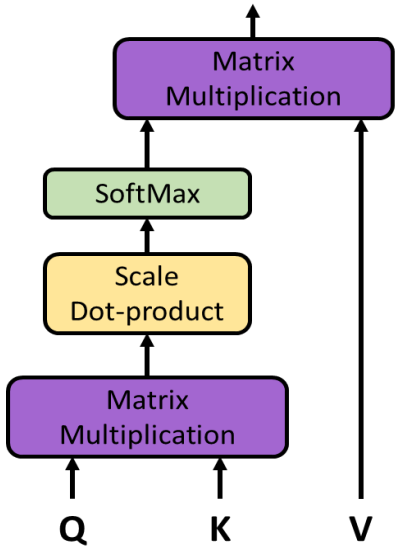
$$\mathrm{PE} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

■ PE矩陣中資料排列示意圖

演算法 - 多頭注意力機制（Multi-head Self-attention）



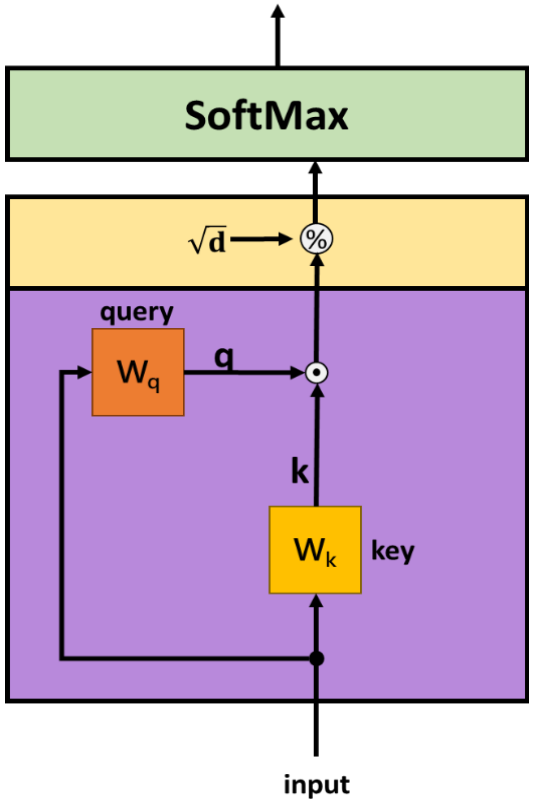
■ 取得Q、K、V過程示意圖



■ Scaled dot product attention

$$\text{Attention}(\text{Q}, \text{K}, \text{V}) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

