Recurrent Neural network (RNN)

2021. 8

Yongjin Jeong, KwangWoon University

Motivation

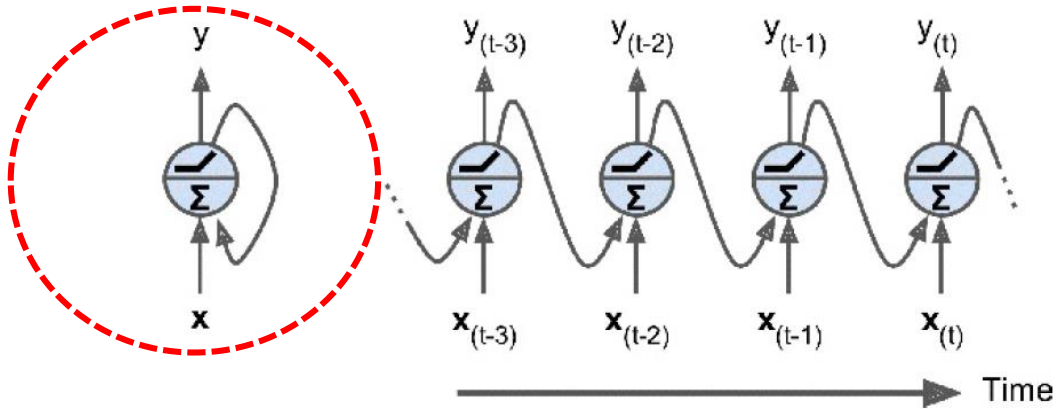
- **합성곱 신경망(CNN)**
  - 2차원(공간) 필터에서 좋은 성과 입증, 이미지 학습 분야에서 널리 사용
- **그러나 CNN은 시간적인 차이를 두고 연속적으로 발생하는 데이터 분석에는 성능이 부족**
  - 연속된 단어로 구성되는, 문장 분석, 자연어 처리, 자동 번역, 시계열 데이터 분석 등
- **이런 데이터에는 순환신경망(Recurrent Neural Network)이 잘 동작**

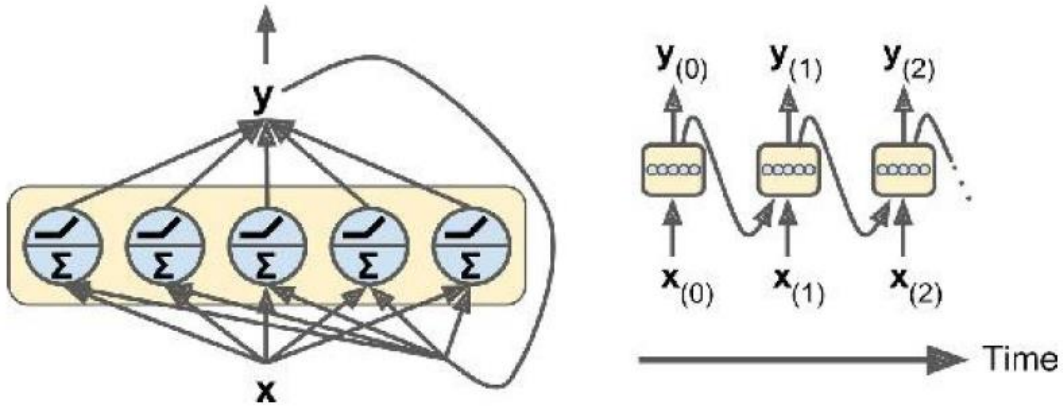
Motivation – Text processing

- 기존 텍스트 분석에서 BOW를 이용
  - 문장 내에 어떤 단어들이 존재하는지는 파악하지만
  - 단어의 배열 정보는 사라지고 이용하지 못함
- 단어의 순서 정보도 분석하려면
  - BoW (bag of words), 단어 벡터 (embedding)의 도입만으로는 부족
  - 단어의 발생 순서 정보를 활용할 수 있는 신경망 모델을 사용해야 한다
  - 이를 위해서 순환신경망(RNN)이 널리 사용
- 주요 응용
  - 음성인식
  - 텍스트 분석
  - 자연어 처리
  - 챗봇
  - 감성 분석
  - 언어 모델링(language modeling)

Recurrent Neurons

- looks very much like a feedforward neural network, except it also has backward connections.
- Can also create a layer of recurrent neurons (for mini-batch input $X_{(i)}$)



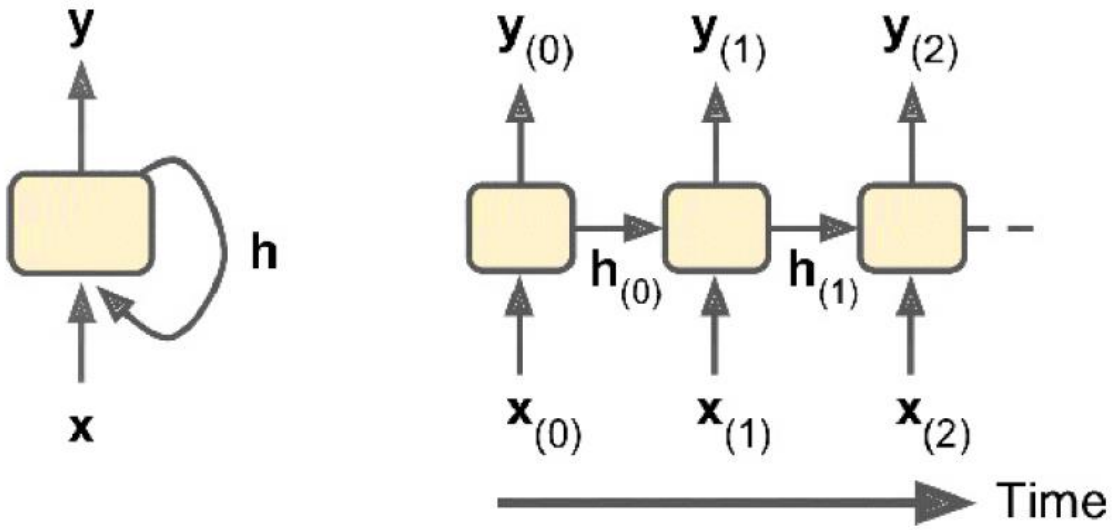


$$\mathbf{y}_{(t)} = \phi\left(\mathbf{x}_{(t)}^{T} \cdot \mathbf{w}_x + \mathbf{y}_{(t-1)}^{T} \cdot \mathbf{w}_y + b\right)$$

