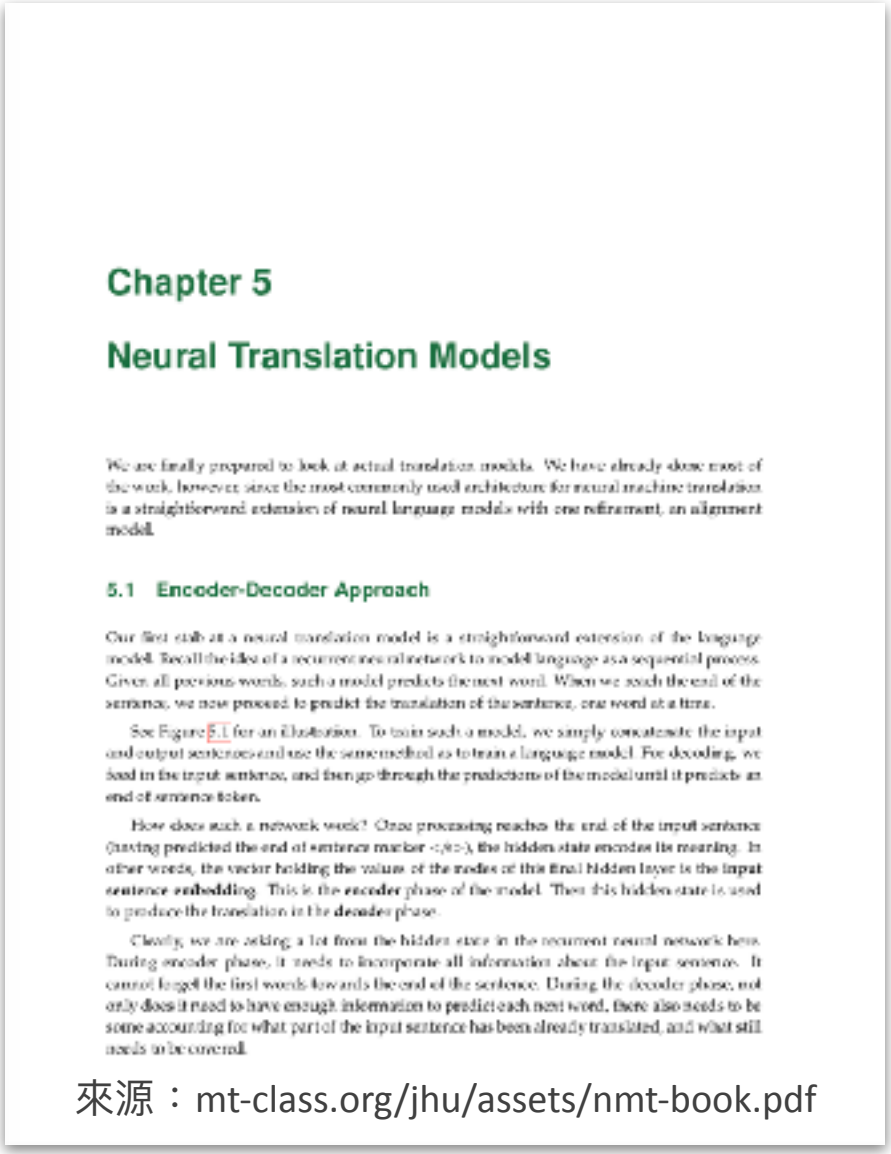
NMT 第 5 章 類神經機器翻譯
Neural MT

教科書與課程網站：mt-class.org/jhu/syllabus.html （草稿）

2018 1120

教科書相關章節



Chapter 5

Neural Translation Models

We are finally prepared to look at actual translation models. We have already done most of the work, however, since the most commonly used architecture for neural machine translation is a straightforward extension of neural language models with one refinement, an alignment model.

5.1 Encoder-Decoder Approach

Our first stab at a neural translation model is a straightforward extension of the language model. Recall the idea of a recurrent neural network to model language as a sequential process. Given all previous words, such a model predicts the next word. When we reach the end of the sentence, we now proceed to predict the translation of the sentence, one word at a time.

See Figure 5.1 for an illustration. To train such a model, we simply concatenate the input and output sentences and use the same method as to train a language model. For decoding, we feed in the input sentence, and then go through the predictions of the model until it predicts an end of sentence token.

How does such a network work? Once processing reaches the end of the input sentence (having predicted the end of sentence marker </s>), the hidden state encodes its meaning. In other words, the vector holding the values of the nodes of this final hidden layer is the **input sentence embedding**. This is the **encoder** phase of the model. Then this hidden state is used to produce the translation in the **decoder** phase.

Clearly, we are asking a lot from the hidden state in the recurrent neural network's brain. During encoder phase, it needs to incorporate all information about the input sentence. It cannot forget the first words towards the end of the sentence. During the decoder phase, not only does it need to have enough information to predict each next word, there also needs to be some accounting for what part of the input sentence has been already translated, and what still needs to be covered.

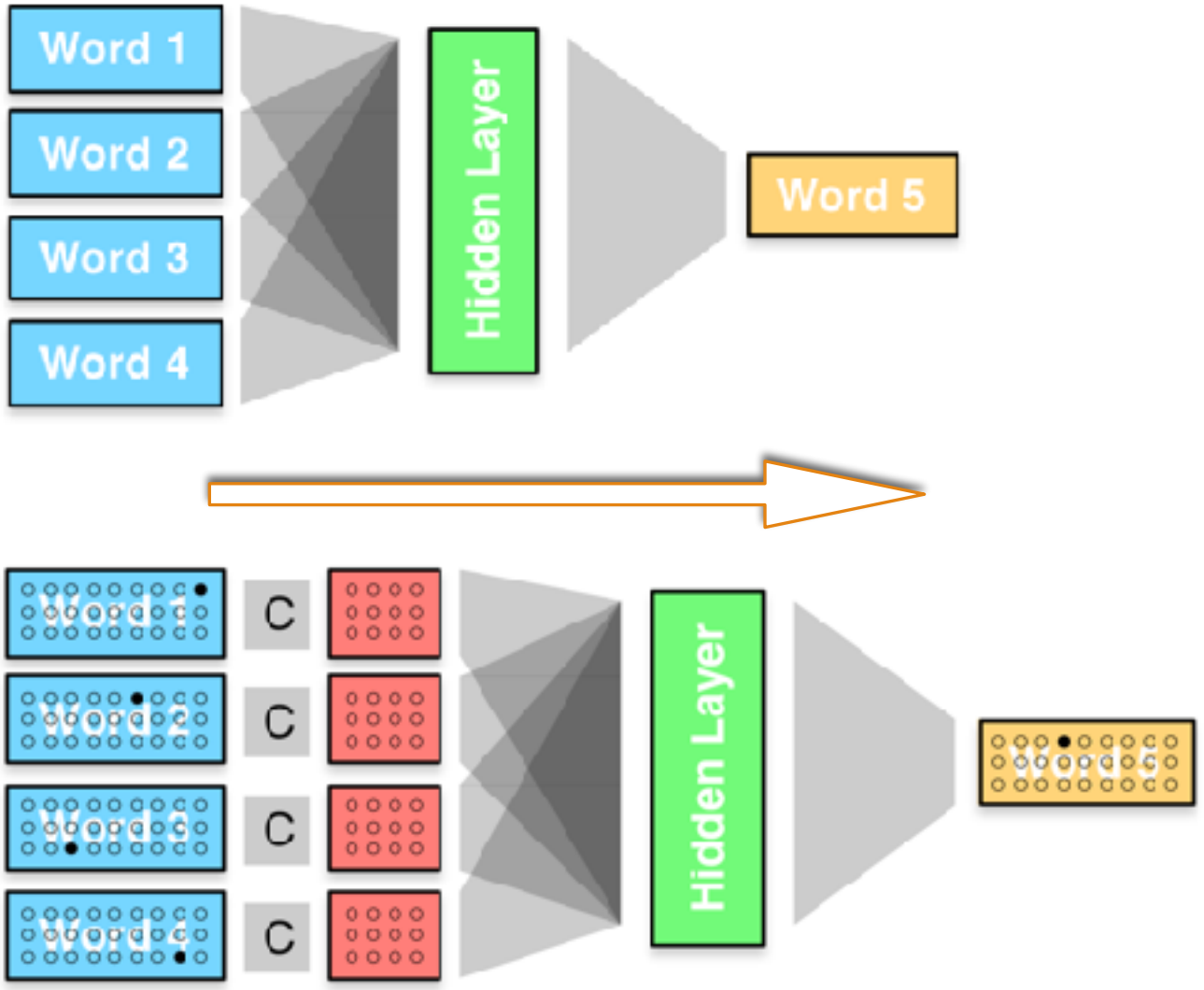
來源：mt-class.org/jhu/assets/nmt-book.pdf

機器翻譯目標語的語言模型

- 語言模型
  - 可以預測下一詞
  - 那麼也可以預測翻譯的下一詞
- 用語言模型的變形，來做機器翻譯
  - 前饋類神經網路 feed-forward neural network
  - 遞迴類神經網路 recurrent neural network
  - 長短期記憶類神經網路 long short term memory neural network
- 用已經產生的輸出＋輸入的文脈
  - 預測下一詞