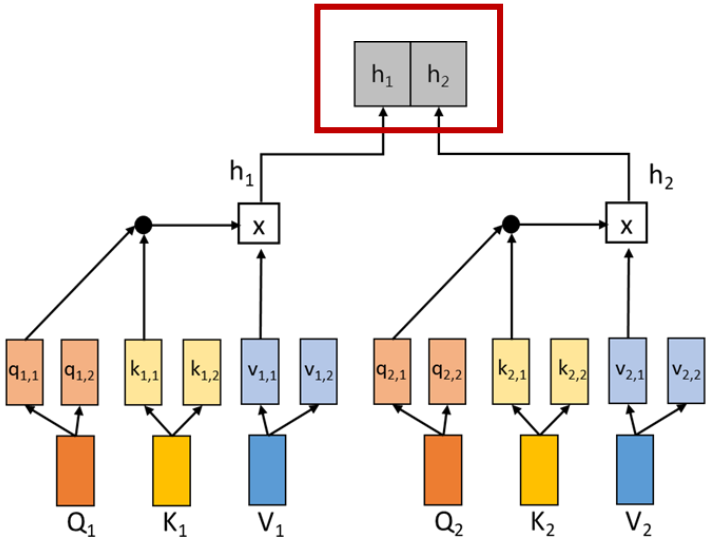
■ Attention 公式



■ 計算attention示意圖

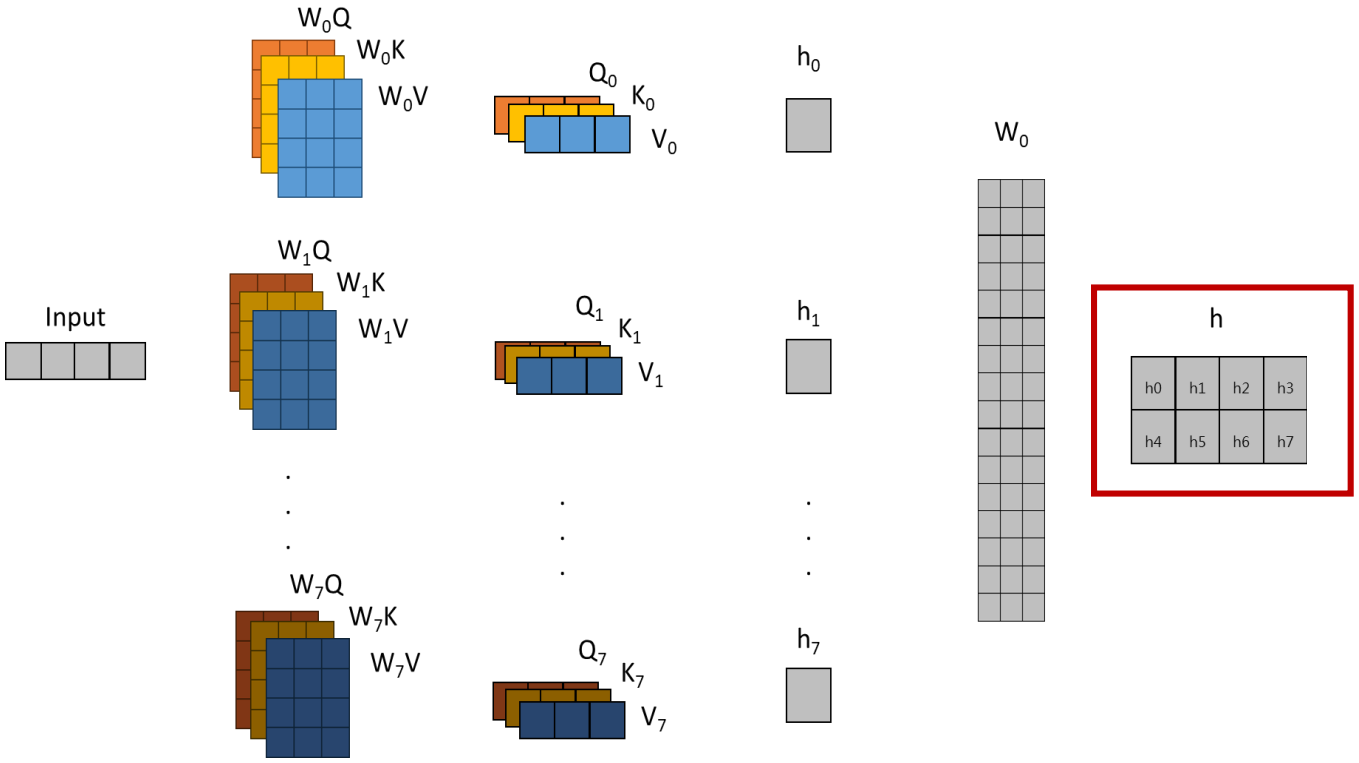
演算法 - 多頭注意力機制（Multi-head Self-attention）

