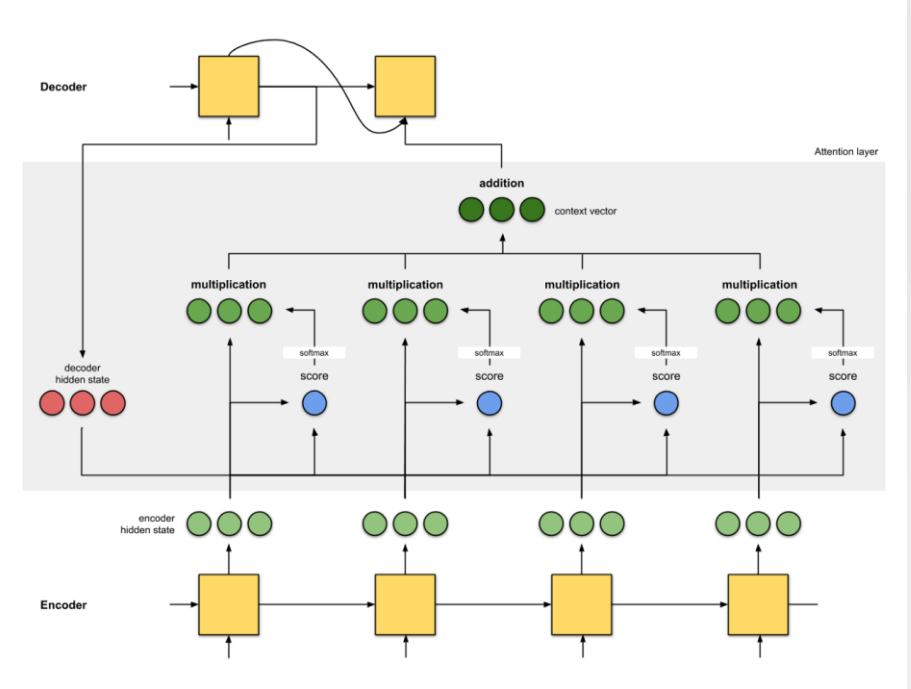
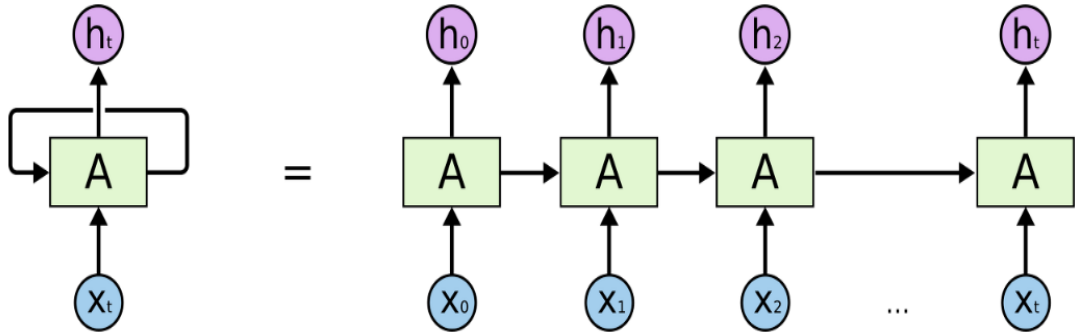
Chapter 09. 시계열을 활용한 딥러닝 (Time Sequence Processing)

RNN Types , Attention



Recurrent Neural Network

- **Recurrent :** 반복되는 -> 시퀀스의 모든 요소에 대해 동일한 작업을 수행하고 출력은 이전 계산에 의존하기 때문에 *반복적* 이라고 합니다.
- RNN은 지금까지 계산 된 것에 대한 정보를 캡처하는 **"메모리"**가 있다는 것입니다



folded

Un folded

RNN 의 특징 :

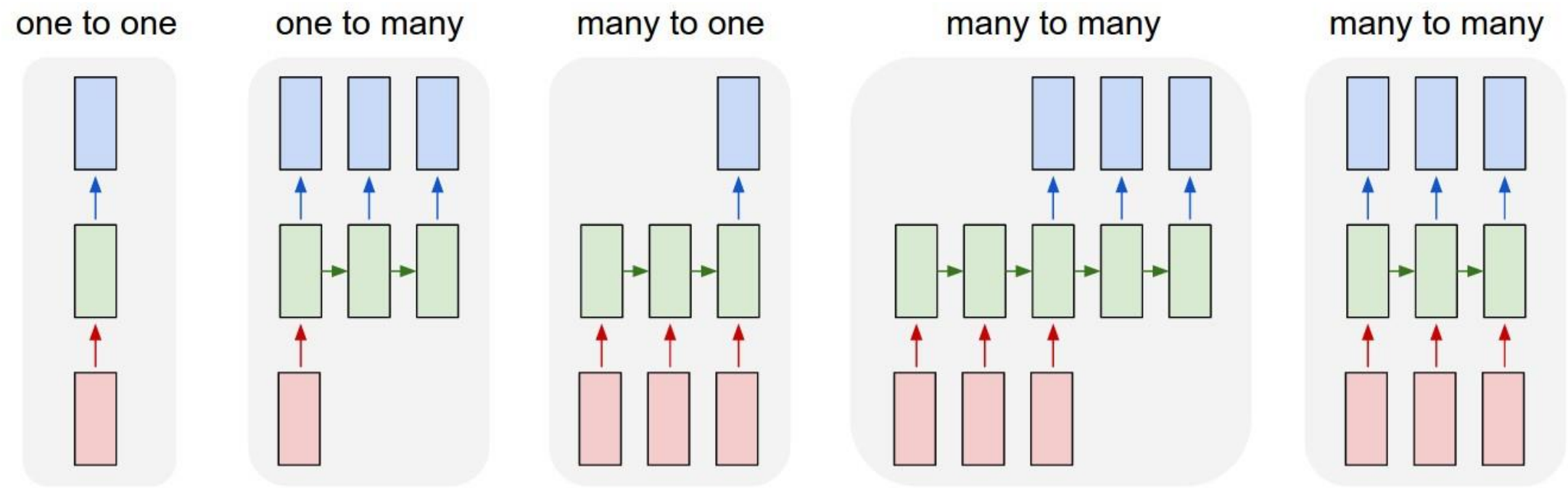
1) 장점 :

- 시퀀스 길이에 관계없는 in put/output

2) 단점 :

- Gradient Vanishing : 상대적으로 짧은 sequence 학습

RNN types



RNN types

(1) Vanilla mode of processing without RNN, (one to one)
from fixed-sized input to fixed-sized output (e.g. image classification).