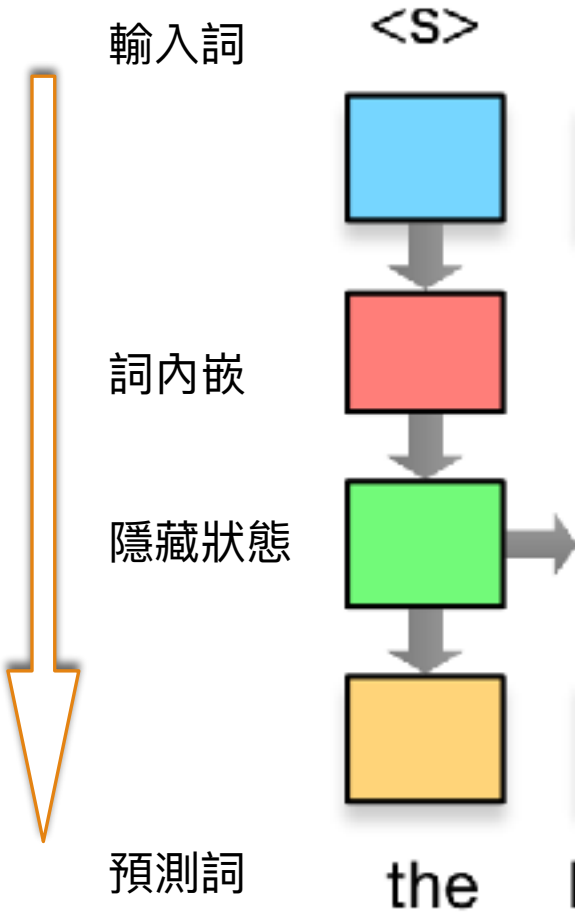
前饋類神經網路 (由左往右)

- 馬可夫假設 (只考慮定長「歷史」前4詞預測第5詞)



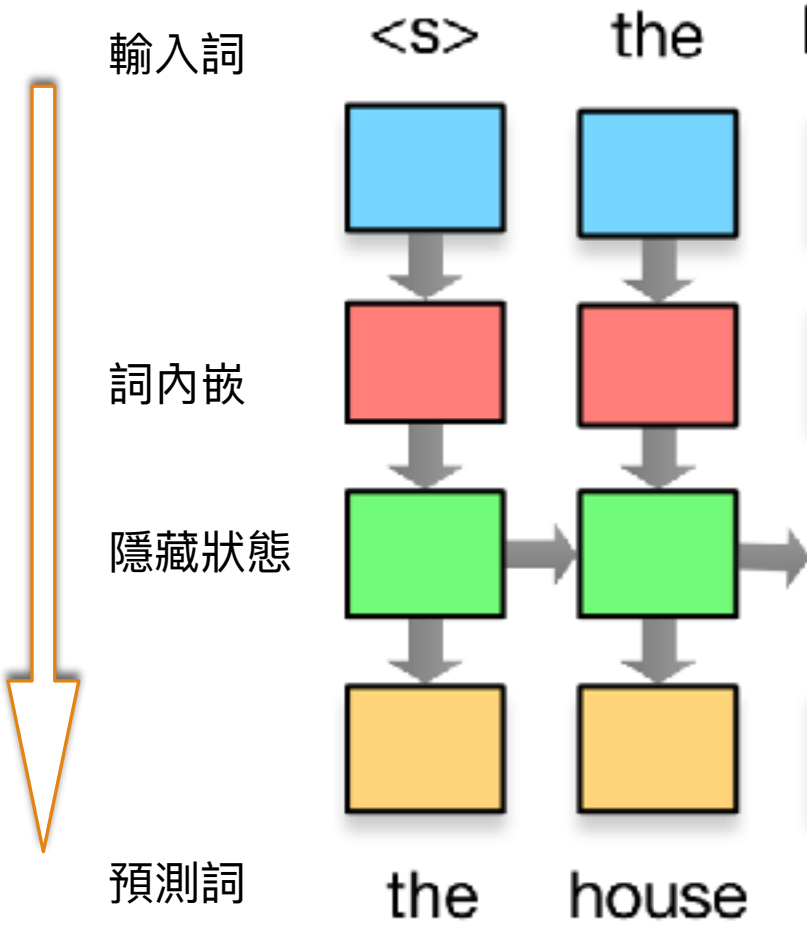
遞迴類神經網路 (由上往下)