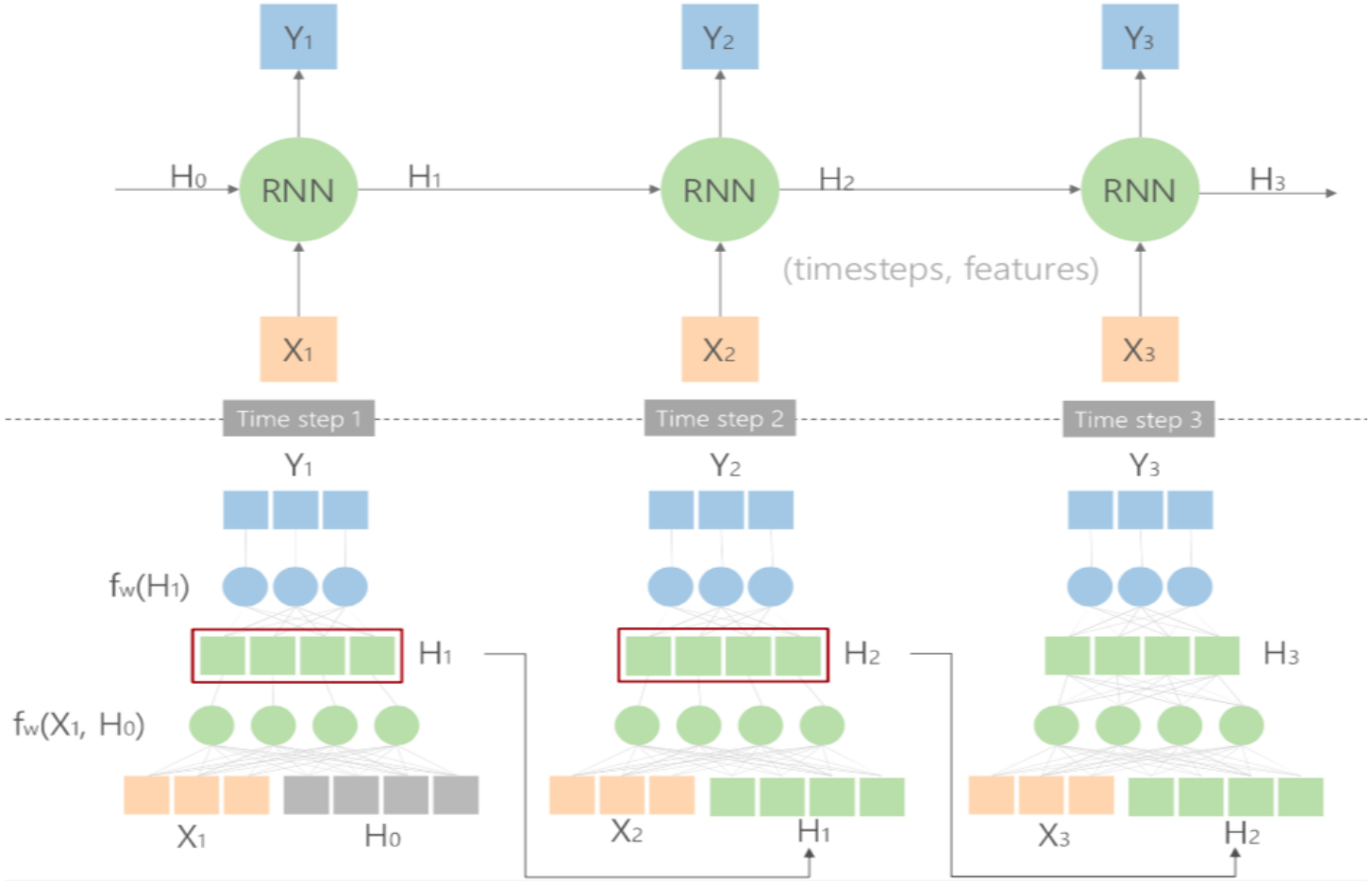
$$\begin{aligned}\mathbf{Y}_{(t)} &= \phi(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b}) \\ &= \phi([\mathbf{X}_{(t)} \quad \mathbf{Y}_{(t-1)}] \cdot \mathbf{W} + \mathbf{b}) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_y\end{bmatrix}\end{aligned}$$

Recurrent Neurons

- In more complex cells, a cell's hidden state, $h_{(t)}$, and its output, $y_{(t)}$, may be different.

