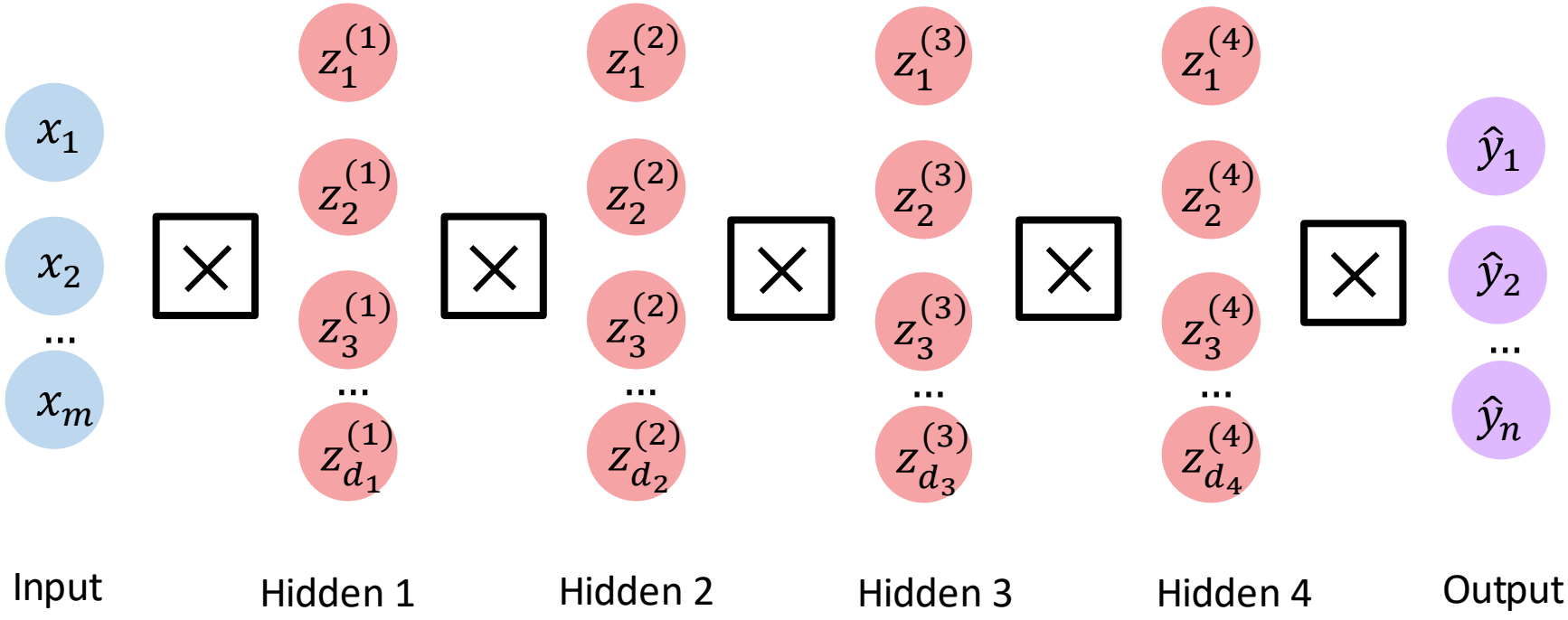
Sequence Modeling

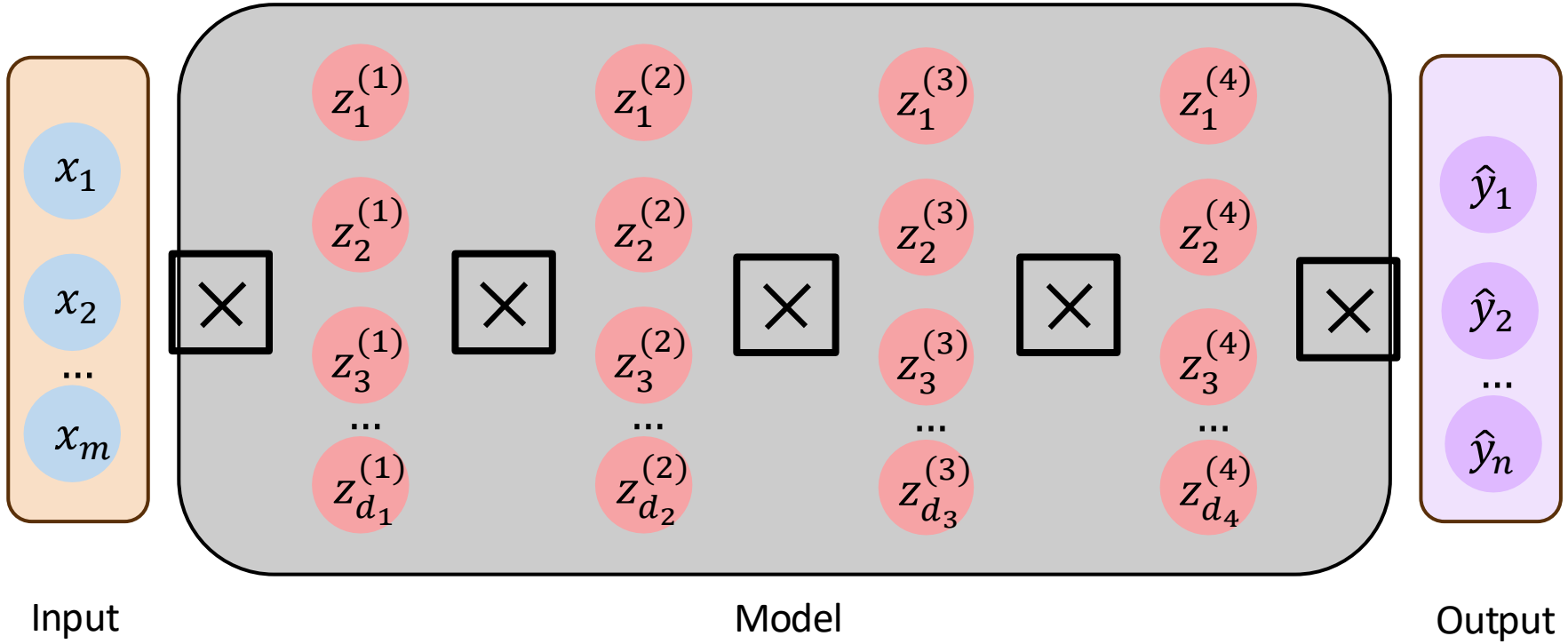
Qiang Sun

University of Toronto

