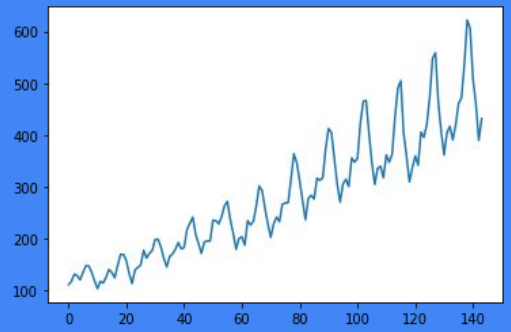
Deep Learning with Tensorflow Keras

5강 RNN

CNN 요약

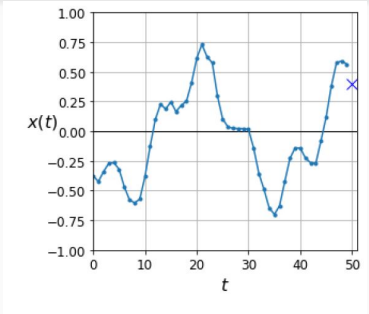
- 이미지, locality(한 점과 그 근처의 데이터가 유사한 것) 데이터에 적용
- 합성곱을 이용해 locality 추출하고 계층적으로 더 복잡한 패턴을 얻어냄
- 경사 소실 문제 해결을 위해 여러가지 구조의 DNN이 나옴
  - 더 층이 많은 DNN을 만들기 위해 여러 연산을 추가함
- 그런데 시간적 순서가 중요한 **Sequential 데이터**는 CNN이 잘 처리할 수 있을까?

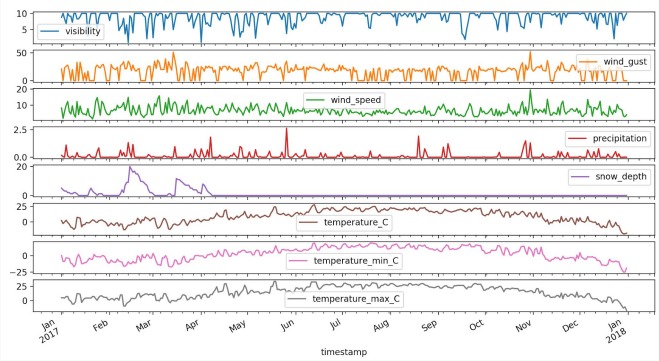
Sequential Data



- 시간적 순서로 나열 가능한 데이터를 일컫는 말
  - 이미지는 시간에 따라 바뀌지 않고 앞에서 본 이미지는 뒤의 이미지와 관계 없음
  - 하지만 음성, 주가, 언어 등은 시간적 순서가 매우 중요
    - 주어 다음에는 동사가 나오는 경우가 많음
    - 주가는 어제 주가에 영향을 크게 받음
- 지금까지 배운 DNN은 이러한 데이터의 특성을 반영할 수 없음
  - 어떤 시점의 데이터가 이전의 데이터와 크게 연관되어도 그 시점의 데이터만 고려함
- 따라서 이러한 데이터를 처리할 새로운 형태의 DNN이 필요함

Univariate/Multivariate

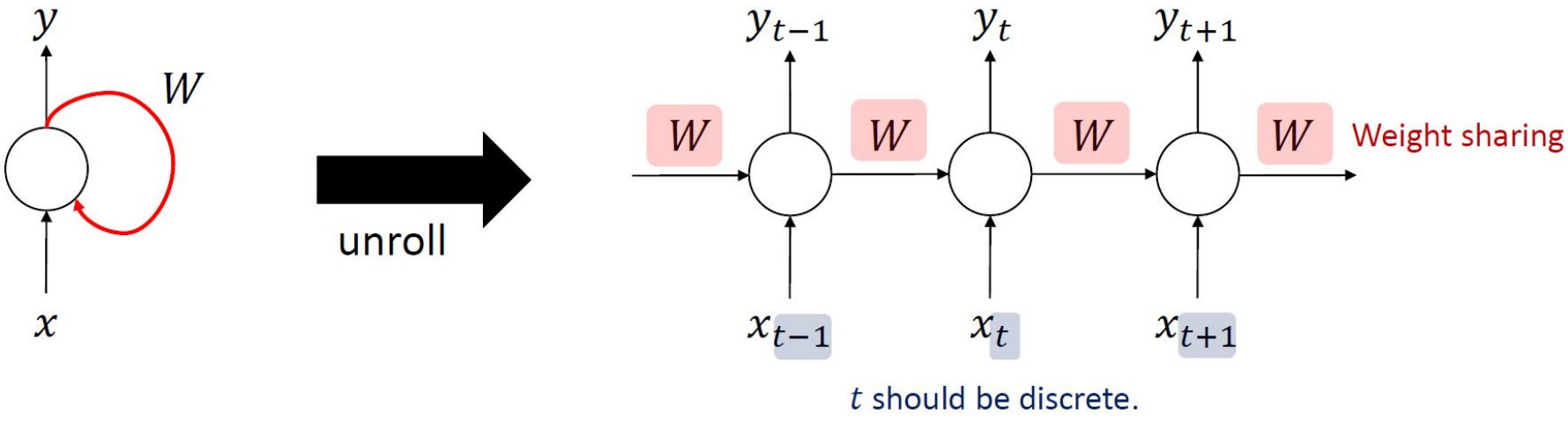




Recurrent Neural Network(RNN)