- 啟動
- 輸入：前一詞 **<S>**，輸出：第一詞 **the** 和 狀態 **h**



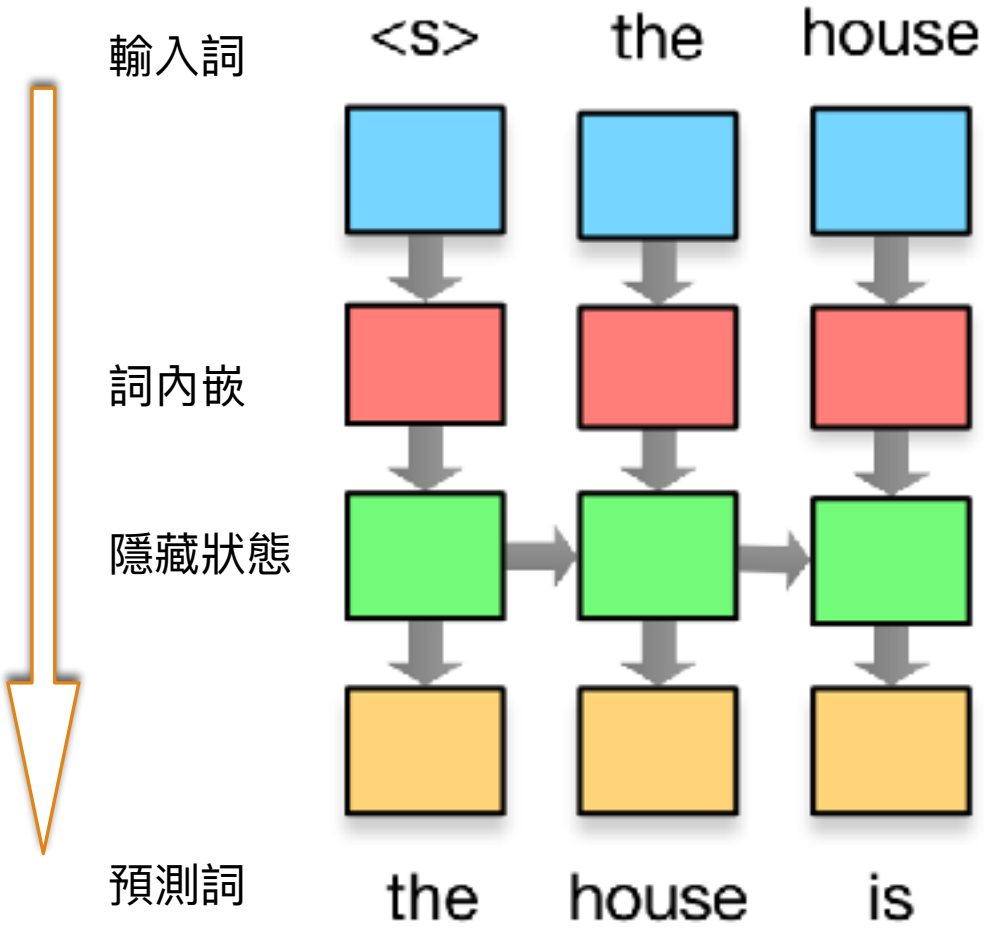
遞迴類神經網路

- 反覆實施
- 輸入：前一詞 **the** 和狀態 **h**，輸出：下一詞 **house** 和 狀態 **h**



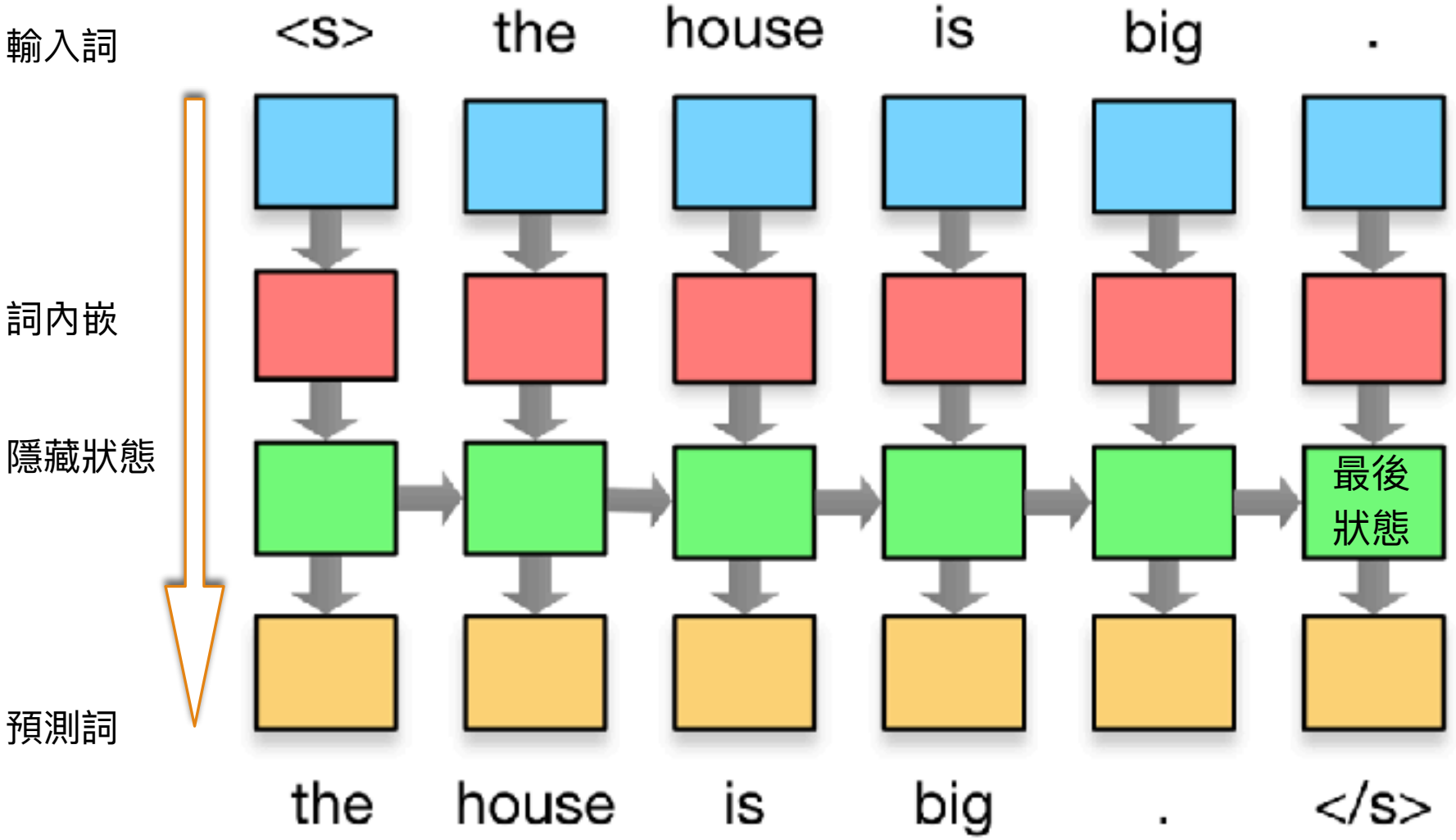
遞迴類神經網路

- 反覆實施
- 輸入：前一詞 **house** 和狀態 **h**，輸出：下一詞 **is** 和 狀態 **h**



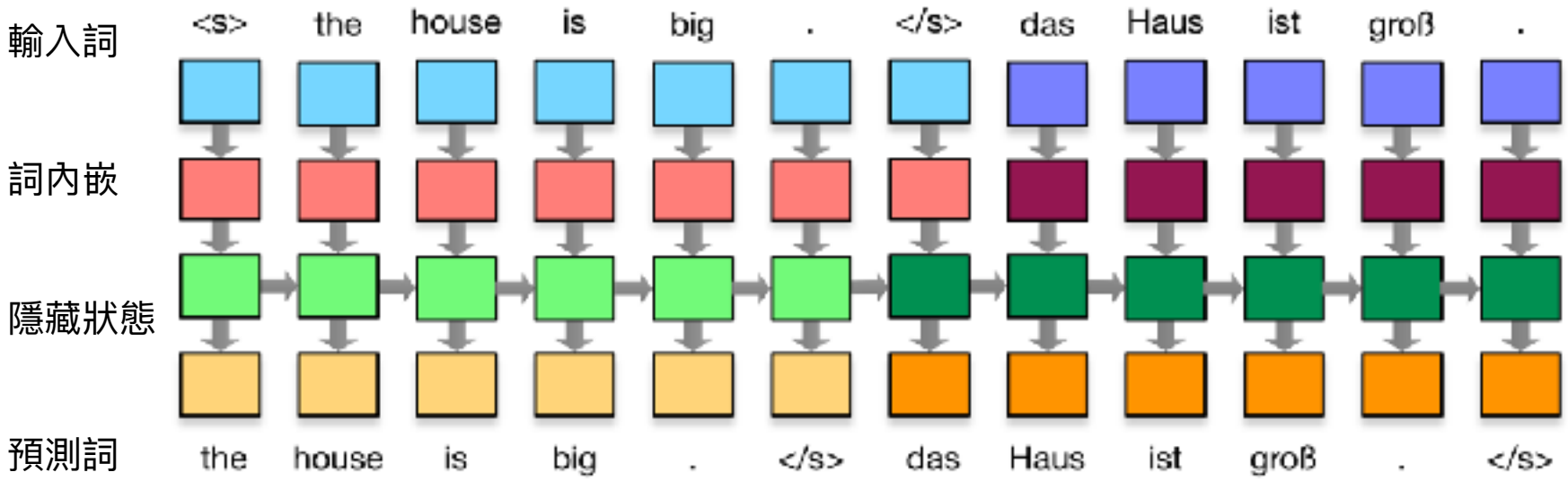
遞迴類神經網路 (不需輸出、只需最後的狀態)