(2) Sequence output (one to many)
(e.g. image captioning takes an image and outputs a sentence of words).

(3) Sequence input (many to one)
(e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

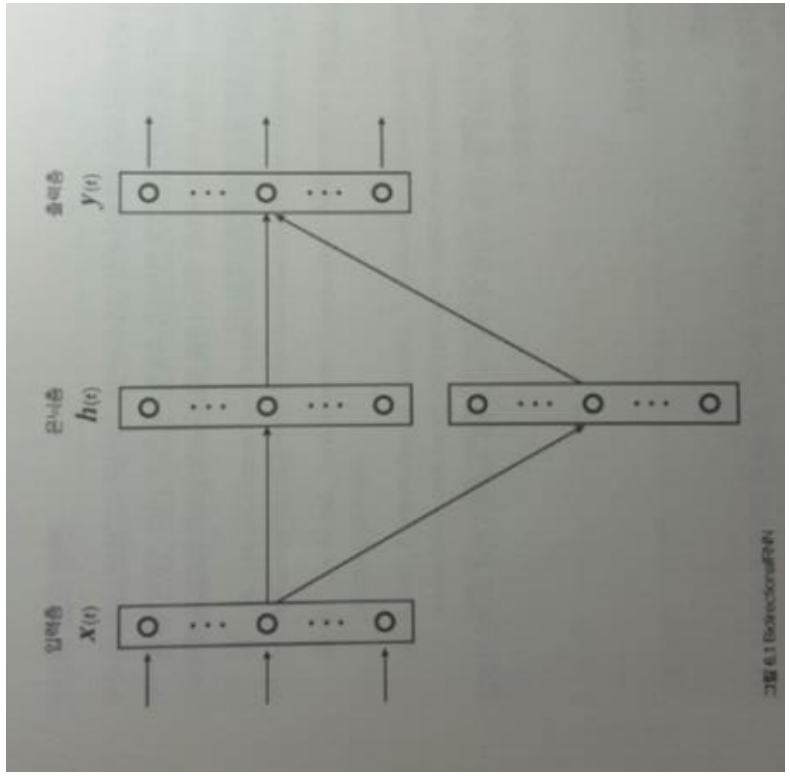
(4) Sequence input and sequence output (many to many)
(e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).

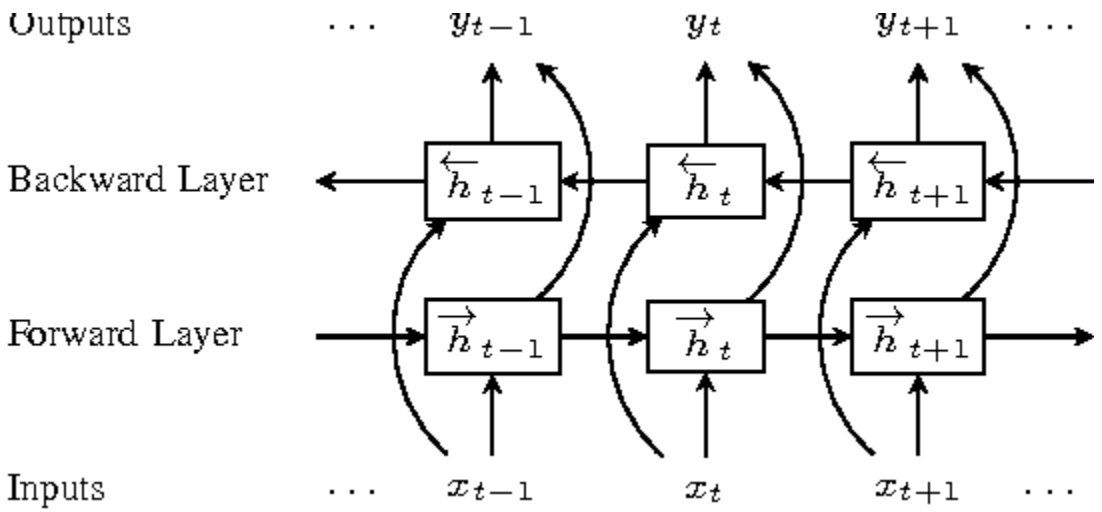
(5) Synced sequence input and output (many to many)
(e.g. video classification where we wish to label each frame of the video).

Bi-directional RNN

양방향 RNN : 과거와 미래의 정보가 모두 현재 스텝에 영향을 주는 경우.

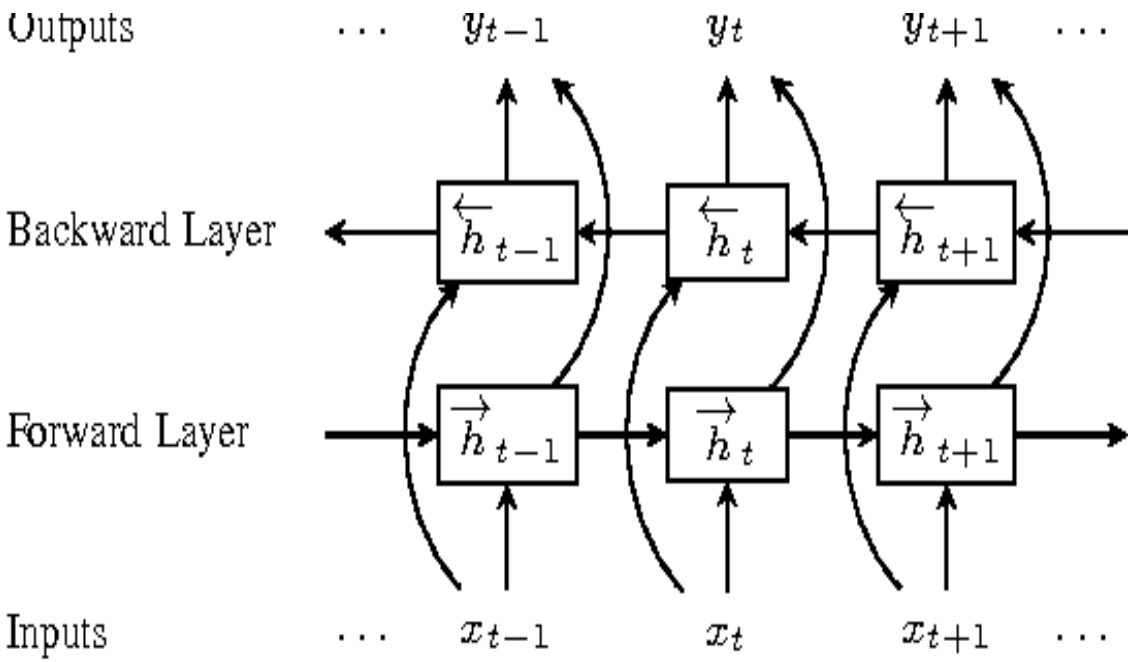
예 : 영어 문제에서 빈칸에 가장 알맞는 단어를 채우기 (앞, 뒤 문장을 모두 확인해야 한다)