- 假設multi-head self-attention參數h等於8時，
  也就是q、k、v各分裂成8個，如右圖。



■ Multi-head self-attention計算示意圖

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

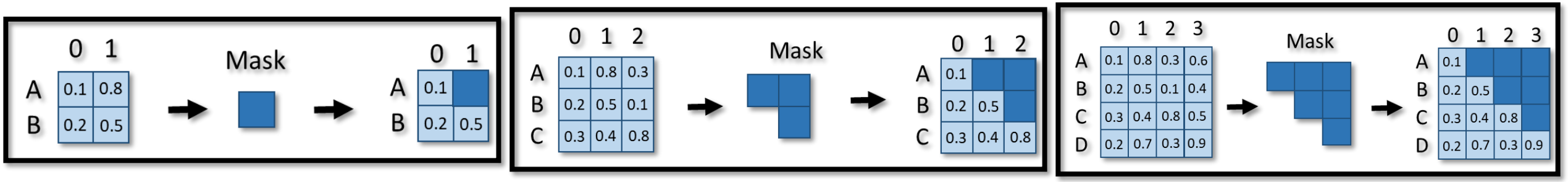


■ Multi-head self-attention計算總圖

演算法 - 具遮罩的多頭注意力機制（Masked Multi-head Attention）

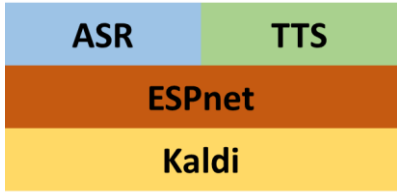
- Masked multi-head attention與multi-head attention運算方法一模一樣，其中為了要實現遮擋未來資訊，遮擋陣列的上三角部分，使其成為下三角矩陣。

- 而decoder與encoder不同，輸入為遞迴輸入，第一次第一筆、第二次第一加二筆...以此類推，而遮罩也會隨著陣列維度增大而改變。



■ Masked multi-head 示意圖

實作 – python (English-to-English)

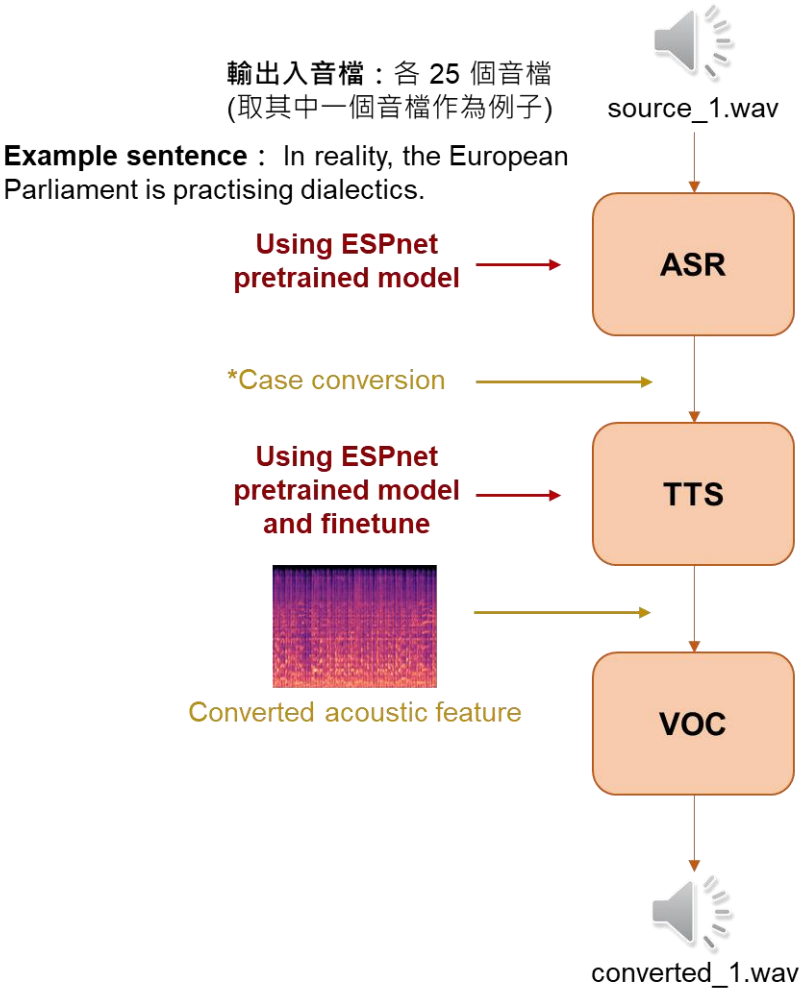


- 預訓練模型、Kaldi、ESPnet之間關係

- 在python版本中，實作是使用VCC2020提供的baseline，其架構為AST+TTS的英文模型，並使用ESPnet所提供的預訓練神經網路來搭建使用pytorch做為深度學習的引擎，並遵循Kaldi風格處理資料，像是特徵提取、資料格式與執行腳本。

