

---

# RNN

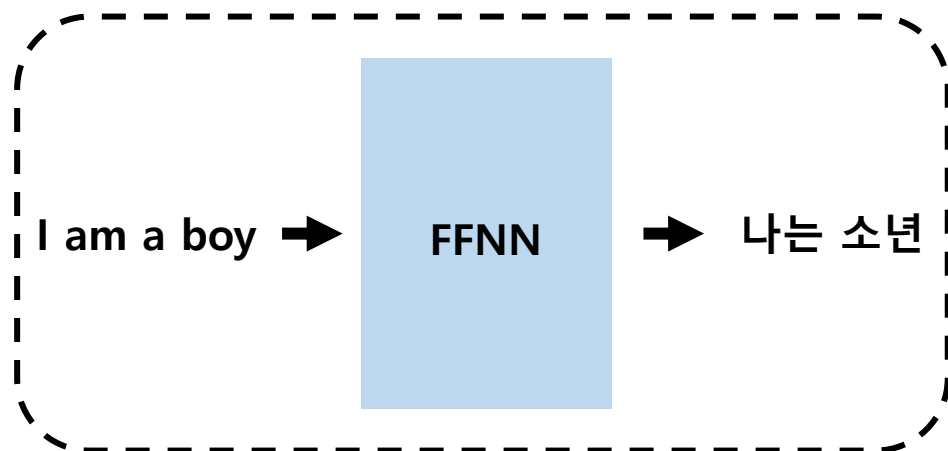
## Recurrent Neural Network

Effective model for sequence data processing

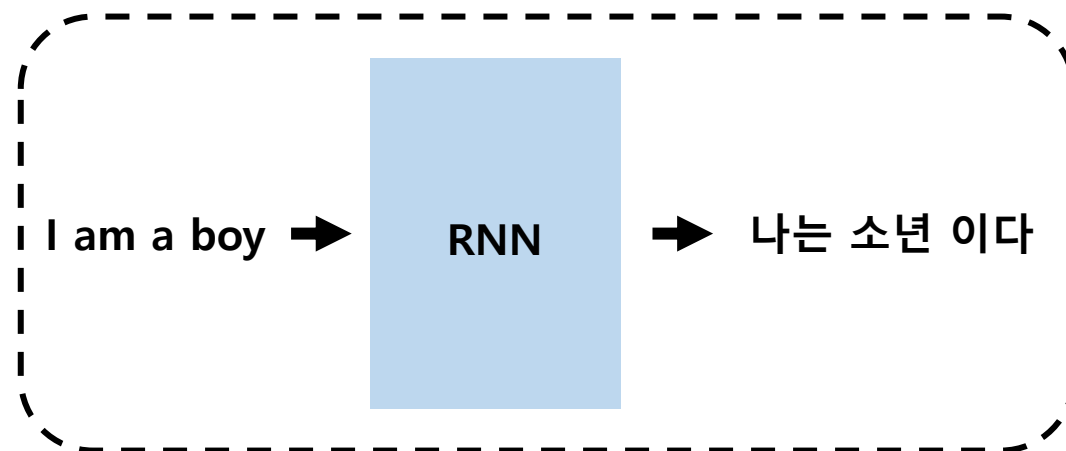
MINGEON PARK



- Sequence data 처리에 용의한 모델에 대한 필요성
  - 기존 NN, CNN은 준수한 성능을 보여주지만, Inputs가 서로 독립적임을 가정하고 있어 Sequence data를 처리하는데 적합하지 않음
  - NN은 고정된 크기의 입력만 처리할 수 있어 문장의 길이가 다양한 Sequence data 처리에 어려움이 있음