Bi-directional RNN

Bi-directional RNN



forward

backward

$$\overrightarrow{h}(t) = f\left(\overrightarrow{U}x(t) + \overrightarrow{W}h(t-1) + \overrightarrow{b}\right)$$

$$\overleftarrow{h}(t) = f\left(\overleftarrow{U}x(t) + \overleftarrow{W}h(t+1) + \overleftarrow{b}\right)$$

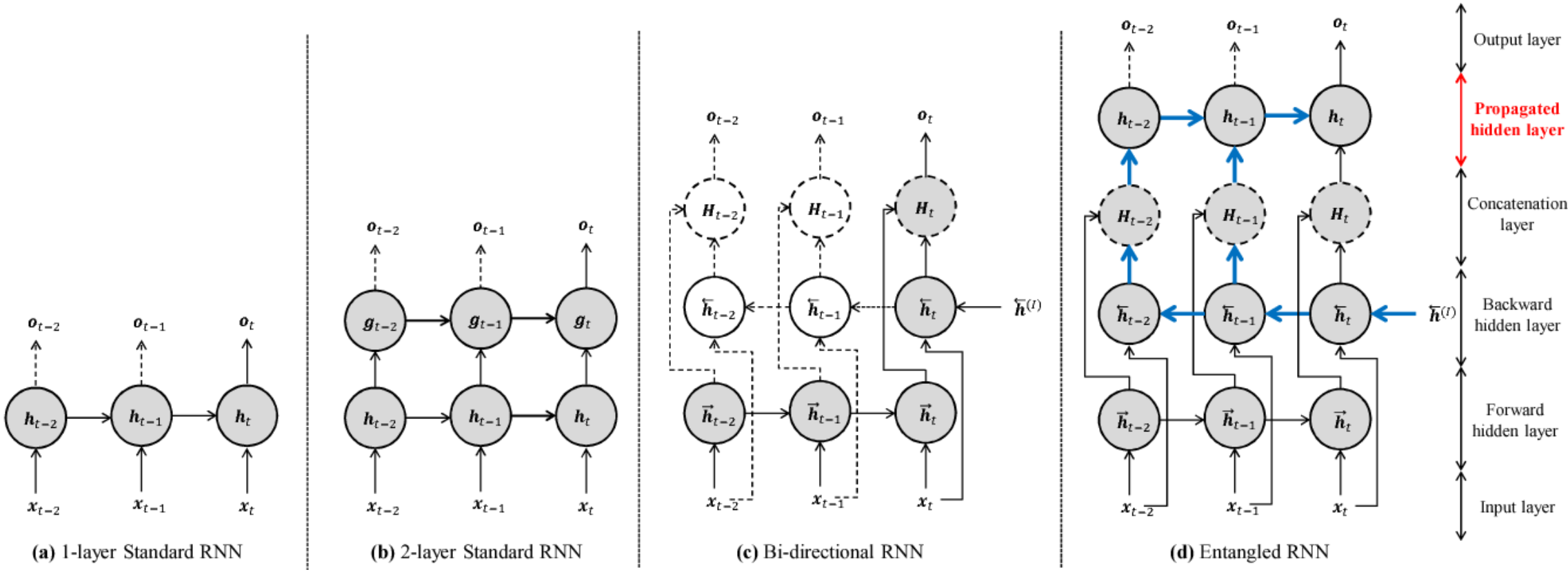
merge

$$h(t) = \begin{pmatrix} \overrightarrow{h}(t) \\ \cdots\cdots \\ \overleftarrow{h}(t) \end{pmatrix}$$