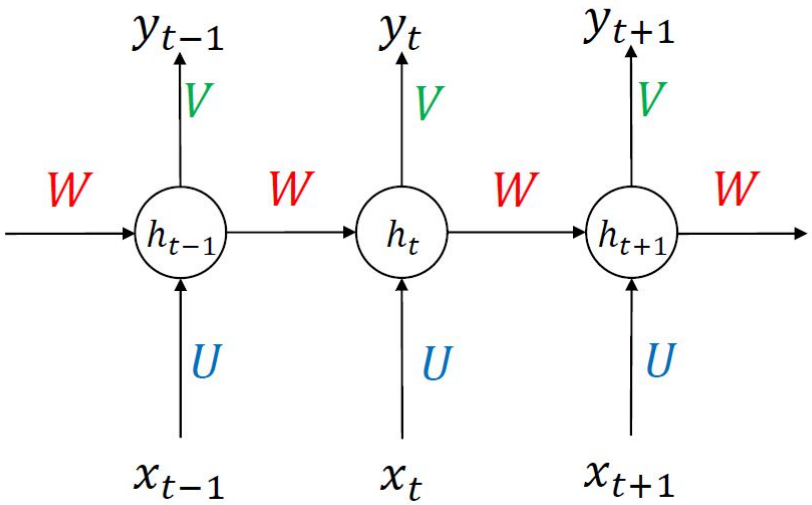
- 지금까지 배운 DNN은 모두 Feed forward network였음
  - 순방향 전파시 NN의 출력이 손실함수까지 바로 전달됨
- 하지만 RNN은 출력이 다시 입력이 됨
  - 즉, 모델 내에 루프가 존재해 피드백을 받게 됨
- 과거의 정보가 현 시점의 모델 결과에 영향을 미치게 됨

Simple RNN 구조



Simple RNN 순방향 전파

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$

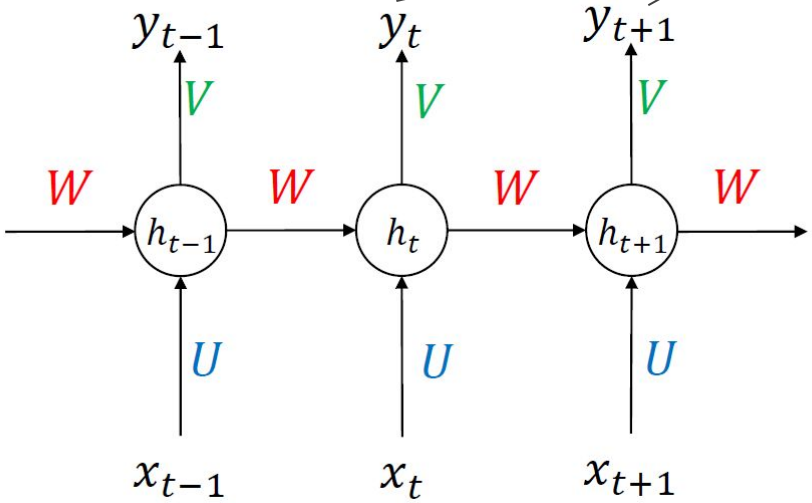


$$h_t = f(Ux_t + Wh_{t-1})$$

$$y_t = f(Vh_t)$$

Simple RNN 순방향 전파

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$



$$h_t = f(Ux_t + Wh_{t-1})$$

$$y_t = f(Vh_t)$$

- Hidden state라고 부름
- RNN 내부에서 다음 데이터가 들어올 때마다 계속 값이 바뀌고 그 다음 시점으로 전달함

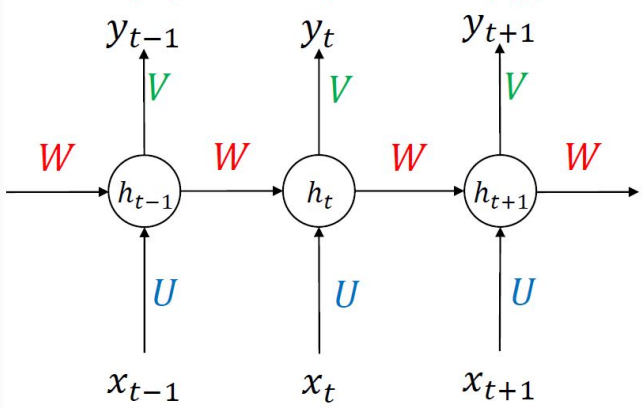
Backpropagation Through Time

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

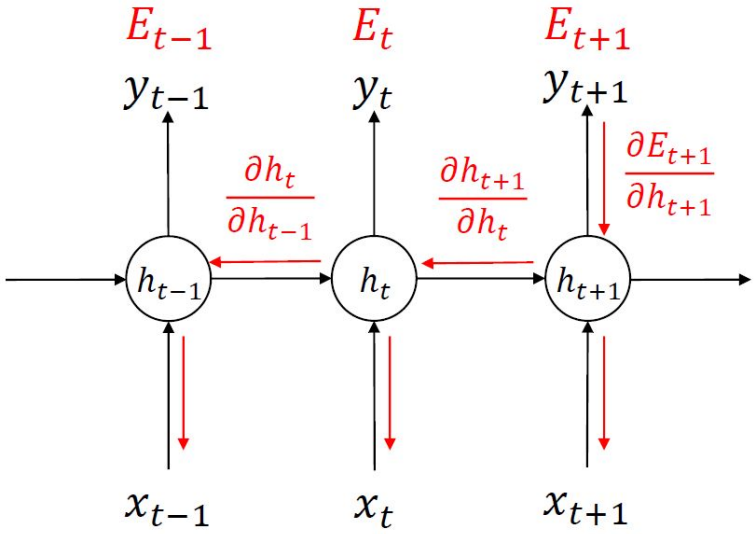
$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial V}$$