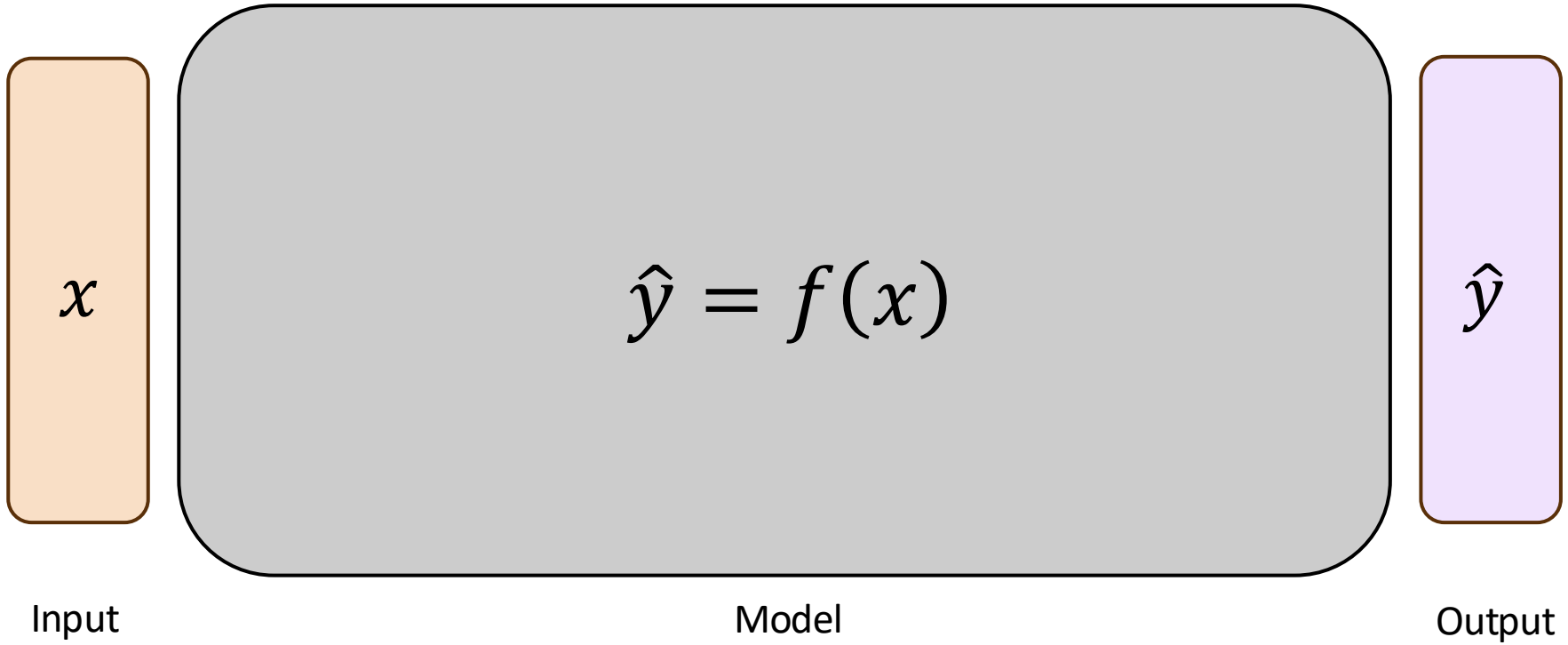
Fully-Connected (Dense) Deep Neural Network



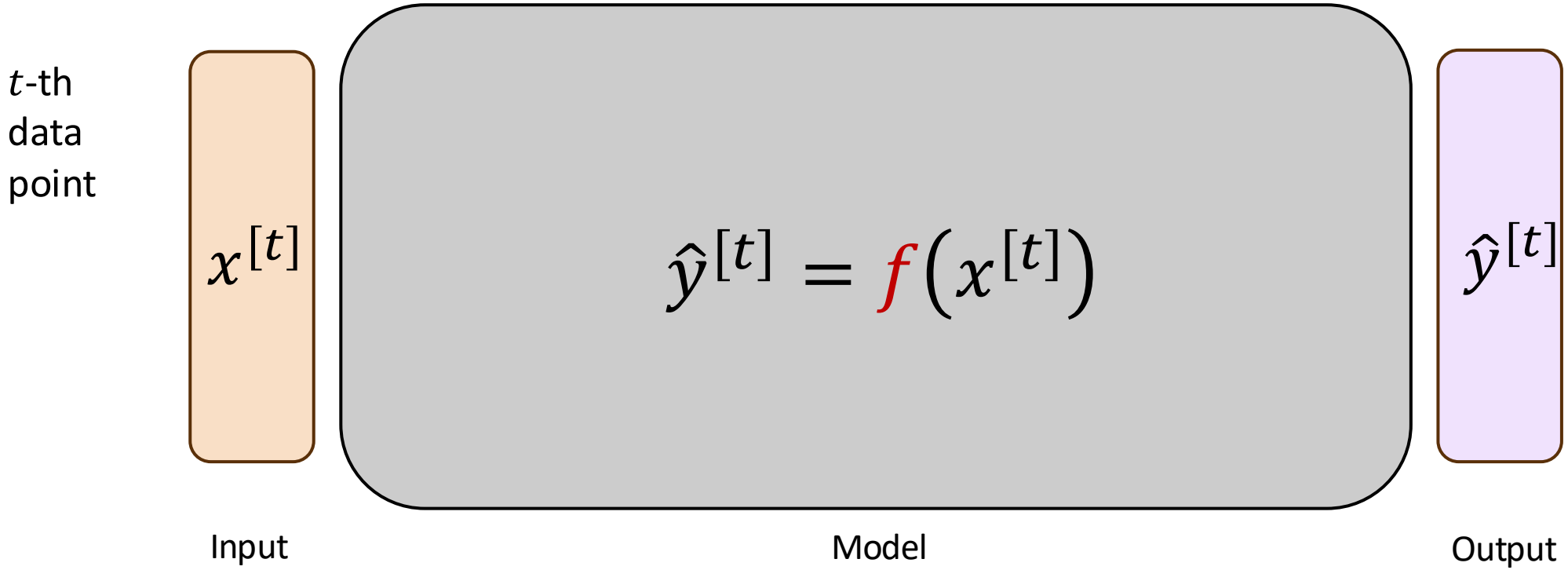
Fully-Connected (Dense) Deep Neural Network



