강의계획표

주	해당 장	주제
1	1장 2장, 3장	머신러닝이란
2		머신러닝을 위한 기초지식, 구현을 위한 도구
3	4장	선형 회귀로 이해하는 지도학습
4	5장	분류와 군집화로 이해하는 지도 학습과 비지도 학습
5	6장	다양한 머신러닝 기법들 - 다항 회귀, Logistic Regression - 정보이론, 결정트리 - SVM, Ensemble
6		
7		
8		중간고사 (04-20)
9	7장	인공 신경망 기초 - 문제와 돌파구
10	8장	고급 인공 신경망 구현
11	9장	신경망 부흥의 시작, 합성곱 신경망
12	10장	순환 신경망
13	11장	차원축소와 매니폴드 학습
14	12장	오토인코더와 잠재표현 학습
15		보강주
16		기말고사

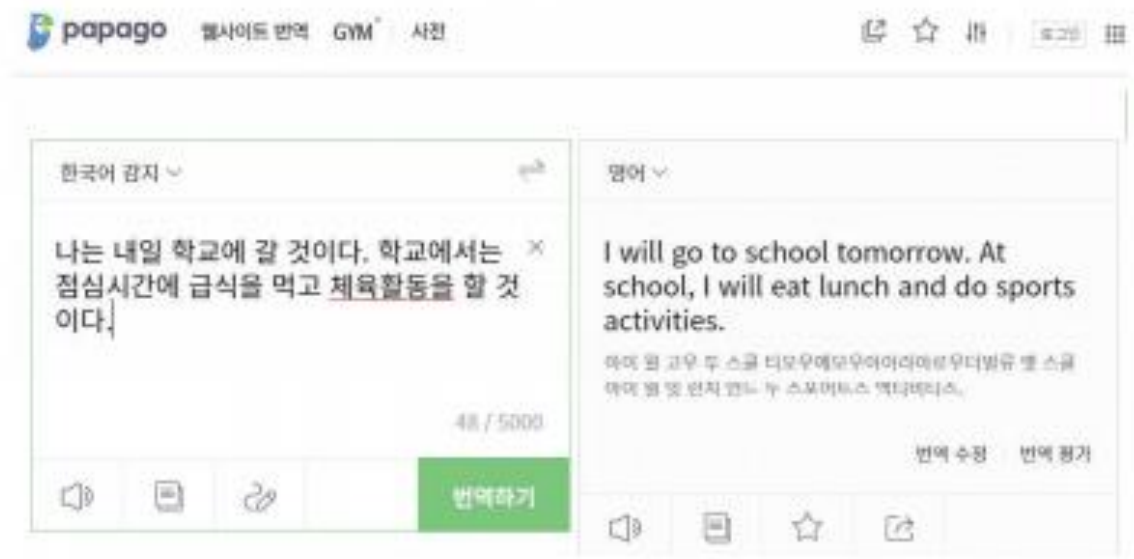
10장 순환 신경망

12 주차

- 시간의 흐름에 따라 연속적으로 입력되는 데이터를 처리하는 방법
- 오래 전에 있었던 입력의 내용을 기억하고 활용하는 방법
- 연속적으로 입력되는 데이터에 대응되는 연속적인 데이터 생성 방법
- 순환 신경망의 다양한 응용 분야

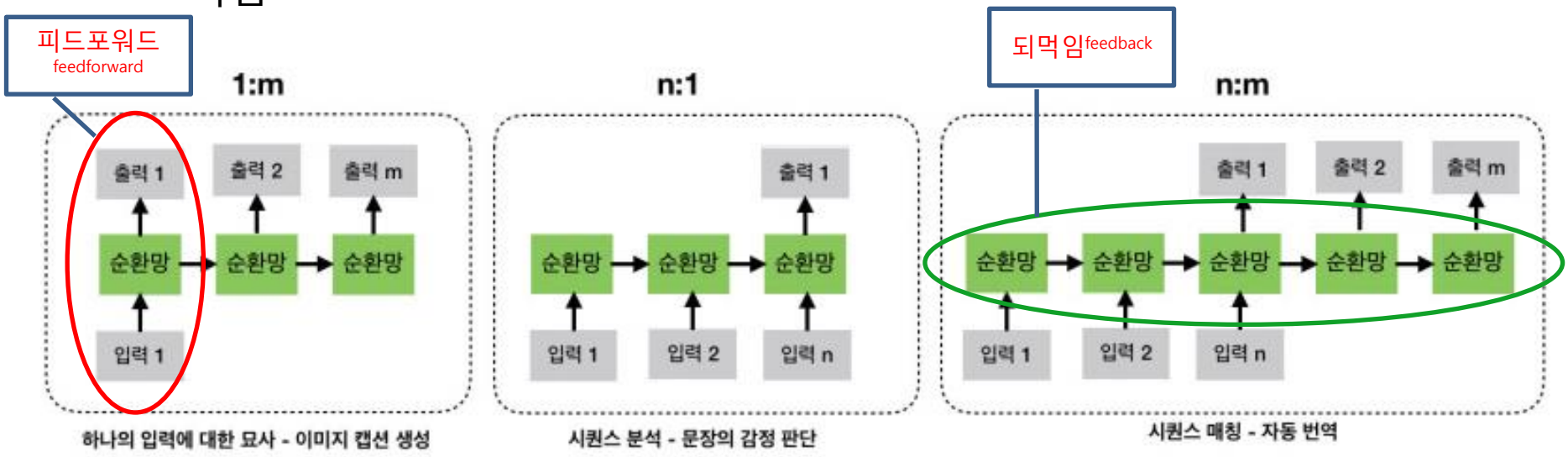
순환 신경망의 적용 분야

- ❖ 피드포워드 신경망feedforward neural network
  - 모든 신호가 오직 출력층 방향으로만 향하는 신경망
  - 특정 한 시점의 이미지나 정보만을 이용, CNN
- ❖ 순환 신경망recurrent neural network:RNN
  - 시간에 따라 순차적으로 제공되는 정보를 다룰 수 있는 신경망
  - 활성화 함수를 통해 나온 출력이 다시 자기 자신에게 입력(feedback)
  - 응용 사례: 음성, 언어이해, 기계번역 프로그램
  - 입력과 출력이 순서를 가지고 연속적으로 나타나는 시퀀스sequence