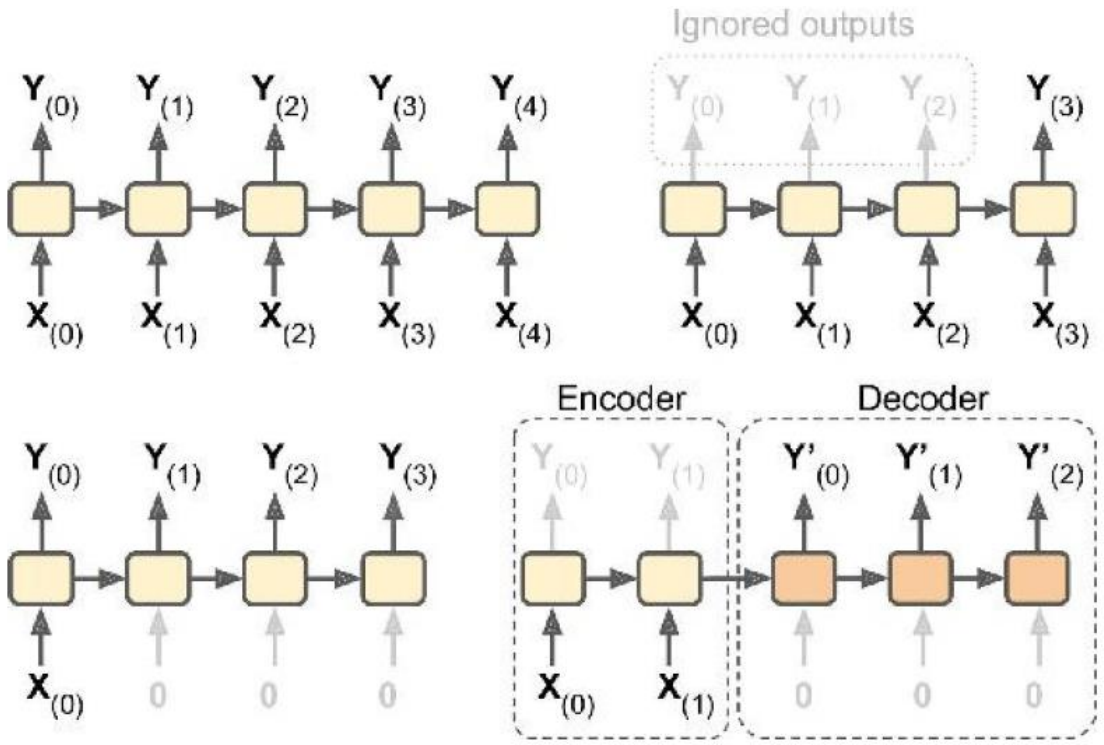


RNN Operation



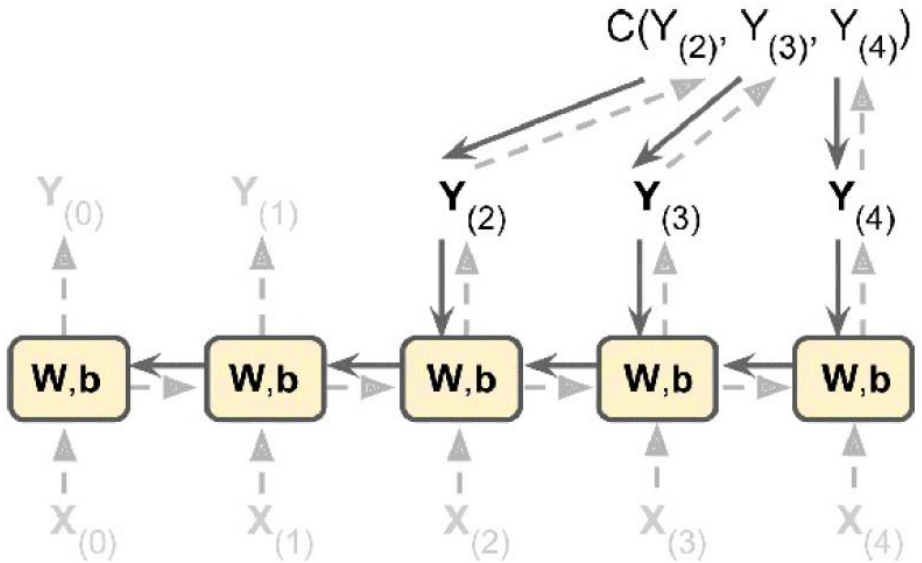
Input and Output Sequences of RNN

- **seq-to-seq**
  - (ex) Predicting time series such as stock prices
- **seq-to-vector**
  - (ex) predicting a sentiment score for a movie review
- **vector-to-seq**
  - (ex) captioning for an input image
- **delayed seq-to-seq (encoder-decoder)**
  - (ex) translating a sentence from one language to another

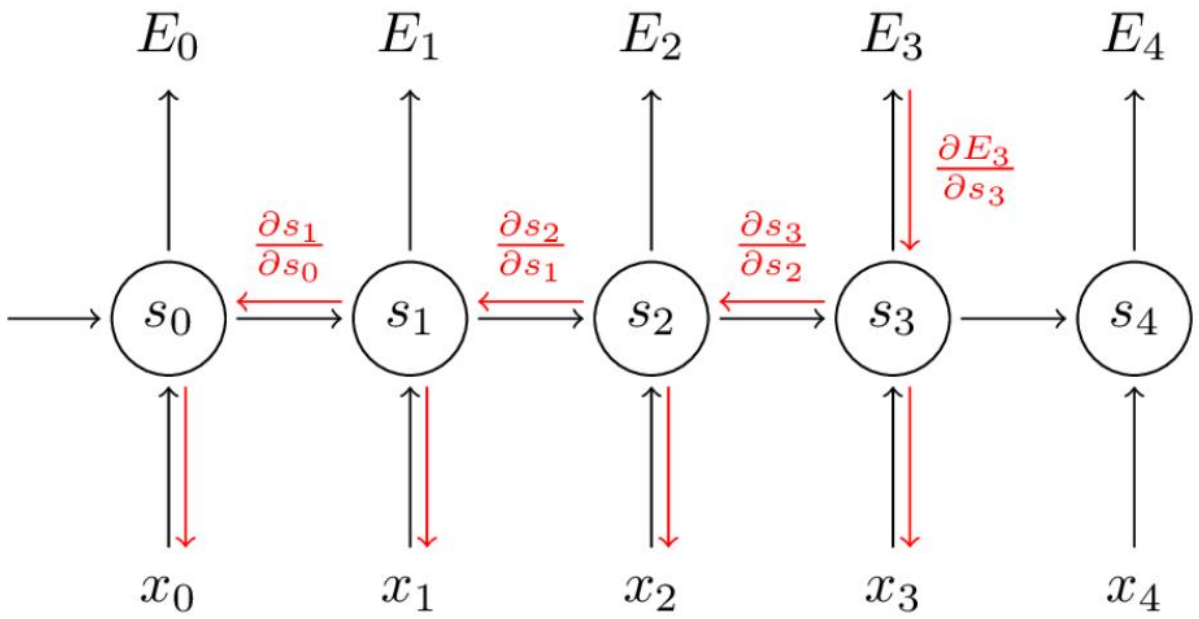


Training RNN

- Backpropagation through time (BPTT)
  - 시간축으로 Unroll 해서 Gradient Descent 방법 사용
  - 학습에 의해서 가중치 매트릭스 W, U, V가 업데이트 되는데 RNN에서는 모든 타임 스텝에서 동일한 가중치를 사용한다는 것을 주의해야 한다.
  - 즉, 현재의 경사(gradient) 변화는 과거 스텝의 계산에도 영향을 미친다.



RNN Backpropagation



- 시퀀스의 처음부터 끝까지를 모두 에러를 역전파하면 계산량이 많기 때문에 이를 줄이기 위해 현재 time step에서 일정시간 이전까지만 (보통 5 time step이전) 계산하는 Truncated-Backpropagation Through Time(생략된-BPTT)를 사용

Problems in RNN

- RNN에서 여러 계층을 거치는 경우, 예를 들어 단어가 수십 개로 된 문장을 해석하는 경우, 오차 역전파를 하면서 경사값이 거의 사라지거나 또는 너무 큰 값으로 발산하여 RNN이 제대로 동작하지 못하기 쉽다
  - 오래된 정보를 모두 중요시하면 정보가 너무 많이 축적되어 **발산**할 우려
  - 오래된 정보를 약하게 반영하면 오래되었지만 중요한 정보를 캐치하지 못하는 즉, **소실**되는 우려

- 이를 해결하기 위해서
  - **LSTM**(Long-Short Term Memory), **GRU** 기법이 제안
  - LSTM에서는 오랜 기간 중요도를 유지할 정보와 그럴 필요 없이 망각해야 할 신호를 구분하여 따로 처리