output

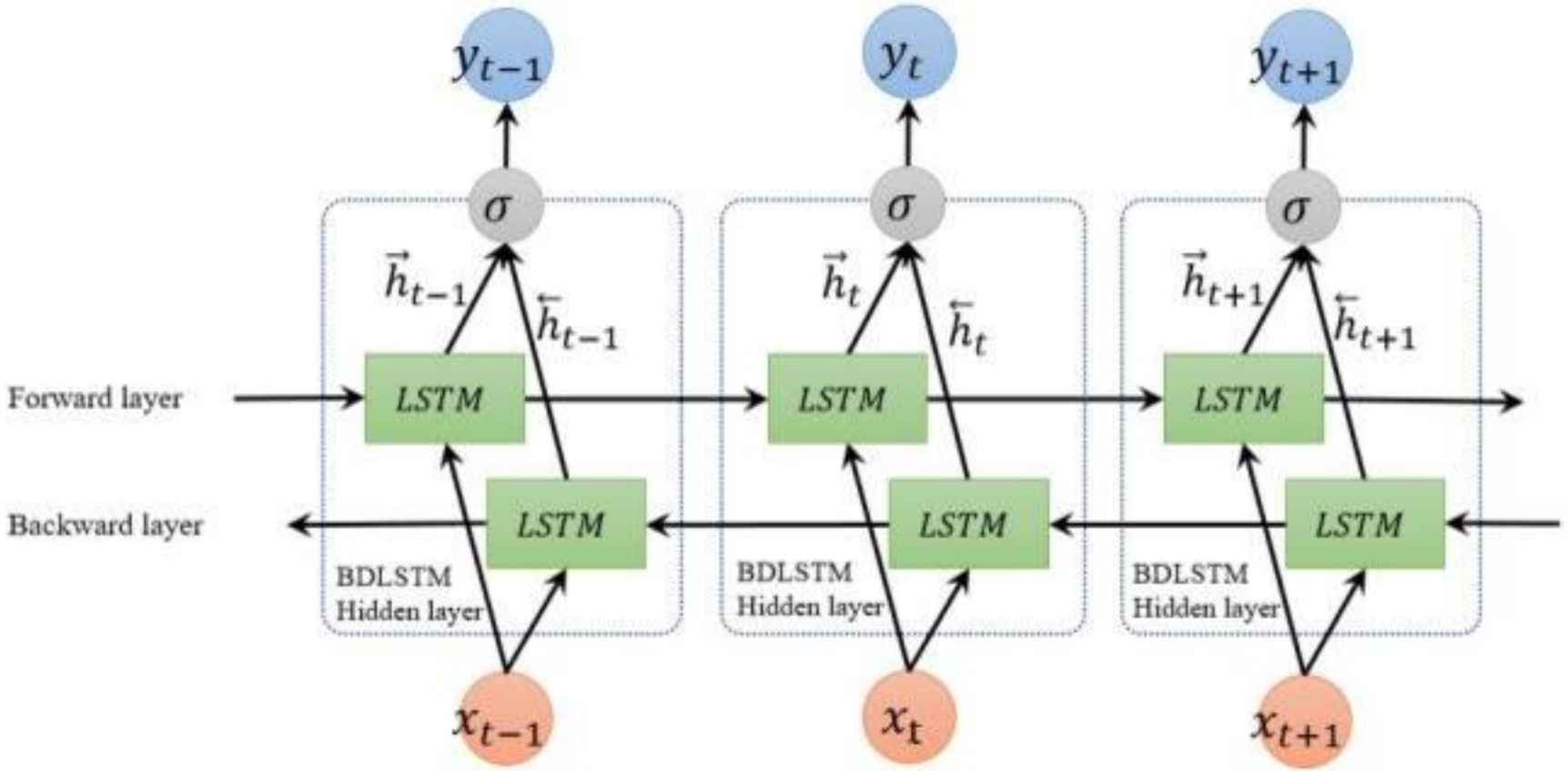
$$y(t) = Vh(t) + c$$

N- layers RNN



(a) 1-layer Standard RNN (b) 2-layer Standard RNN (c) Bi-directional RNN (d) Entangled RNN

Bidirectional LSTM

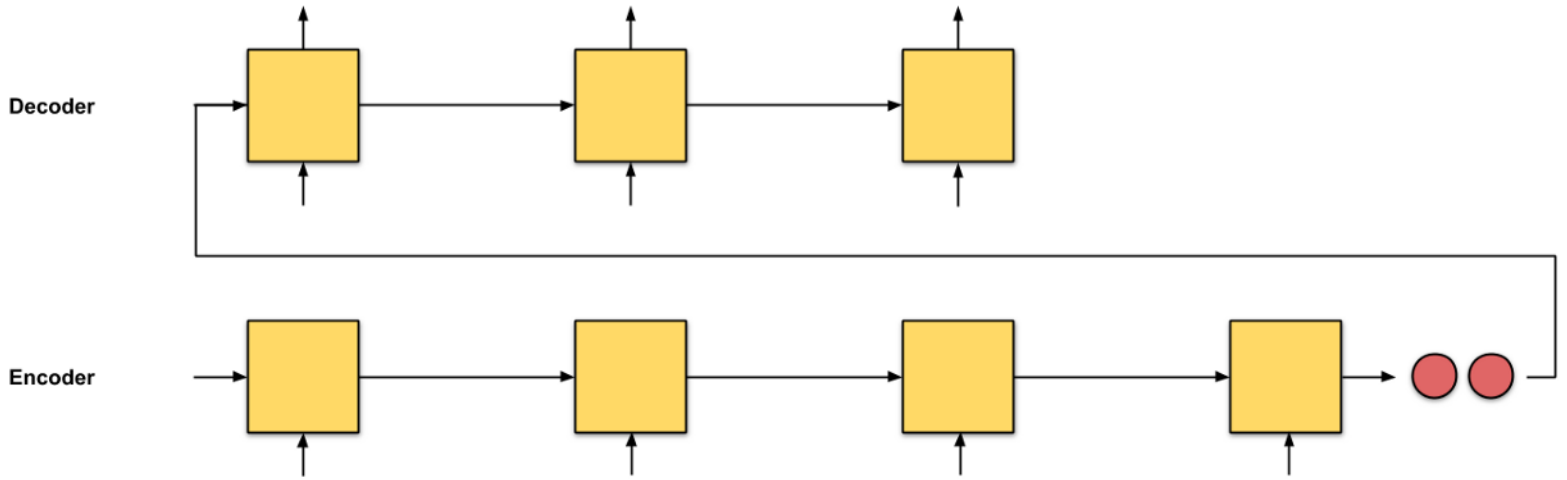


Attention

https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3

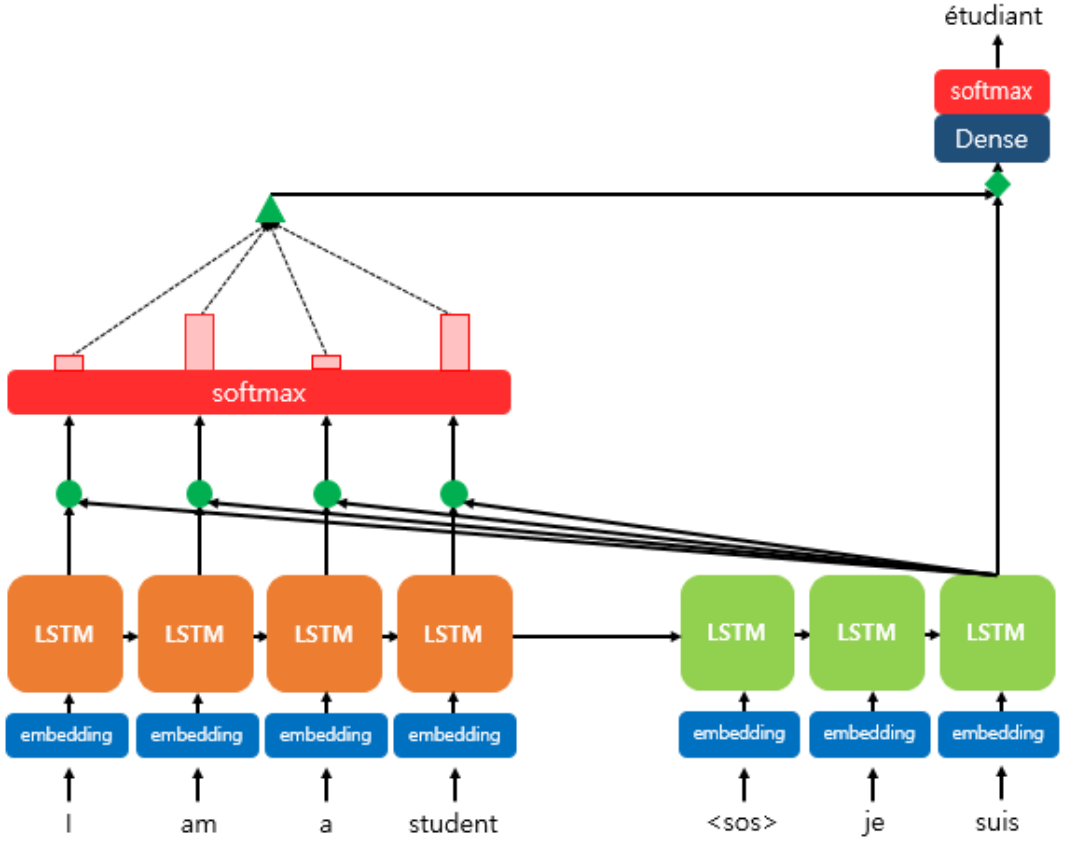
Attention

eq2seq의 문제점: 디코더가 인코더로부터 수신 하는 정보는 **마지막 인코더 숨겨진 상태** 뿐.
Attention 은 모든 인코더 숨겨진 상태의 정보를 디코더에 제공하는
인코더와 디코더 간의 인터페이스입니다



Attention

https://wikidocs.net/22893



Seq2Seq :

번역기는 독일어 텍스트를 처음부터 끝까지 읽습니다. 완료되면, 그는 한 단어 씩 영어로 번역하기 시작합니다. 문장이 매우 길면 본문의 앞 부분에서 읽은 내용을 잊었을 수 있습니다.