순환 신경망의 적용 분야 (2)

- 시퀀스-대-시퀀스sequence to sequence, seq2seq 매칭
  - n개의 순차적으로 입력을 m개의 순차적으로 출력으로 나열
  - 회색 사각형 입력이 녹색의 순환망에 입력되고, 순환망이 자기 자신에게 신호를 되먹임feedback하면서 위쪽으로 회색 사각형의 출력을 내보냄
  - 입력: 1개 혹은 n개, 출력: 1개 혹은 m개
  - 아래 위로 한 줄의 연결이 피드포워드feedforward 신경망이고, 오른쪽으로 한 칸씩 이동할 때마다 시간이 한 단계씩 증가하여 새로운 신호를 받거나 순환 신경망이 자기 자신에게 신호를 되먹임feedback하여 출력을 만들어냄



순환 신경망의 적용 분야 (3)

❖ 이미지 캡션caption 생성

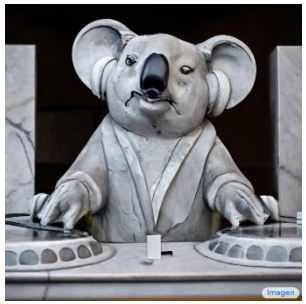
- 1:m
- 합성곱 신경망을 이용하여 이미지의 특징을 파악한 뒤, 이 특징을 임베딩 공간이라는 벡터 공간에 보낸 뒤, 해당 이미지를 설명할 수 있는 적절한 캡션을 생성

"Two young girls are playing with lego toy"

❖ AI image생성

- n:1
- Text-to-image
- 2023년 최고의 AI 이미지 생성기 - Ecommerce Platforms

"A marble statue of a Koala DJ in front of a marble statue of a turntable. The Koala has wearing large marble headphones."

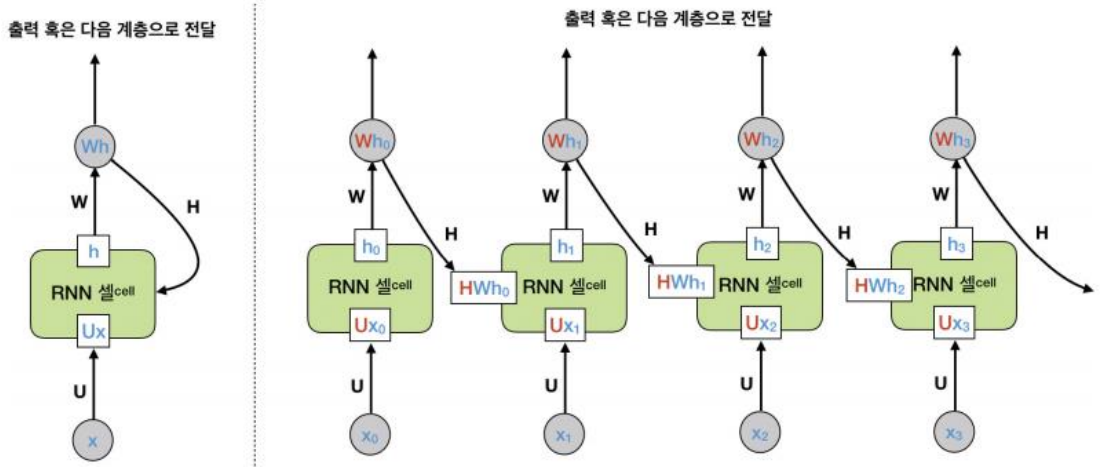


순환 신경망 구조

❖ 바닐라 순환 신경망vanilla RNN

- Feedforward & feedback

- U x 입력신호가 cell에 전달
- h = 활성화함수(Ux)
- 출력= Wh
- 되먹임: HWh_{t-1}



단순화한 그림

좌측의 그림을 펼친 그림

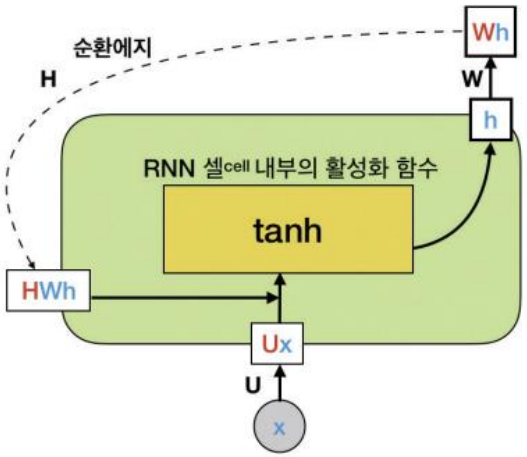
- 순환 에지는 **이전 순간$(t-1)$에 발생한 정보를 다음 순간(t)에 전달할 수 있기** 때문에 순환 신경망의 뉴런은 일정한 지속시간을 갖고, 길이가 가변적이며, 문맥 의존적인 입력을 처리할 수 있음
- 시간의 흐름에 따라 새로운 벡터를 입력받지만 모든 순간마다 동일한 U, W, H 라는 연결강도를 공유

순환 신경망 셀의 구조와 유닛 (1)