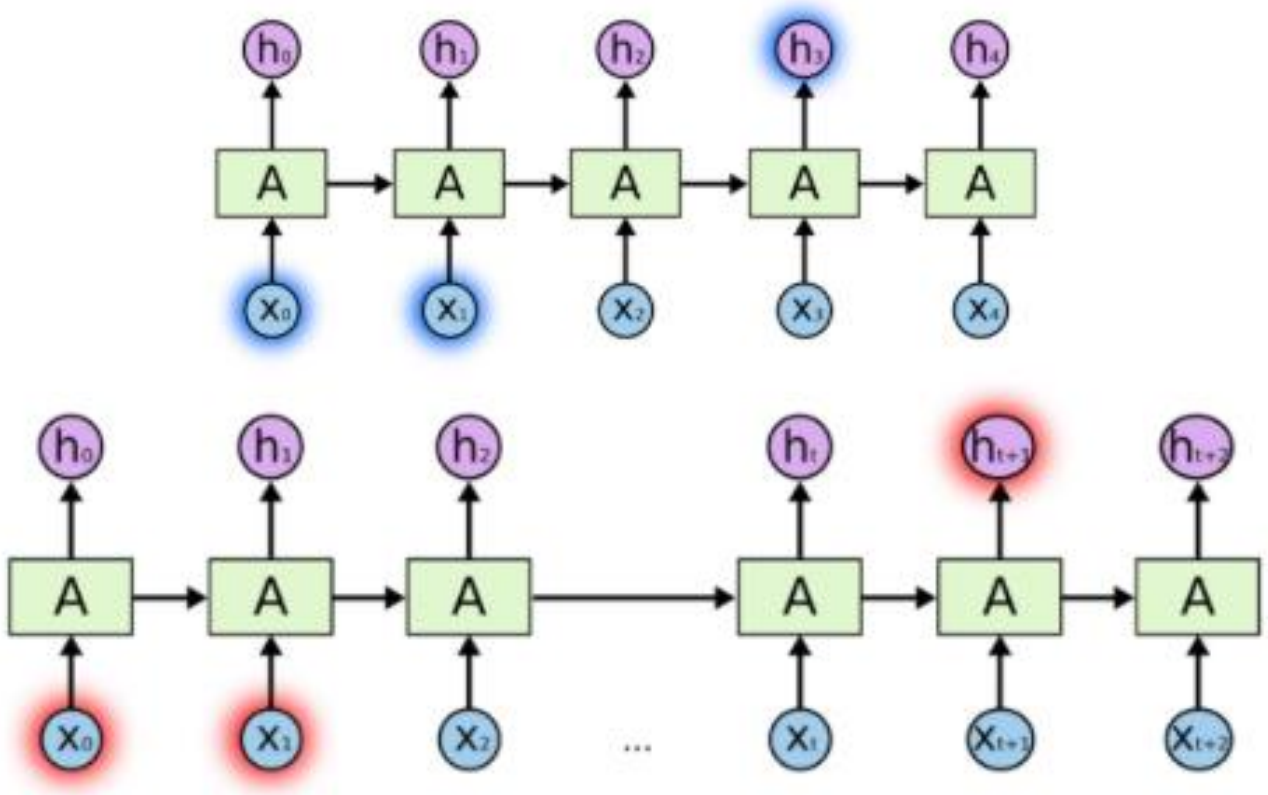
LSTM

- 단순 RNN 구조 (케라스의 SimpleRNN)
  - 실제로는 **경사 소실-발산** 등의 문제로 인해 잘 사용하지 않는다
  - 즉, 오래된 정보가 마지막 출력단에서는 매우 약해져서 학습에 사용하기 부족해진다. 따라서 **step 수가 늘면 학습이 잘 되지 않는다**.

- LSTM
  - RNN의 단점을 극복하기 위해서 제안
  - 여러 스텝 앞의 정보를 놓치지 않고 따로 뒷단으로 보내주는 **채널**을 하나 더 **추가** (오래된 정보가 스텝을 지나면서 사라지지 않고 뒤에 영향을 미치도록 하는 것이 목적)
  - 우리가 대화를 할 때에도 바로 최근의 단어들을 듣고 뜻을 파악하지만 오래 전에 한 말을 통해서 전체적인 맥락이나 목적 등을 꾸준히 파악하는 것과 같은 의미
  - 즉, 시퀀스로 입력되는 데이터의 단기(short) 정보와 함께, 오래된(long) 정보를 병행해서 사용하고 학습한다는 의미로 LSTM이라는 이름을 붙임

LSTM - Vanishing Gradient Problem

- **RNN은 관련 정보와 그 정보를 사용하는 지점 사이의 거리가 멀 경우**
  - 역전파 시, gradient 정보가 점차 줄어 들어 **학습능력**이 **현저히 저하**

