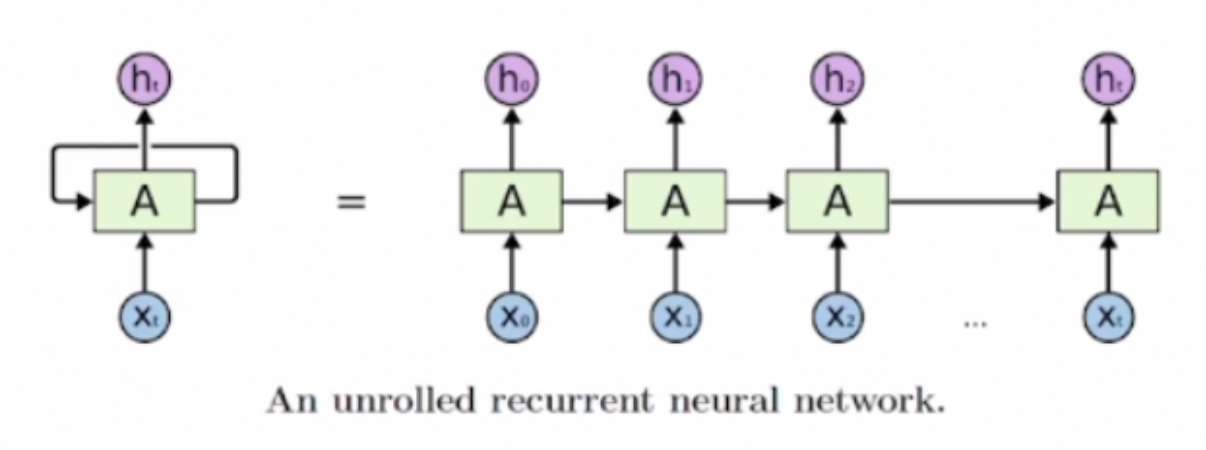
2. 자연어 처리와 딥러닝

1. Recurrent Neural Network (RNN)

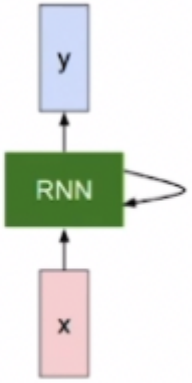


An unrolled recurrent neural network.

- RNN: 현재 타임스텝에 대해 이전 스텝까지의 정보를 기반으로 예측값을 산출하는 구조의 딥러닝 모델.
  - 매 타임스텝마다 동일한 파라미터를 가진 모듈을 사용
  - '재귀적인 호출'의 특성

- 계산

$$h_t = f_W(h_{t-1}, x_t)$$

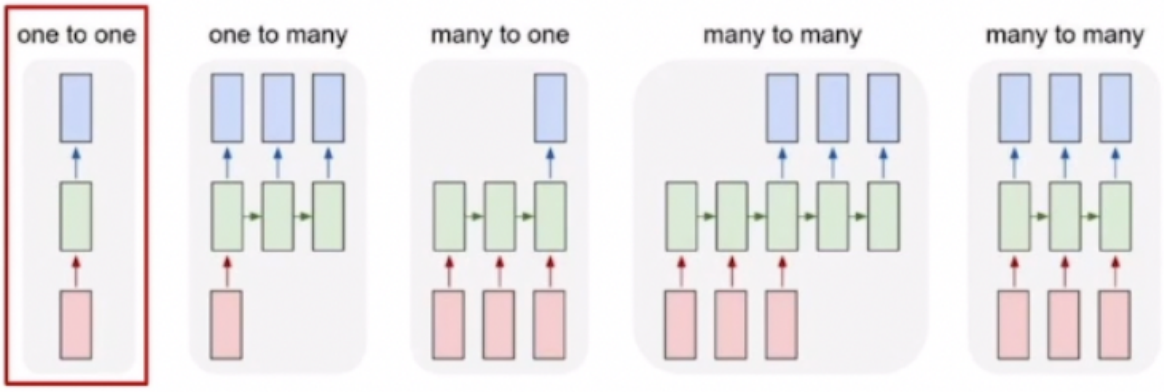


- 인자
- 계산 순서

- w : weight
- t : 현재 time step
- h_t-1*ht*−1 : old hidden-state vector
- x_t*xt* : input vector at some time step
- h_t*ht* : new hidden-state vector
- f_w*fw* : RNN function with parameters W
- y_t*yt* : output vector at time step t