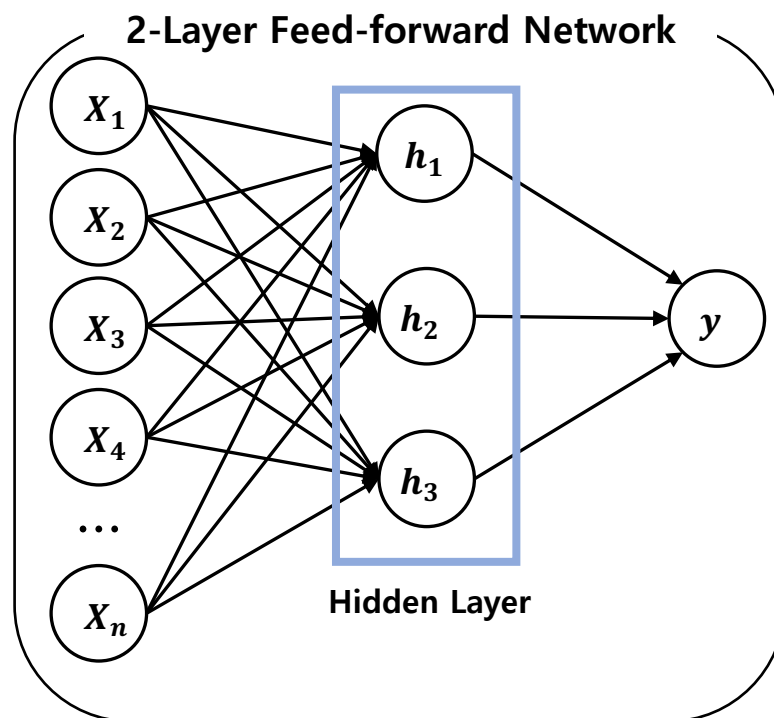


<FFNN 기계번역 단순화한 기계 번역 프로세스>



<RNN 기계번역 단순화한 기계 번역 프로세스>

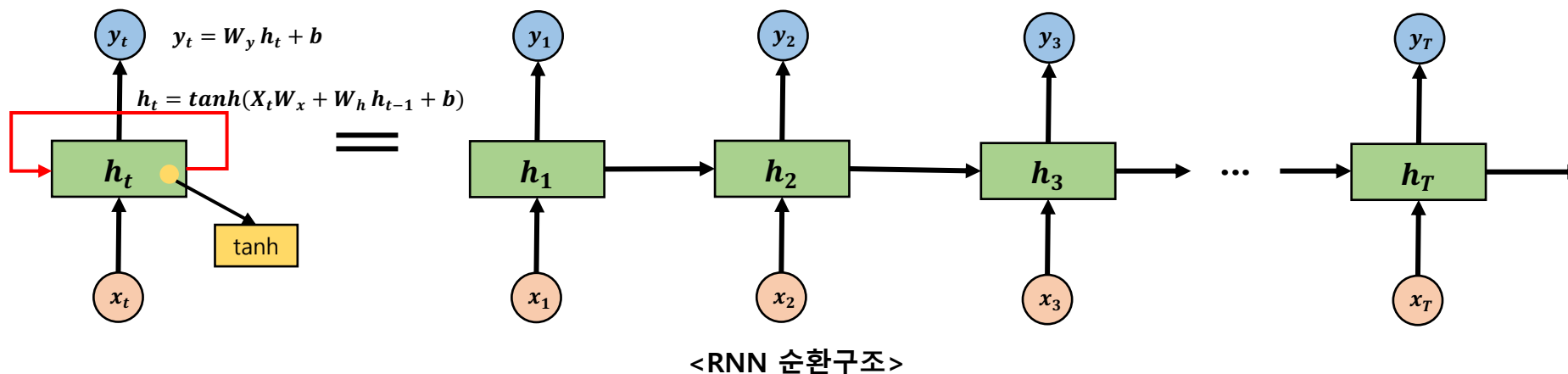
- 기존의 NN, (Feedforward) Network는 신경망의 기본 모델
- 순방향 신경망으로 기계번역을 실행하면  $x \rightarrow y$ 를 향한 방향으로 번역이 이뤄지고 모든 입력데이터는 동일한 가중치를 공유
- 각 입력이 독립적으로 처리되며, 은닉층에서 Activation Function을 지난 값은 출력층으로 향하는 Feed-Forward 특징을 지녀서 시간적인 관계를 학습할 방법이 없음
- 일반적인 Fully Connected Neural Network는 고정된 크기의 입력을 받아 고정된 크기의 출력을 하는 구조
- 복잡하고 다양한 길이의 데이터를 다루는 시계열 데이터 분석이나 자연어 처리에는 적합하지 않음