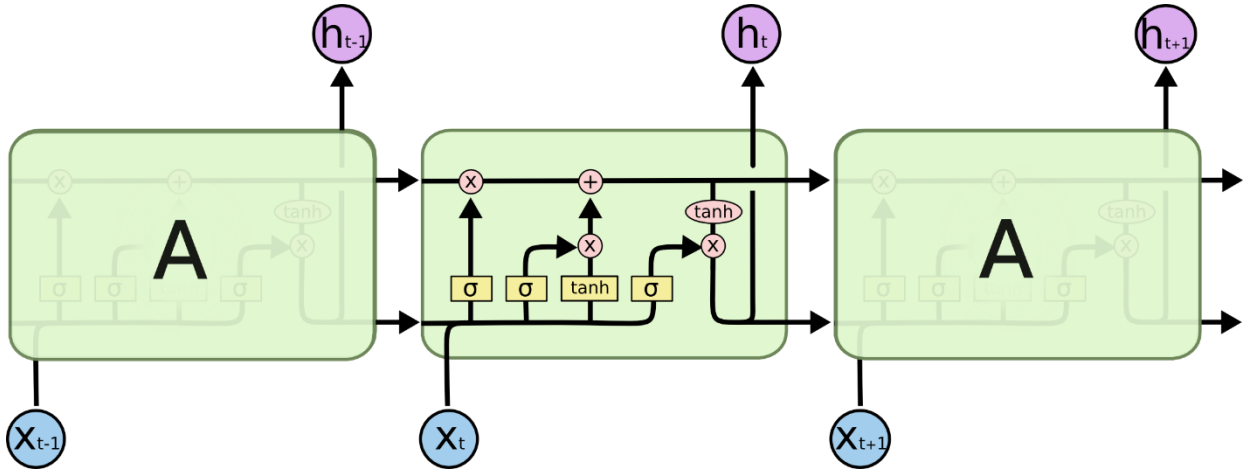


LSTM - Concept

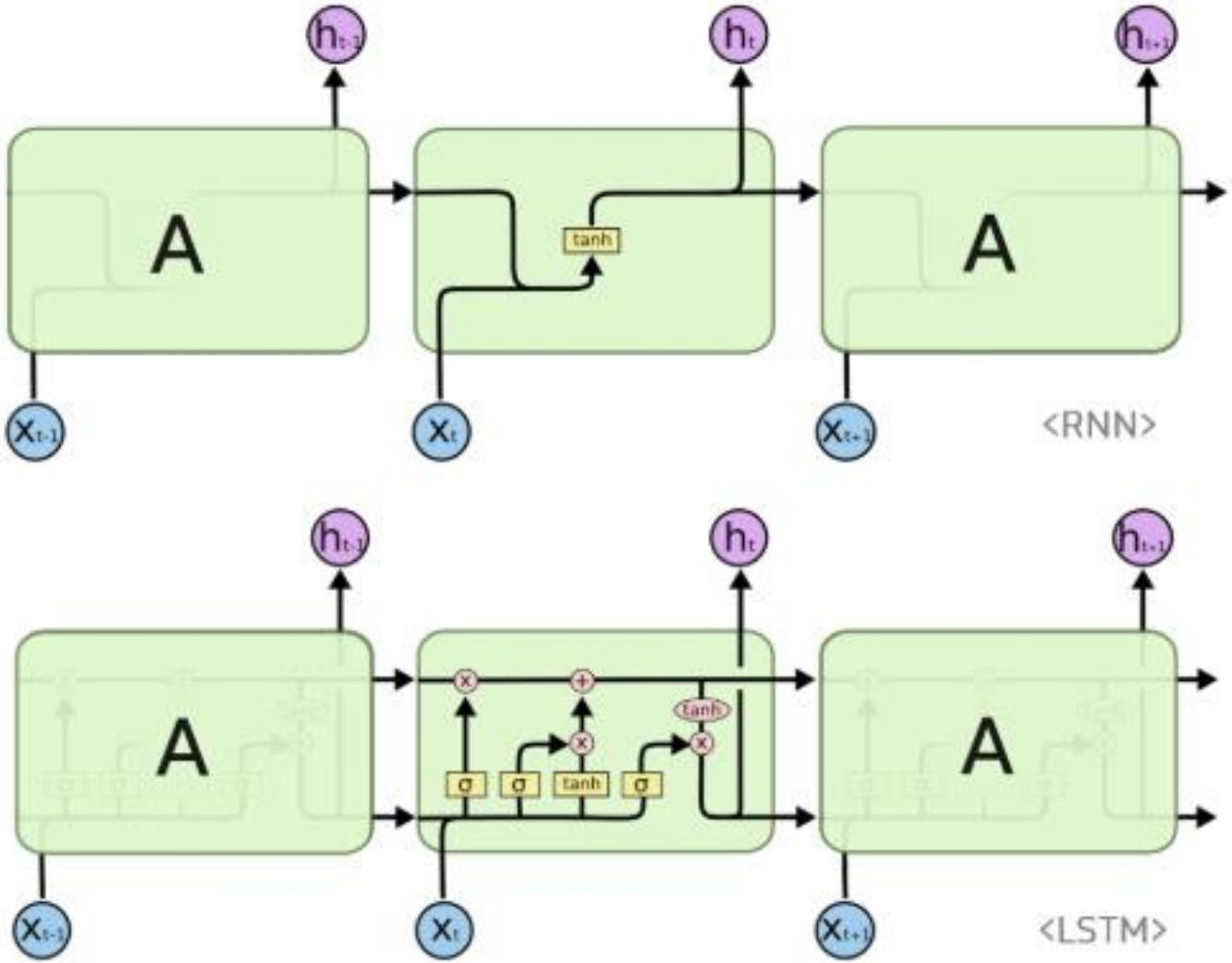
- 각 cell은 장, 단기 2개의 채널 사용
  - 단기적으로 학습한 정보가 전달되는 채널 (short-term state, h(t))
    (현재의 입력 정보와 상태 정보에서는 필요한 부분을 선택)
  - 장기적으로 살아남아 전달되는 채널 (long-term state, C(t))
    (과거의 전달 정보에서도 필요한 정보를 필터링하는 작업을 수행)
- LSTM에서 선택해야 할 하이퍼 파라미터
  - 임베딩 차원
  - 출력 차원
- 망각(forget), 입력(input), 출력(output) 게이트를 사용
  - 장기적인 상호작용을 학습시키는데 유용
  - 새롭고 관련성이 있는 정보를 선호하여 기억하도록 하고, 관련이 적은 정보를 잊도록 학습
- 문서 번역, 질의 응답(QA), 대화 서비스 (챗봇) 등에서 좋은 성능

LSTM cell structure

- 오래된 정보를 전달하는 Long-term state (Carry) 별도의 채널이 있다
- 각 셀에서는
  - 입력 신호(x), 이전 단계의 상태 정보(t), 그리고 이 전달 정보(c) 세 가지 정보의 가중치 합을 구하고
  - 활성화 함수를 통과하여 출력(t)을 만든다

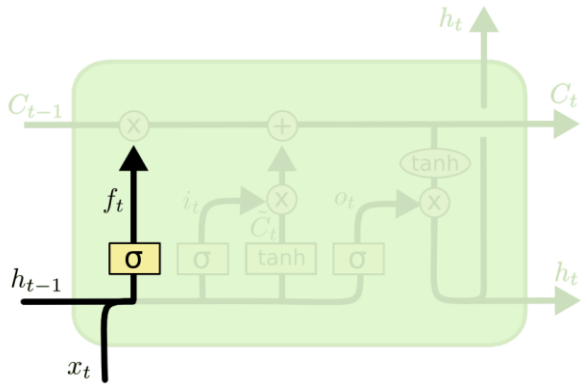


RNN and LSTM



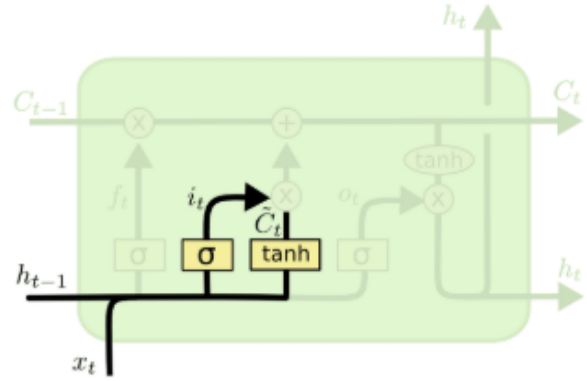
LSTM

- forget gate: cell state (C_{i-1})로부터 어떤 정보를 버릴 것인지를 결정 (sigmoid 사용)



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

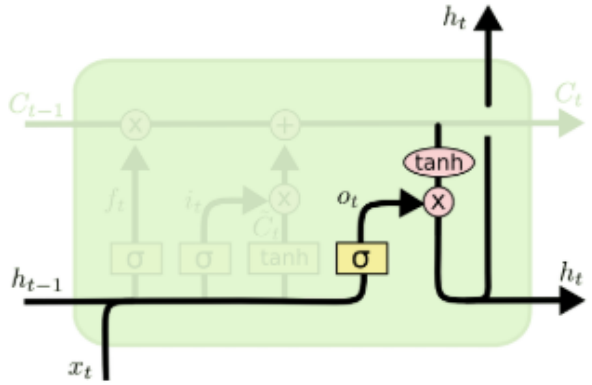
- input gate: 앞으로 들어오는 새로운 정보 중 어떤 것을 cell state에 저장할 것인지를 결정