- h_t = f_w(h_t-1,x_t)*ht*=*fw*(*ht*−1,*xt*) 의 함수를 통해 매 타임스텝마다 hidden state를 다시 구하기
- W와 입력값(x_t, h_t-1*xt*,*ht*−1)으로 tan_h*tanh* 를 곱해서 h_t*ht* 를 구하기
- 구해진 h_t, x_t*ht*,*xt* 를 입력으로 y_t*yt* 값을 산출

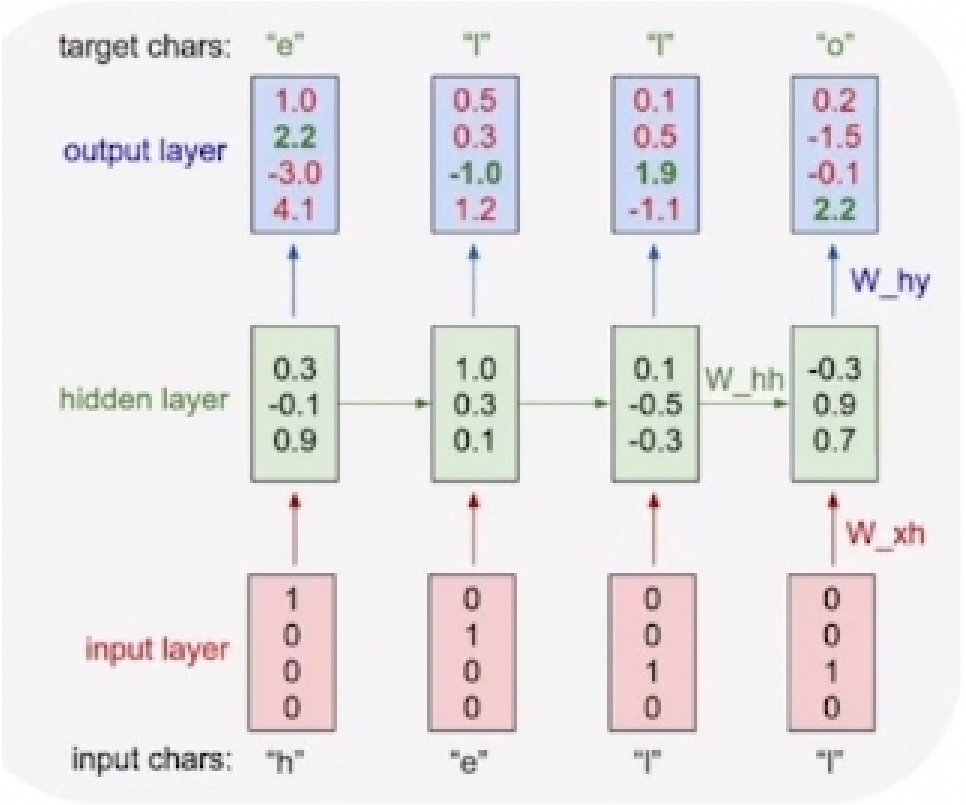
- RNN 모델 종류



1	one to one	[키, 몸무게, 나이]와 같은 정보를 입력값으로 할 때, 이를 통해 저혈압/고혈압인지 분류하는 형태의 태스크
2	one to many	'이미지 캡셔닝'과 같이 하나의 이미지를 입력값으로 주면 설명글을 생성하는 태스크
3	many to one	감성 분석과 같이 문장을 넣으면 긍/부정 중 하나의 레이블로 분류하는 태스크
4	many to many	기계 번역과 같이 입력값을 끝까지 다 읽은 후, 번역된 문장을 출력해주는 태스크
5	many to many	비디오 분류와 같이 영상의 프레임 레벨에서 예측하는 태스크 혹은 각 단어의 품사에 대해 태깅하는 POS와 같은 태스크

- 들어오는 입력값에 대해서, 많은 유연성을 가지고 학습되는 딥러닝 모델
- 그레디언트 소실/증폭 문제가 있어 실제로 많이 사용되지는 않지만, RNN 계열의 LSTM, GRU 모델은 현재도 많이 사용됨.