- 環境架設完成後，可直接使用專案中轉換前後皆是英文的版本，而右圖為轉換流程。

- 接著要使用中文轉換成中文，須將目前ASR與TTS模型都替換掉，在ESPnet github中可以找到相同架構的Transformer並使用中文語料預訓練而成的ASR與TTS模型。



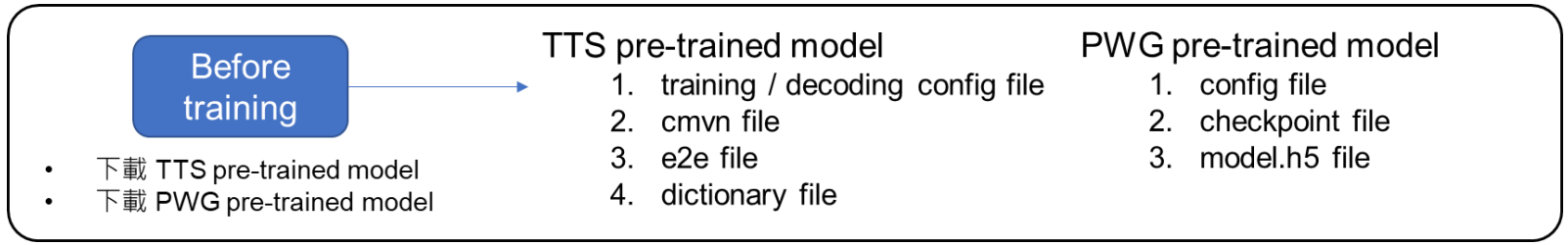
- conversion process

實作 – python (Chinese-to-Chinese)

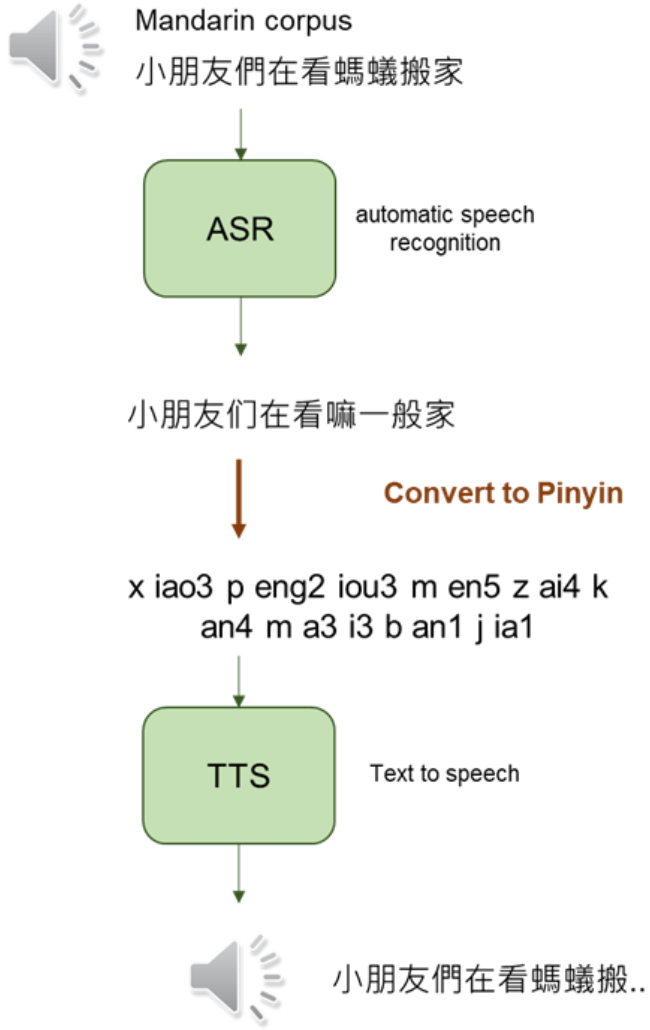
- 因為要更換的模型有兩種，因此我先更換了ASR模型，並使用中文ASR模型+英文TTS模型，可以正常運作後再比照辦理更換成中文TTS模型，最後再使用中文語料輸入。