- 反覆實施，直到出現 </S>



遞迴類神經翻譯網路

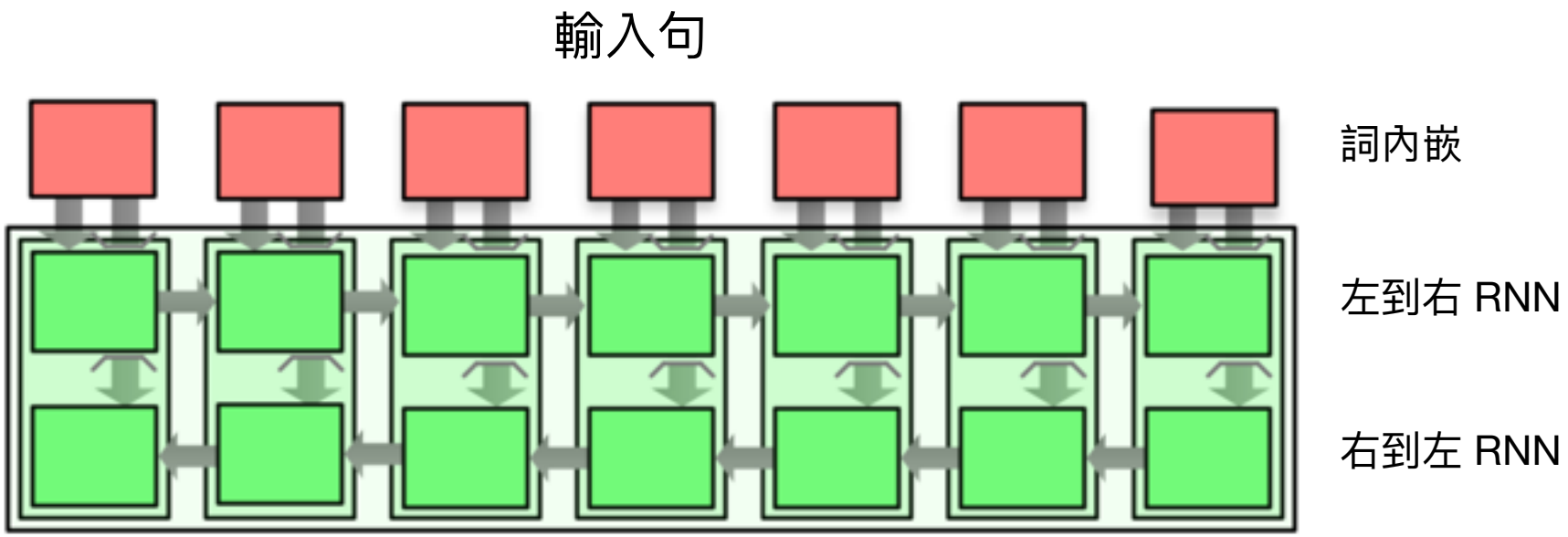
- Google 研究員 Sutskever et al. (2014) 提出「編碼解碼模型」
  - 語言模型可預測下一詞，或許可接著預測翻譯 (好像狂想)
- 訓練的時候：知道編碼器、解碼器的輸入和輸出 (兩者同時執行)
  - 訓練資料輸入和輸出，在時間軸，差一個位置 (預先安排好)



- 執行的時候：不需要編碼器的輸出，不知道解碼器的輸入 (兩者分開執行)
- 缺乏輸入詞、輸出詞對應 (word alignment) 的考慮
  - 解決之道：注意機制 attention mechanism

雙向 RNN 的編碼模型效果更好

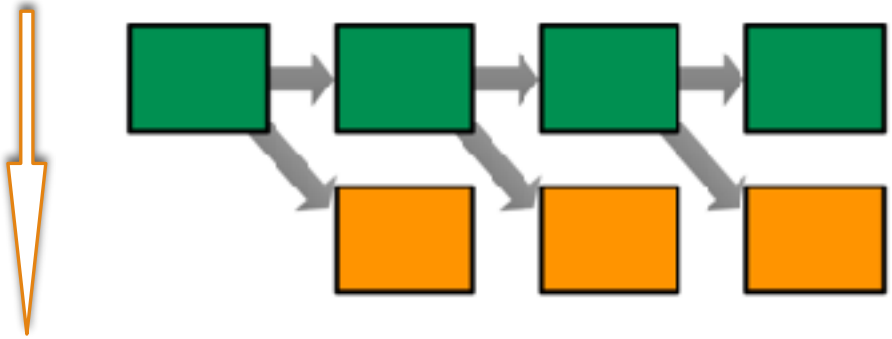
- 觀察 Sutskever 編碼解碼模型的隱藏層
- 正向 H_i 記錄第 i 詞左邊的文脈
- 反向 $\hat{H}_i$ 記錄第 i 詞右邊的文脈



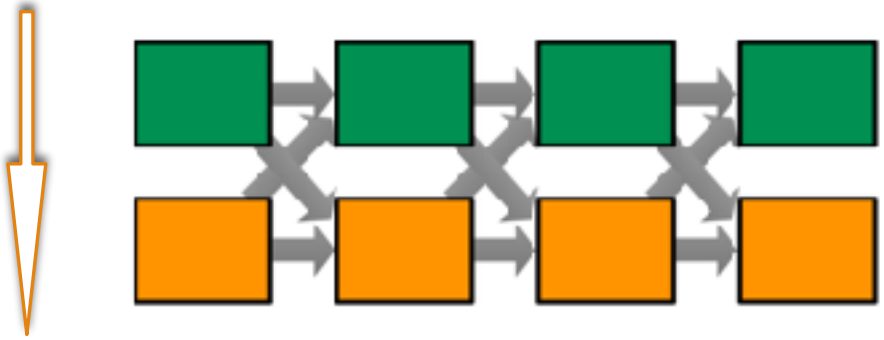
5.2.2 解碼器的設計的改善

- RNN 解碼器的考量

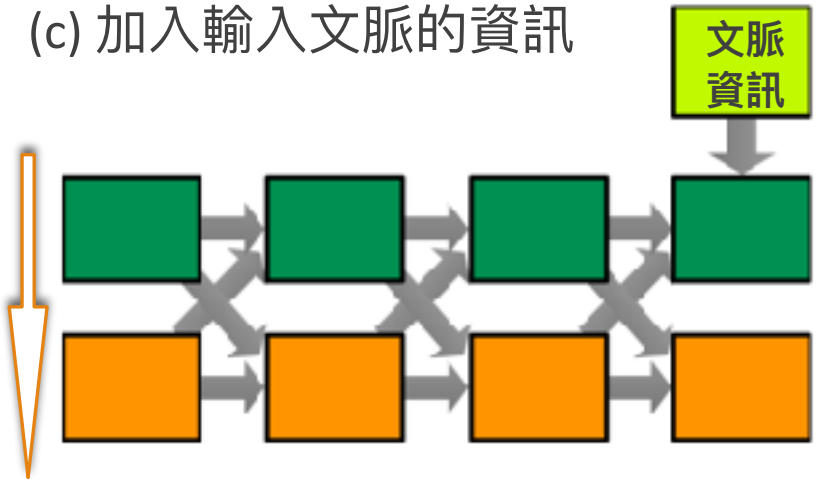
(a) 前狀態 ⟹ 輸出＋狀態



(b) 前狀態+前輸出 ⟹ 輸出＋狀態

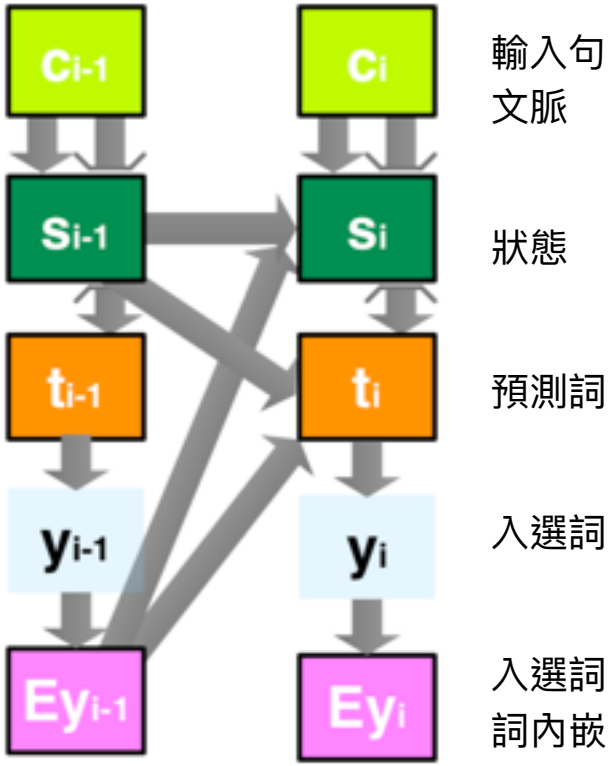


(c) 加入輸入文脈的資訊



5.2.2 解碼器 (詳細分解)