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

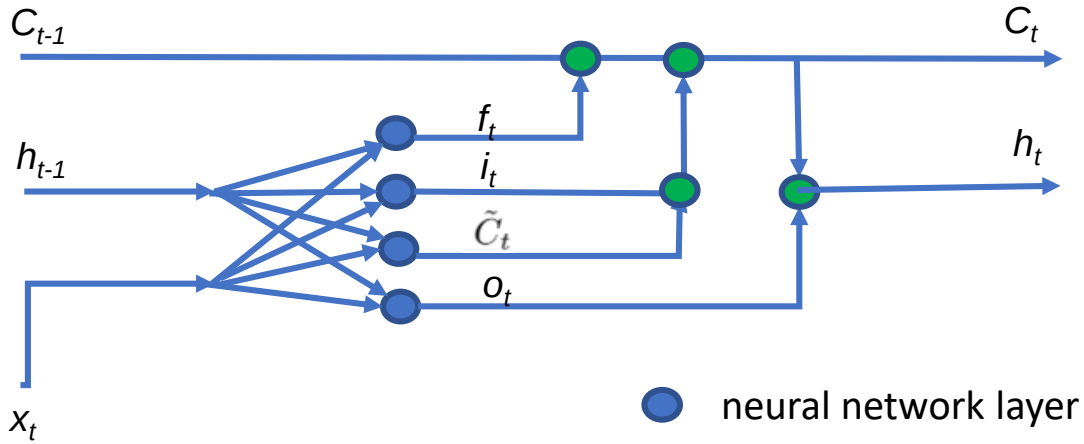
LSTM

- output gate: 무엇을 output으로 내 보낼 지를 결정



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- summary

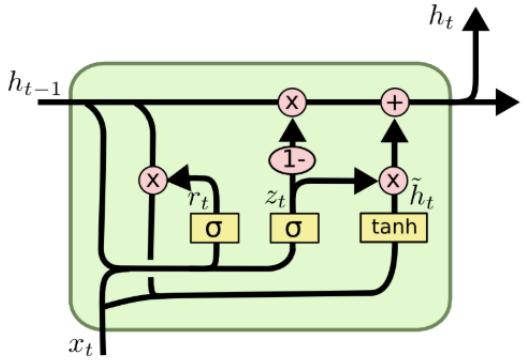


$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$

GRU

- Simplified version of LSTM (KyungHoon Cho, 2014)
  - 응용에 따라서 LSTM보다 성능이 우수하기도 하고 떨어지기도 함

- 2개의 게이트 사용
  - 리셋 게이트
  - 업데이트 게이트 : forget과 input 게이트를 합한 기능을 수행



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

임시 cell state $\tilde{c}_t = tanh(W_{xh_g} x_t + W_{hh_g}(r_t \otimes h_{t-1}) + b_{h_g})$

최종 cell state (=hidden state) $h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes \tilde{c}_t$

forget gate 역할 (과거정보)

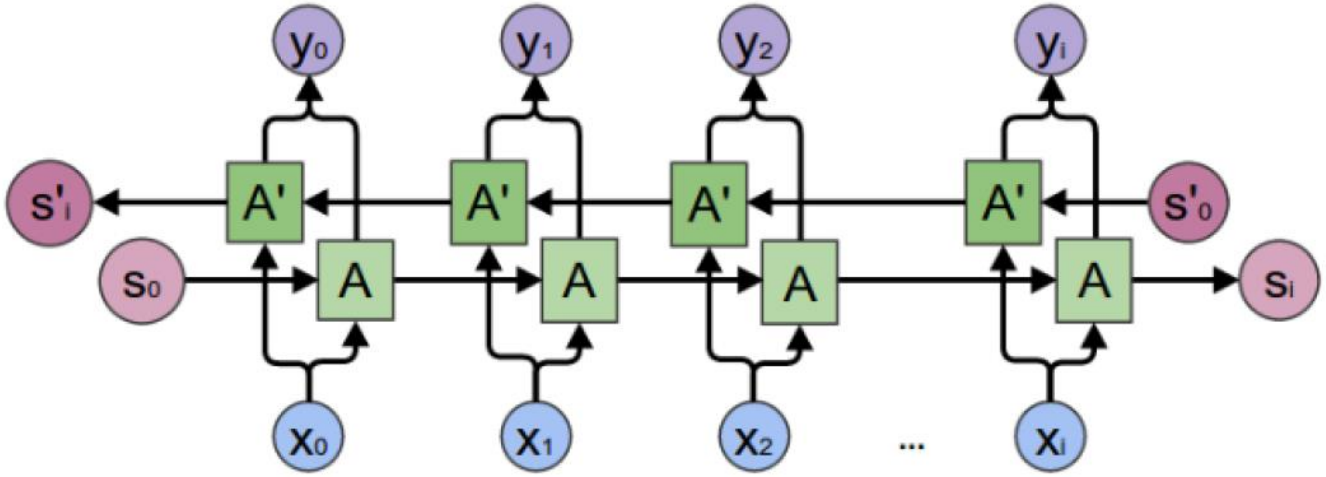
input gate 역할 (현재정보)

- It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state and makes some other changes, making it simpler than standard LSTM models.