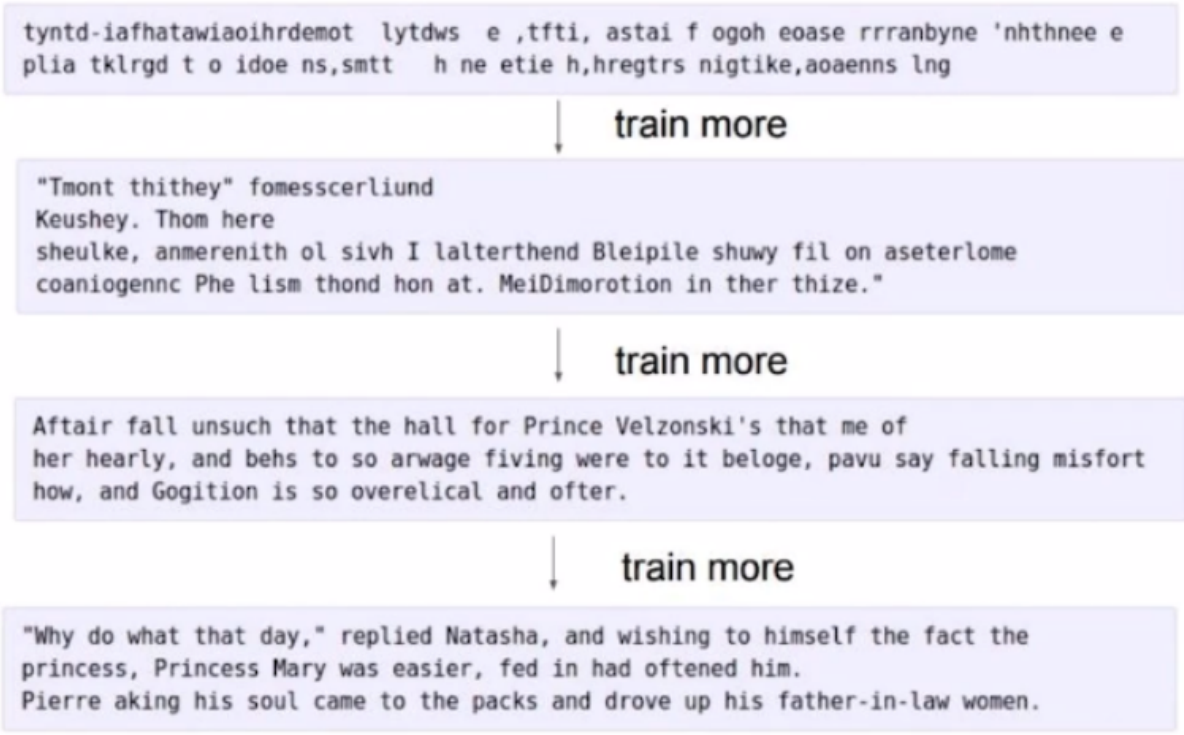
2. Character-level Language Model



- 언어 모델: 이전에 등장한 문자열을 기반으로 다음 단어를 예측하는 태스크
  - 캐릭터 레벨 언어 모델(character-level Language Model): 문자 단위로 다음에 올 문자를 예측하는 언어 모델.
  - ex) 맨 처음에 "h"가 주어지면 "e"를 예측하고, "e"가 주어지면 "l"을 예측하고, "l"이 주어지면 다음 "l"을 예측하도록 hidden state가 학습되어야 함.
  - 각 타임스텝별로 output layer를 통해 차원이 4(유니크한 문자의 개수) 벡터를 출력

    → logit이라고 부르며, softmax layer를 통과시키면 원-핫 벡터 형태의 출력값이 나옴.

