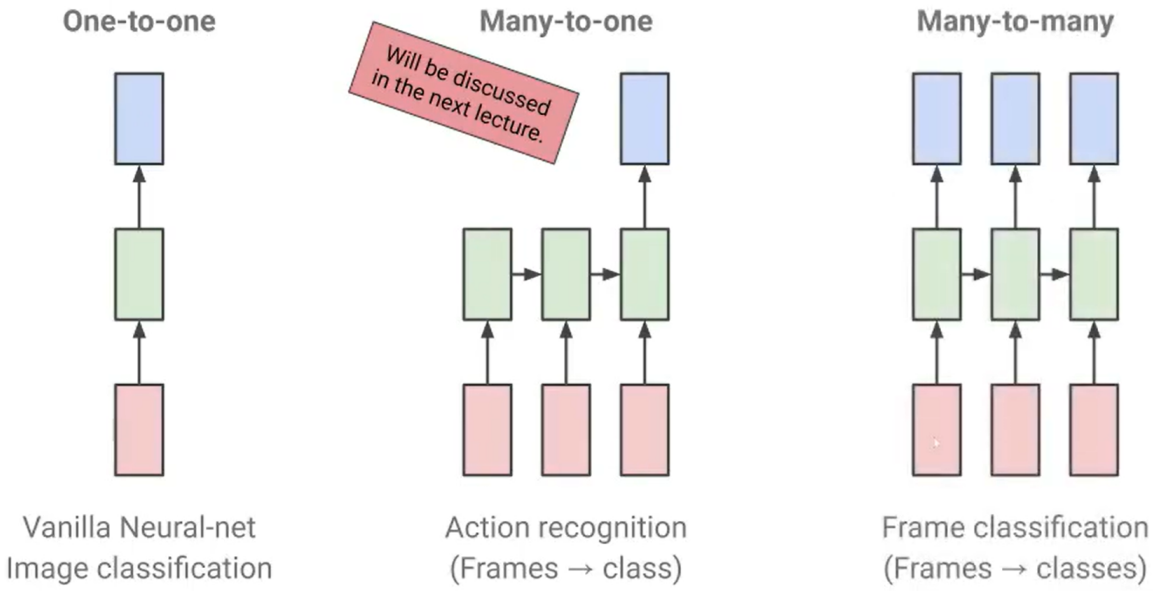
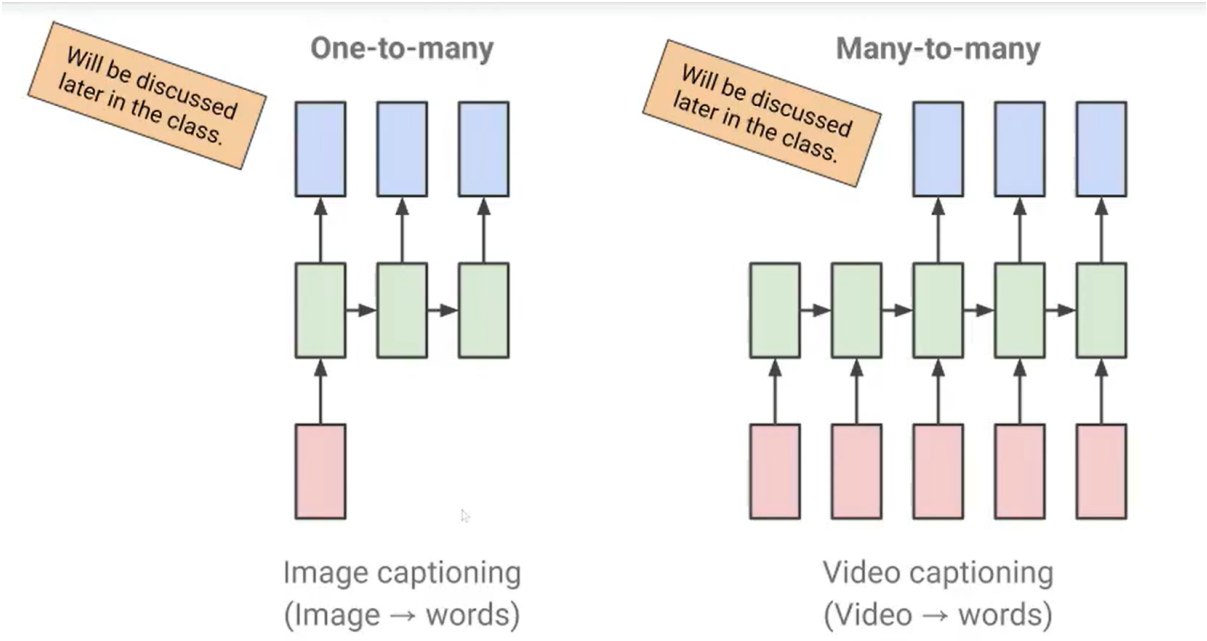
✅