Bidirectional RNN

- 시퀀스를 양쪽 방향으로 처리하여(즉, 두 번 프로세싱함), 한쪽 방향으로만 볼 때 놓치기 쉬운 패턴을 찾아보는 방법

- 즉, 현재에서 과거로의 추정과 함께 과거로부터 현재로의 추정을 동시에 진행하여 이 중에 좋은 패턴을 활용한다는 개념



Bidirectional RNN

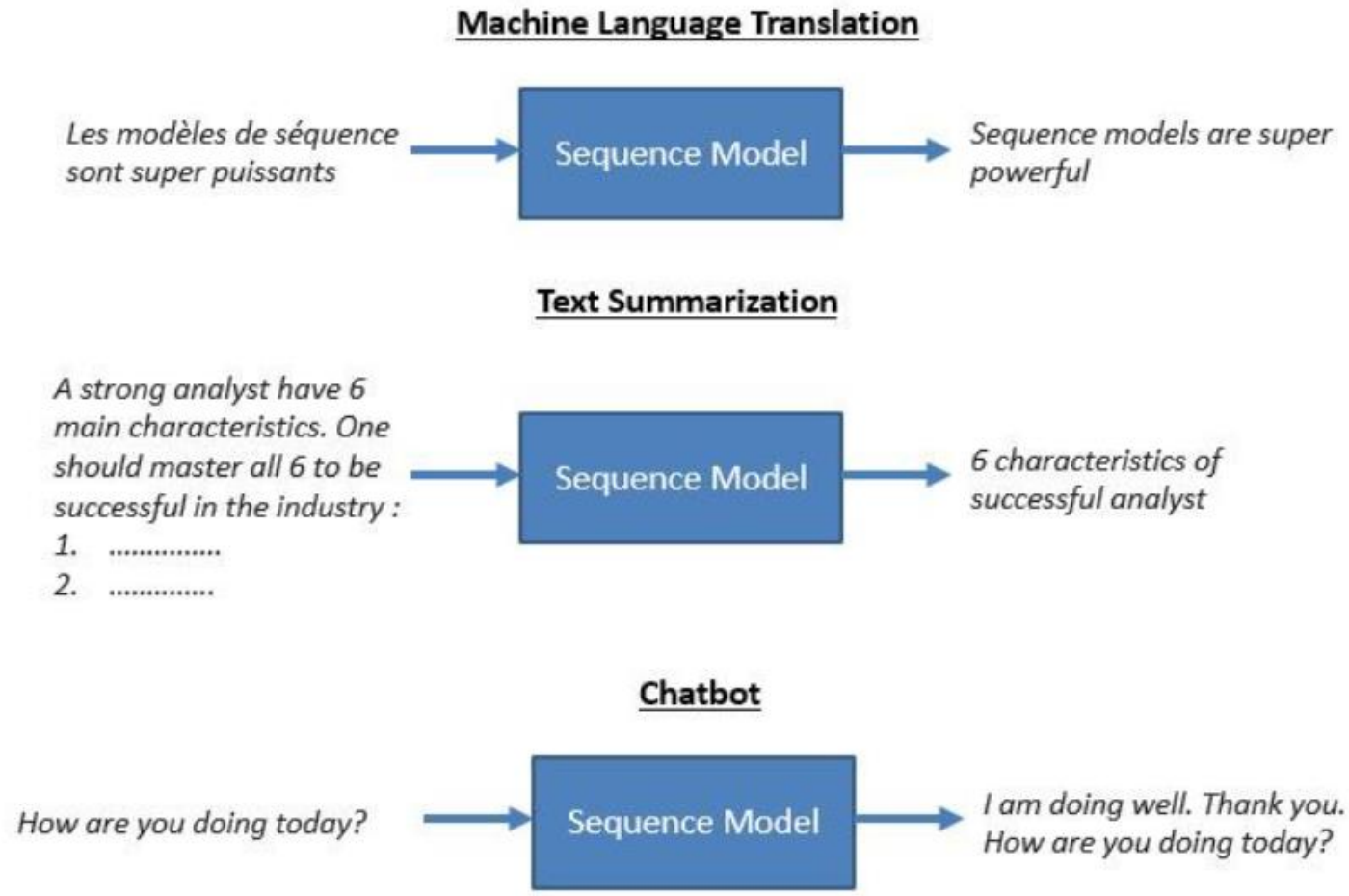
- 이러한 양방향 RNN이 항상 잘 성능을 개선하는 것은 아니나 좋은 성능을 나타내는 경우가 있다. 특히 자연어 처리에서 좋은 성능을 낸다.
- 문장과 같이 단어의 나열의 경우에 반대 방향으로 단어의 순서를 뒤집어서 예측하는 경우도 비슷한 성능을 보임
- 이는 언어에서 반대의 순으로 말을 하여도 컴퓨터를 거의 비슷하게 학습할 수 있다고 볼 수 있다
- 같은 정보를 다른 방법으로 표현하는 것을 활용하여 앙상블을 취하면 더 좋은 성능을 낼 수 있다는 가정에서 양방향 RNN을 도입
- Keras에서는 양방향 RNN을 구축하기 위한 Bidirectional 계층을 지원한다. 그러나 두 배 많은 파라미터를 사용하게 되어 과대적합이 될 가능성도 더 높아진다

Seq2Seq and Attention

- **Seq2seq**: Encoder-decoder architecture
  - the encoder: processes the input sequence and encodes/compresses/summarizes the information into a context vector (also called as the “thought vector”) of a fixed length.
  - This representation is expected to be a good summary of the entire input sequence.
  - The decoder: initialized with this context vector and starts generating the transformed output

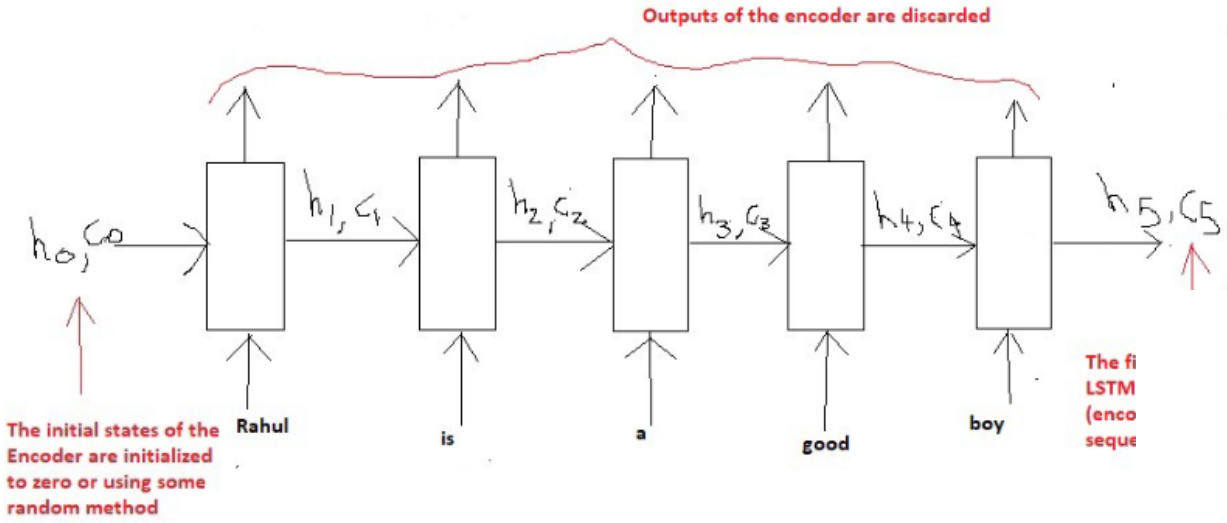
- A critical disadvantage of this fixed length context vector
  - Incapability to remember longer sequences
  - Often is has forgotten the earlier parts of the sequence once it has processed the entire the sequence.
  - The **attention** mechanism was born to resolve this problem.