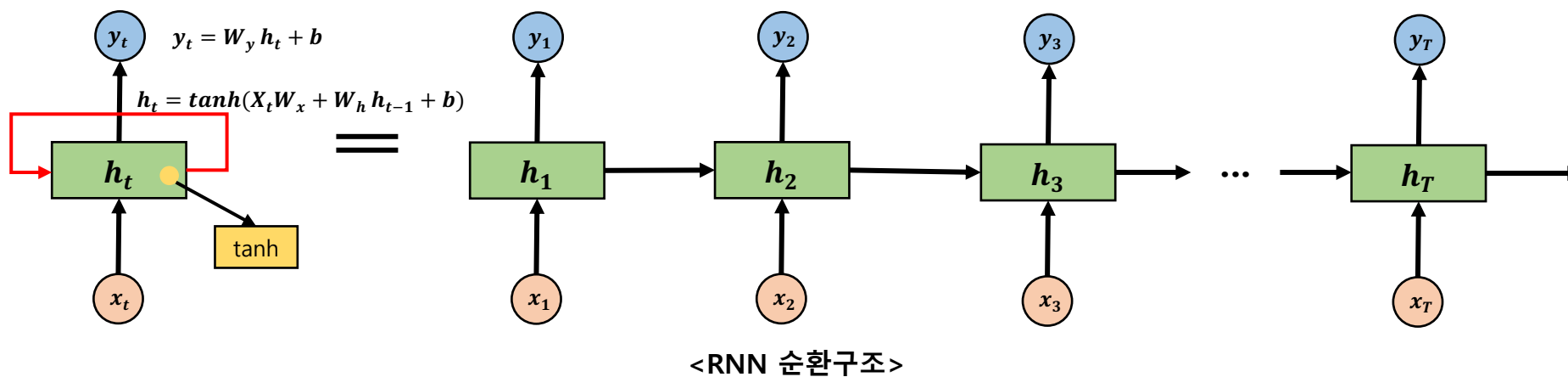
<2-Layer 순방향 신경망>

- **Recurrent Neural Network**

- 과거의 출력 데이터를 재귀적으로 참조하여 Sequence Data를 처리하는 신경망
- 출력  $h_t$ 를 hidden state라고 하며 스텝에 따라 갱신하는 state로 활용하여 이전 step의 결과를 반영
- 결국 RNN은 기존의 Neural Network의 가중치가 동일하고 독립적인 곳에서 발생하는 시간에 대한 참조 불가능이라는 문제를 **hidden state**라는 **이전 step에서 갱신되는 state를 이용하여 해결하려 함**
- memory sell : 은닉층 -> 활성화 함수로 결과를 내보내는 node