Lecture 13

Types of Neural Network

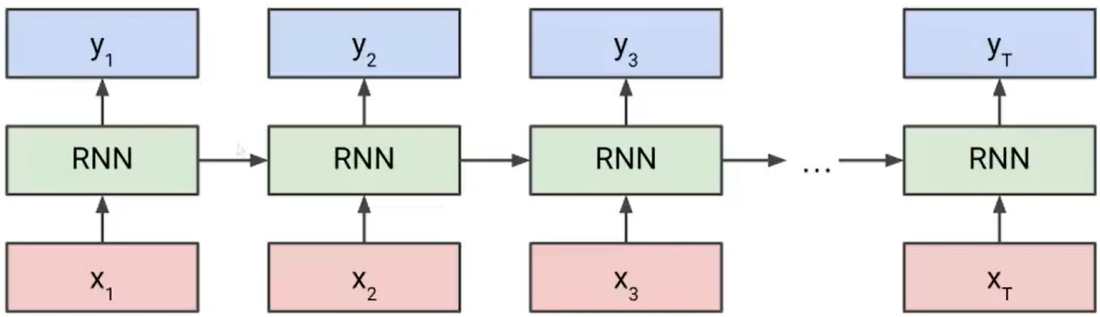
- One-to-one : input도 하나 output도 하나
- many-to-one : input은 여러개 (연속적인 데이터) output은 하나 (ex. 비디오)
- many-to-many : input도 output도 sequential함
- one-to-many : input은 하나 output은 여러 개 (이미지 한장에 대해 문장을 내놓는 작업)





Recurrent Neural Network

- RNN
  - input이 sequential하게 들어옴
  - 다음의 input이 들어오면 현재 state를 이용해 다음 state를 update시키고 그에 대한 output

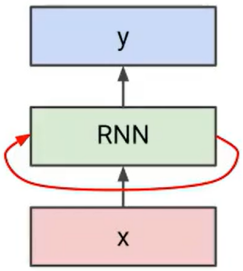


  - 각각의 RNN cell은 xi를 인풋으로 받고, hi-1로부터 hi층의 hidden state를 업데이트시켜 yi라는 output을 (선택적으로) 반환함
  - 이때 초기값은 initialize 필요
  - 식

- A sequence of vectors $\{x_1, x_2, ..., x_T\}$ is processed by applying a recurrence formula at every time step:

$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

- It is important to note that the **same function** and the **same set of parameters** are used at every time step.



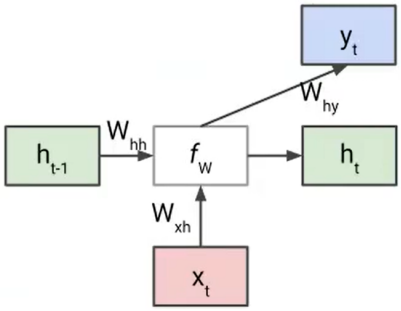
- Vanilla RNN

- Then what should we do inside the RNN cell?
  - One simple way is taking **linear transformations** (with W_{hh} and W_{xh}) on the two inputs (previous hidden state h_{t-1} and input x_t),
  - then take a **nonlinearity** before updating it as h_t.
  - For the output, we may put another **linear transformation** from h_t.