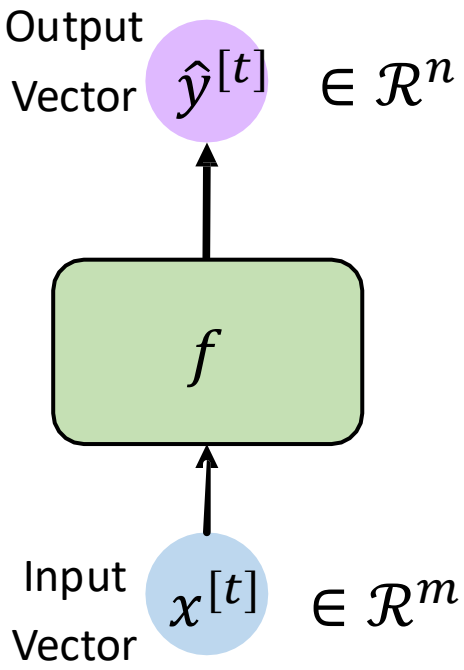
Fully-Connected (Dense) Deep Neural Network



Fully-Connected (Dense) Deep Neural Network

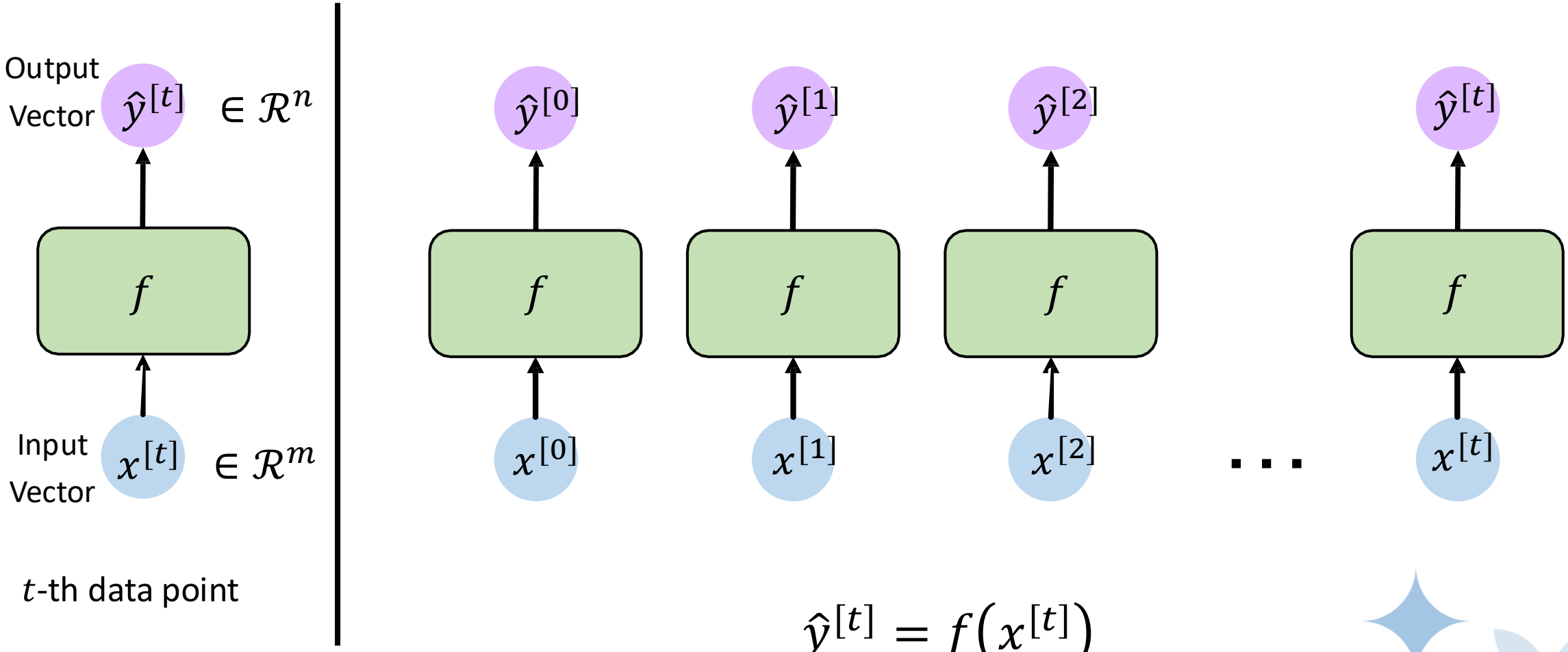


Sequence Model f



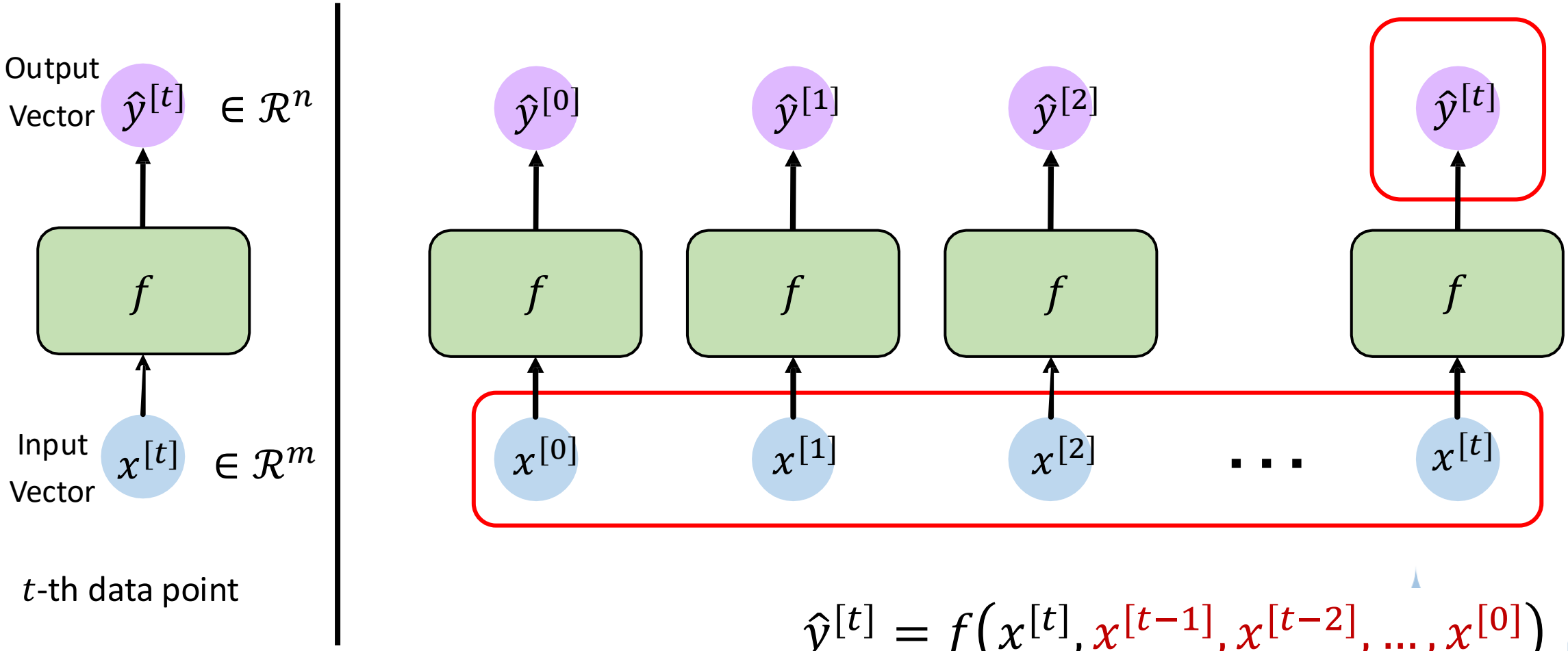
t-th data point

Sequence Model f



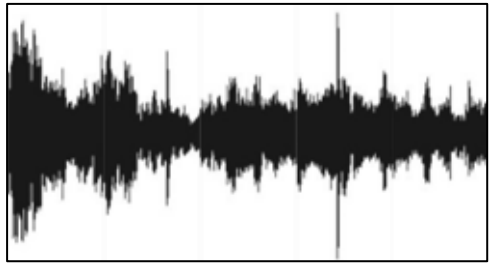
$$\hat{y}^{[t]} = f\left(x^{[t]}\right)$$

Sequence Model



$$\hat{y}^{[t]} = f\left(x^{[t]}, x^{[t-1]}, x^{[t-2]}, \ldots, x^{[0]}\right)$$

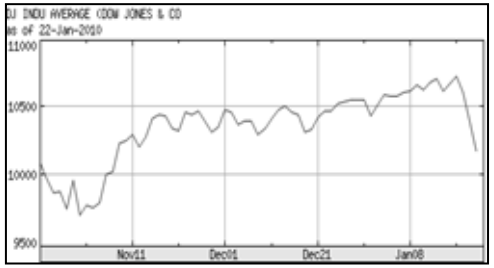
Sequence Modeling



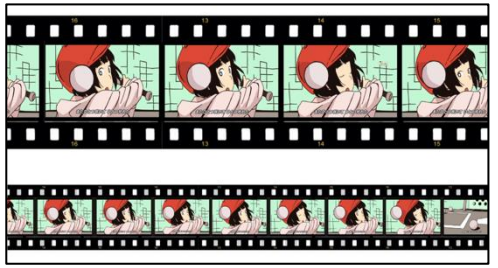
Voice

"Why do we care about sequences?"