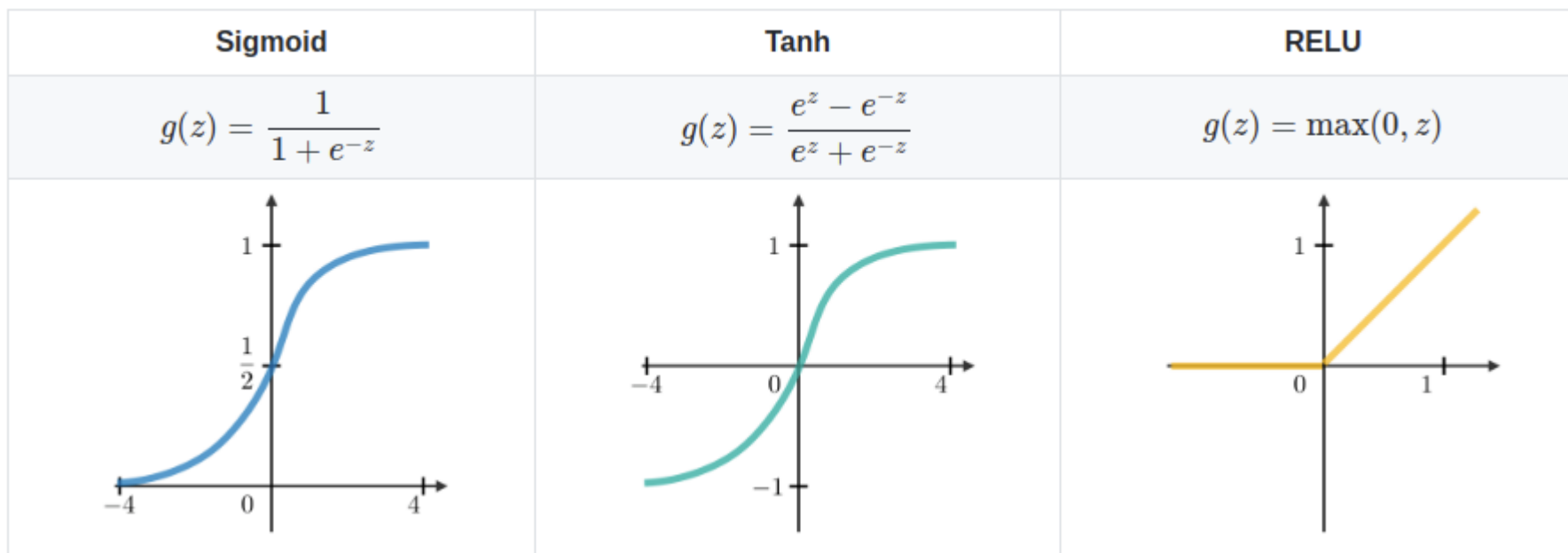


- $h_t = \tanh(X_t W_x + W_h h_{t-1} + b)$
- $h_1 = \tanh(X_1 \textcolor{red}{W}_x + \textcolor{blue}{b})$
- $h_2 = \tanh(X_2 \textcolor{red}{W}_x + h_1 \textcolor{green}{W}_h + \textcolor{blue}{b})$
- $h_3 = \tanh(X_3 \textcolor{red}{W}_x + h_2 \textcolor{green}{W}_h + \textcolor{blue}{b})$
- $\hat{y} = h_3 W_y + b_y$
- 원하는 시점에서  $\hat{y}$  출력 가능
- 같은 색깔은 같은 값
- $\hat{y}$ 의 Activation Function은 풀고자 하는 문제에 따라 다름



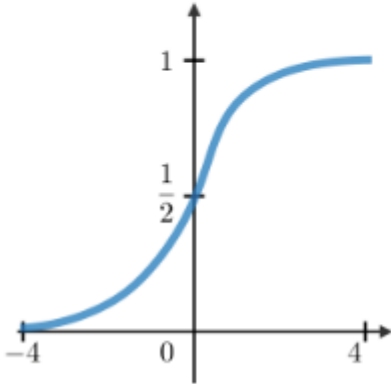
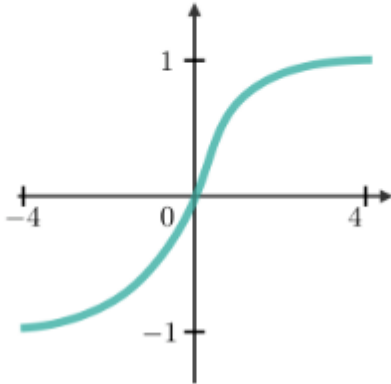
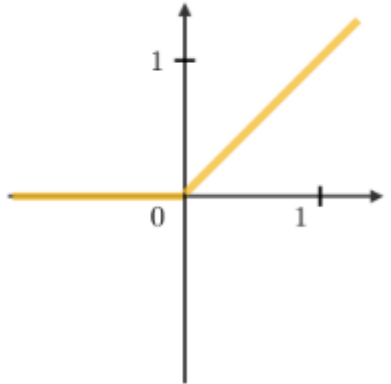
- **tanh vs sigmoid**