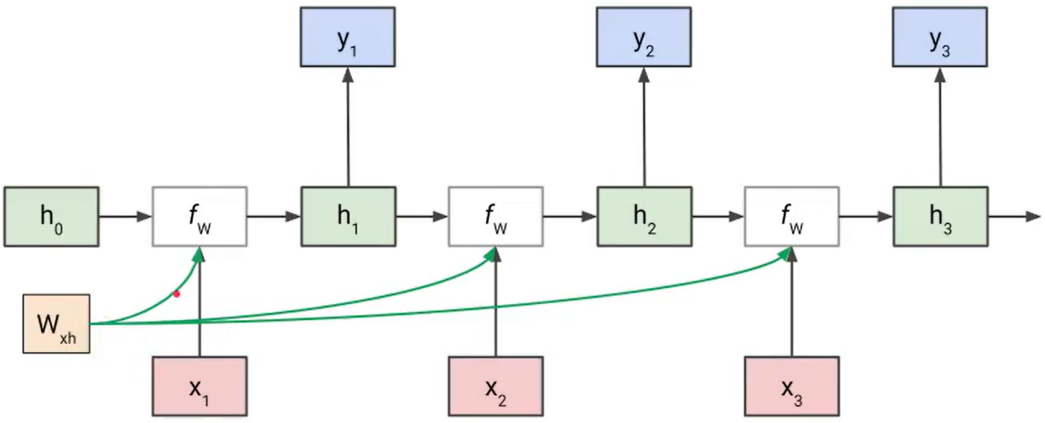
$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t$$

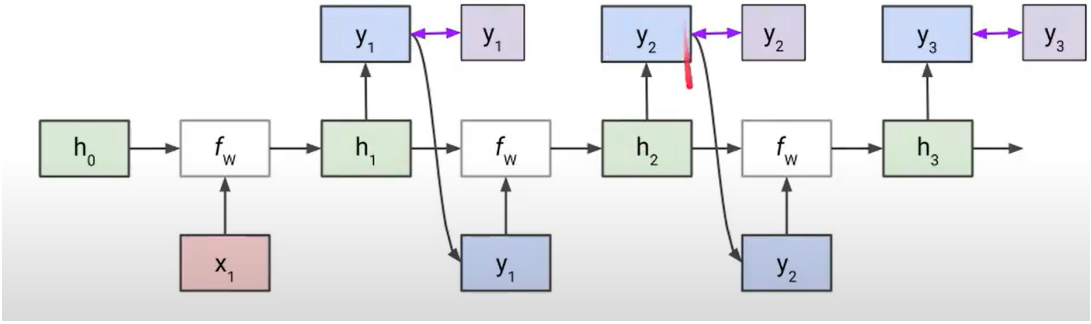


  - Weight은 모든 단계에서 항상 동일하다!! → 이렇지 않으면 RNN은 동작하지 못함
  - many-to-many인 경우



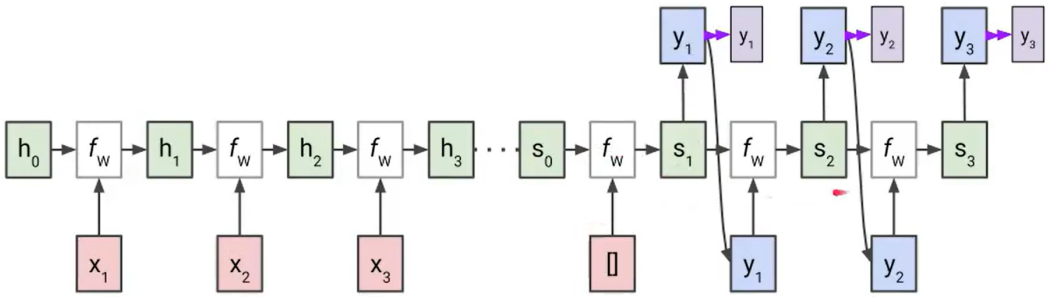
  - one-to-may인 경우

- What about one-to-many?
  - We still must input something at each step, given the formula: $\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$
  - **Autoregressive** input: For time series data, the lagged (autoregressive) values of the time series are used as inputs to a neural network.