Seq2Seq + Attention :

번역자 는 처음부터 끝까지 **키워드를 쓰면서** 독일어 텍스트 를 읽은 후 영어 번역을 시작합니다. 각 독일어 단어를 번역하는 동안 그는 적어 둔 키워드를 사용합니다.

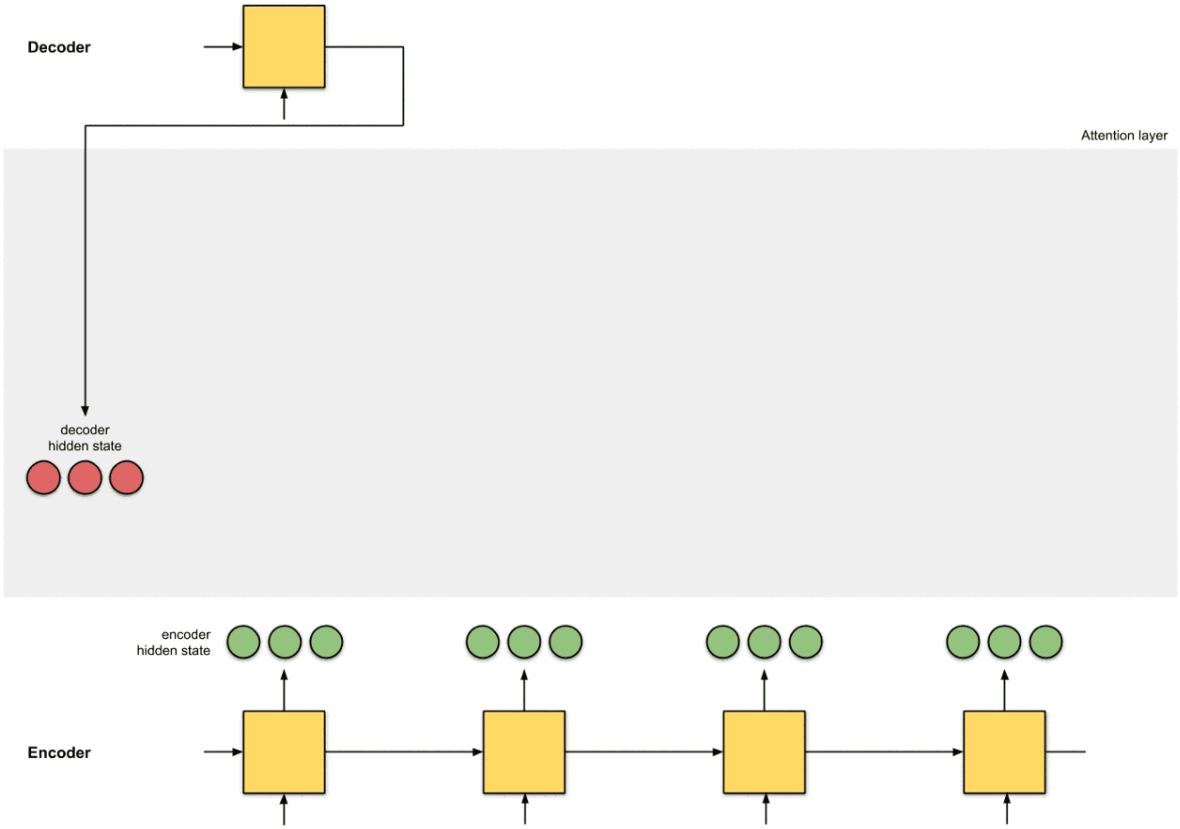
Attention

Step 0: Prepare hidden states.

Encoder

Attention

Step 1: Obtain a score for every encoder hidden state.

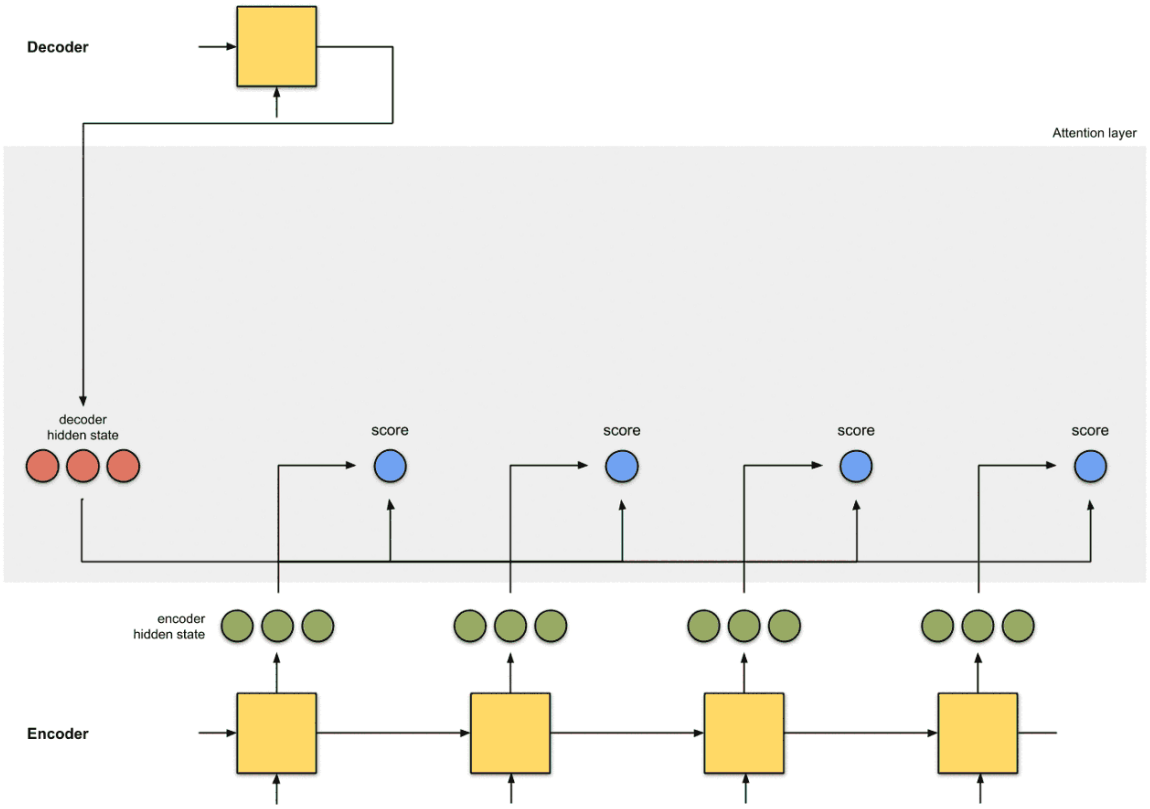



```
decoder_hidden = [10, 5, 10]

encoder_hidden  score
----------------------
     [0, 1, 1]     15 (= 10×0 + 5×1 + 10×1, the dot product)
     [5, 0, 1]     60
     [1, 1, 0]     15
     [0, 5, 1]     35
```