$V h_t$



Backpropagation Through Time

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

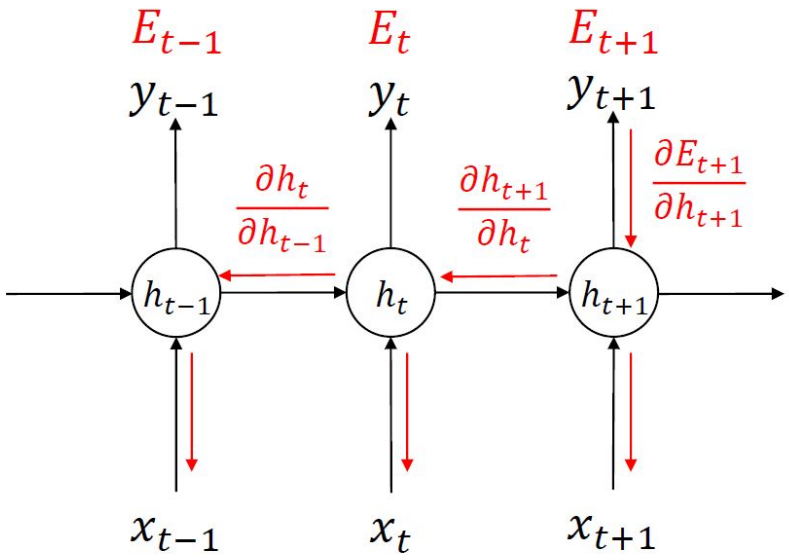


$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial W}$$

$$h_t = f(Ux_t + Wh_{t-1})$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

RNN 경사소실 문제

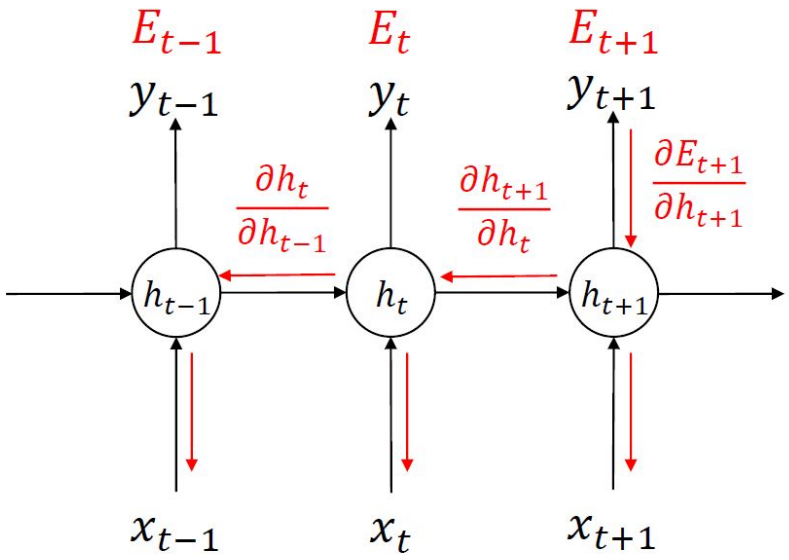


$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j}$$

피드백 루프에 의해 미분값을 여러번 곱하게 되어 경사가 소실되고 **장기적인 데이터의 상관관계(longterm dependency)**를 모델링할 수 없게됨

RNN 경사소실 문제

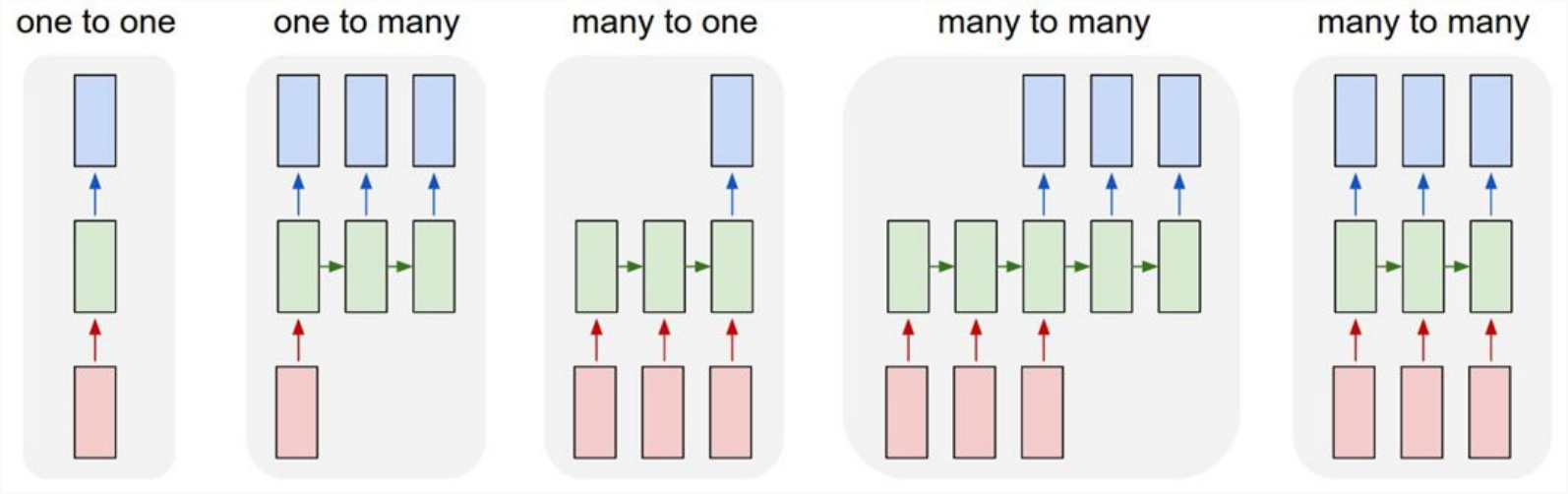


$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j}$$

또한 미분값이 커지면 경사값이 너무 커져버릴 수 있어 활성화함수를 보통 sigmoid같은 함수를 사용함(최대 미분값 제한)