Seq2Seq

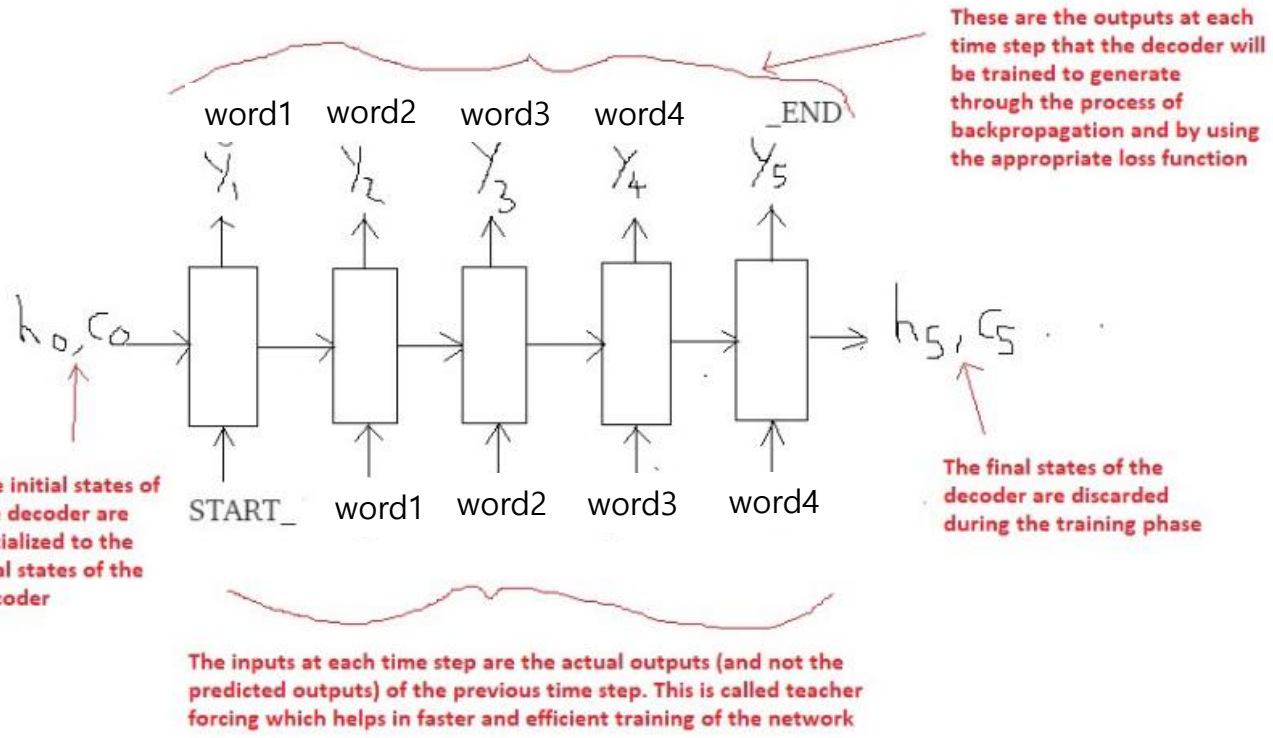


ource https://www.analyticsvidhya.com/blog/2018/04/sequence-modelling-an-introduction-with-practical-use-cases/

Seq2Seq

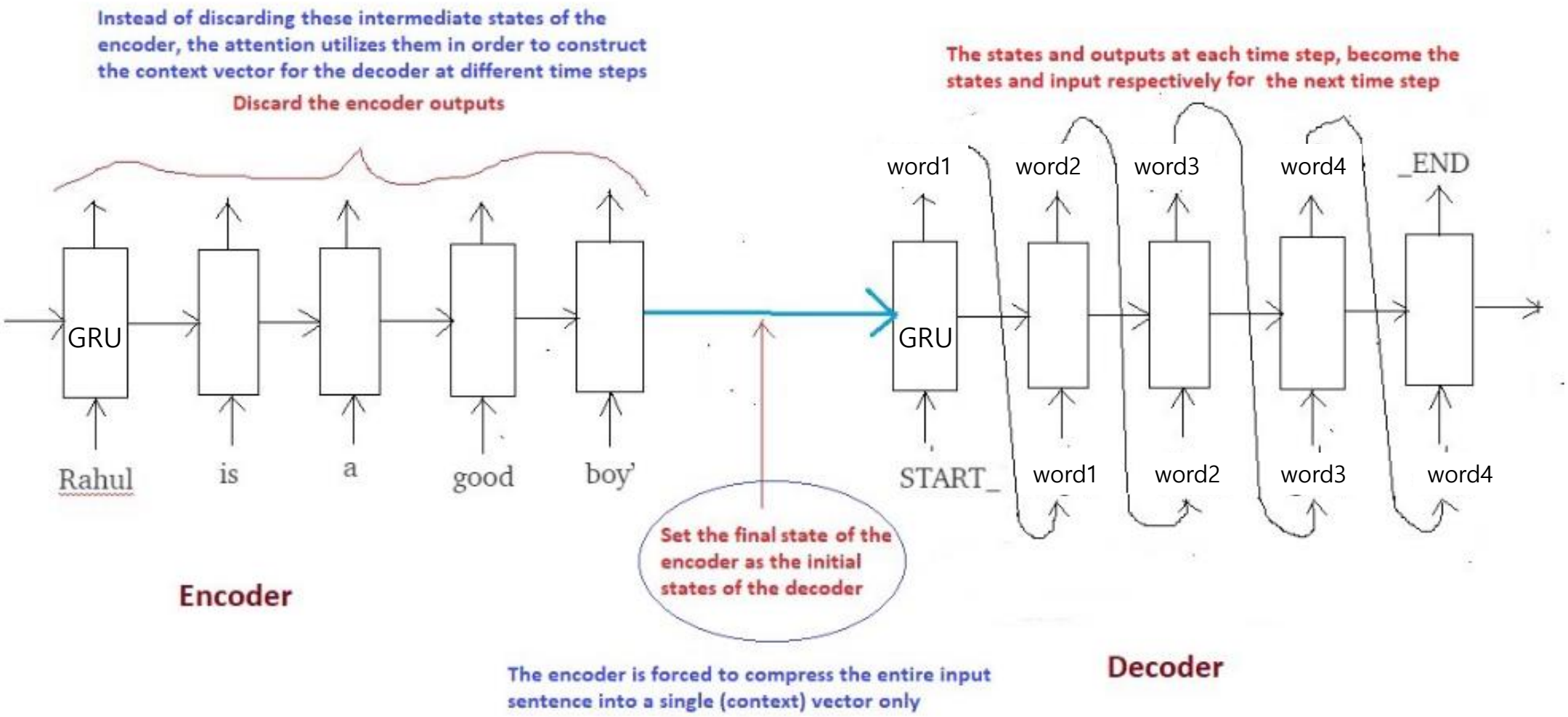


Encoder LSTM



Decoder LSTM — Training Mode

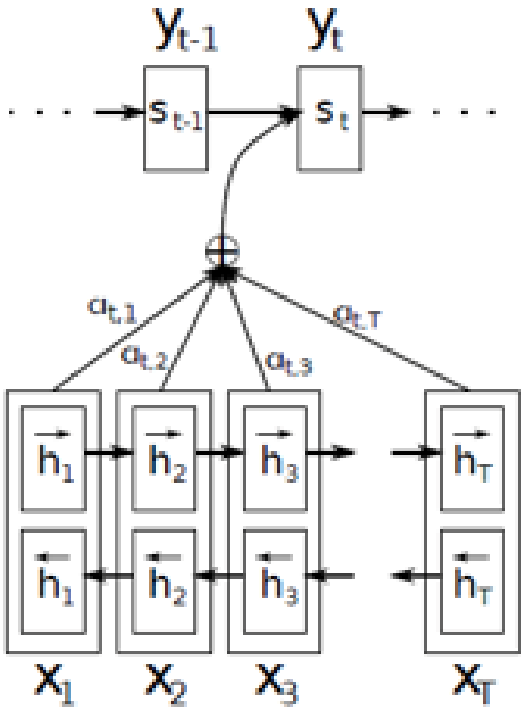
Seq2Seq



https://towardsdatascience.com/intuitive-understanding-of-attention-mechanism-in-deep-learning-6c9482aecf4f

Attention

- **Attention Mechanism (**from original paper – bidirectional**)**



$$h_j = \left[\overrightarrow{h}_j^\top ; \overleftarrow{h}_j^\top \right]^\top$$

- Sequence of annotations (h_1, h_2, …, h_T) for each input sequence (T: number of words in the input sequence)

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

- Context vector for the output word y_i: simply a weighted sum of hidden states

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

- Weight (also learned by a feed forward network)

$$e_{ij} = a(s_{i-1}, h_j)$$

- output score of a feedforward network (function a attempts to capture the alignment between input at j and output at i)