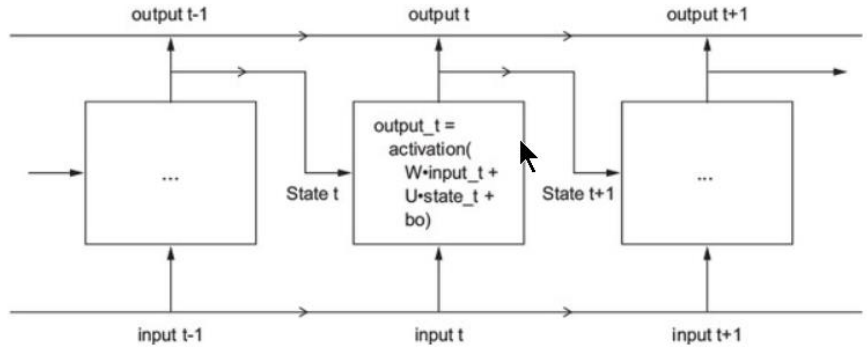
RNN(Recurrent Neural Network)

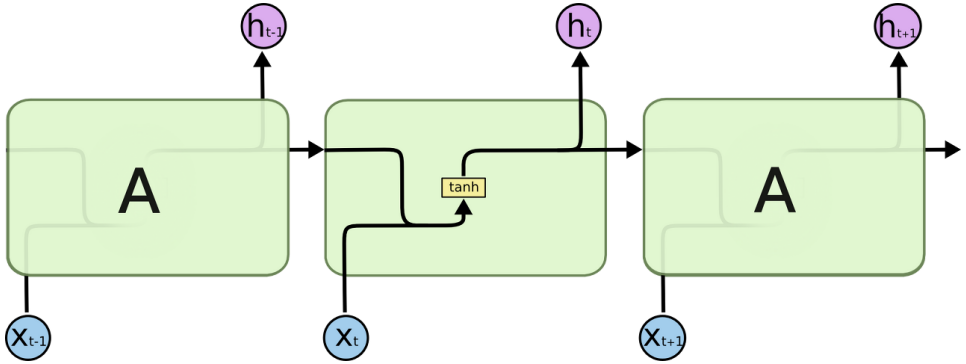
- 시퀀스 데이터를 처리하는 기법
  - 시계열 데이터
  - 텍스트 데이터
  - 기타 순서가 있는 데이터

- 기존 신경망(완전연결 신경망, 컨브넷)의 특징
  - 상태를 유지하는 메모리가 없음.
  - 학습, 예측과정에서 각 입력을 독립적으로 처리하므로 입력데이타의 순서개념이 없음.

- RNN(Recurrent Neural Network, 순환신경망)
  - 시퀀스의 원소를 한번에 처리하지 않고 순회하면서 처리 및 처리결과를 저장하는 기능이 있음.

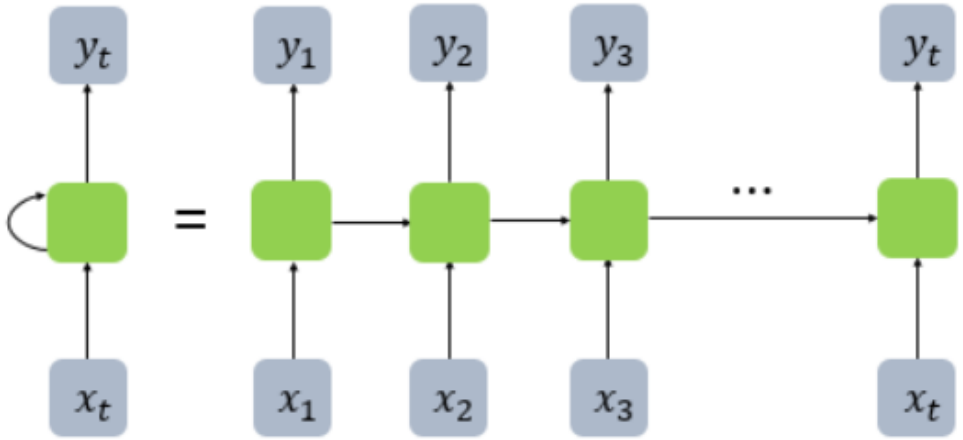
Figure 6.8. A simple RNN, unrolled over time