Text



Stock price

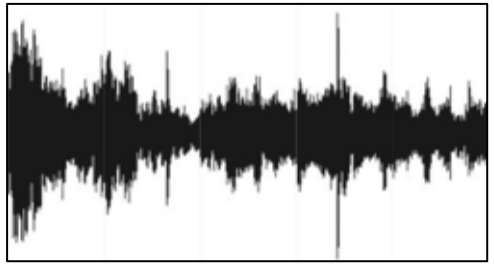


Video

- Order of the input matters
- Valuable information can be of variable (potentially infinite) length

$$x^{[0]} \rightarrow x^{[1]} \rightarrow x^{[2]} \rightarrow \ldots \rightarrow x^{[t]}$$

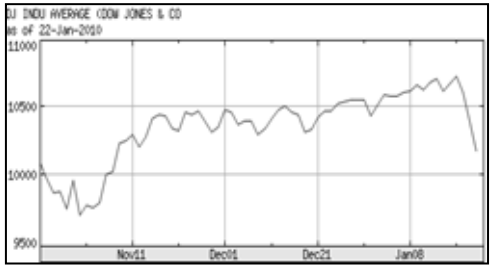
Sequence Modeling



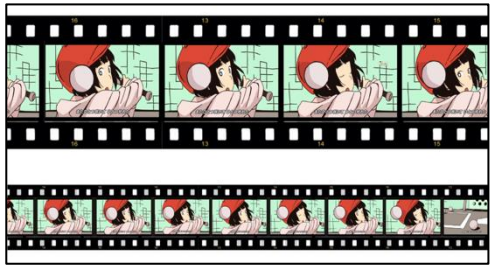
Voice

"Why do we care about sequences?"

Text



Stock price



Video

- Order of the input matters
- Valuable information can be of variable (potentially infinite) length

$$x^{[0]} \rightarrow x^{[1]} \rightarrow x^{[2]} \rightarrow \ldots \rightarrow x^{[t]}$$

Sequence Modeling Tasks

Sentiment Classification

Music Generation

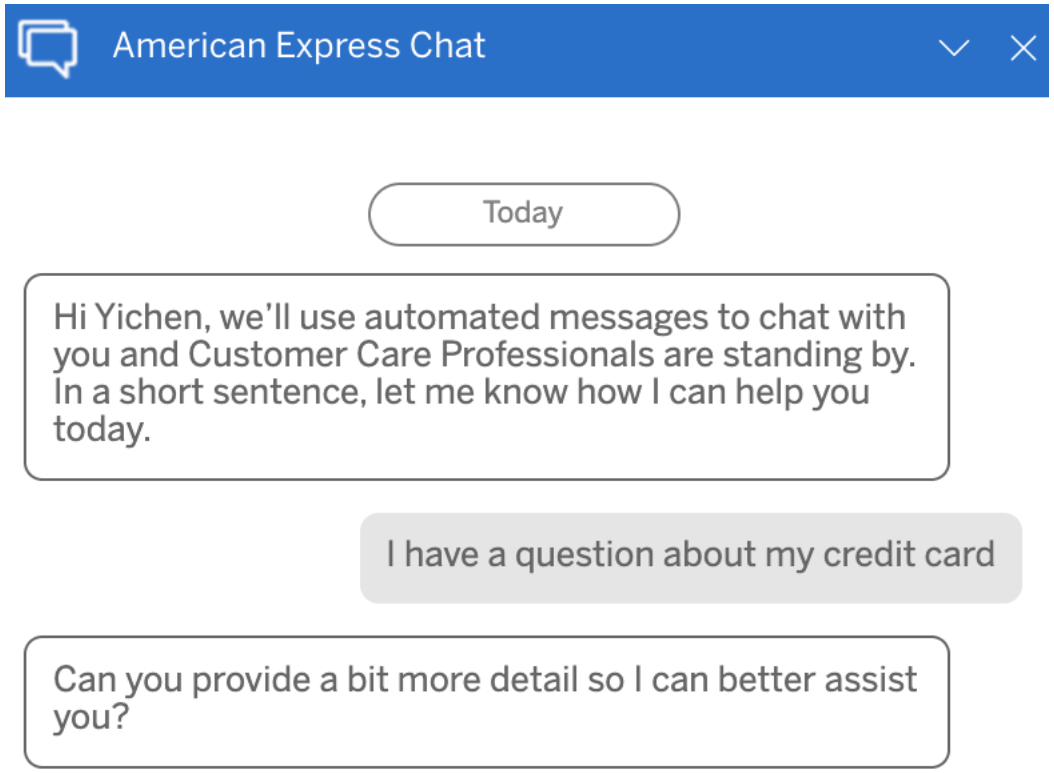
Sequence Modeling Tasks

Automated Speech Recognition

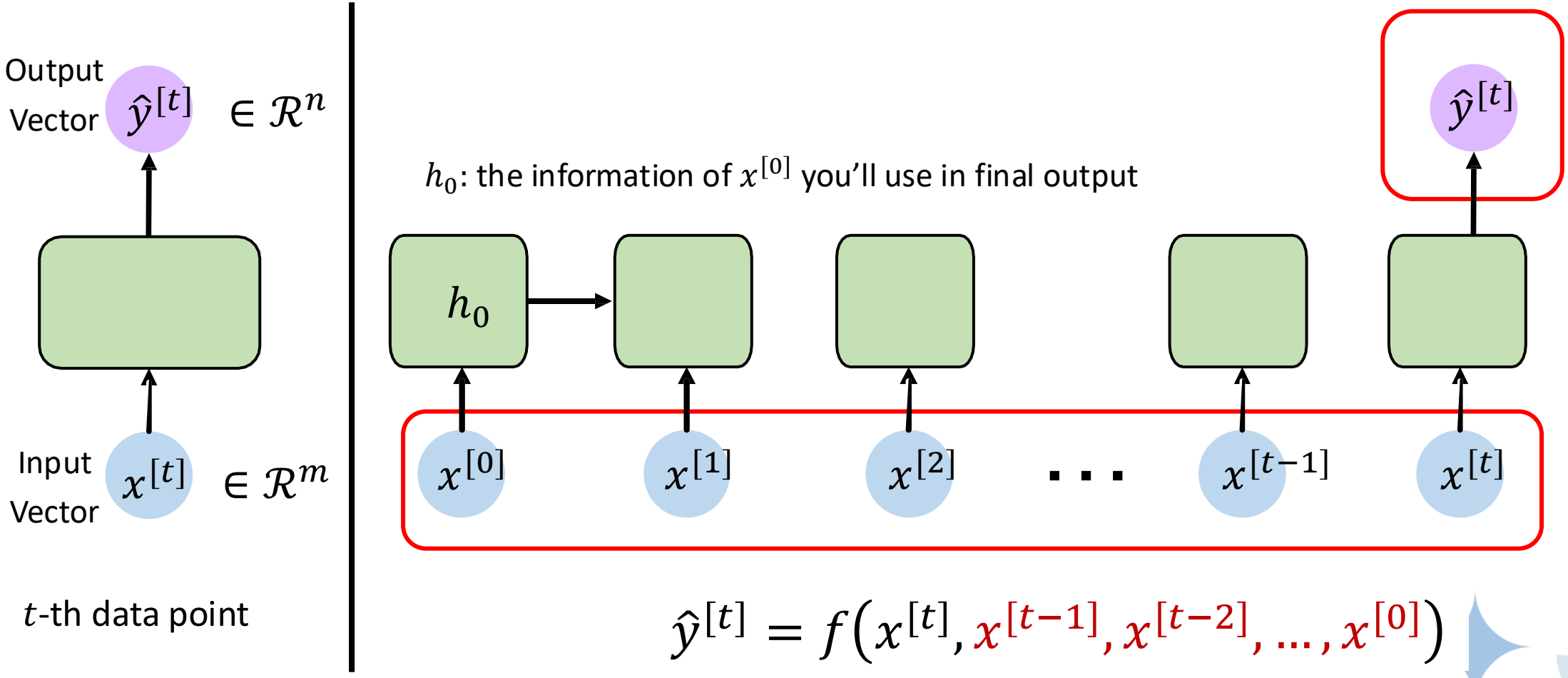
Translating voice calls and video calls in 11 languages and instant messages.