Attention

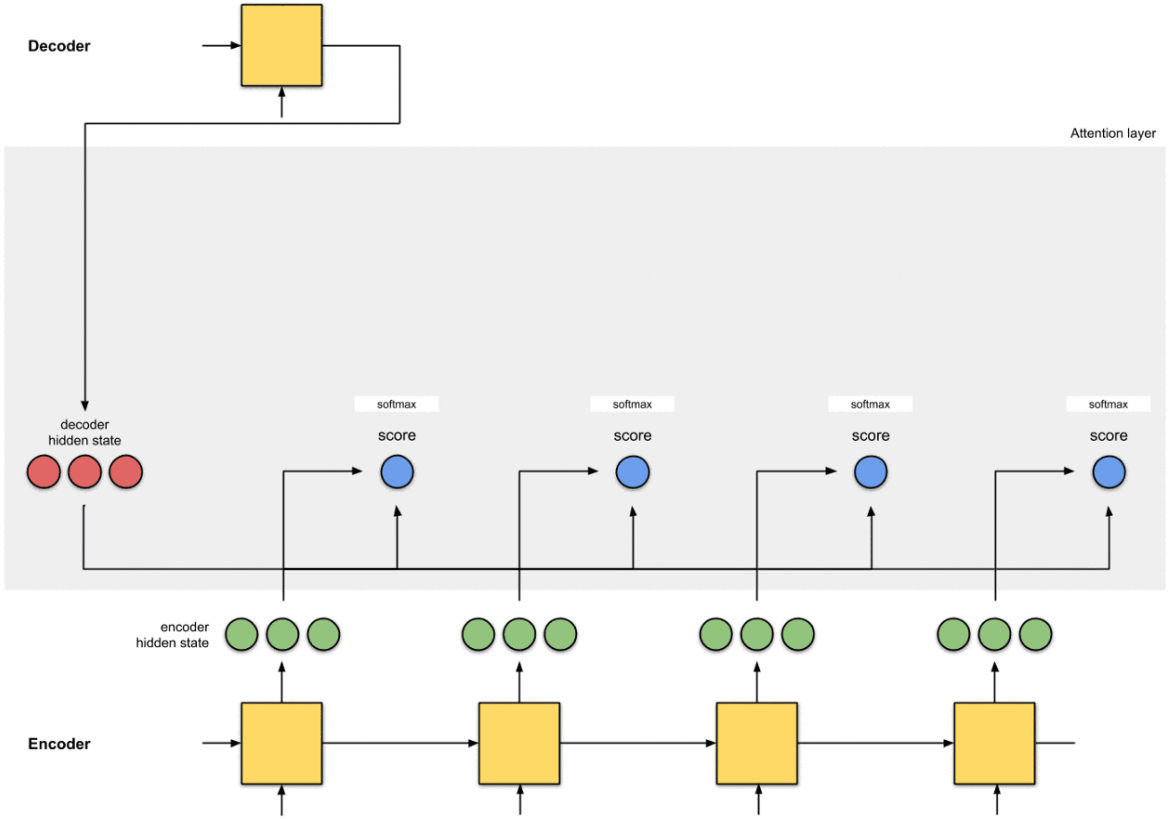
Step 2: Run all the scores through a softmax layer.



encoder_hidden	score	score^
[0, 1, 1]	15	0
[5, 0, 1]	60	1
[1, 1, 0]	15	0
[0, 5, 1]	35	0

Attention

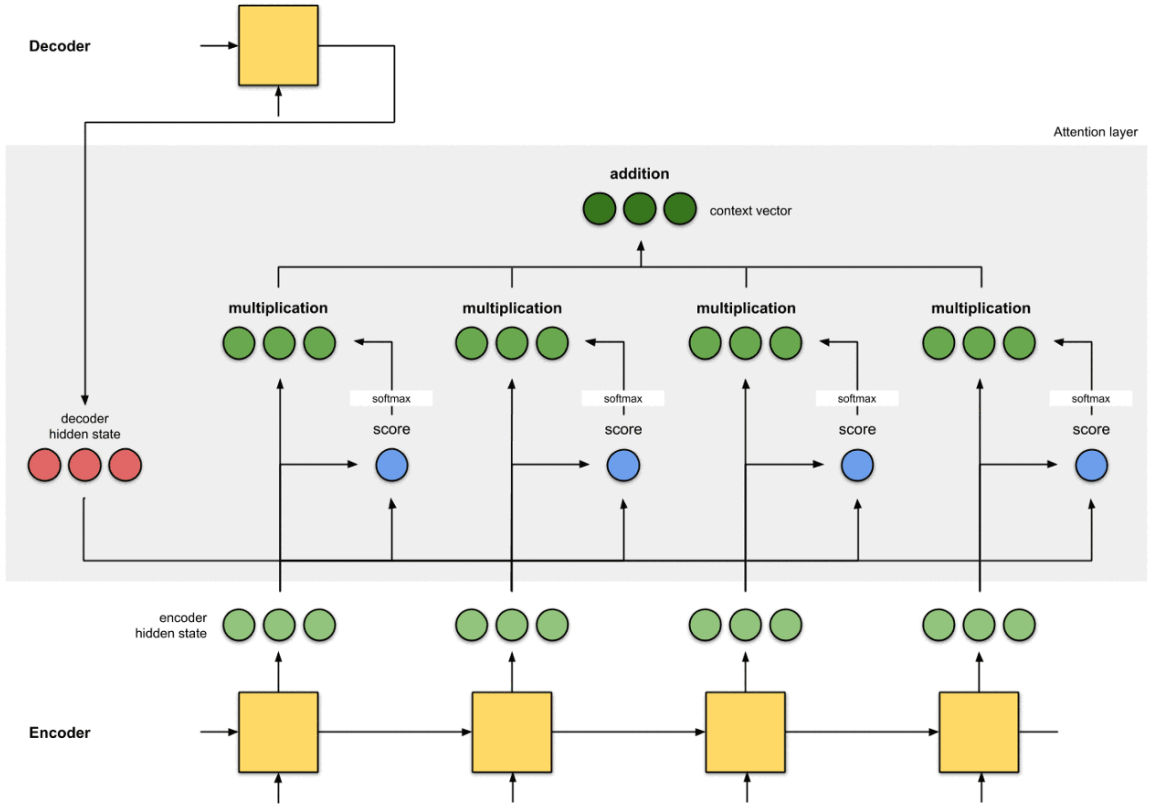
Step 4: Sum up the alignment vectors.