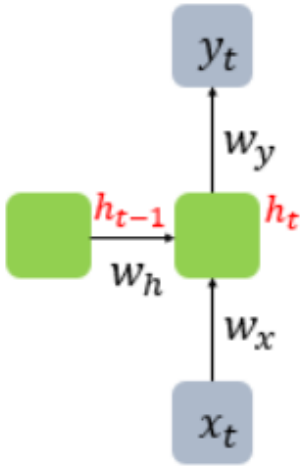




RNN(Recurrent Neural Network)

- 메모리 셀 :
  - 셀은 이전의 상태값을 기억하려고 하는 일종의 메모리 역할을 수행.
- 시점 t에서의 은닉 상태값(ht) 계산 :
  - 입력층을 위한 가중치 Wx, t-1의 은닉 상태값인 ht−1을 위한 가중치 Wh 사용.
- 출력(yt) 계산 :
  - ht와 출력을 위한 가중치 Wy를 계산 후 비선형 활성함수 적용.





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

new state / some function with parameters W / old state / input vector at some time step

은닉층 : $h_t = tanh(W_x x_t + W_h h_{t-1} + b)$

출력층 : $y_t = f(W_y h_t + b)$

RNN(Recurrent Neural Network)

- Simple RNN의 문제점
  - 최근의 정보는 유지할 수 있지만 오래전의 정보는 유지하기 어렵다.

the clouds are in the ***sky.***

vs.

I grew up in France… I speak fluent ***French.***

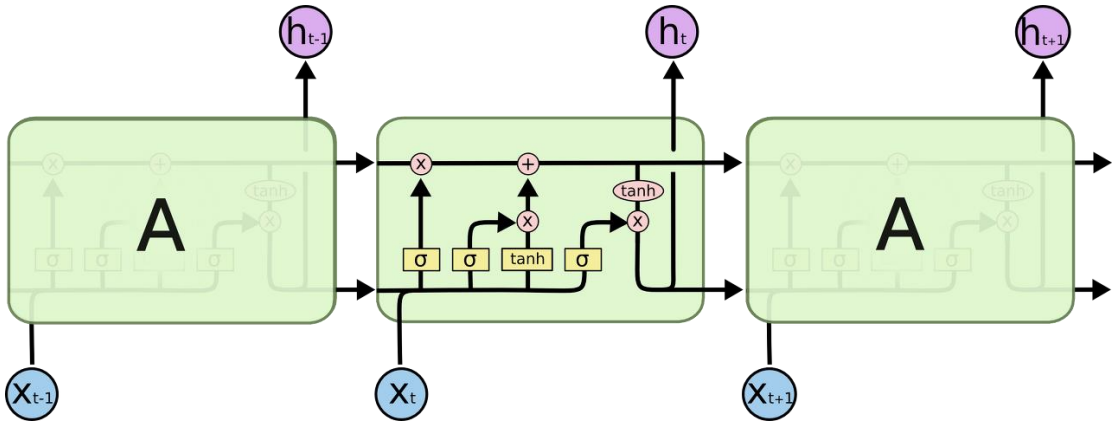
  - 모든 타임 스텝마다 처음부터 끝까지 역전파 수행.
  - 타임 스텝이 길면 매우 깊은 네트워크가 되며 이러한 네트워크는 Gradient vanishing / exploding 문제 유발.
  - 장기간의 패턴을 학습하기 어려움.

LSTM(long-Short term Memory)

- Simple RNN(Vanilla RNN)보다 향상된 RNN
  - RNN cell은 은 단순한 신경망(tanh 활성함수 포함) 사용.
  - 기존의 RNN이 출력과 먼 위치에 있는 정보를 기억할 수 없다는 단점을 보완하여 장/단기 기억을 가능하게 설계한 신경망의 구조.

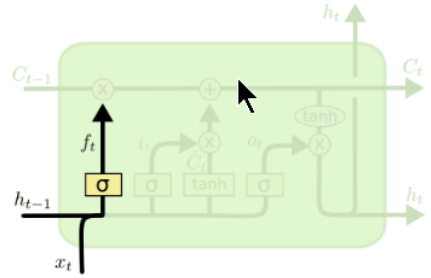
- 주요 구조
  - cell state 가 추가됨.
  - LSTM cell에는 4개의 상호작용하는 기능이 들어있다.
  - ht : 단기 상태(short-term state)이면서 출력.
  - ct : 장기 상태(long-term state) 표현.
  - 장기상태(cell state)가 구조의 핵심으로 셀 내부의 다른 gate들에서 가공되는 정보를 유지.전달하는 역할.