https://www.skype.com/en/features/skype-translator/

Automated Chatbot

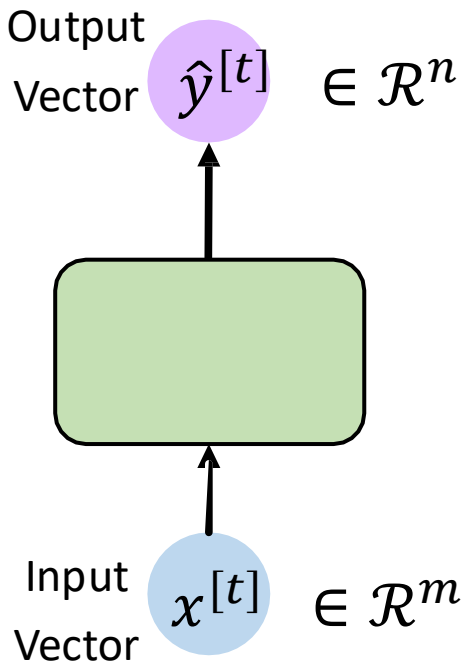


How do we model that?



$$\hat{y}^{[t]} = f\left(x^{[t]}, x^{[t-1]}, x^{[t-2]}, \ldots, x^{[0]}\right)$$

How do we model that?