❖ 셀의 구조

- 하나의 unit을 사용하는 경우 H=1

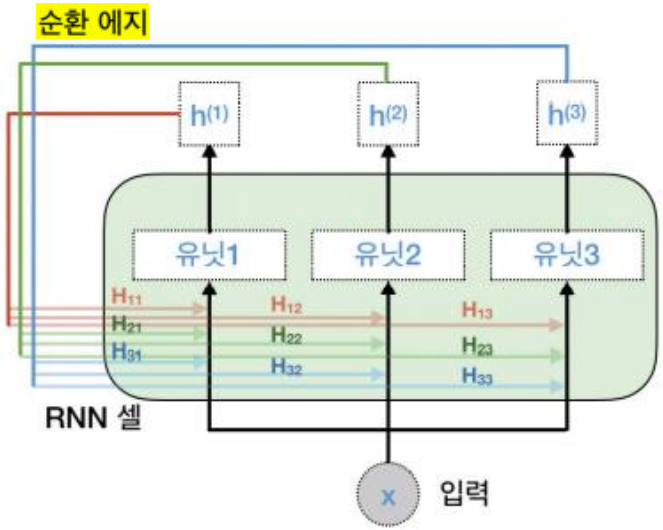
$$h_t = fw(h_{t-1}, x_t) = \tanh(\mathbf{W}h_{t-1} + \mathbf{U}x_t)$$



- 활성화 함수 하이버볼릭 탄젠트hyperbolic tangent
  - ReLU를 사용하는 경우 되먹임 구조 때문에 신호가 지나치게 커지는 오버플로우overflow가 발생, -1< tanh() < 1
  - Sigmoid의 사라지는 기울기vanishing gradient 문제 해결

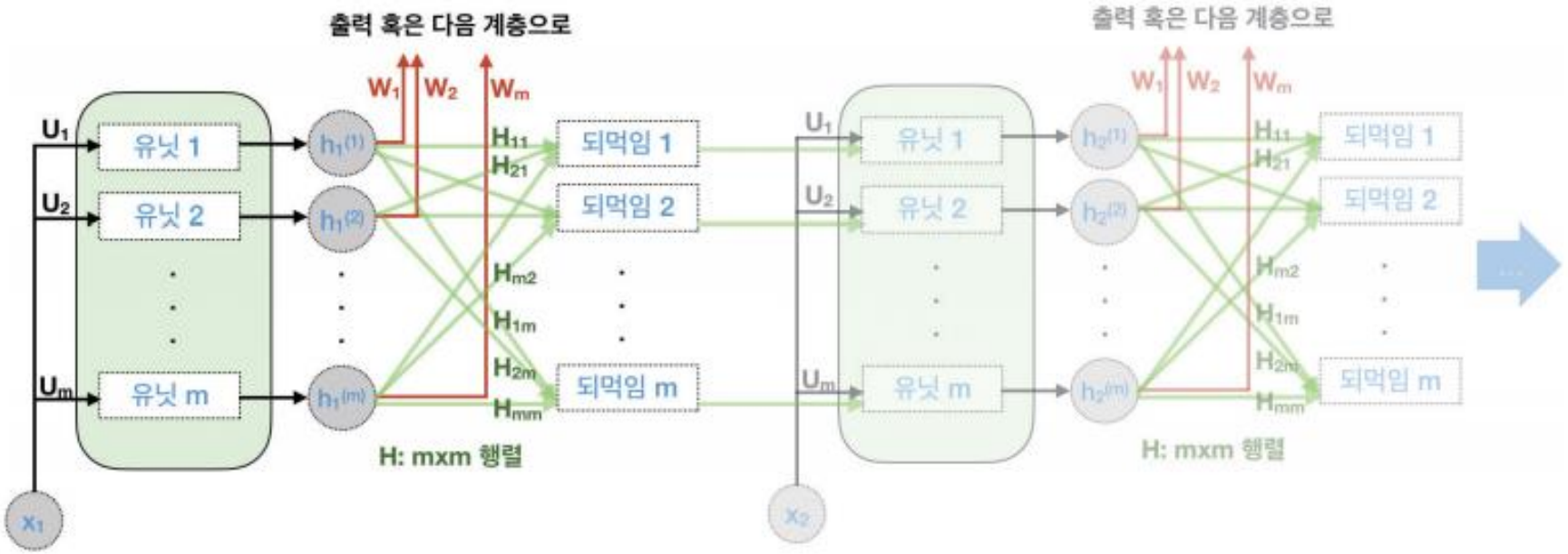
❖ 유닛unit

- 하나의 셀 내에서 병렬적으로 구성



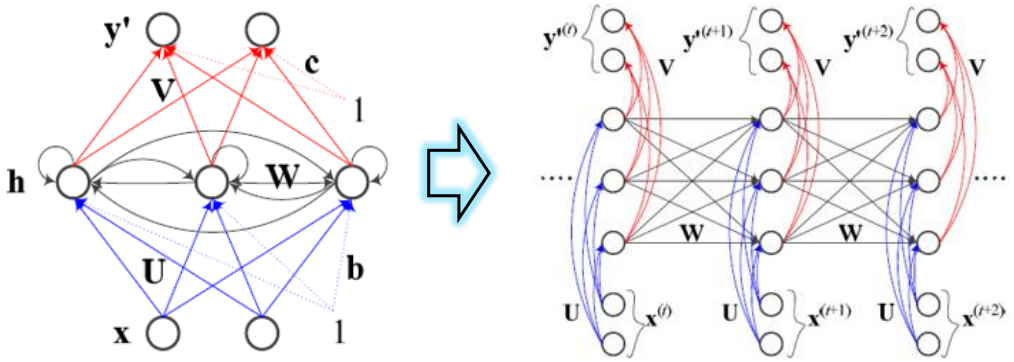
순환 신경망 셀의 구조와 유닛 (2)