- LSTM(Long-Short Term Memory)
- GRU(Gated Recurrent Unit) - LSTM과 같은 원리지만 더 간결하여 자원 덜 사용함)



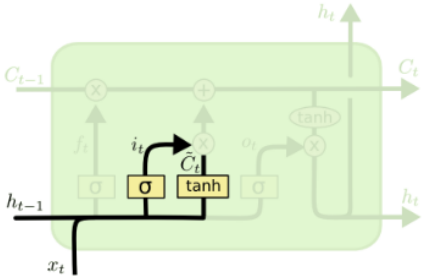
LSTM 4개의 중요 요소

Forget gate에서 이전 cell state 값(Ct-1)을 어느 정도 버릴 지 계산.
이전출력(ht−1)과 입력(x_t)을 받아서 cell state에 일정량(0 ~1) 전달.
1이면 "모든 정보 보존", 0이면 무시.



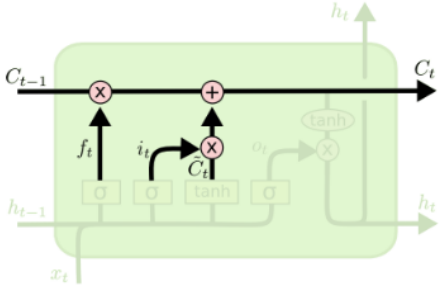
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

It : 새로운 정보를 어느 정도 기억할 지.
Ct : 새로운 정보.
위 두 정보를 곱하여 cell state를 업데이트할 내용 생성 .



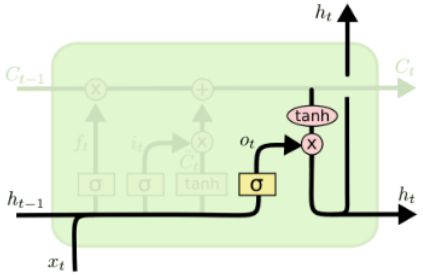
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Cell state update Ct-1 → Ct
기존의 정보를 얼만큼 잊고, 새로운 정보로 얼만큼 대체 할 것인가



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

ht(hidden state), output 출력
cell state 정보와 입력 정보를 연산하여 새로운 hidden state, output 생성



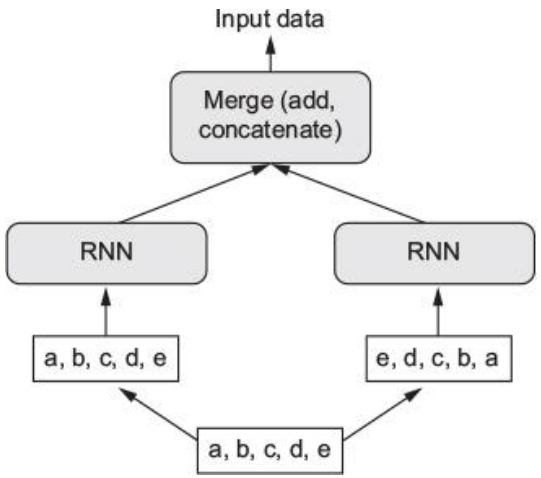
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://warm-uk.tistory.com/54

LSTM 성능을 향상하기 위한 기법

- 드롭아웃 적용
  - 과대적합을 억제하고 성능을 향상하는 기법
- Stacking
  - 층 추가 및 층내 유닛 추가방법
  - 과대적합이 일어날 때까지 네트워크 용량을 늘임.

- 양방향 순환 층
  - 같은 정보를 역방향으로 구성하여 동시에 주입
  - 데이터를 다른 시각에서 보기 때문에 일방향 순환망에서 놓칠 수 있는 패턴을 추가로 확습할 수 있음.