- Tanh는 기울기의 최대값이 0 ~ 1
- sigmoid는 기울기의 최대값이 1/4
- Sigmoid보다 tanh가 Gradient Vanishing problem에 강함
- Sigmoid는 0 ~ 1사이에서 값을 출력하며, 기울기는 x=0에서 최대값을 가지고, x가 양의 무한대 또는 음의 무한대로 갈수록 기울기가 0에 가까워 신경망이 깊어질 수록 작은 기울기가 곱해져 gradient 소실의 원인이 됨
- Tanh는 -1에서 1사이에서 값을 출력하며, 중심점은 0으로 sigmoid보다 출력범위가 넓고 0을 중심으로 대칭되기에 초기 학습 과정에서 가중치 조절이 효율적이고 상대적으로 gradient 소실에 강함



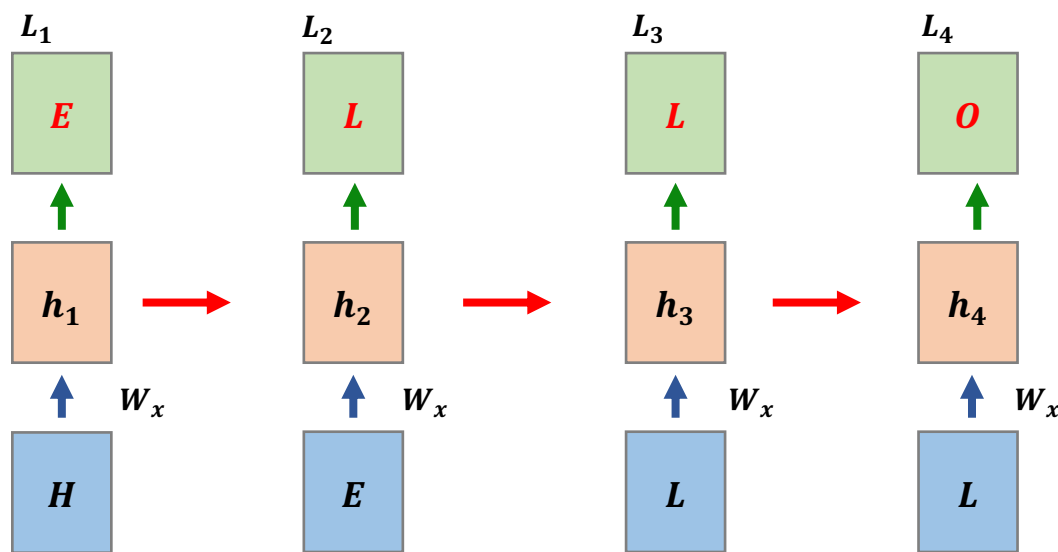
<Activation Function 비교>

- ReLU를 사용하지 않는 이유
  - RNN은 이전 step의 정보를 hidden state로 가져와 사용하므로 ReLU를 이용하게되면 이전의 값이 커져서 전체적으로 출력이 발산하는 문제가 발생