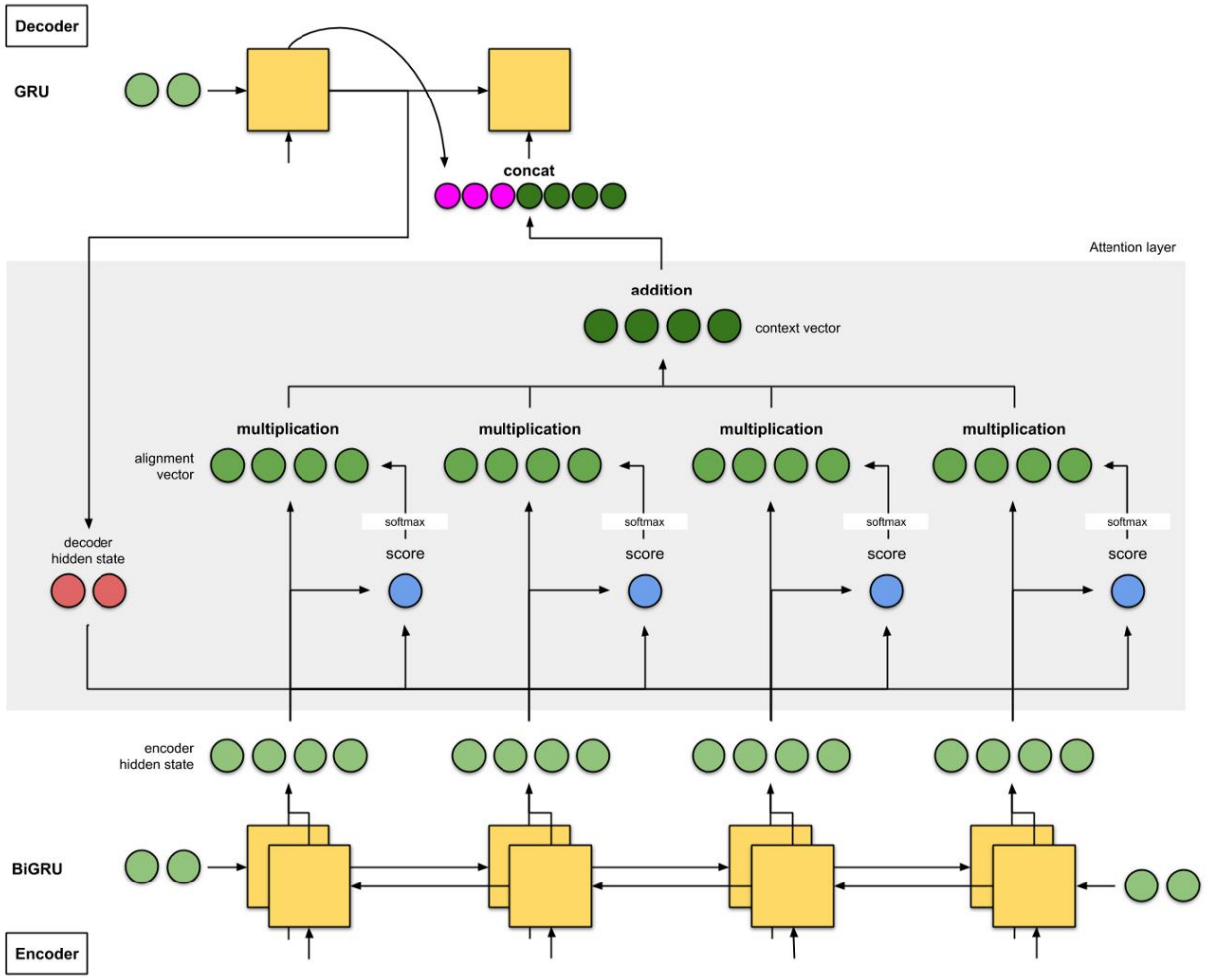
encoder	score	score^	alignment
[0, 1, 1]	15	0	[0, 0, 0]
[5, 0, 1]	60	1	[5, 0, 1]
[1, 1, 0]	15	0	[0, 0, 0]
[0, 5, 1]	35	0	[0, 0, 0]

Attention

Step 5: Feed the context vector into the decoder.