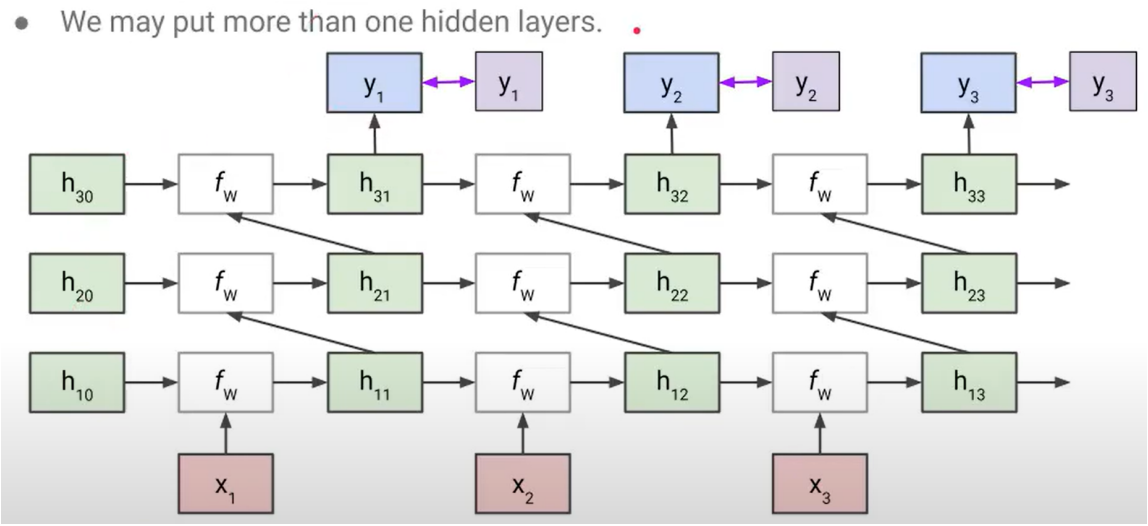
  - many-to-many인 경우

- Lastly, how to implement many-to-many?
  - Many-to-one as an **encoder**, then one-to-many as a **decoder**.
  - The input sequence is encoded as a single vector at the end of the encoder.
  - From this single vector, the decoder generates output sequence.
  - Called Sequence-to-sequence, or **seq2seq**.



- RNN trade-off
  - 장점
    - input이 아무리 커져도 모델 크기가 커지지 않음
    - 이론적으로는 오래 전의 정보들도 사용할 수 있음
  - 단점
    - parallelirzation이 잘 안됨
    - vanishing gradient 문제가 발생함 (→ 이걸 해결하는게 LSTM)
    - long-rage dependence가 발생해 마지막 데이터에 의존함