RNN 실습

- 데이터 : 독일 예나시의 2009 ~ 2016년 사이의 기상데이타
- 기온, 기압, 습도, 풍향 등 14개 관측치가 10분 단위로 기록됨.
- 예측목표 : 24시간 후의 기온

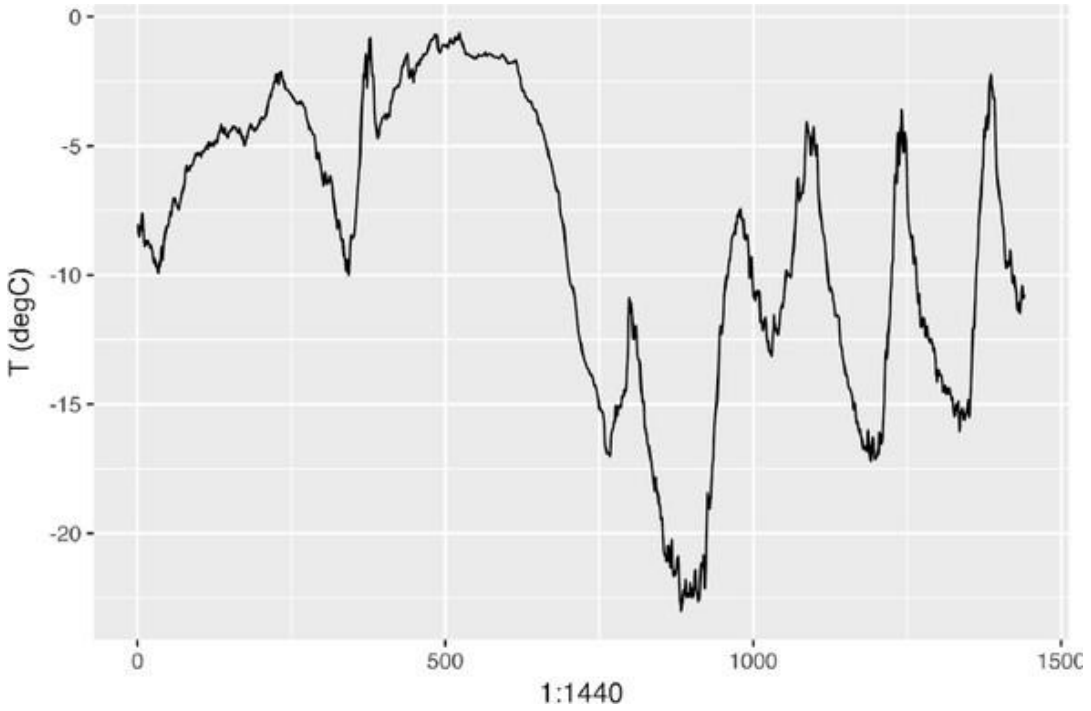
1. 기존 신경망을 이용한 예측
   - 32 unit 을 가지는 기본 신경망을 구성하여 예측
   - 2.57도 오차를 보임.

2. 순환신경망을 이용한 예측
   - 32 unit 을 가지는 GRU 를 구성하여 예측
   - 2.35도 오차를 보임

3. 드롭아웃 적용하여 예측
   - 과대적합 방지 목적
   - 과대적합 경향이 크게 줄어듬.