다양한 RNN 구조



Simple RNN 구현

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])
```

출력 차원

출력 횟수

입력 차원

many to many case
if False, many to one case

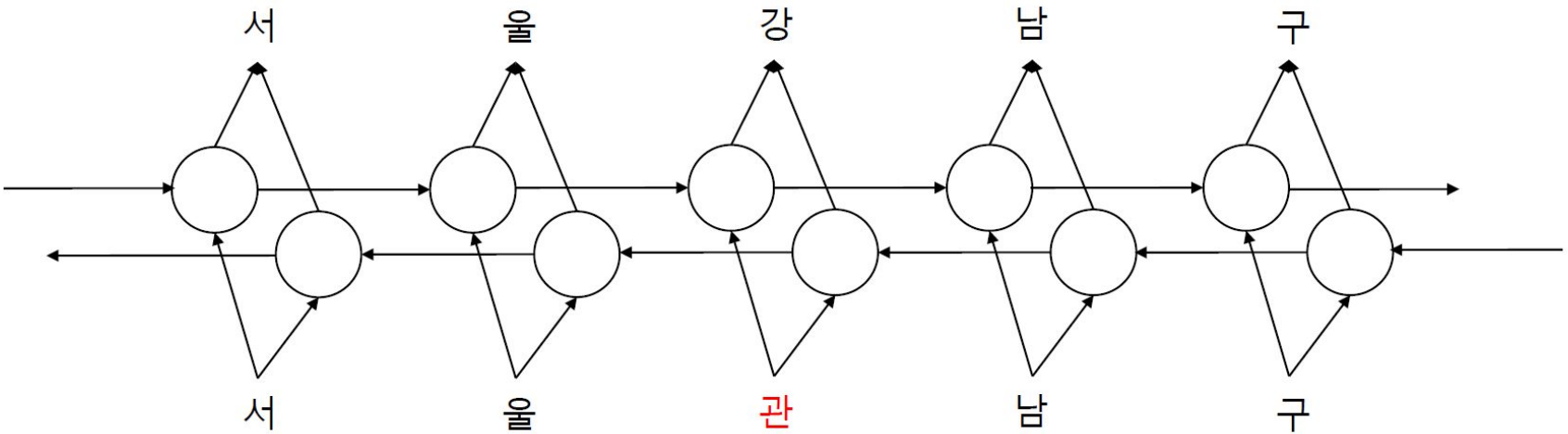
For just one timestamp prediction

여러 미래값을 예측하는 경우 구현

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

- 순차적인 미래값 10개 예측하는 예시
- 10개 값에 대해 따로 Dense layer를 만들어냄

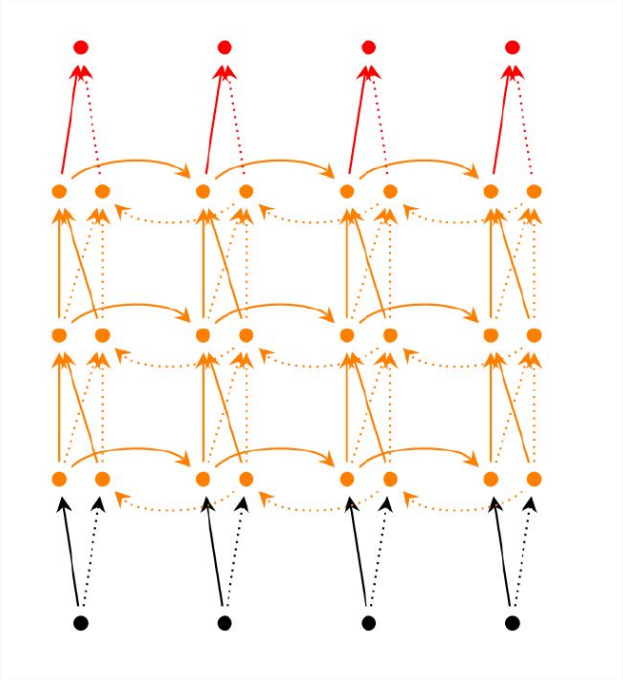
양방향 RNN



- Feedback 루프가 거꾸로 한번 더 작동함(왼쪽에서 오른쪽으로, 오른쪽에 왼쪽으로)
- 오타 수정시에 사용될 수 있음

심층 양방향 RNN

- RNN을 여러층을 쌓아서 만듦
- 각 층의 RNN은 입력이 들어올 때마다 출력 발생
  - 아니게 설정할 수도 있음
  - 그렇게 되면 RNN을 쌓을 수 없음
- 맨 위층에 FCN을 붙일수도 있고 CNN을 붙일 수도 있음



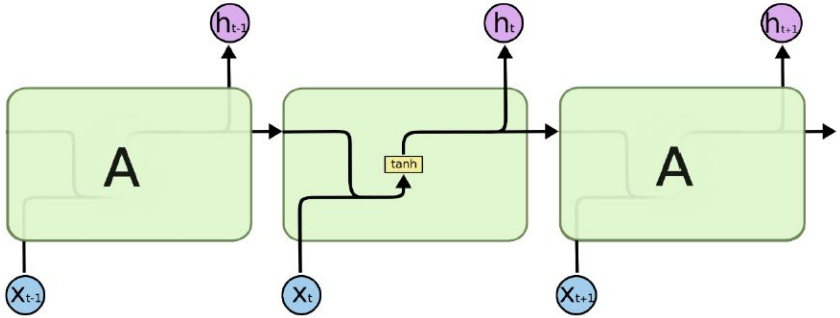
장기간 연관성을 모델링하려면?

- 경사소실이 일어나지 않는 새로운 '길'이 필요함
  - 오래 전에 들어왔던 데이터에 대한 정보를 보관해주는 경로가 필요함

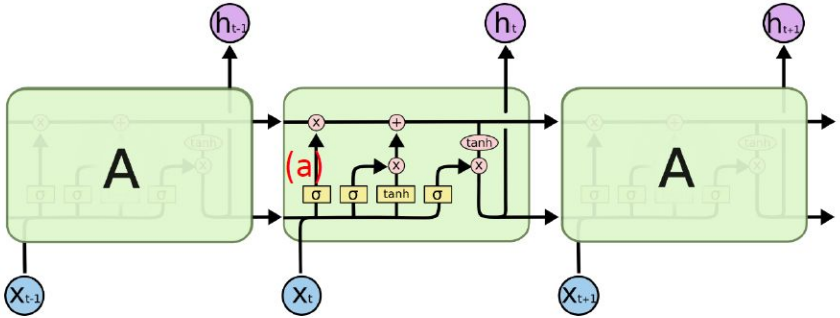
- SimpleRNN은 Hidden State 하나에 의존하고 있음
  - h_t에 t 시점의 데이터와 그때까지 들어온 데이터들 전부의 은닉값이 혼재
  - 그래서 먼 과거의 데이터가 잊혀질 수 있음

Long Short Term Memory(LSTM)

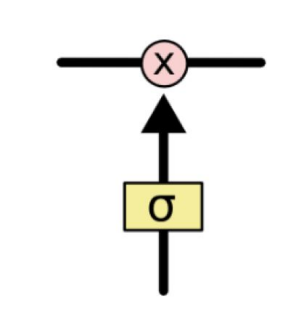
Standard RNN



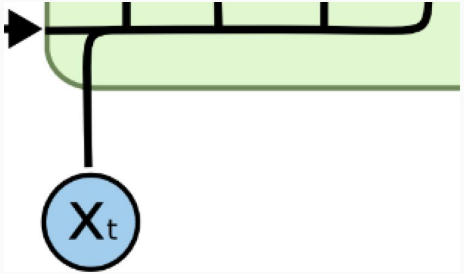