- m 개의 unit을 가진 RNN
  - 입력 데이터를 유닛에 전달할 때 곱해지는 연결강도 **U:** m개
  - 각 유닛의 값이 출력으로 전달될 때 곱해지는 연결 강도 **W:** m개
  - 연결강도 **H**는 모두 $m \times m$개의 성분을 가진 행렬
  - Parameter 수: m x m + 2m = m(m+2)

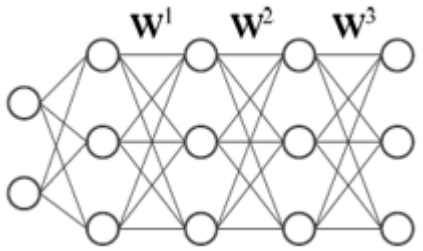


순환 신경망의 학습

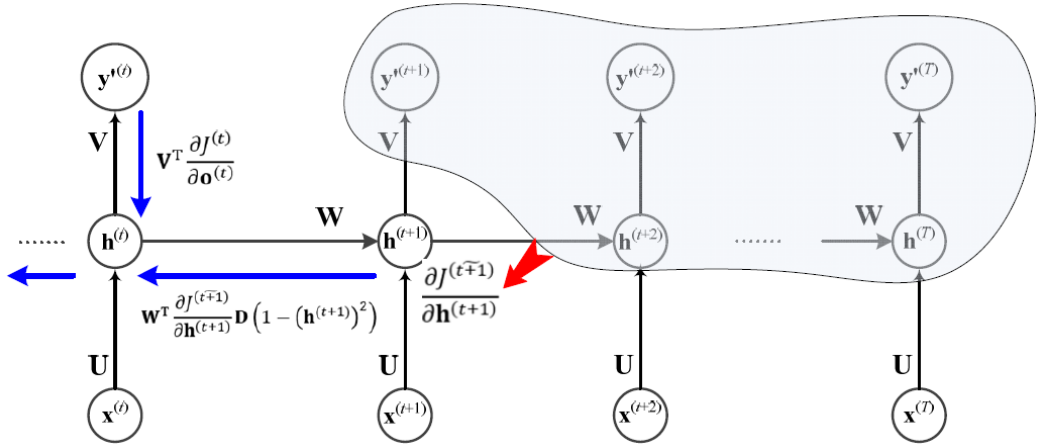
- BPTT(Backpropagation Through Time)
  - Barack A. Pearlmutter, 1995



(a) RNN



(b) MLP



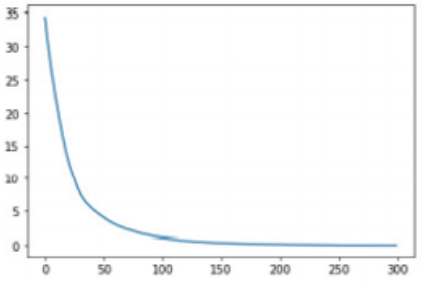
$$
\begin{cases}
\dfrac{\partial J}{\partial \mathbf{V}} = \displaystyle\sum_{t=1}^{T} \frac{\partial J^{(t)}}{\partial \mathbf{o}^{(t)}} \mathbf{h}^{(t)\mathrm{T}} \\
\dfrac{\partial J}{\partial \mathbf{W}} = \displaystyle\sum_{t=1}^{T} \mathbf{D}\left(1-\left(\mathbf{h}^{(t)}\right)^2\right) \frac{\partial J^{(\tilde{t})}}{\partial \mathbf{h}^{(t)}} \mathbf{h}^{(t-1)\mathrm{T}} \\
\dfrac{\partial J}{\partial \mathbf{U}} = \displaystyle\sum_{t=1}^{T} \mathbf{D}\left(1-\left(\mathbf{h}^{(t)}\right)^2\right) \frac{\partial J^{(\tilde{t})}}{\partial \mathbf{h}^{(t)}} \mathbf{x}^{(t)\mathrm{T}} \\
\dfrac{\partial J}{\partial \mathbf{c}} = \displaystyle\sum_{t=1}^{T} \frac{\partial J^{(t)}}{\partial \mathbf{o}^{(t)}} \\
\dfrac{\partial J}{\partial \mathbf{b}} = \displaystyle\sum_{t=1}^{T} \mathbf{D}\left(1-\left(\mathbf{h}^{(t)}\right)^2\right) \frac{\partial J^{(\tilde{t})}}{\partial \mathbf{h}^{(t)}}
\end{cases}
$$

텐서플로를 이용하여 단순 RNN 모델 만들기 (1)

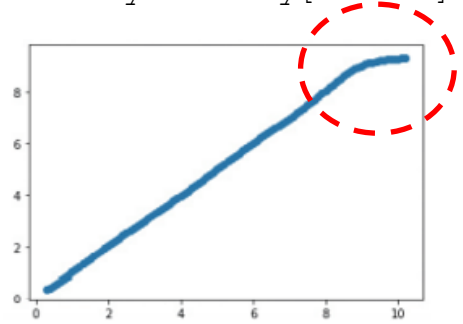
- ❖ 프로그램
  - https://colab.research.google.com/drive/1m2OM3Vpgx_dn1Pzua7N1KqJ_nvVLN5nQ
- ❖ Sequence data
  - 입력: [[0.0] [0.1] [0.2]], [[0.1] [0.2] [0.3]], ..., [[9.9] [10.0] [10.1]], 100개
  - 출력: 0.3, 0.4, ..., 10.2
- ❖ Keras SimpleRNN model1 (unit=20, epoch=300)

```
import tensorflow as tf
# units는 SimpleRNN 레이어에 있는 뉴런의 수
# return_sequences는 출력으로 시퀀스 전체를 출력할지 묻는 옵션
# input_shape [3, 1]에서 3는 timesteps, 1은 입력차원
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units = 20, return_sequences=False,
                              input_shape=[3, 1]),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer ='adam', loss = 'mse')
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_16 (SimpleRNN)    (None, 20)                440
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 21
=================================================================
Total params: 461
Trainable params: 461
Non-trainable params: 0
_________________________________________________________________
```



history.history['loss']



scatter(Y, y_hat)