4. Stacking
   - 네트워크의 표현능력 증가시킴
   - 성능 약간 향상에 그침.

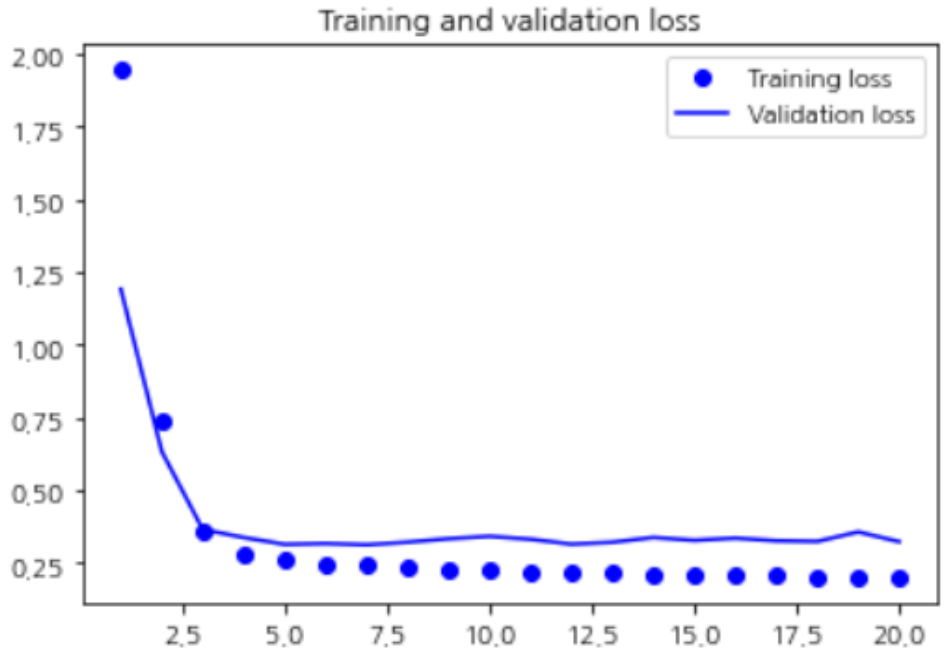
5. 양방향 RNN 을 이용한 예측
   - 같은 정보를 반대방향으로도 주입



RNN 실습 - 기본 신경망

- 기본 완전연결 네트워크
  - ➢ RNN처럼 복잡하고 연산 비용이 많이 드는 모델을 시도하기 전에 간단한 소규모의 완전 연결 네트워크 시험.

```
model.add(layers.Flatten(input_shape=(lookback // step, float_data.shape[-1])))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))
```

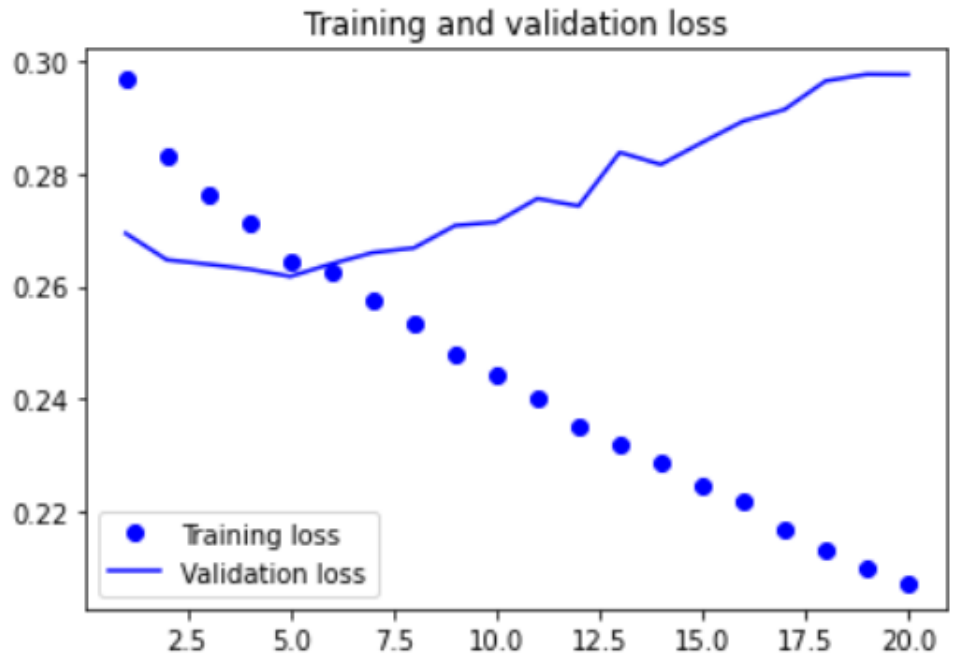


- (분석)
- 머신 러닝 모델이 핵심 정보를 찾지 못했을 가능성.