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

<Activation Function 비교>

- 다음 단어를 예측하는 문제
  - H -> E -> L -> L -> O를 예측
  - 다음 알파벳(one-hot encoded)을 예측하는 다중 분류 문제
  - $\hat{y}$ 을 softmax 통과시켜 cross-entropy 계산 (softmax는 RNN의 출력을 확률화)
  - ELLO 네 가지 글자에 대한 cross-entropy를 모두 합산
  - $\frac{\partial L_t}{\partial W_x}$ 는 각 가중치가 최종 손실에 미치는 영향의 정도 (e.g.  $L_1$ 은 E 단어의 가중치가 손실에 미치는 영향)
  - 그렇다면 RNN 단어 예측에서 마지막 문자인 O가 나오기 위해 H가 gradient에 미치는 영향력은 ?  
 $\Rightarrow \frac{\partial L_4}{\partial W_x}$ 에 해당 (값의 결과는 weigh들의 합에 해당)



&lt;RNN Hello 예측&gt;

$$L - Total$$

$$= L_1 + L_2 + L_3 + L_4$$

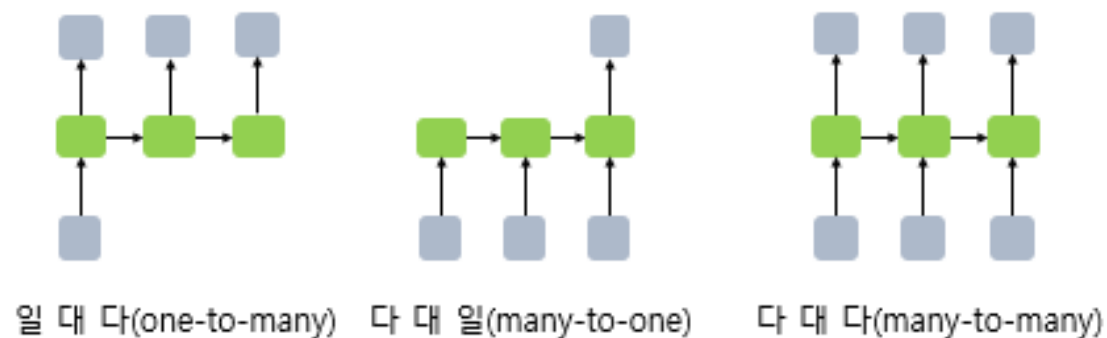
$$W_x \leftarrow W_x - \alpha \frac{\partial L_{total}}{\partial W_x}$$

$$\downarrow$$

$$\frac{\partial L_1}{\partial W_x} + \frac{\partial L_2}{\partial W_x} + \frac{\partial L_3}{\partial W_x} + \frac{\partial L_4}{\partial W_x}$$



- **One to many**
  - 하나의 입력에 여러 출력을 가지는 모델
  - 주로 이미지 캡셔닝 분야에 이용
- **Many to one**
  - 여러 입력에 하나의 출력을 가지는 모델
  - 스팸 분류, 감성 분류 등의 분야에 이용
- **Many to many**
  - 여러 입력에 여러 결과를 반환하는 모델
  - 챗봇이나 번역기 등의 기초



<RNN 종류>

- 장기 의존성 문제(Long-Term Dependencies)
  - Gradient Descent 미치는 영향력이 계산 시점에서 멀어질 수록 감소하여 잊혀진다.
  - (back propagation 을 구할 때 처음의 값이 결과에 영향이 적어지며 사라지는 현상)
- Vanishing Gradient
  - RNN back propagation에서 tanh에 의해 기울기가 압축되면서 매우 작아져 소실되고, 이로 인해 가중치 업데이트가 제대로 이루어지지 않아 학습이 멈추는 현상
- 위 구조적 문제들을 해결하기 위해 LSTM, GRU가 등장

# Thank you

Thank you for listening to my  
presentation

rschrmin7985@gmail.com  
MINGEON PARK

<https://saugatbhattarai.com.np/what-is-activation-functions-in-neural-network-nn/logistic-sigmoid-unipolar-tanh-bipolar/>

<https://velog.io/@koreanlperson/RNN-%EC%9D%B4%EC%95%BC%EA%B8%B0>

<https://twitter.com/JulianoDag>