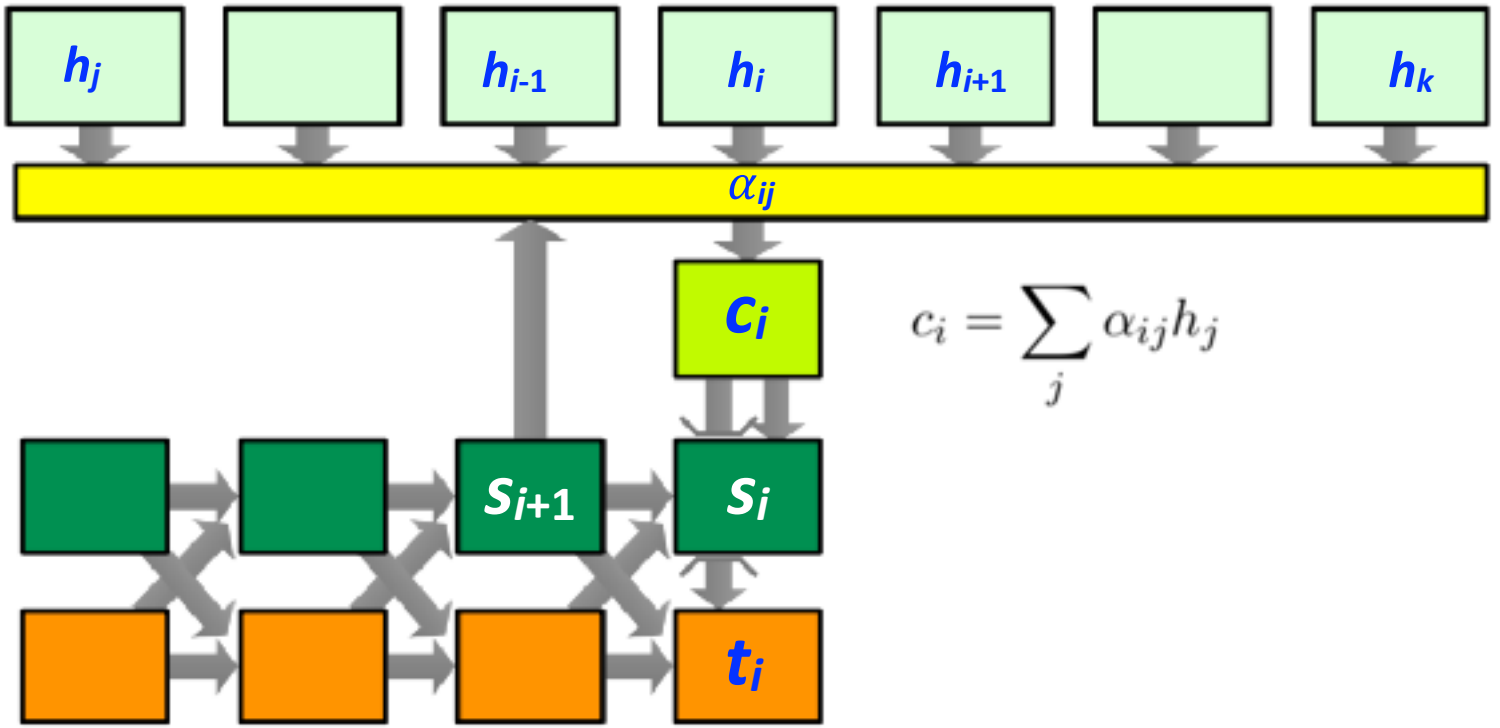
- 解碼器產生隱藏狀態序列 $s_i = f(s_{i-1}, Ey_{i-1}, c_i)$
- 其中的 f() 函數可使用各種元件實現
  - feed-forward layer
  - GRU
  - LSTM 等等
- 產生輸出詞 y_i 的過程
  - 計算向量 (詞彙表的大小)
    $t_i = W(Us_{i-1} + V\,Ey_{i-1} + Cc_i)$
  - 正規化 t_i, 成機率分布
  - 找機率最高值向量 $y_i = \mathbf{max}\ t_i$
  - 向量 Ey_i = 輸出 y_i 的詞向量$_i$



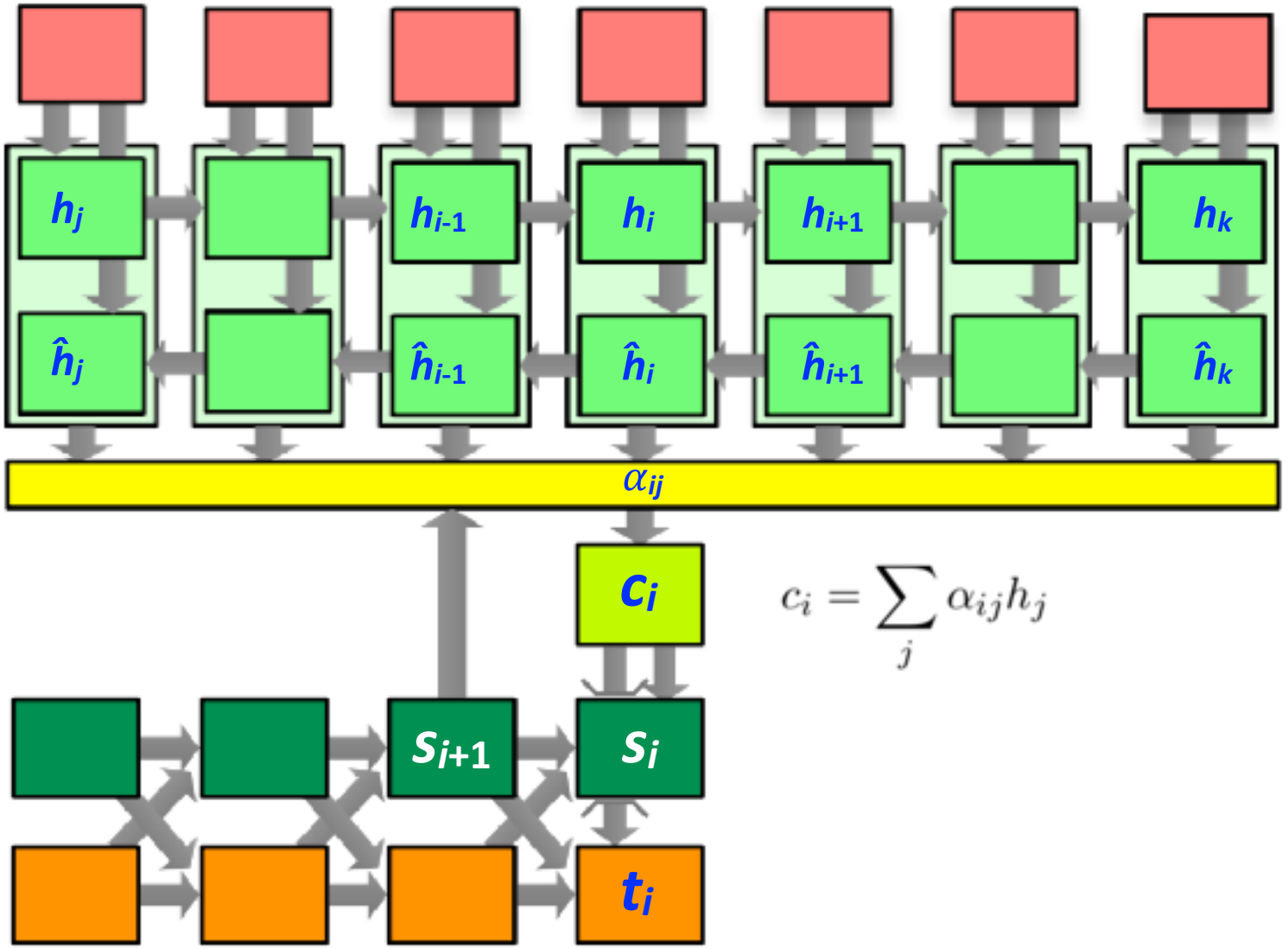
5.2.3 注意機制 (難以視覺化，用公式表達)