LSTM



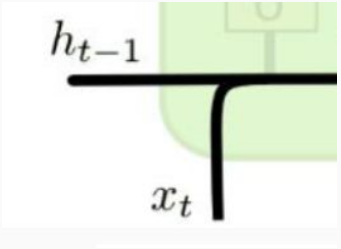
기호 범례



Gate: sigmoid함수를 취한 후 원소별로 곱함

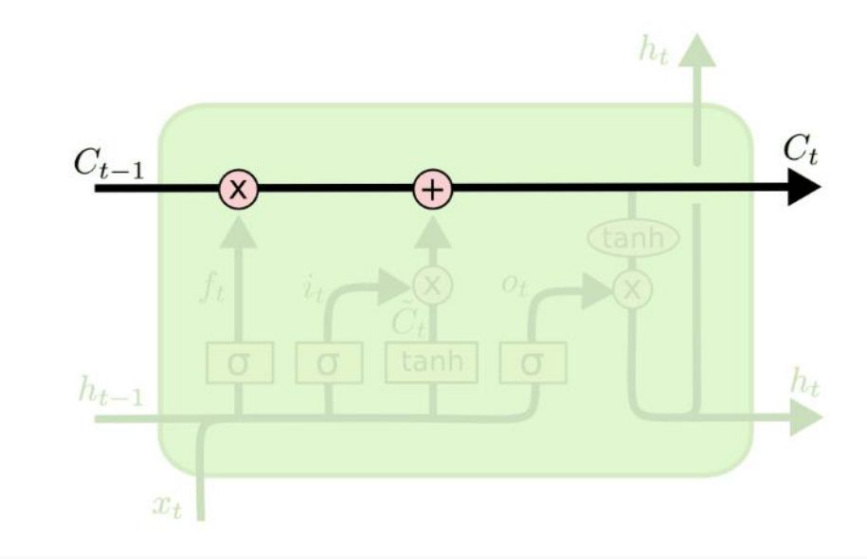


X_t를 복사해서 전달함



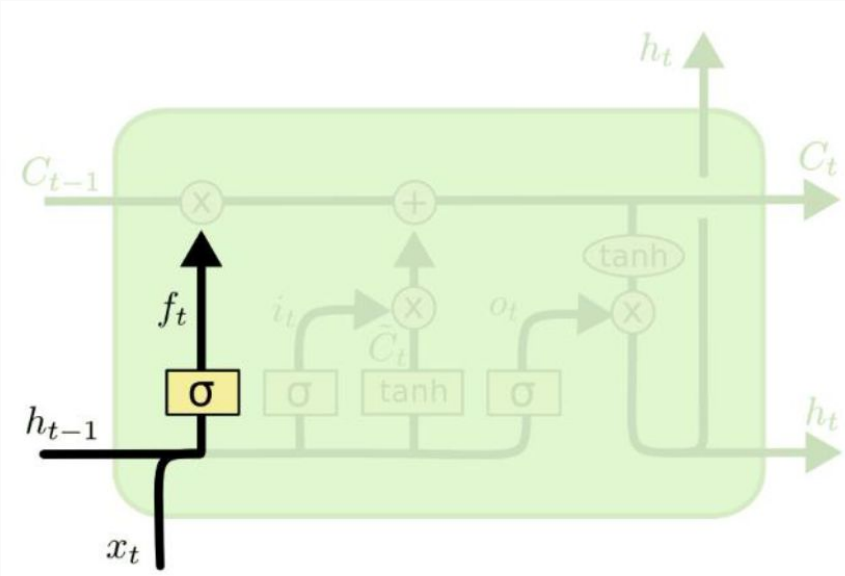
두 벡터를 이어붙임

LSTM의 Cell State



- Cell state: 각 시점에 들어온 입력값을 따로 저장할 수 있는 또 다른 state
- 여러 gate를 이용해 각 시점의 입력 데이터를 얼마나 넣을지 결정하게 됨
- LSTM과 RNN의 가장 큰 차이점

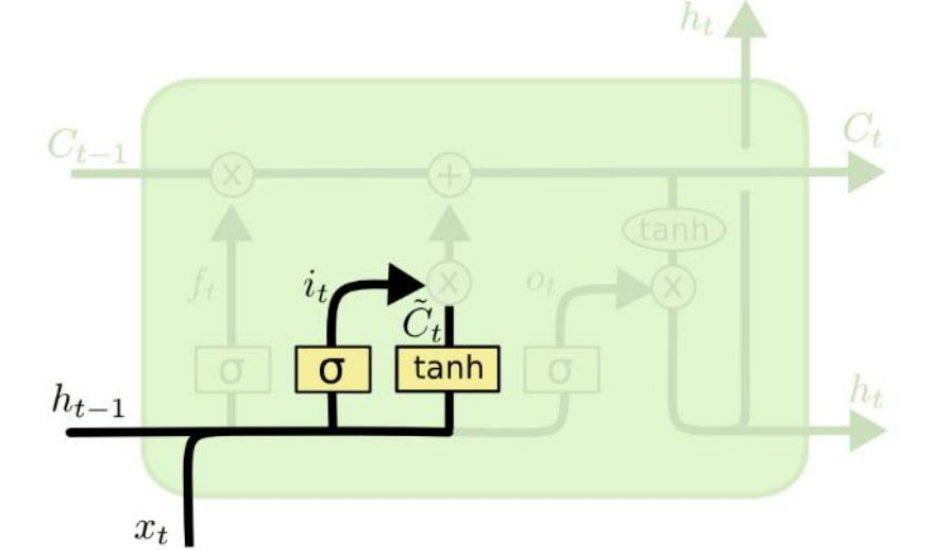
Forget Gate



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

- 가중치 W_f와 이전의 h_(t-1), 이번에 들어온 입력 데이터로 얼마나 이전의 cell state을 기억할지 결정함
- Sigmoid값이 0이면 들어온 cell state를 무시하게 되고, 1이면 모두 기억하게 됨

Input Gate

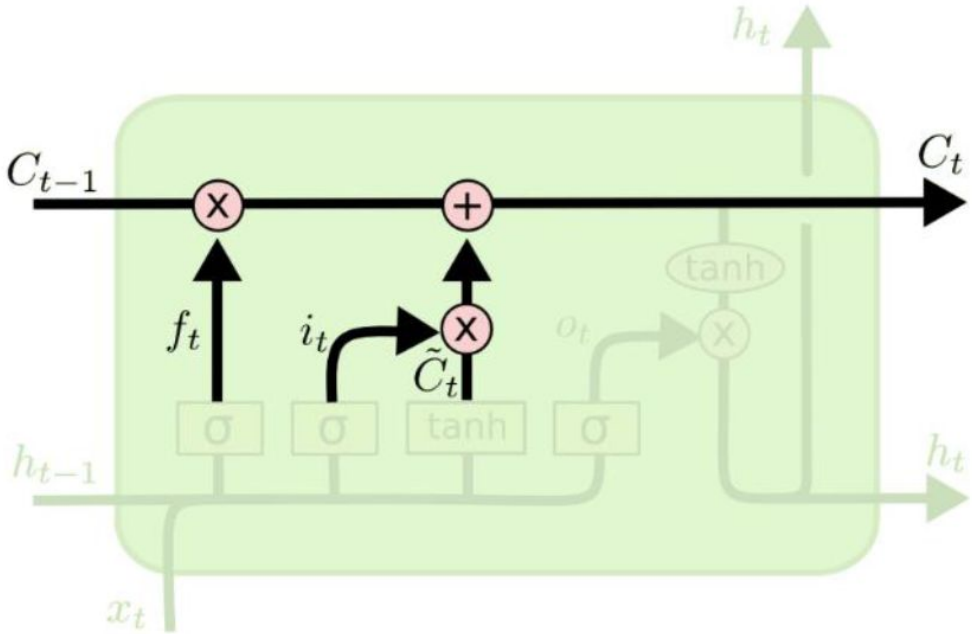


$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_C)$$

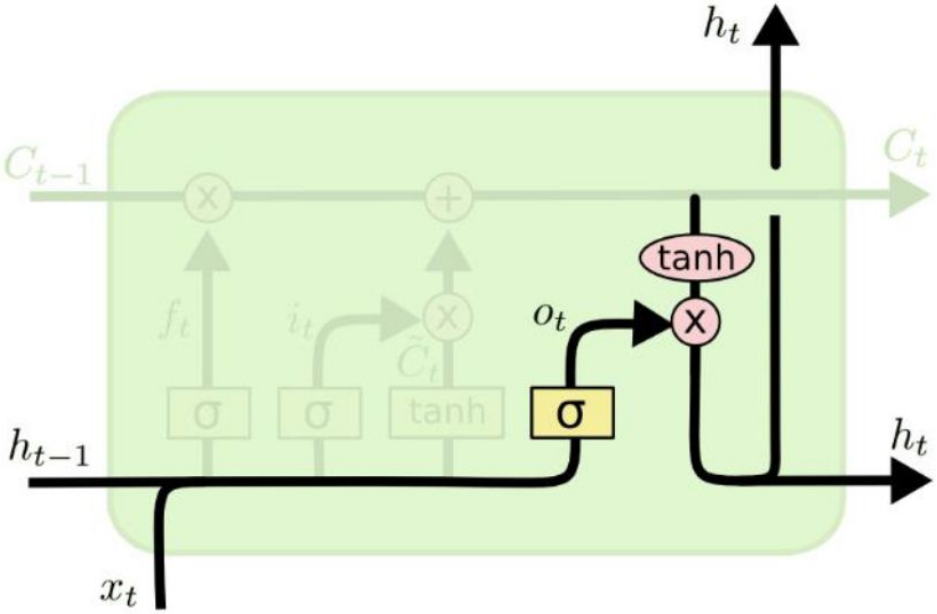
- 가중치 W_i, W_c를 통해 각 시점의 입력 데이터와 이전의 hidden state가 얼마나 Cell state에 기억될지 학습함
- i_t를 통해 얼마나 저장할지 조절

Cell State 업데이트



$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

Output State

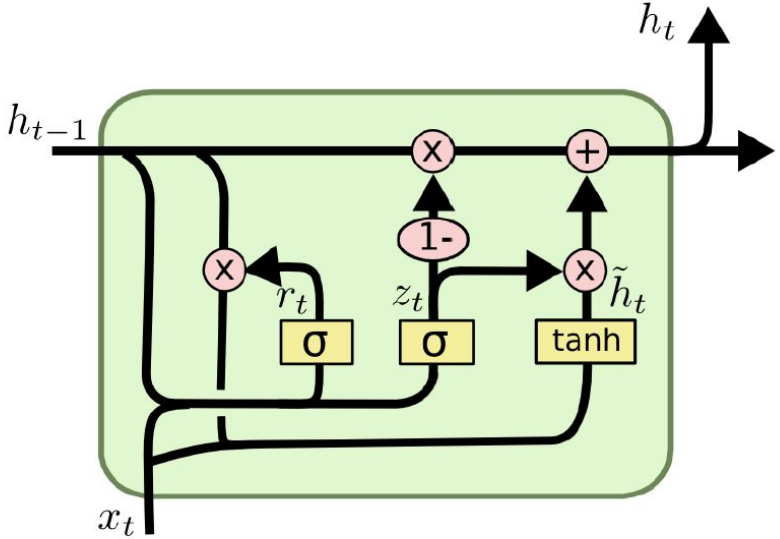


- 업데이트된 cell state와 이전 hidden state, 지금 입력데이터로 출력값을 결정함