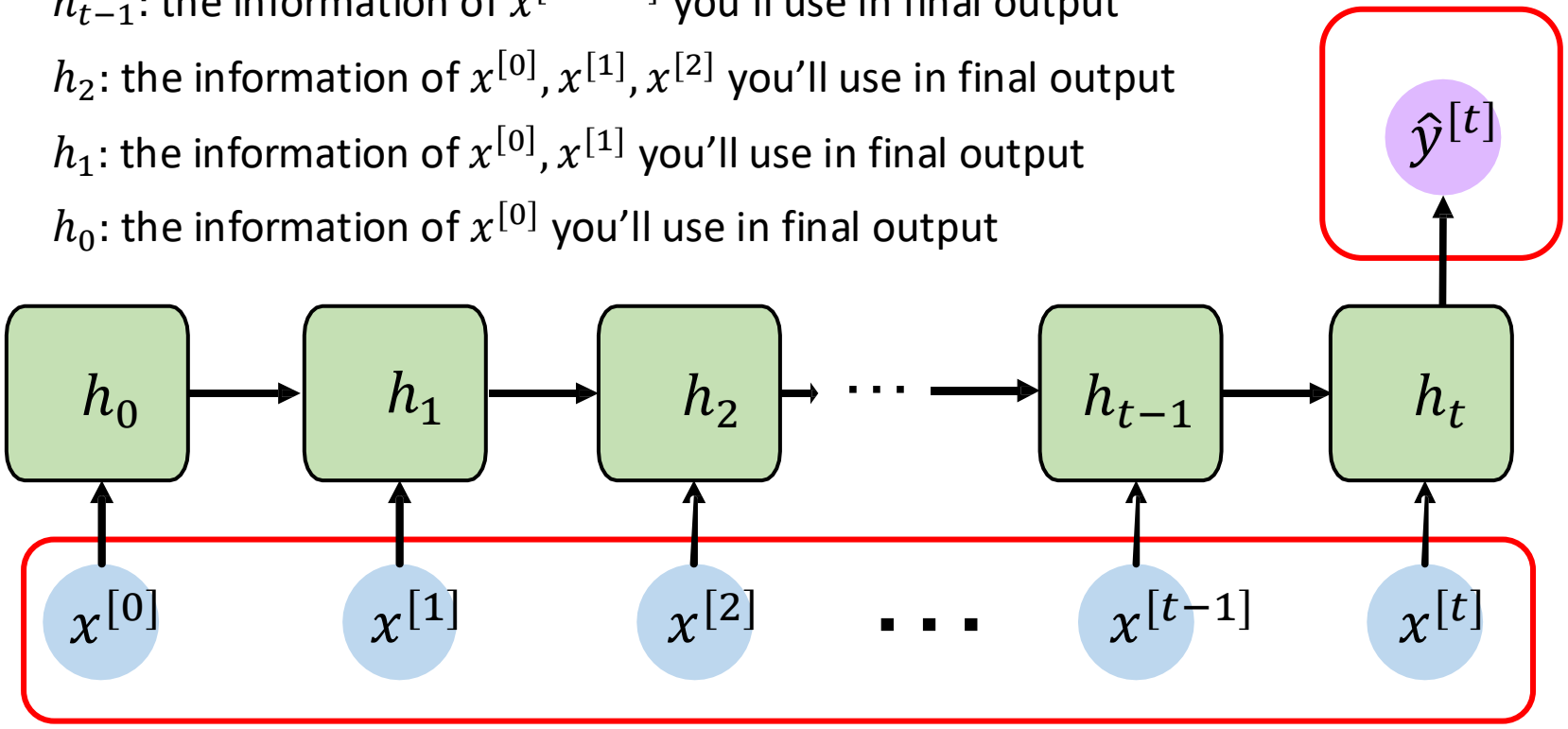
t-th data point

h_{t-1}: the information of $x^{[0\sim t-1]}$ you'll use in final output

h_2: the information of $x^{[0]}, x^{[1]}, x^{[2]}$ you'll use in final output

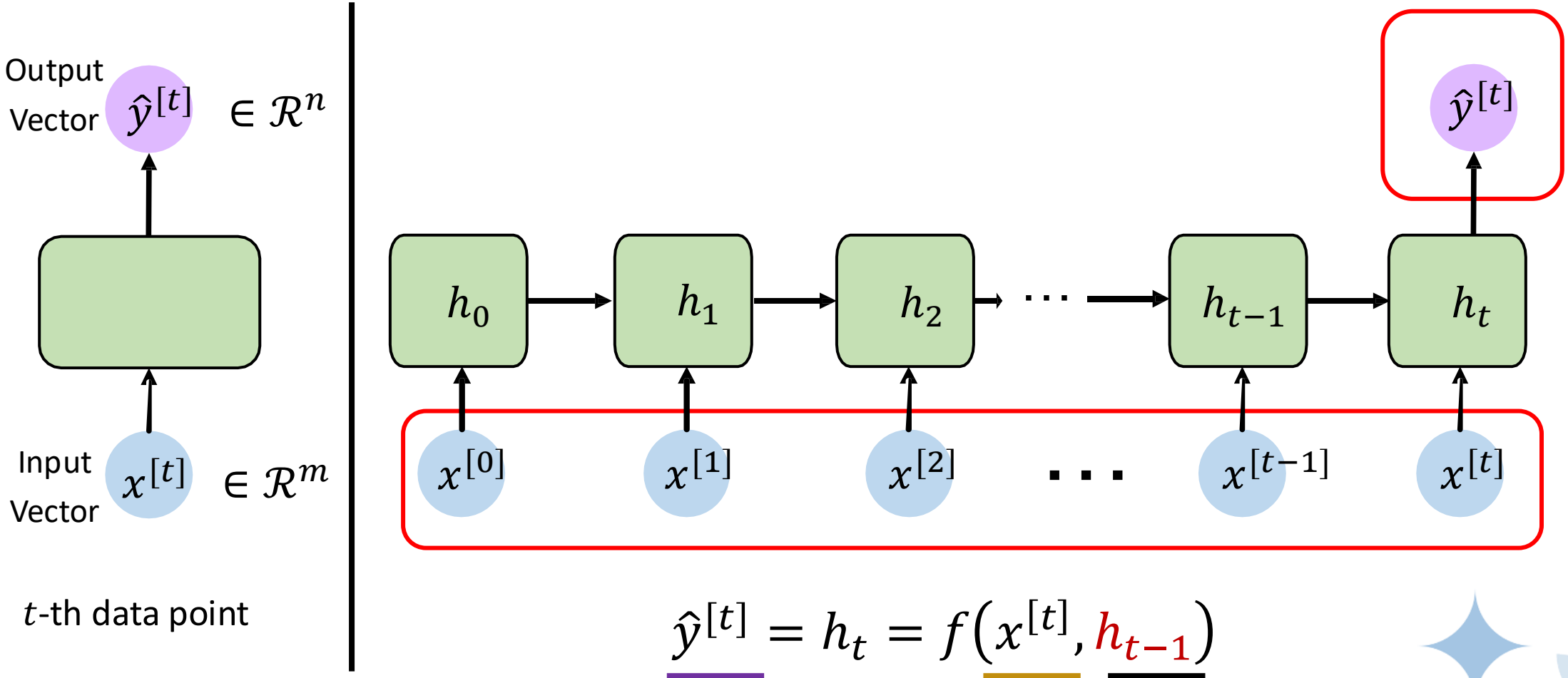
h_1: the information of $x^{[0]}, x^{[1]}$ you'll use in final output

h_0: the information of $x^{[0]}$ you'll use in final output



$$\hat{y}^{[t]} = f\left(x^{[t]}, x^{[t-1]}, x^{[t-2]}, \ldots, x^{[0]}\right)$$

How do we model that?