기본 순환 신경망

- 기본신경망은 시계열 데이터를 펼쳤기 때문에 입력 데이터에서 시간 개념을 잃어버림.
- 순서가 의미가 있는 시퀀스 데이터 그대로 사용해서 시험.

```
model = Sequential()
model.add(layers.GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))
```

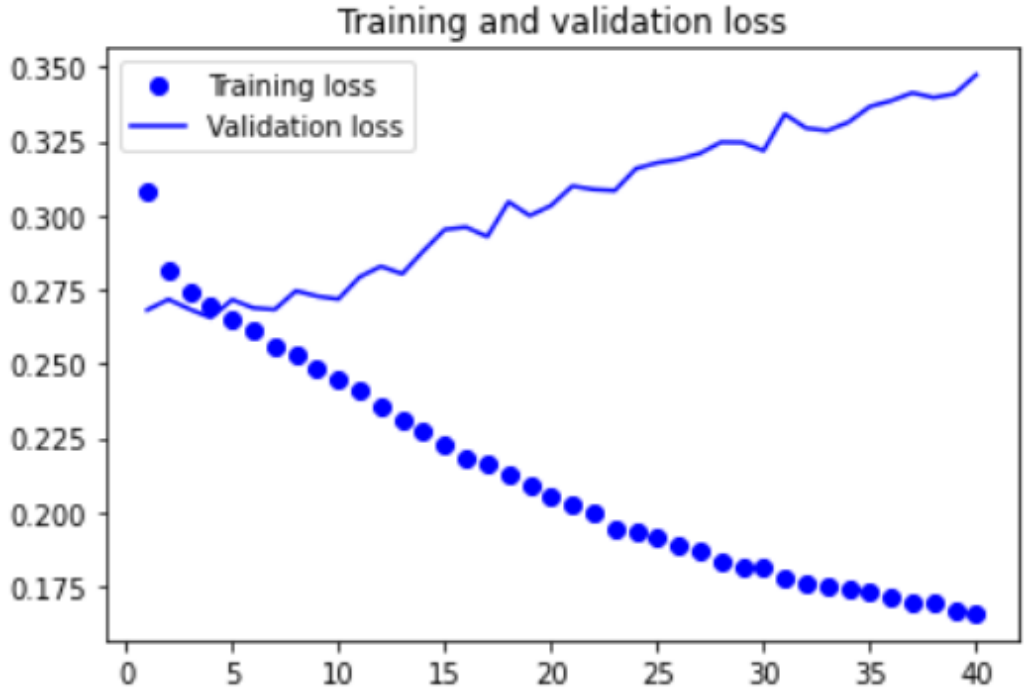


- (분석)
- 기본성능을 앞지름.
- 시퀀스를 펼쳐서 처리하는 완전 연결 네트워크에 비해서 순환 네트워크가 이런 작업에 적합함.
- 과대적합 발생 : 몇 번의 에포크 이후에 훈련 손실과 검증 손실이 현저하게 벌어짐.

과대적합을 감소하기 위해 드롭아웃 사용

- RNN에 기본 Dropout을 잘못 적용하면 과거 정보를 잃어버릴 확률이 높아지게 되므로 Model의 성능이 나빠질 수 있음.
  - * `dropout` : 층의 입력에 대한 드롭아웃 비율
  - * `recurrent_dropout` : 순환 층 내부 계산에 사용된 활성화 함수에 타임스텝마다 동일한 드롭아웃 마스크를 적용

```
model.add(layers.GRU(32,
                     dropout=0.2,
                     recurrent_dropout=0.2,
                     input_shape=(None, float_data.shape[-1])))
```

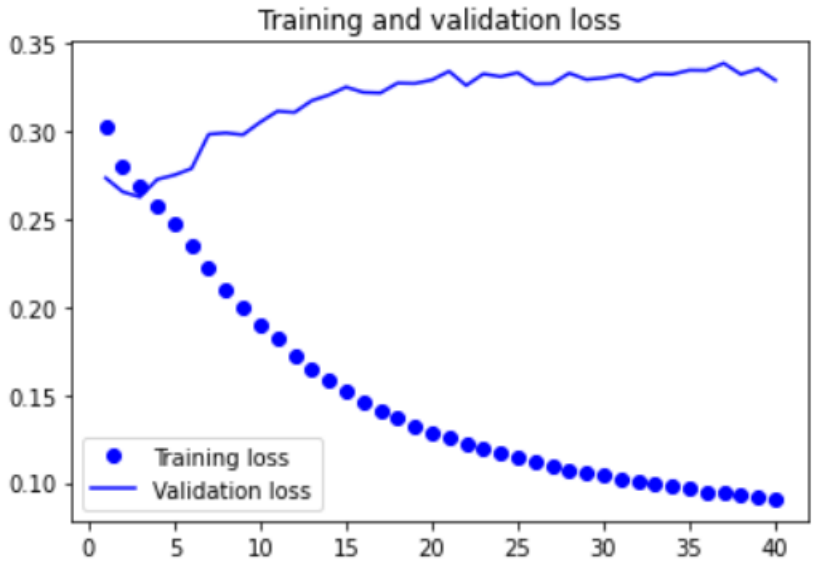


- (분석)
- 과대적합경향 감소.
- 평가 점수는 안정적이지만 이전보다 더 나아지진 않았음.