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \tanh(C_t)$$

GRU(Gated Recurrent Unit)



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

- Cell state와 Hiddent state가 하나로 합쳐져 있음

LSTM/GRU 성능차: 거의 없음

Arch.	N	N-dropout	P
Tanh	3.612	3.267	6.809
LSTM	3.492	3.403	6.866
LSTM-f	3.732	3.420	6.813
LSTM-i	3.426	**3.252**	6.856
LSTM-o	3.406	**3.253**	6.870
LSTM-b	3.419	3.345	6.820
GRU	3.410	3.427	6.876
MUT1	**3.254**	3.376	**6.792**
MUT2	3.372	3.429	6.852
MUT3	3.337	3.505	6.840

Table 2. Negative Log Likelihood on the music datasets. N stands for Nottingham, N-dropout stands for Nottingham with nonzero dropout, and P stands for Piano-Midi.

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	**4.380 (79.83)**
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

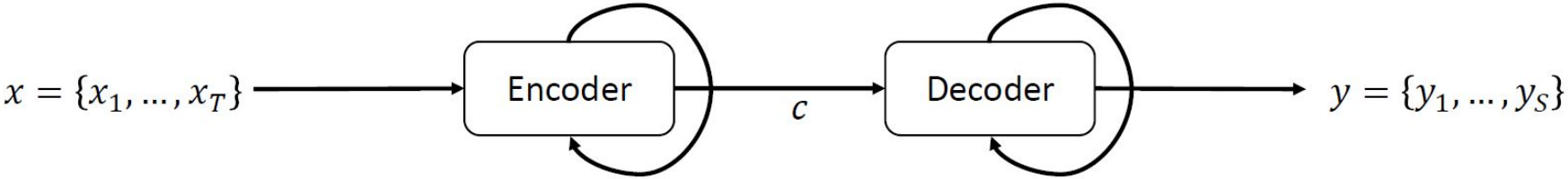
Table 3. Perplexities on the PTB. The prefix (e.g., 5M) denotes the number of parameters in the model. The suffix "v" denotes validation negative log likelihood, the suffix"tst" refers to the test set. The