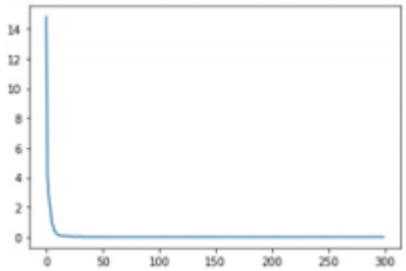
텐서플로를 이용하여 단순 RNN 모델 만들기 (2)

❖ Model2

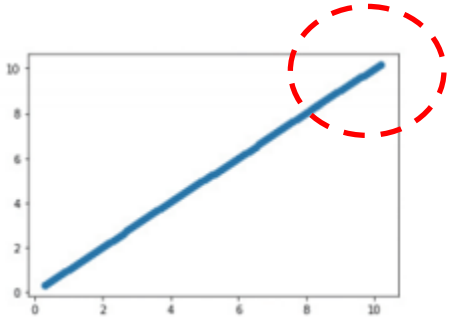
- unit=256, epoch=300

```
model256 = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units = 256, return_sequences=False,
                              input_shape=[3, 1]),
    tf.keras.layers.Dense(1)
])

model256.compile(optimizer ='adam', loss = 'mse')
model256.summary()
history = model256.fit(X, Y, epochs=300)
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])

y_hat = model256.predict(X)
plt.scatter(Y, y_hat)
```

```
simple_rnn_1 (SimpleRNN)     (None, 256)               66048
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 66,305
Trainable params: 66,305
```



history.history['loss']



scatter(Y, y_hat)

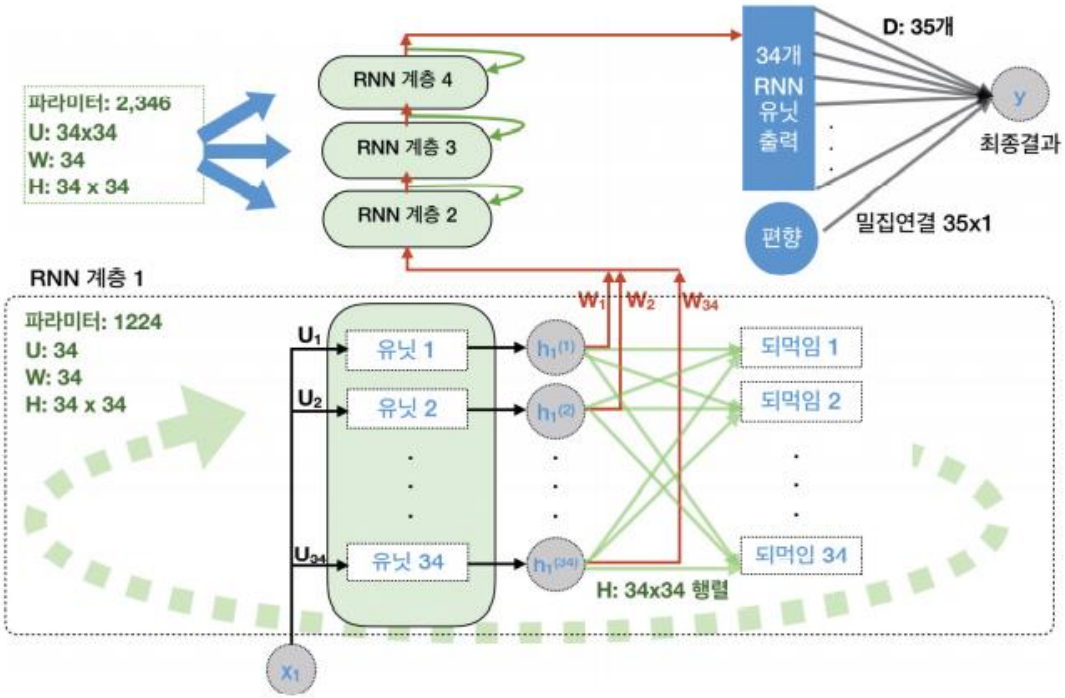
텐서플로를 이용하여 단순 RNN 모델 만들기 (3)

❖ 다층 RNN

```
model_multilayer = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units = 34, input_shape=[3, 1],
                                          return_sequences=True),
    tf.keras.layers.SimpleRNN(units = 34, return_sequences=True),
    tf.keras.layers.SimpleRNN(units = 34, return_sequences=True),
    tf.keras.layers.SimpleRNN(units = 34),
    tf.keras.layers.Dense(1) ])

model_multilayer.compile(optimizer ='adam', loss = 'mse')
history = model_multilayer.fit(X, Y, epochs=300)
```

Layer (type)	Output Shape	Param #
simple_rnn_7 (SimpleRNN)	(None, 3, 34)	1224
simple_rnn_8 (SimpleRNN)	(None, 3, 34)	2346
simple_rnn_9 (SimpleRNN)	(None, 3, 34)	2346
simple_rnn_10 (SimpleRNN)	(None, 34)	2346
dense_4 (Dense)	(None, 1)	35