- 輸入：來源句文脈 $h_j=(h_j, \hat{h}_j)$、解碼隱藏狀態 s_{i-1}
- 輸出：(焦點或注意) 文脈狀態 c_i.
- 權重：w^a, u^a, b^a

$$a(s_{i-1}, h_j) = w^{aT} s_{i-1} + u^{aT} h_j + b^a \qquad \alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}$$

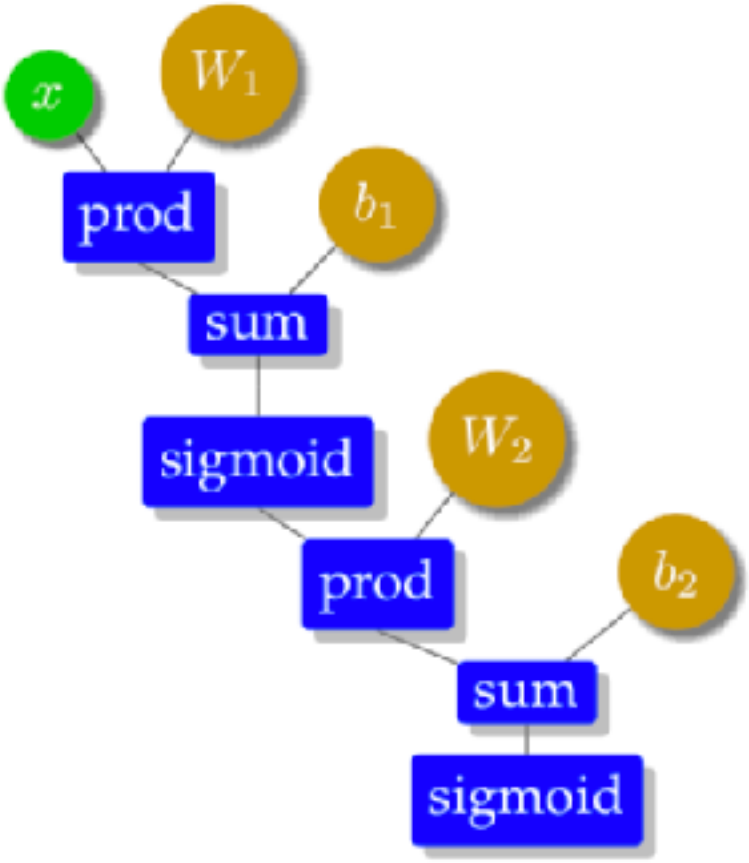


5.2.3 雙向 RNN 編碼＋注意＋解碼機制



5.4 訓練—計算圖

- 訓練的數學基礎
  - 類神經網路定義了一個計算圖 computation graph
  - 用計算圖向前計算計算誤差
  - 用計算圖向後擴展計算梯度
    - 修正模型權重

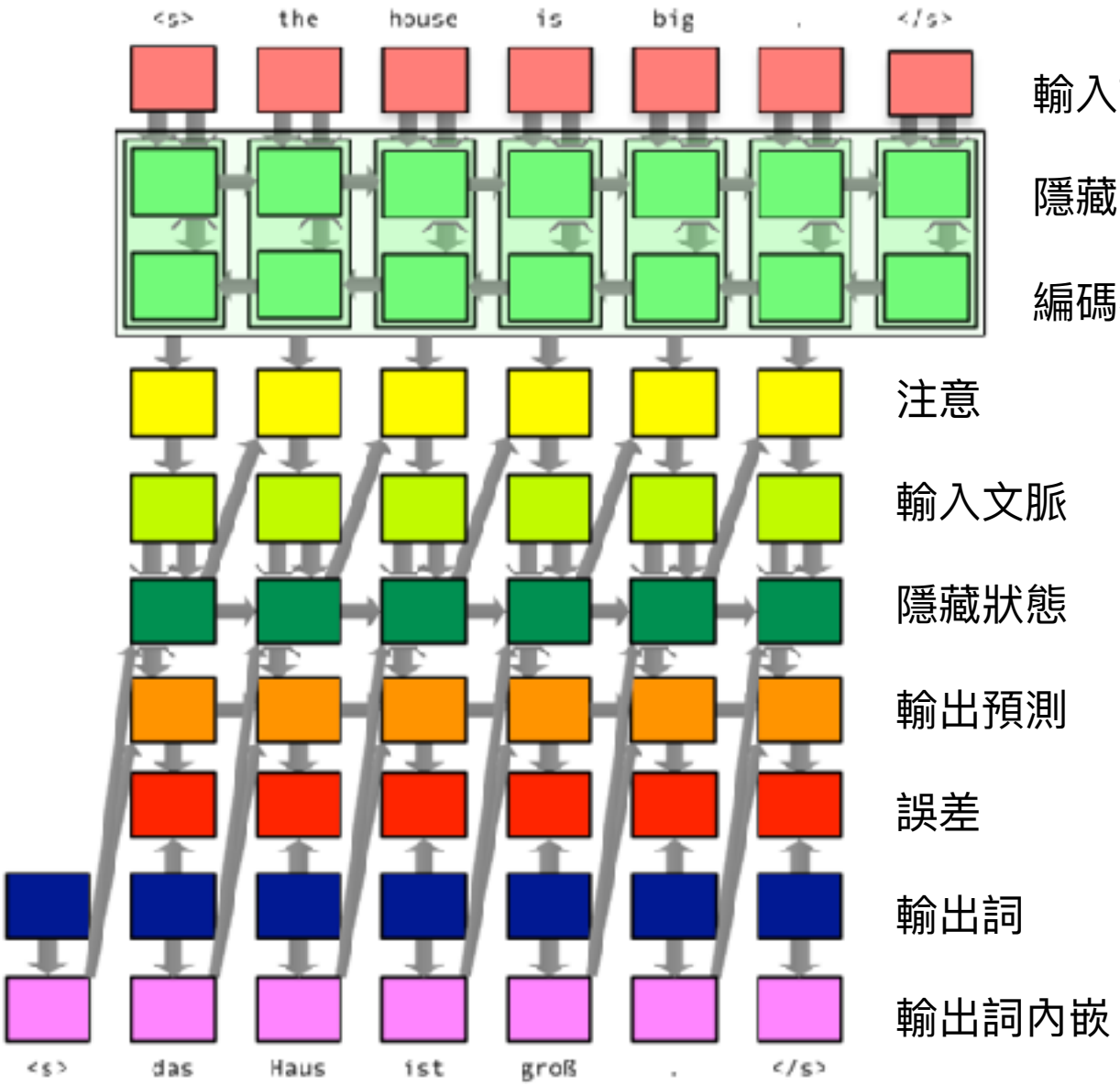


5.4 訓練—時間軸上展開 (不需要編碼器輸出)