perplexity for select architectures is reported in parentheses. We used dropout only on models that have 10M or 20M parameters, since the 5M models did not benefit from dropout at all, and most dropout-free models achieved a test perplexity of 108, and never greater than 120. In particular, the perplexity of the best models without dropout is below 110, which outperforms the results of Mikolov et al. (2014).

LSTM/GRU 구현

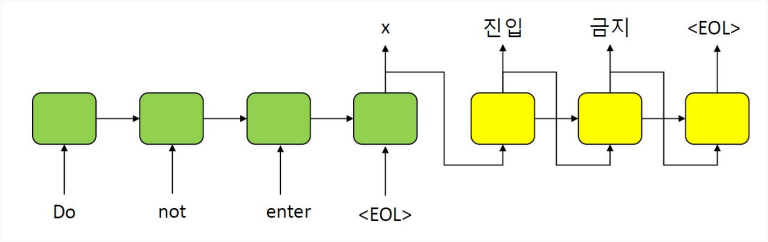
```
model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.LSTM(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

```
model = keras.models.Sequential([
    keras.layers.GRU(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.GRU(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

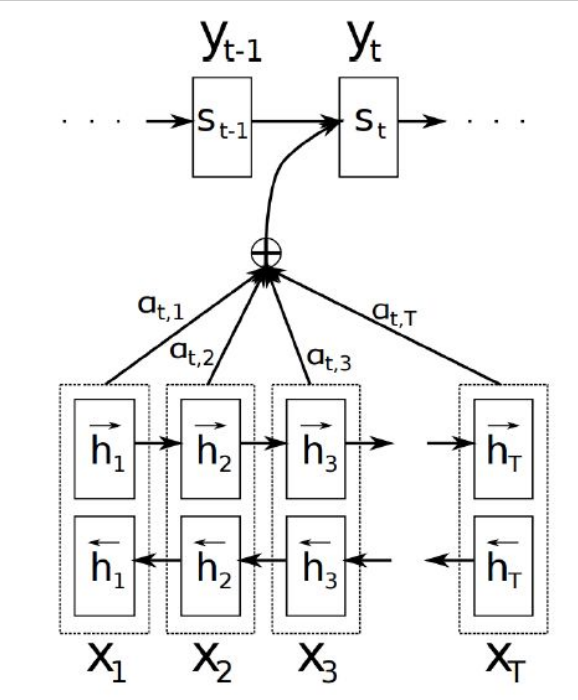
인코더 디코더 프레임워크



- 인코더는 입력을 받아 문맥 벡터 c로 데이터를 변환
- 디코더는 c를 받고 y를 연속적으로 출력
  - 이때 y_t를 출력할 때 y_t-1을 입력으로 사용함



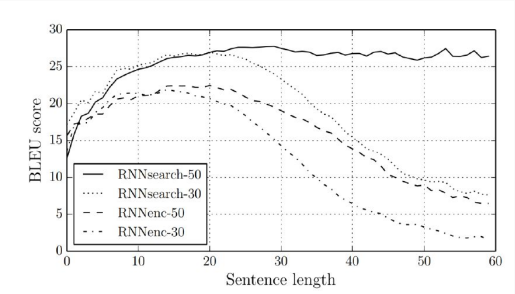
Attention 네트워크

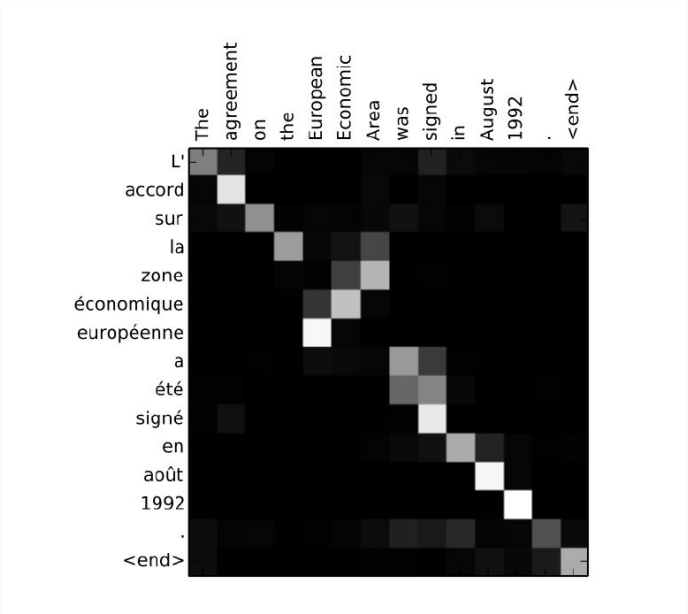