Total params: 8,297
Trainable params: 8,297
Non-trainable params: 0

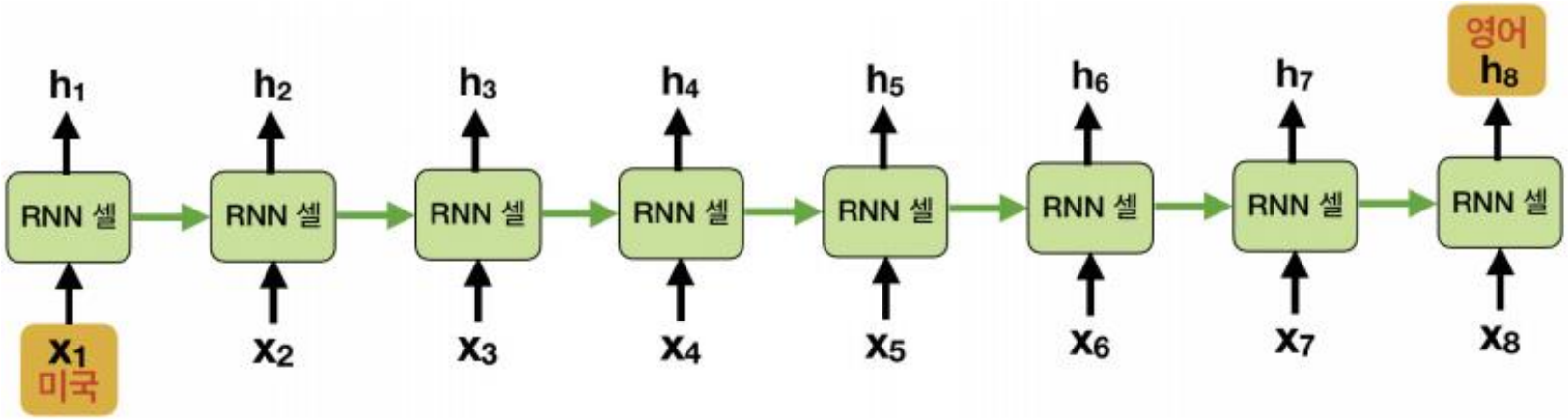


- 계층1 (U:34, W:34, H:34x34)
- 계층2, 3, 4 (U:34x34, W:34, H:34x34)
- 출력계층(W:34, b:1)

256개 유닛을 사용한 모델보다는 예측 능력이 떨어지지만, 파라미터 수를 크게 늘리지 않고도 성능이 대폭 개선된 것을 확인할 수 있음

장기 의존성 문제와 RNN의 한계 (1)

- 장기 의존성long-term dependency 문제
  - 연관성이 있는 두 단어가 멀리 떨어져 있을 경우 단순 RNN은 이 두 단어의 연관성 정보를 잘 전달할 수 없다
  - 예)
    - "남해 바다는 정말 푸르다." (O)
    - 나는 미국 텍사스에서 태어났어. 그리고 10살 무렵 한국으로 이사를 왔지. 한국에서 우리말을 열심히 익혔지만, 쉽지 않았어. 그래서 나는 아직 우리말보다는 영어를 잘해!" (X)



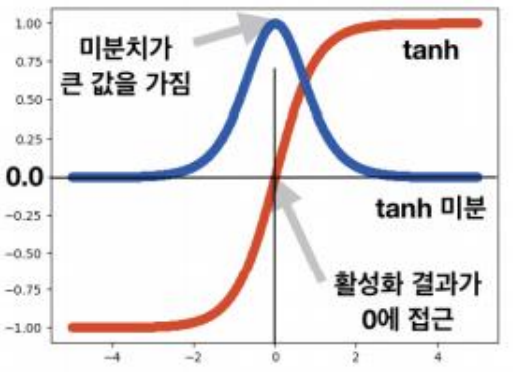
장기 의존성 문제와 RNN의 한계 (2)

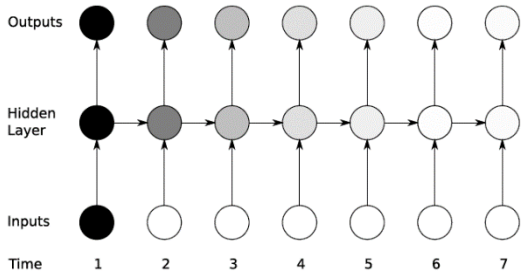
- 장기 의존성long-term dependency 원인
  - 활성화함수 tanh() 사용
    - $h_t = \tanh(Wh_{t-1} + Ux_t) = \tanh(f(h_{t-1}, x_t))$

      $h_{t-2} = \tanh(f(h_{t-3}, x_{t-2}))$

      $h_{t-1} = \tanh(f(\tanh(f(h_{t-3}, x_{t-2})), x_{t-2}))$

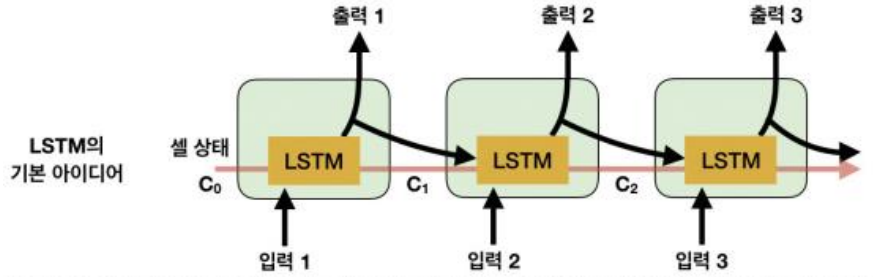
      $h_t = \tanh(f(\tanh(f(\tanh(f(h_{t-3}, x_{t-2})), x_{t-2}), x_t)))$
    - 원점을 기준으로 대칭이며 -1에서 1 사이의 값을 가짐
    - 미분치가 시그모이드 함수에 비해 훨씬 더 크기 때문에 기울기를 전달하는 역전파 과정에서 sigmoid에 비해 훨씬 더 좋은 특성을 보임
  - BPTT: 오차는 tanh()의 미분을 통해 역으로 전파
    - 미분값은 0~1이기 때문에 특별한 증폭 연산을 거치지 않을 경우 시간 t에서 멀어질수록(장기의존성) 오차 값이 사라지는 문제를 완전히 해결할 수는 없음
    - **단순 RNN은 사라지는 경사값 문제를 피하기 어려움**