스태킹 순환 층

- 성능상의 병목 해결방법의 하나는 네트워크의 용량을 늘이는 것임.
- 과대적합을 줄이는 기본 단계를 거친 후 과대적합이 일어날 때까지 네트워크의 용량을 늘림.
- 층에 있는 유닛의 수를 늘리거나 층을 더 많이 추가

```
model.add(layers.GRU(32,
                     dropout=0.1,
                     recurrent_dropout=0.5,
                     return_sequences=True,
                     input_shape=(None, float_data.shape[-1])))
model.add(layers.GRU(64,
                     activation='relu',
                     dropout=0.1,
                     recurrent_dropout=0.5
                    ))
```

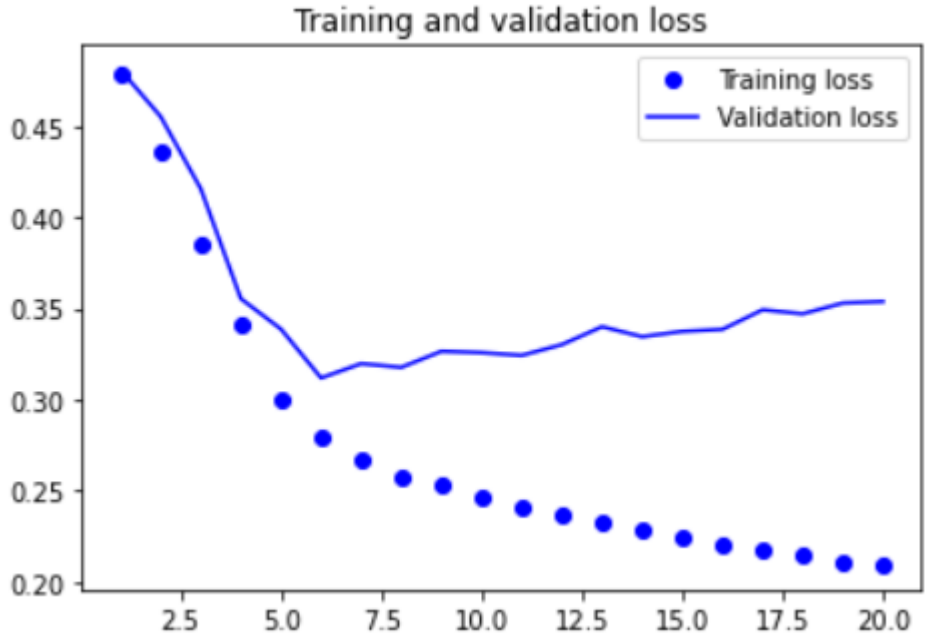


- (분석)
- 층을 추가하였으나 성능은 조금 향상
- 아직 과대적합을 만들지 못했기 때문에 층의 크기를 늘릴 수 있음. 적지 않은 계산 비용이 추가.

양방향 순환 층

- RNN의 한 변종이고 특정 작업에서 기본 RNN 보다 훨씬 좋은 성능을 냄(자연어 처리분야).
- GRU나 LSTM 같은 RNN 두 개를 사용
- 각 RNN은 입력 시퀀스를 한 방향(시간의 순서나 반대 순서)으로 처리한 다음 각 표현을 합침.
- 시퀀스를 양쪽 방향으로 처리하기 때문에 단방향 RNN이 놓치기 쉬운 패턴을 감지할 수 있음.

```
model.add(layers.Bidirectional(
    layers.GRU(32), input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))
```



- (분석)
- 일반 GRU 층과 비슷한 성능
- 시간 반대 순서로 처리하는 절반은 예측에 덜 중요(최근의 정보가 오래 전의 정보보다 더 중요함).