- 下圖為將英文模型更換成中文模型，我所更換的文件，分別為訓練配置檔、解碼配置檔，以及神經網路訓練完成的二進為參數檔。

- 更換完成即可開始訓練並提取訓練完成的參數。



■ 轉換模型需更換的文件



■ 轉換流程

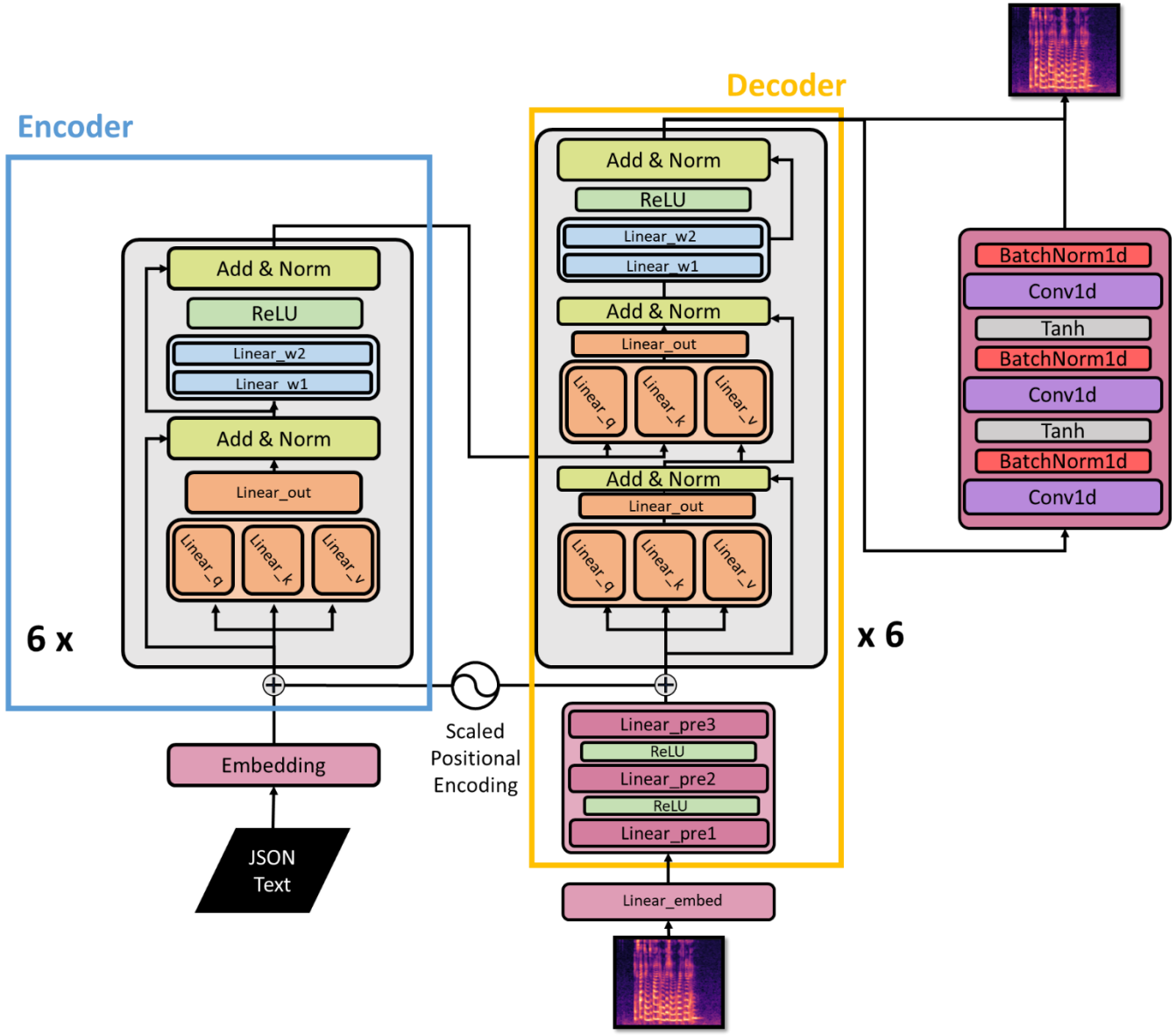
實作 – C code

- 在baseline版本中使用了shell、perl、python三種語言，因此第一步驟我先簡化問題，其步驟如下：
  - 將腳本指令移除(shell、perl)，僅留下必要的執行檔(ASR、TTS、PWG)，單獨取出 TTS 與 PWG。
  - 對照步驟一取出的檔案，將其每個步驟、區塊、函數的輸入輸出透過 C 重新實現。