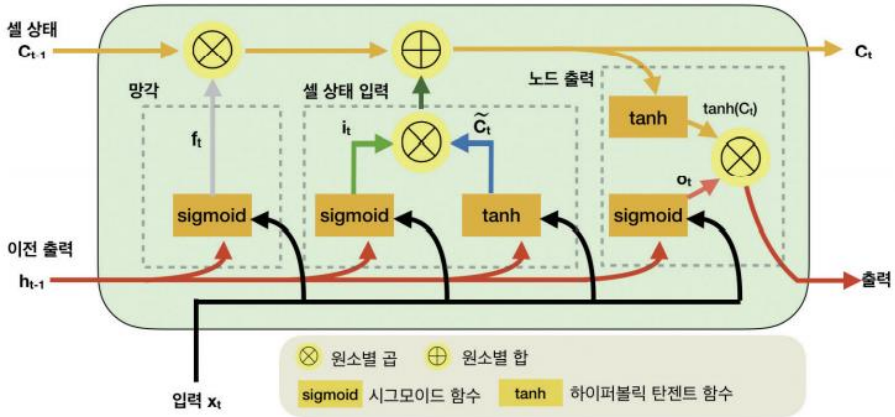


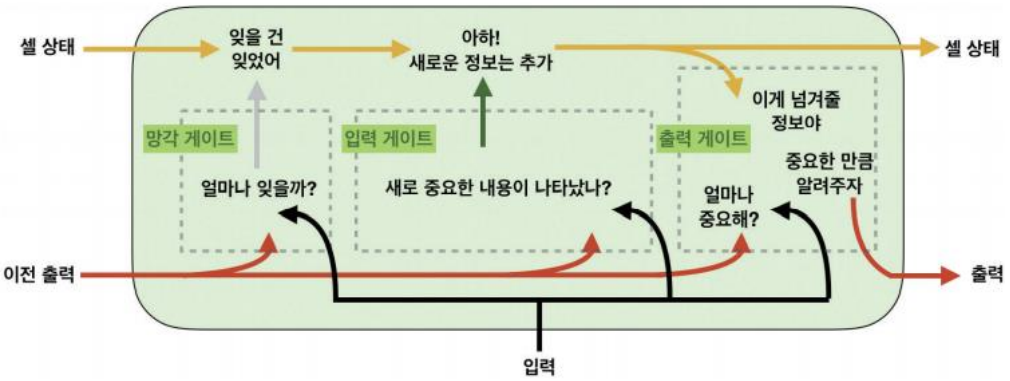


장기 의존성 문제를 해결하는 LSTM (1)

- LSTM long short term memory
  - 순환의 대상으로 출력만을 되먹임시키는 것이 아니라, 셀 상태를 같이 전달
  - 게이트gate : 셀 상태 정보를 변경하는 구조
    - 망각 게이트forget gate
    - 입력 게이트input gate
    - 출력 게이트output gate



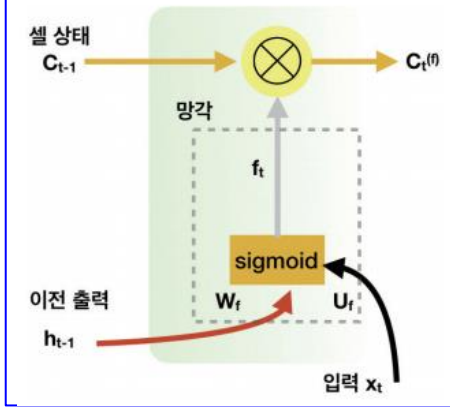




장기 의존성 문제를 해결하는 LSTM (2)

❖ 망각 게이트forget gate

- 정보를 얼마나 잊어야 할지를 결정

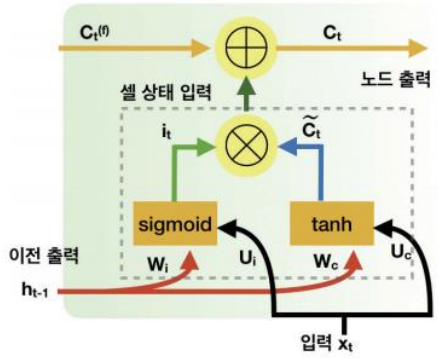


$$f_t = \sigma(\mathbf{W}_f h_{t-1} + \mathbf{U}_f x_t)$$

$C_t^{(f)}$: $[0, c_{t-1}]$의 범위

❖ 입력 게이트input gate

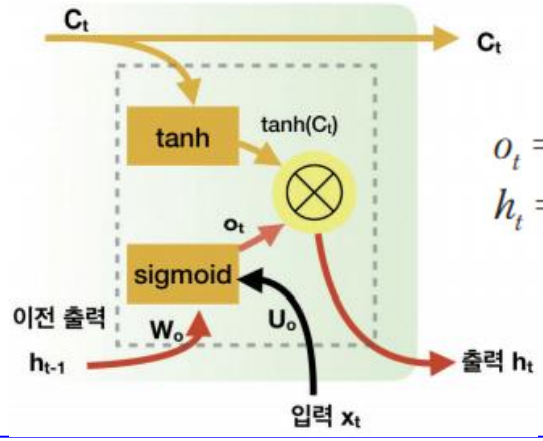
- 후보값($\tilde{C}_t$)을 얼마나 전달할지(i_t) 결정



$$i_t = \sigma(\mathbf{W}_i h_{t-1} + \mathbf{U}_i x_t)$$
$$\tilde{c}_t = \tanh(\mathbf{W}_c h_{t-1} + \mathbf{U}_c x_t)$$
$$c_t = c_t^{(f)} \oplus i_t \otimes \tilde{c}_t$$