- 언어 모델 예시



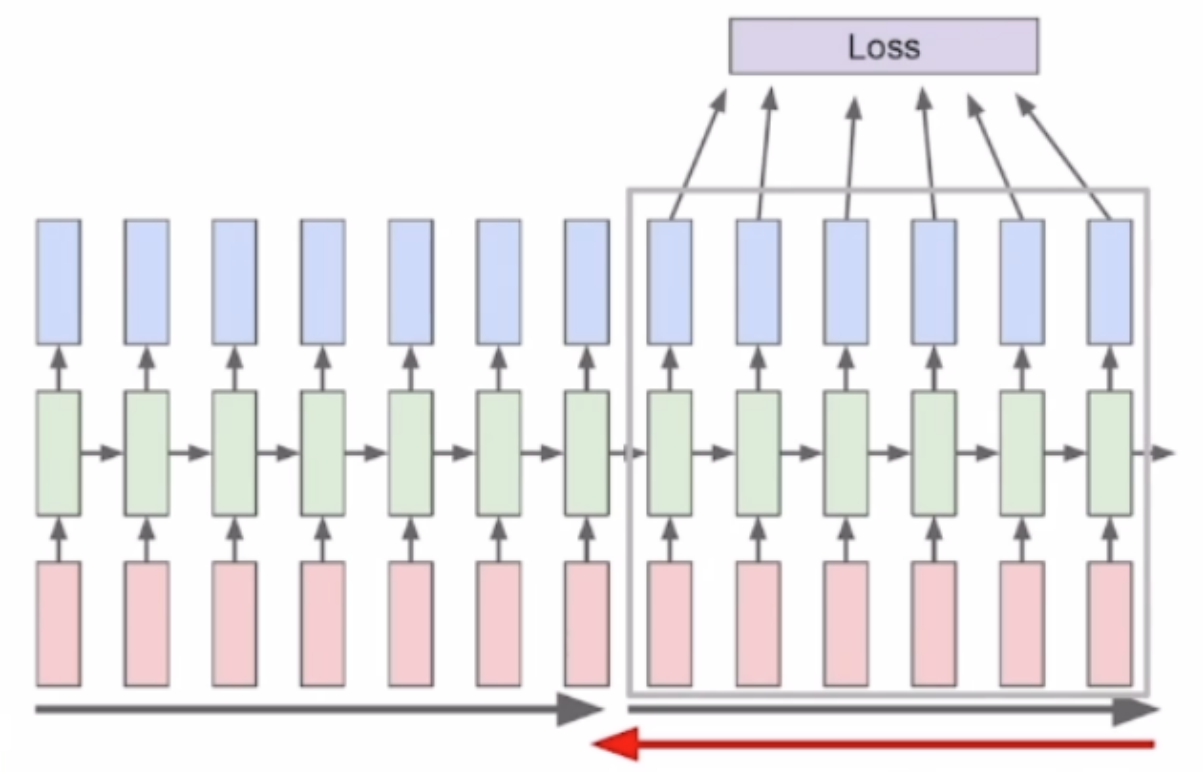
- 셰익스피어의 글을 활용해 더 많은 학습이 진행될수록 완전한 형태의 문장을 출력
- 전 타임 스텝까지의 주식값을 활용하여 다음날 주식값을 예측하는 형태로도 수행 가능
- 인물 별 대사, Latex로 쓰여진 논문, C 프로그래밍 언어와 같은 경우에도 다양한 언어적 특성을 학습하여 텍스트 생성 가능

3. Backpropagation through time and Long-Term-Dependency

<모델 학습 방법>

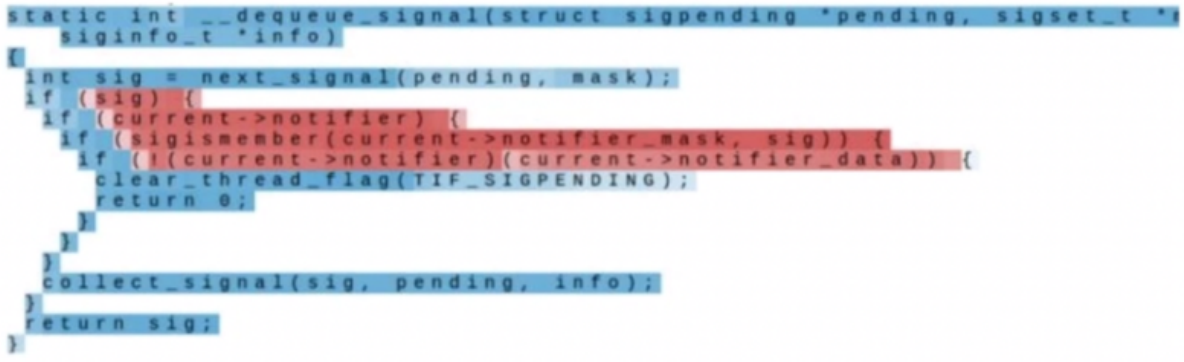
1. **Truncation**

- 제한된 리소스(메모리) 내에서 모든 시퀀스를 학습할 수 없기때문에 사진과 같이 잘라서 학습에 사용하는 것



- 딥러닝 모델은 **forward propagation을 통해 계산된 W**를, **backward propagation을 지나면서 W를 미분한 값인 gradient**를 통해 학습