- 其中使用到最多的是pytorch中的linear函數，其功能為對輸入資料做轉置（transformation）。

$$y = xA^T + b$$

■ pytorch中的linear函數



■ C-based Transformer TTS架構圖

Demo

插入實作影片

Outline

- Introduction
  - TTS、Transformer、 Transformer-based TTS
- Transformer-based TTS
  - 系統架構
  - 系統流程
  - TTS 合成結果
- Embedded system implementation
  - 演算法
  - 實作
  - Demo
- Conclusion

結論

- 在文字語音轉換研究領域中，對於序列型資料，我們有著許多不同的處理方式，而本論文在此領域使用了基於變換器的架構，其優點為使用特別的結構「多頭注意力機制」，不僅使資料可以平行輸入，達到訓練時間大幅降低的效果，而且其結果在自然度、相似度上也有著出色的表現。

- 現今大型神經網路為建構方便，可以常見到使用python語言進行實作，因此難以直接於嵌入式系統中應用，因此本論文將原先2020年語音轉換挑戰大賽中提供的英文版本模型更改承中文版本模型，並使用C語言實現，並且最後合成結果在自然度上與pytorch版本相當。

- 有了Transformer的嵌入式系統，利於未來對於嵌入式系統的其它應用，或是優化系統的執行時間，以進行硬體上的即時呈現。

參考文獻

1. A. Vaswani, et al., “Attention is all you need,” in *Proc. NIPS*, 2017.
2. N. Li, et al., “Close to human quality TTS with transformer,” *arXiv preprint arXiv:1809.08895*, 2018.
3. T. Hayashi, et al., “ESPnet-TTS: unified, reproducible, and integratable open source end-to-end text-to-speech toolkit,” in *Proc. IEEE ICASSP*, 2020, pp. 7654-7658.
4. N. Li, et al., “Neural speech synthesis with transformer network,” in *Proc. AAAI*, 2019, pp. 6706-6713.
5. J.-X. Zhang, et al., “Voice conversion by cascading automatic speech recognition and text-to-speech synthesis with prosody transfer,” *arXiv preprint arXiv:2009.01475*, 2020.
6. D. Snyder, D. G.-Romero, G. Sell, D. Povey, S. Khudanpur, “X-vectors: robust DNN embeddings for speaker recognition,” in *Proc. IEEE ICASSP*, 2018, pp. 5329-5333.
7. W.-C. Huang, T. Hayashi, S. Watanabe, T. Toda, “The sequence-to-sequence baseline for the voice conversion challenge 2020: cascading ASR and TTS,” *arXiv preprint arXiv:2010.02434*, 2020.
8. M. Huang, “Development of Taiwan mandarin hearing in noise test,” *Department of speech language pathology and audiology, National Taipei University of Nursing and Health Science*, 2005.

參考文獻

9. Z. Yi, et al., "Voice conversion challenge 2020: Intra-lingual semi-parallel and cross-lingual voice conversion," *arXiv preprint arXiv:2008.12527*, 2020.
10. X. Tan, T. Qin, F. Soong and T.-Y. Liu "A survey on neural speech synthesis," *arXiv preprint arXiv:2106.15561*, 2021.
11. S. Watanabe, et al., "Espnet: end-to-end speech processing toolkit," *arXiv preprint arXiv:1804.00015*, 2018.
12. V. Popov, et al., “Fast and Lightweight On-Device TTS with Tacotron2 and LPCNet,” in *Proc. INTERSPEECH*, 2020.
13. Z. Ying and X. Shi, “An RNN-based algorithm to detect prosodic phrase for Chinese TTS,” in *Proc. IEEE ICASSP*, 2001, Vol. 2, pp. 809-812.
14. H. Tachibana, K. Uenoyama and S. Aihara, “Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention,” in *Proc. IEEE ICASSP*, 2018, pp. 4784-4788.