- 각 시점의 hidden state에 파라미터를 곱한 후 다시 새로운 RNN에 그 값을 넣어줌
- 이 파라미터는 학습되어 어떤 t가 y_t을 결정할 때 가장 중요한지 보여주는, **해석할 여지**를 주는 값이 됨

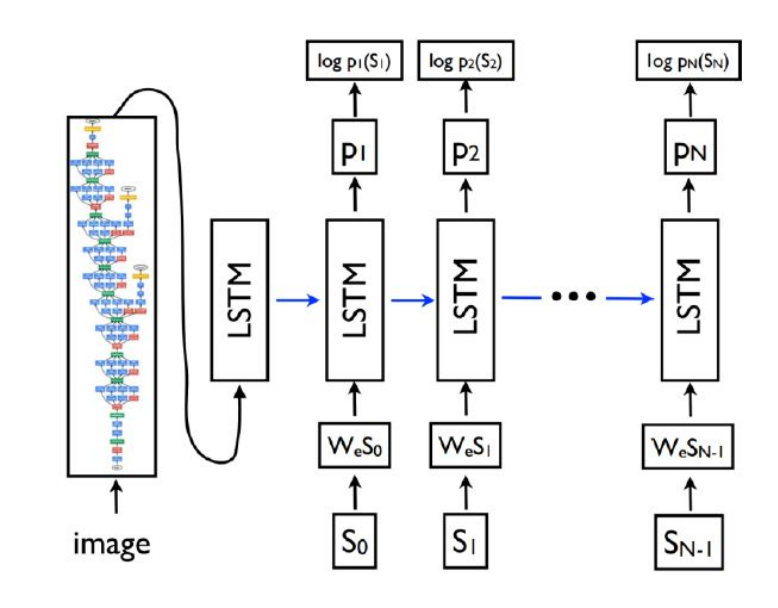
Attention을 사용한 프랑스어-영어 번역

- Attention 파라미터가 높을수록 흰색
- 유럽을 뜻하는 두 언어의 단어의 위치가 달라도 attention에 의해 모델이 더 주목했다는 점을 확인할 수 있음
- 문장이 매우 길어질 때 효과적임





이미지 설명


log p1(S1)
log p2(S2)
log pN(SN)
p1
p2
pN
LSTM
LSTM
LSTM
LSTM
WeS0
WeS1
WeSN-1
image
S0
S1
SN-1


A person riding a motorcycle on a dirt road.
Two dogs play in the grass.
A skateboarder does a trick on a ramp.
A dog is jumping to catch a frisbee.
A group of young people playing a game of frisbee.
Two hockey players are fighting over the puck.
A little girl in a pink hat is blowing bubbles.
A refrigerator filled with lots of food and drinks.
A herd of elephants walking across a dry grass field.
A close up of a cat laying on a couch.
A red motorcycle parked on the side of the road.
A yellow school bus parked in a parking lot.
Describes without errors
Describes with minor errors
Somewhat related to the image
Unrelated to the image