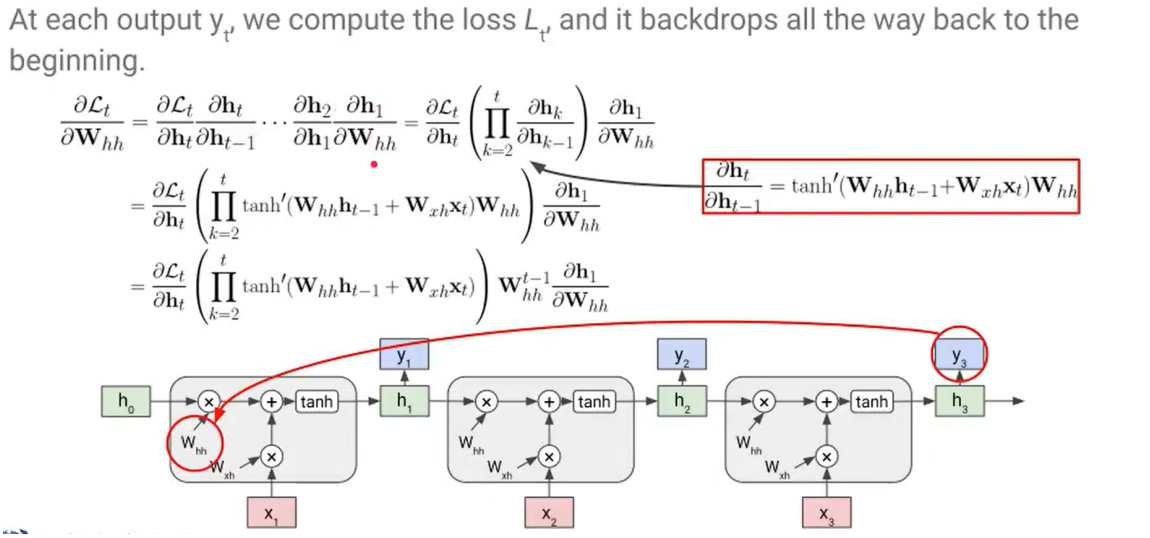
- Multi-layer RNN

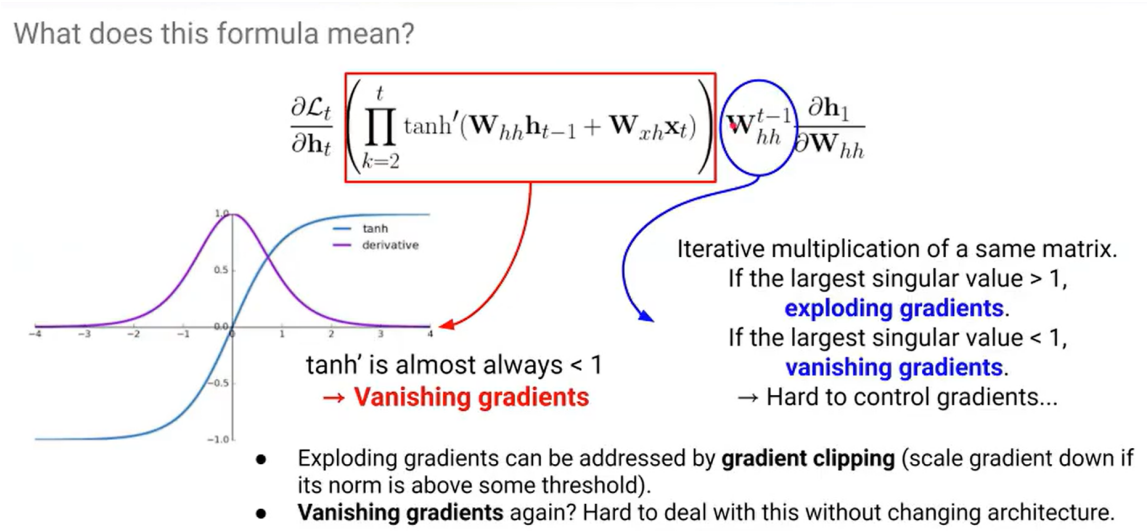


- RNN 사용하는 곳
  - image captioning
  - visual dialog
  - visual language navigation
  - 등등

Towards Modeling Longer Dependence

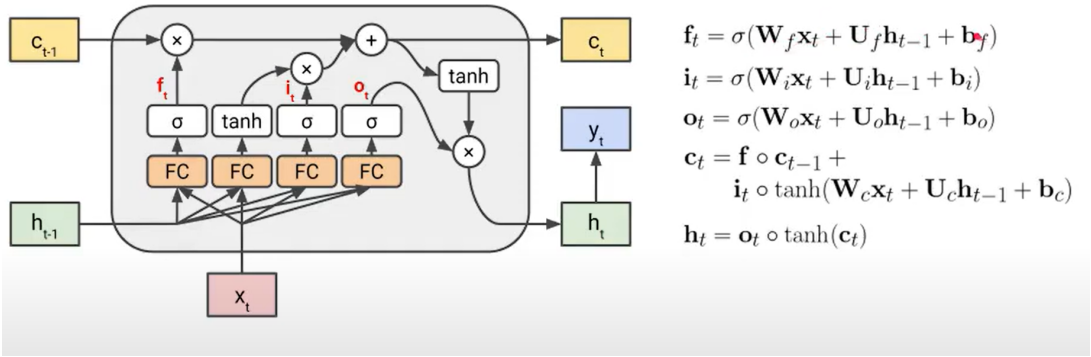
- gradient flow problem with Vanilla RNN





- fully-connected layer에서 문제가 발생했기 때문에 이를 없애고자 함

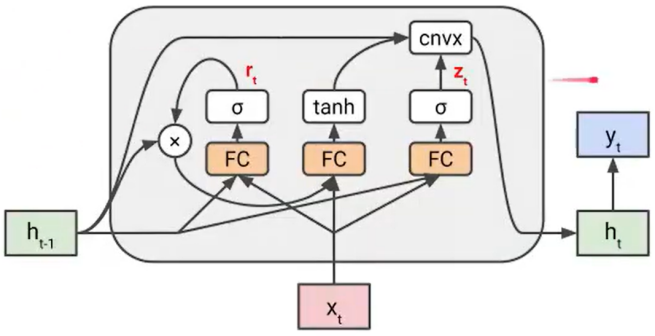
- Long Short Term Memory
  - c state : long term memory를 기억하는 layer
  - forget gate : 지금 input과 state을 기준으로 long term data를 지울지, 유지할지를 결정
  - input gate : input을 통제하는 게이트
  - output gate



  - 사용하는 이유
    - long-range information을 rnn보다 더 잘 보존한다

- Gated Recurrent Units (GRU)

- LSTM과 비슷한 모델이지만 구체적으로 게이트들이 계산하는 방식은 조금씩 다르다
- h가 가는 길에 두개의 way를 만들어서 하나는 weight, 다른 하나는 input에 의해 결정되도록



$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r)$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z)$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{b}_h)$$

https://arxiv.org/pdf/1406.1078.pdf