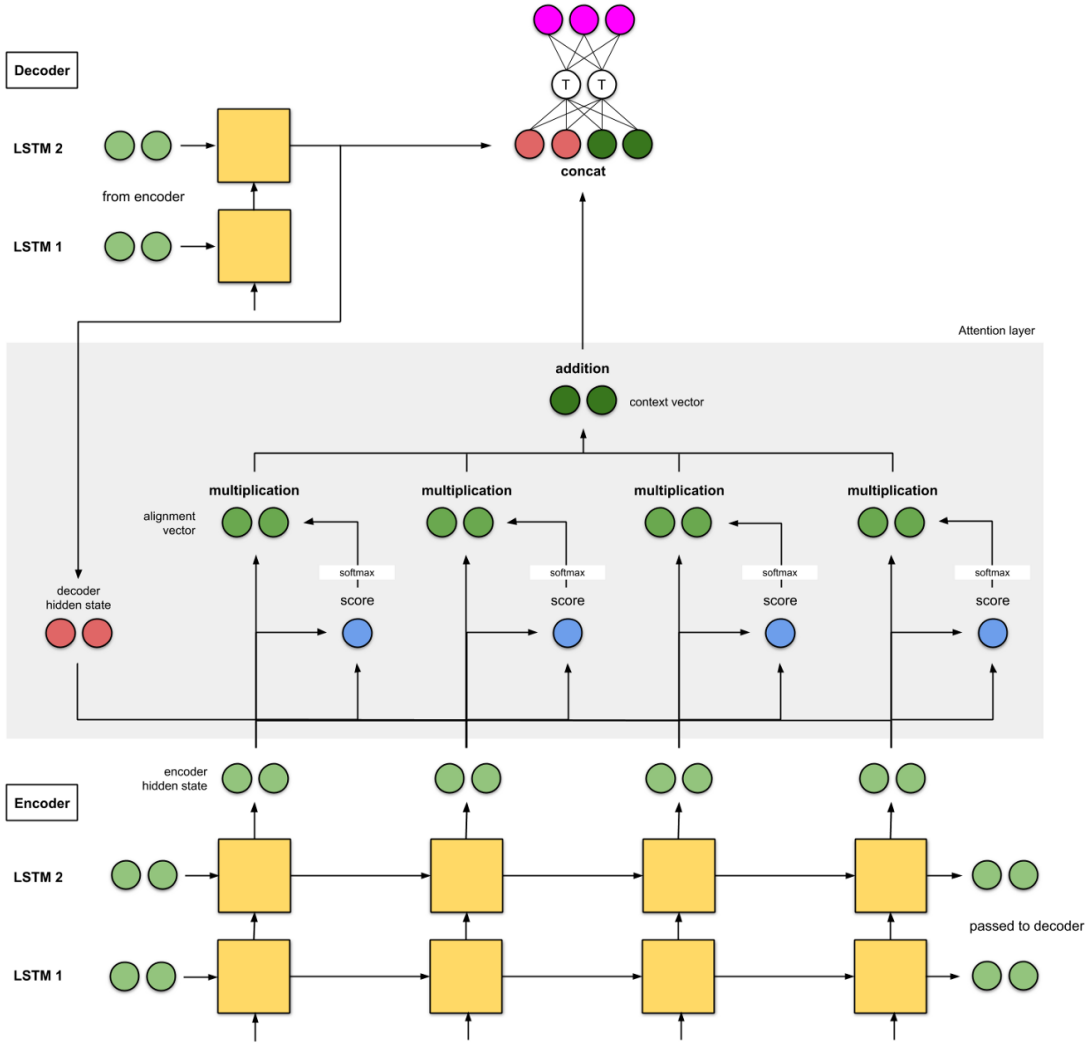
Attention



Bahdanau et. al (2015)
BiGRU / GRU

additive/concat.

Attention



Luong et. al (2015)

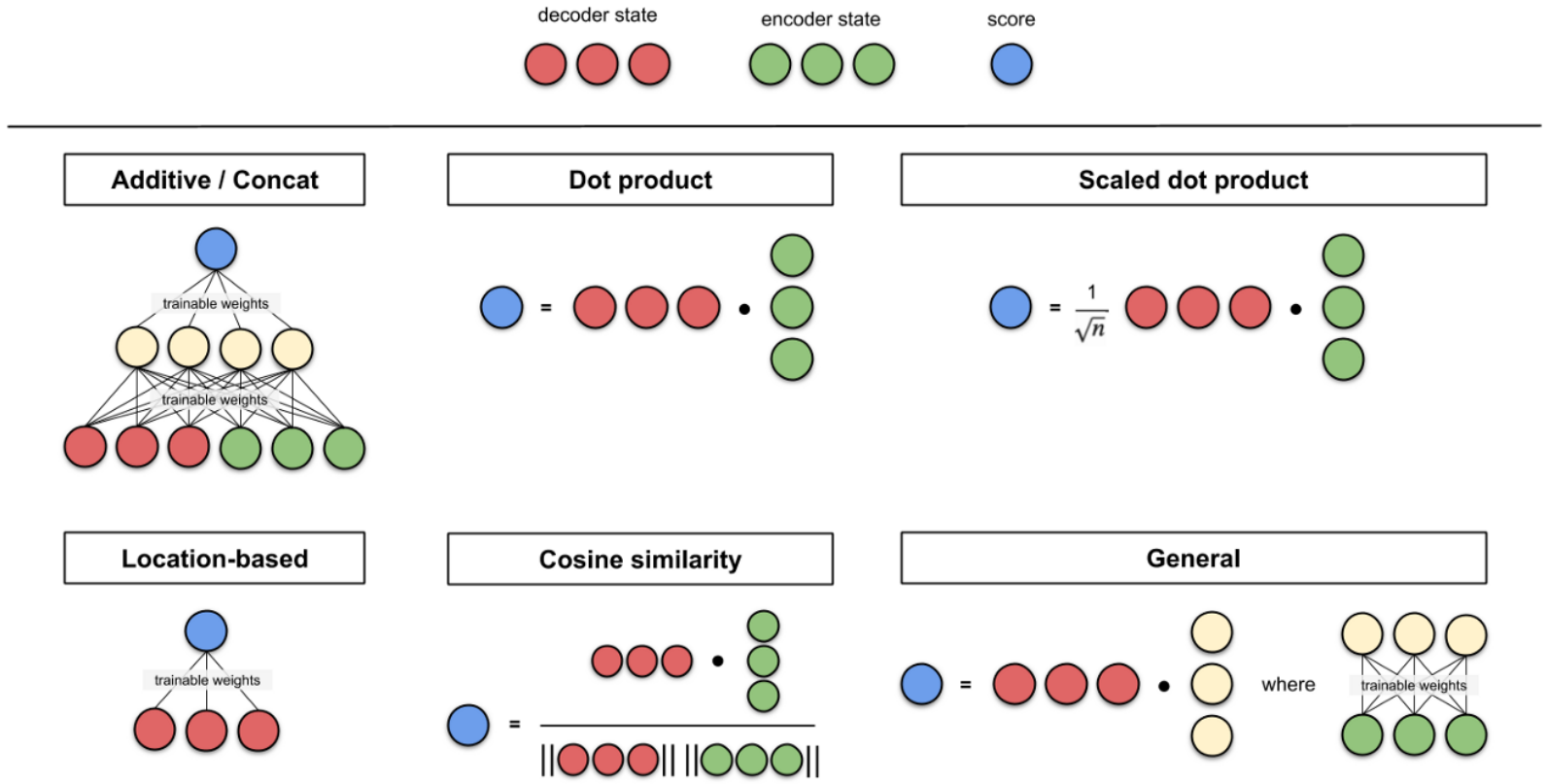
two-stacked long short-term memory (LSTM)

(i) **additive/concat**
(ii) **dot product**
(iii) **location-based**
(iv) **general**

Attention

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Attention



- *Thank you*