2. **BPTT**: Backpropagation through time의 줄임말
    - RNN에서 타임스텝마다 계산된 weight를 backward propagation을 통해 학습하는 방식



→ 특정 hidden state를 시각화한 그림으로, BPTT를 반복하게 되면 빨강(긍정)과 파랑(부정)으로 해당 time step에서의 중요한 부분을 잘 학습하는 것

- Vanilla RNN으로는 위와 같이 학습 X

→ gradient가 전파되면서 소실되거나 증폭되면서 멀리까지 학습정보를 잘 전달하지 못하는 Long-Term-Dependency가 발생하기 때문

- **Vanishing/Exploding Gradient Problem in RNN**

Toy Example

- $h_t = \tanh(w_{xh}x_t + w_{hh}h_{t-1} + b), t = 1,2,3$
- For $w_{hh} = 3, w_{xh} = 2, b = 1$

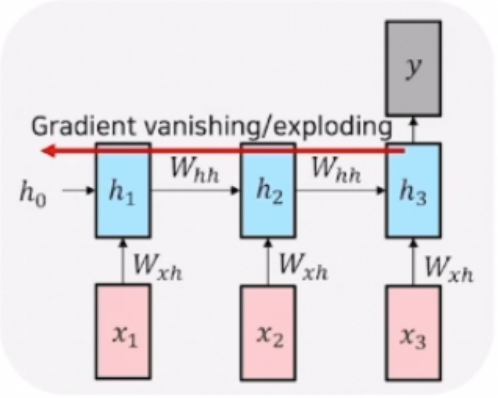
$h_3 = \tanh(2x_3 + 3h_2 + 1)$

$h_2 = \tanh(2x_2 + 3h_1 + 1))$

$h_1 = \tanh(2x_1 + 3h_0 + 1))$

...

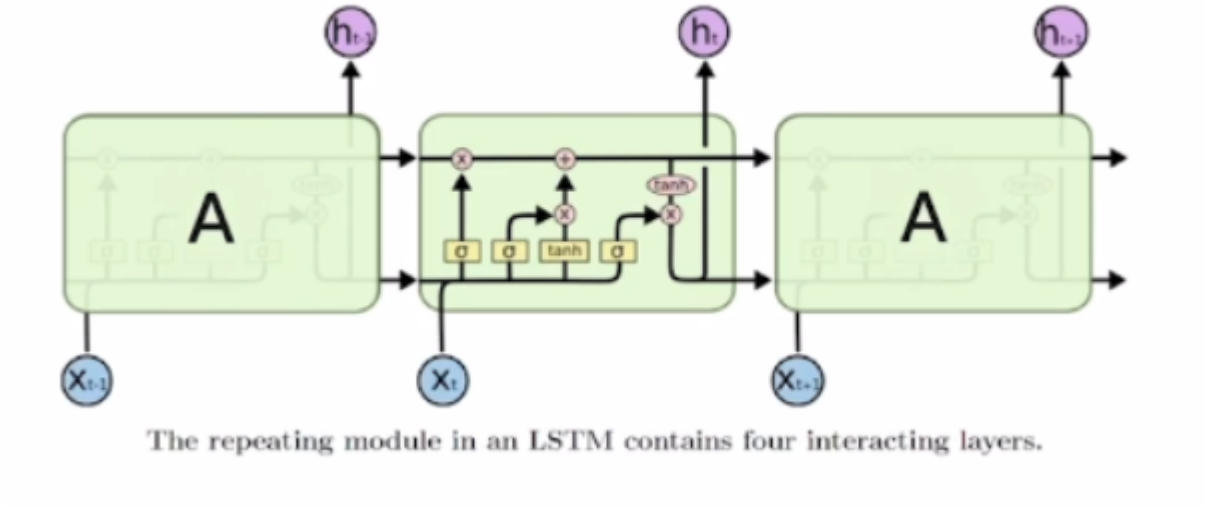
$h_3 = \tanh(2x_3 + 3\tanh(2x_2 + 3h_1 + 1) + 1)$