❖ 출력 게이트output gate

- o_t 만큼 셀 상태를 출력

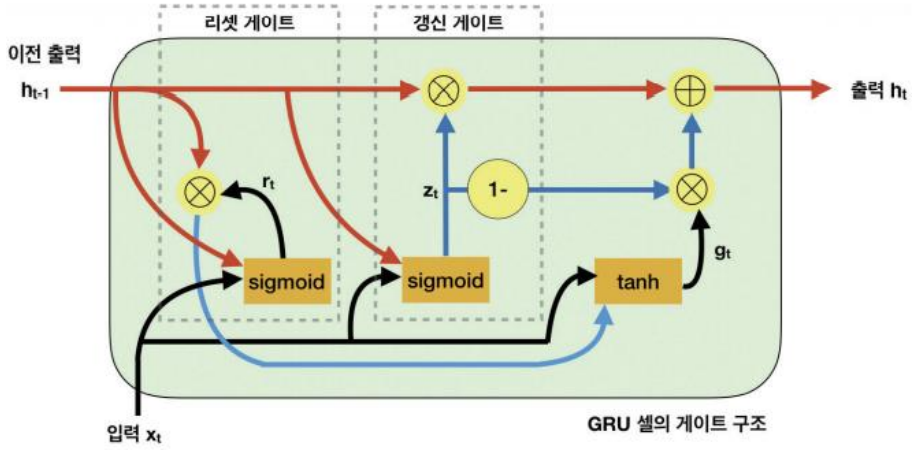


$$o_t = \sigma(\mathbf{W}_o h_{t-1} + \mathbf{U}_o x_t)$$
$$h_t = o_t \otimes \tanh(c_t)$$

GRU

❖ GRU gated recurrent unit

- LSTM의 셀 상태 cell state 정보는 장기 기억을 위한 상태 정보
- LSTM은 셀의 중요한 입력값을 인식해서 장기 상태에 저장하고 필요한 기간 동안 이를 보존하는 기능과 함께 필요할 때 이를 추출할 수 있는 능력을 학습 => 파라미터의 수가 많음
- LSTM을 좀 더 단순화 시킨 구조 (2014, 조경현)



$$z_t = \sigma(\mathbf{W}_u h_{t-1} + \mathbf{U}_u x_t)$$
$$r_t = \sigma(\mathbf{W}_r h_{t-1} + \mathbf{U}_r x_t)$$
$$g_t = \tanh(\mathbf{W}_g (h_{t-1} \otimes r_t) + \mathbf{U}_g x_t)$$
$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes g_t$$

- 리셋 gate: 이전 출력을 얼마나 제공할 것인지를 결정
- 갱신 gate(z_t): 1이면 망각게이트, 0이면 입력게이트 역할

RNN, LSTM, GRU 비교

❖ 프로그램

- https://colab.research.google.com/drive/1m2OM3Vpgx_dn1Pzua7N1KqJ_nvVLN5nQ

❖ Sequence

훈련용 데이터

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5] 1.6
[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6] 1.7
[0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7] 1.8
[0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8] 1.9
[0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9] 2.0
[0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ] 2.1
[0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1] 2.2
[0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2] 2.3
```

검증용 데이터

```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5] 2.6
[1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6] 2.7
[1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7] 2.8
[1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8] 2.9
```

❖ Simple RNN

```
simpleRNN_model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units = 10, return_sequences=False,
                              input_shape=[seq_len, 1]),
    tf.keras.layers.Dense(1)
])
```

❖ LSTM

```
LSTM_model = tf.keras.Sequential([
    tf.keras.layers.LSTM(units = 10, return_sequences=False,
                         input_shape=[seq_len, 1]),
    tf.keras.layers.Dense(1)
])
```

❖ GRU

```
GRU_model = tf.keras.Sequential([
    tf.keras.layers.GRU(units = 10, return_sequences=False,
                         input_shape=[seq_len, 1]),
    tf.keras.layers.Dense(1)
])
```

Program

❖ LAB10-1 비선형 시퀀스를 순환 신경망으로 예측

실습 목표

사인 곡선에서 일부분을 잘라 만든 시퀀스를 보고, 이 시퀀스 다음에 나타날 값을 예측하는 순환 신경망 모델을 만들어 보자.

힌트 선형 특성을 가진 시퀀스와 동일한 방법으로 훈련할 수 있다. 이 실습에서 단순 RNN와 LSTM, GRU의 예측 성능을 비교해 보자.

❖ LAB10-2 기억이 필요한 시퀀스 예측

실습 목표

사인 곡선에서 일부분을 잘라 만든 시퀀스의 각 요소 각각에 임의의 난수 인덱스를 부여하자. 이번에는 이 시퀀스의 다음 값을 예측하는 것이 아니라, 시퀀스의 각 요소들 가운데 짝수 인덱스를 가진 요소들의 평균값을 계산하는 모델을 만들어 보자.

힌트 이것은 단순히 시퀀스의 변화만 살펴본다고 해결할 수 있는 문제가 아니라, 시퀀스에서 짝수 인덱스를 부여받은 데이터들이 어떤 것이 있는지 따로 기억을 해야 하는 문제이다. 이러한 기억 능력의 측면에서 LSTM과 GRU가 더 나은 동작을 보이는지 확인해 보자.

참고 사이트

- 딥 러닝을 이용한 자연어 처리 입문
- https://wikidocs.net/45101