編碼器

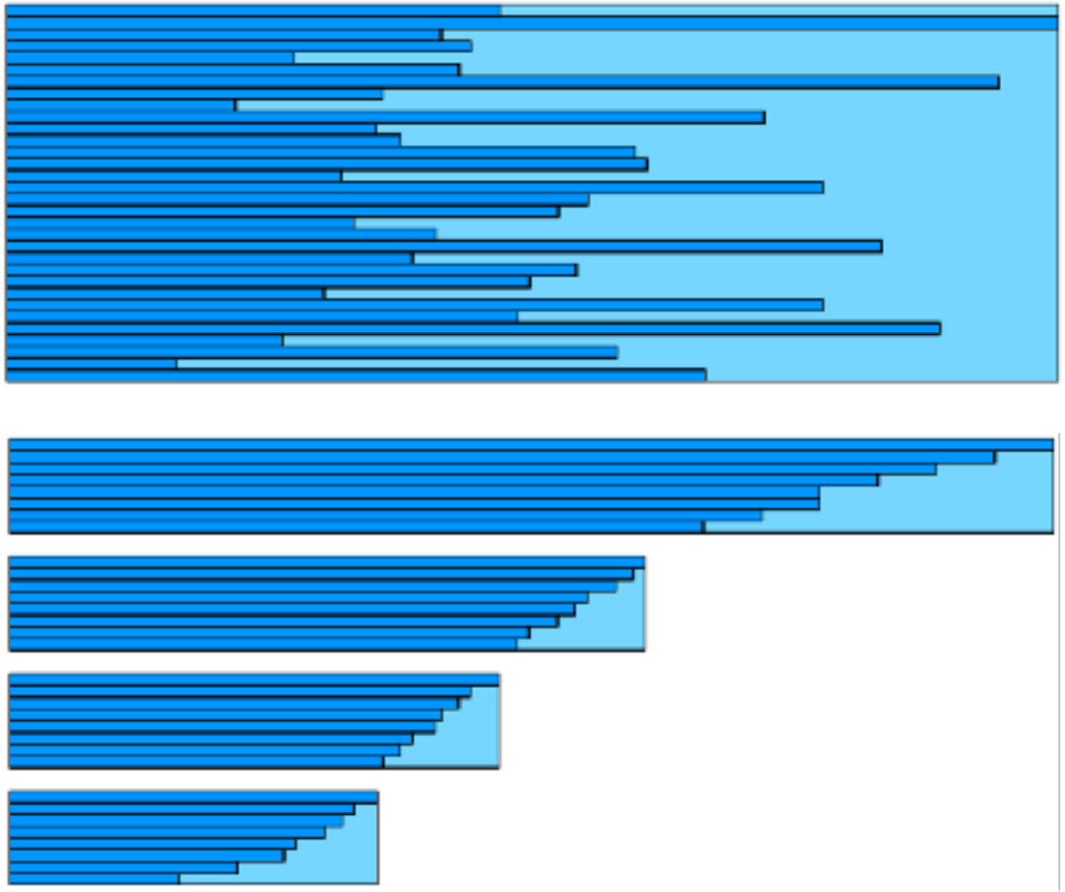
— — —

解碼器



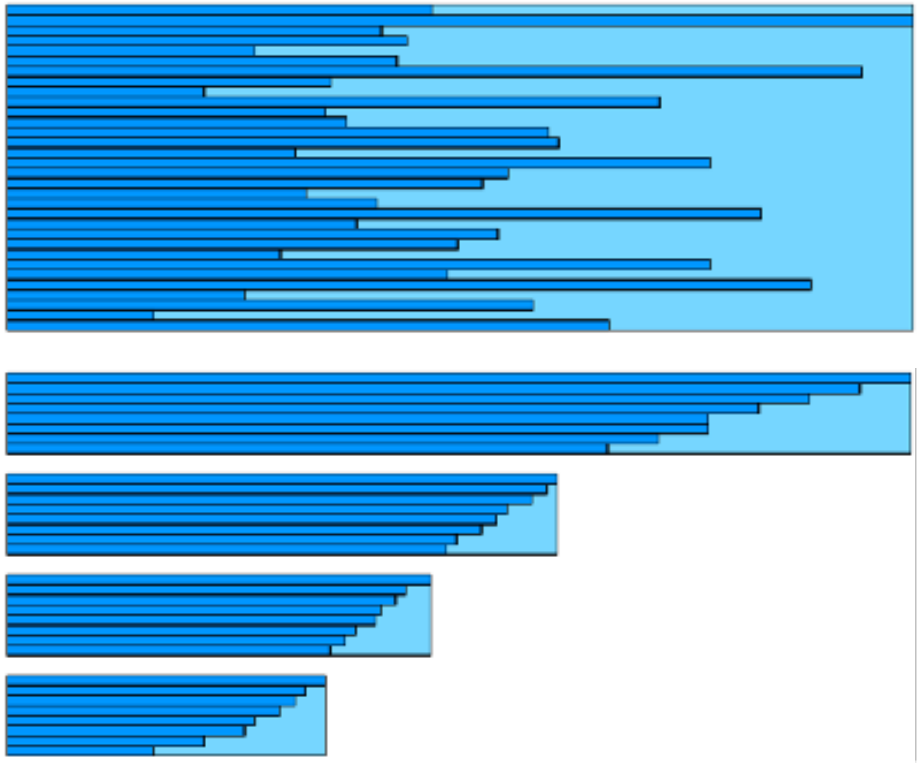
5.3 訓練的批次安排

- 句子的長度不一，整批處理時必須補滿空隔
  ⇒ 浪費很多空間、時間
- 解決之道
  - 句子排序
  - 照長度分批次
- 例子
  - 大批次：1600 句配對
  - 小批次：80 句配對



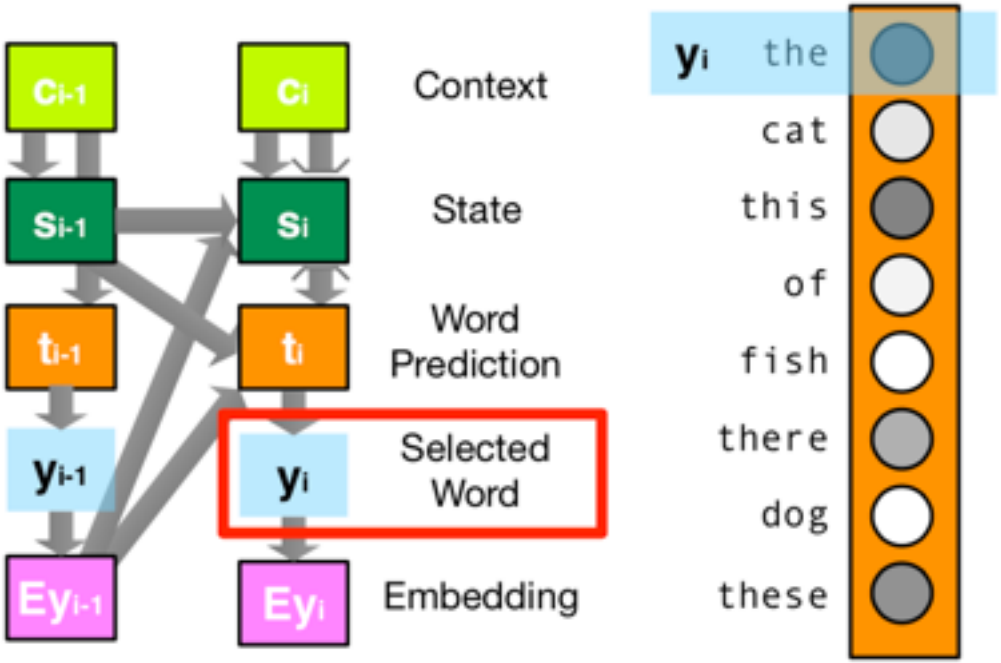
5.3 訓練的批次安排

- 調動語料庫的句子
- 分成大批次 maxi-batches，再分成小批次 mini-batches
- 處理小批次，更新權重
- 一回接著一回反覆處理
- 典型回合 epoch 次數
  - 5-15 回合 epochs



5.4 最可能預測與光束搜尋法 (1)

- 預測並輸出最可能的第一詞



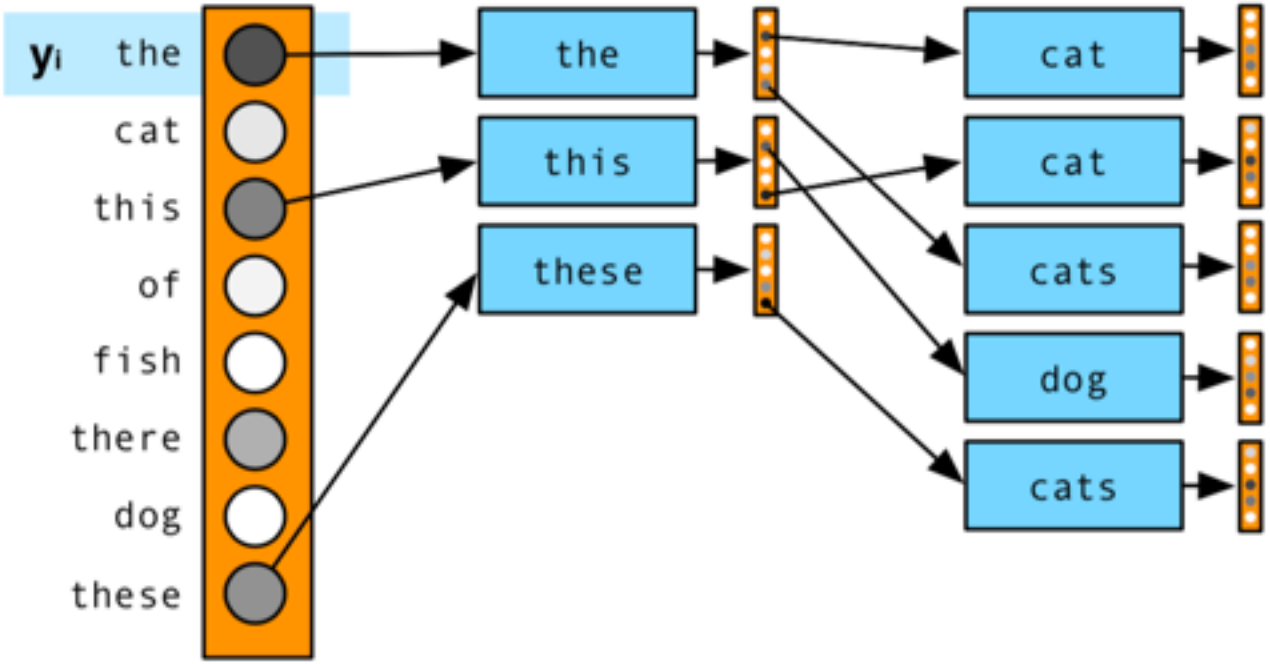
5.4 最可能預測與光束搜尋法 (2)