: time step이 3인 RNN의 BPTT과정

- 3번째 time step의 hidden state 인 h_3 를 h_1 으로 표현하면, 맨 아랫줄의 식과 같이 표현
- BPTT를 통해 gradient를 계산해주면, tanh로 감싸진 괄호안의 값들 중에 3 값이 속미분되어 나오게 됨.
- 위 예시는 time step이 3이므로 3이 2번 곱해지지만, 만약 길이가 더 길어진다면 미분값은 기하급수적으로 커질 것
- 만약 속미분되어 나오는 W의 값이 1보다 작다면 미분값은 기하급수적으로 작아질 것
- 이 계산 과정을 통해 Gradient Vanishing/Exploding 문제가 발생하고 이 문제가 Long-Term-Dependency를 일으킴.

4. LSTM



The repeating module in an LSTM contains four interacting layers.

- 단기 기억으로 저장하여, 때에 따라 꺼내 사용함으로 더 오래 기억할 수 있도록 개선하는 것
- Cell state에는 핵심 정보들을 모두 담아두고, 필요할 때마다 Hidden state를 가공해 time step에 필요한 정보만 노출하는 형태로 정보가 전파됨.

<LSTM의 여러 gate 설명>

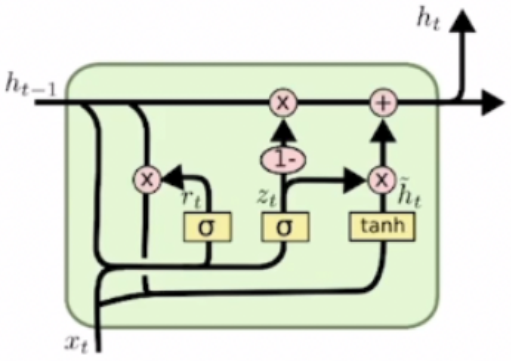
- inputL x_t, h_t - 1
- input를 W에 곱해준 후 각각 sigmoid, tanh로 연산
- I : Input gate로 불리며, cell 에 쓸 지말지를 결정하는 게이트
  - 들어오는 input에 대해서 마지막에 sigmoid를 거쳐 0-1 사이 값으로 표현
  - 표현식: sigmoid(W(xt, ht-1))
- f: Forget gate, 정보를 어느정도로 지울지를 0~1 사이의 값으로 나타냄.

    - 표현식: $sigmoid(W(xt,ht-1)$
- o: Output gate, Cell 정보를 어느정도 hidden state에서 사용해야할 지를 0~1사이 값으로 나타냄
    - 표현식 : sigmoid(W(xt, ht-1)$sigmoid(W(xt,ht-1)$
- g : Gate gate로 불리며, 어느정도로 Cell state에 반영해야할 지를 -1 ~ 1 사이의 값으로 나타냄
    - 표현식 : tanh(W(xt, ht-1)$tanh(W(xt,ht-1)$

- **LSTM vs RNN**
    - LSTM의 특징: 각 time step마다 필요한 정보를 단기 기억으로 hidden state에 저장하여 관리되도록 학습하는 것.
    - 오차역전파(backpropagation) 진행시 가중치(W)를 계속해서 곱해주는 연산이 아니라, forget gate를 거친 값에 대해 필요로하는 정보를 덧셈을 통해 연산하여 그레디언트 소실/증폭 문제를 방지

- **GRU: Gated Recurrent Unit**

- $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
- $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$
- $\tilde{h_t} = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t])$
- $h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h_t}$
- c.f) $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C_t}$ in LSTM



- LSTM과 전체적인 동작원리는 유사
- Cell state, Hidden state를 일원화하여 경량화한 모델
- GRU에서 사용되는 h_t-1은 LSTM에서의 c_t와 비슷한 역할
- forget gate 대신 (1-input gate)를 사용하여 h_t를 구할때 가중평균의 형태로 계산.
- 계산량과 메모리 요구량을 LSTM에 비해 줄여준 모델이면서 동시에 성능면에서도 LSTM과 비슷하거나 더 좋은 성능을 내는 모델

- LSTM과 GRU 모델은 RNN과 달리 가중치를 곱셈이 아닌 덧셈을 통한 그레디언트 복사로 그레디언트 소실/증폭 문제를 해결