- 預測並輸出
  - Top N: 同義詞、構詞變化
- 每次都輸出最佳預測詞
- 問題 (garden-path problem)
  - 直到最後才發現最早的決定並非最佳
- 解答： 光束搜尋
  - 考慮、組合 Top N 預測
  - 計算最佳「路徑」
  - 在路徑節點保持少量最佳候選詞 (即光束寬度)

Best		Alternatives
but	(42.1%)	*however* (25.3%), *I* (20.4%), *yet* (1.9%
I	(80.4%)	*also* (6.0%), , (4.7%), *it* (1.2%), *in* (0.7
also	(85.2%)	*think* (4.2%), *do* (3.1%), *believe* (2.9%)
believe	(68.4%)	*think* (28.6%), *feel* (1.6%), *do* (0.8%), .
he	(90.4%)	*that* (6.7%), *it* (2.2%), *him* (0.2%), ...
is	(74.7%)	*'s* (24.4%), *has* (0.3%), *was* (0.1%), ...
clever	(99.1%)	*smart* (0.6%), ...
enough	(99.9%)	
to	(95.5%)	*about* (1.2%), *for* (1.1%), *in* (1.0%), *of*
keep	(69.8%)	*maintain* (4.5%), *hold* (4.4%), *be* (4.2%
his	(86.2%)	*its* (2.1%), *statements* (1.5%), *what* (1.
statements	(91.9%)	*testimony* (1.5%), *messages* (0.7%), *co*
vague	(96.2%)	*v@@* (1.2%), *in* (0.6%), *ambiguous* (0.
enough	(98.9%)	*and* (0.2%), ...
so	(51.1%)	, (44.3%), *to* (1.2%), *in* (0.6%), *and* (0.
they	(55.2%)	*that* (35.3%), *it* (2.5%), *can* (1.6%), *yo*
can	(93.2%)	*may* (2.7%), *could* (1.6%), *are* (0.8%),
be	(98.4%)	*have* (0.3%), *interpret* (0.2%), *get* (0.2
interpreted	(99.1%)	*interpre@@* (0.1%), *constru@@* (0.1%)
in	(96.5%)	*on* (0.9%), *differently* (0.5%), *as* (0.3%
different	(41.5%)	*a* (25.2%), *various* (22.7%), *several* (3.
ways	(99.3%)	*way* (0.2%), *manner* (0.2%), ...
.	(99.2%)	</s> (0.2%), , (0.1%), ...
</s>	(100.0%)	

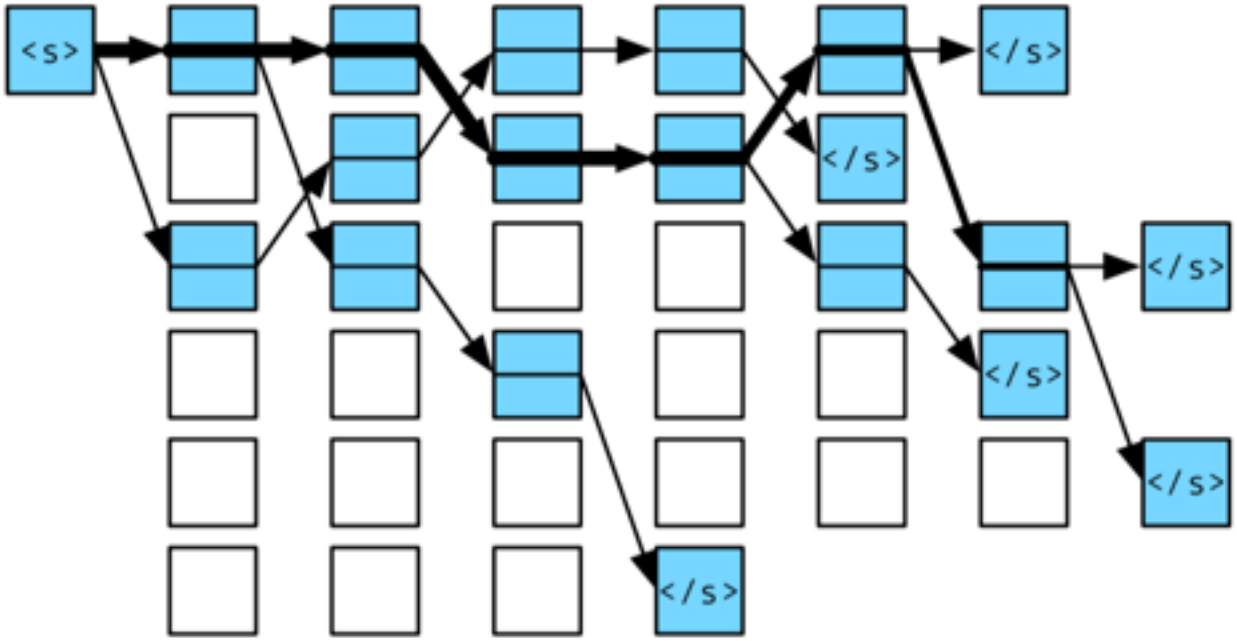
5.4 光束搜尋法 (3)

- 光束搜尋
  - 考慮、組合 Top N 輸出詞
- 在路徑的每點都保持少量最佳候選詞 (即光束)
- 計算最佳「路徑」——RNN 模型或 N-gram 模型



5.4 光束搜尋法 (4)

- 光束搜尋
  - 考慮、組合 Top N 預測詞
- 在路徑的每點都保持少量最佳候選詞 (即光束寬度)
- 輸出最佳「路徑」



資源

- Keras 範例
  - https://github.com/keras-team/keras
  - https://github.com/awslabs/keras-apache-mxnet/blob/master/examples/lstm_seq2seq.py
- TensorFlow 範例
  - nlp.stanford.edu/projects/nmt/Luong-Cho-Manning-NMT-ACL2016-v4.pdf
  - sites.google.com/site/acl16nmt/home/resources
  - www.tensorflow.org/tutorials/
    - github.com/tensorflow/nmt
- PyTorch 範例
  - https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

程式範例 (lstm_seq2seq.py)

- We start with input sequences from English to French sentences

- An encoder LSTM turns input sequences to 2 state vectors
  (we keep the last LSTM state and discard the outputs).

- A decoder LSTM is trained to turn the target sequences into
  the same sequence but offset by one timestep in the future

  Uses as initial state the state vectors from the encoder.
  Learns to generate `targets[t+1…]`given `targets[...t]`,
  conditioned on the input sequence.

- In inference mode, to decode unknown input sequences
  - Encode the input sequence into state vectors
  - Start with a target sequence of size
  - Feed the state vectors and target sequence to the decoder
    to produce predictions for the next character
  - Use argmax to produce the next character using predictions
  - Append the character to the target sequence
  - Repeat until generating end-of-sequence/hitting length limit