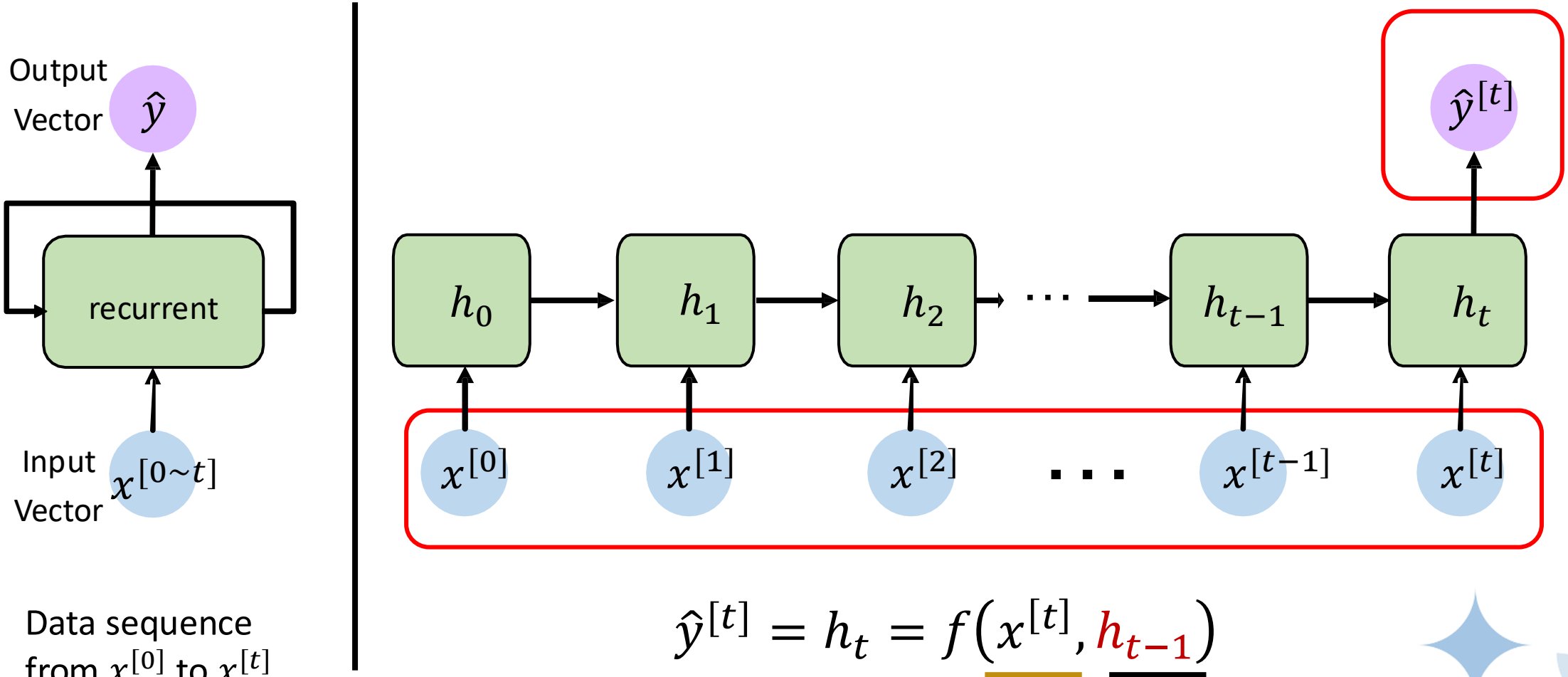
$$\hat{y}^{[t]} = h_t = f\left(x^{[t]}, h_{t-1}\right)$$

Current Output

Current Input

Past Memory

How do we model that?



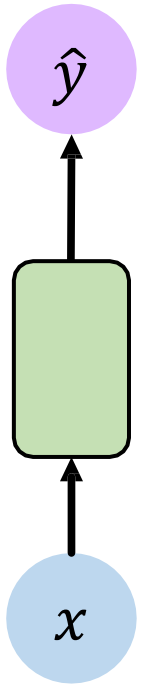
Data sequence from $x^{[0]}$ to $x^{[t]}$

$$\hat{y}^{[t]} = h_t = f\left(x^{[t]}, h_{t-1}\right)$$

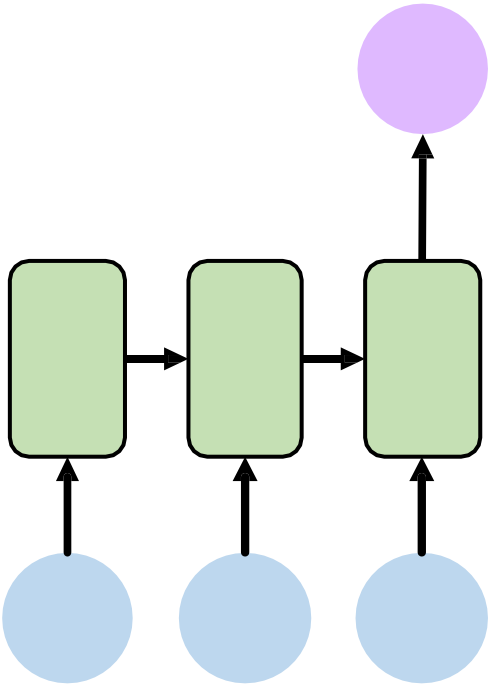
Current Input

Past Memory

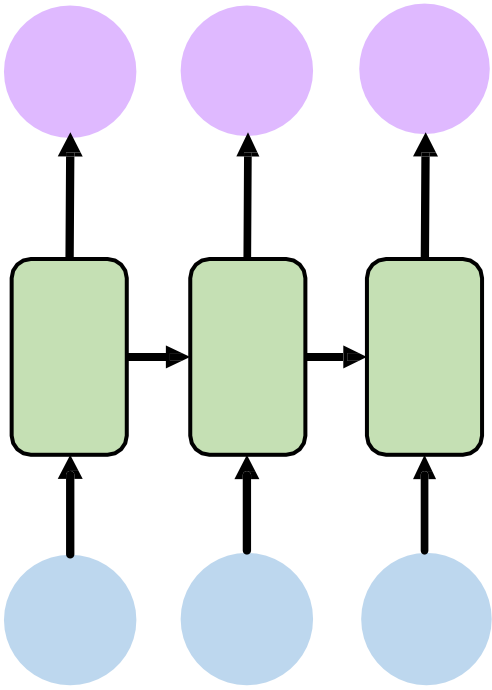
Deep Sequence Modeling



One to One
"Vanilla" neural network

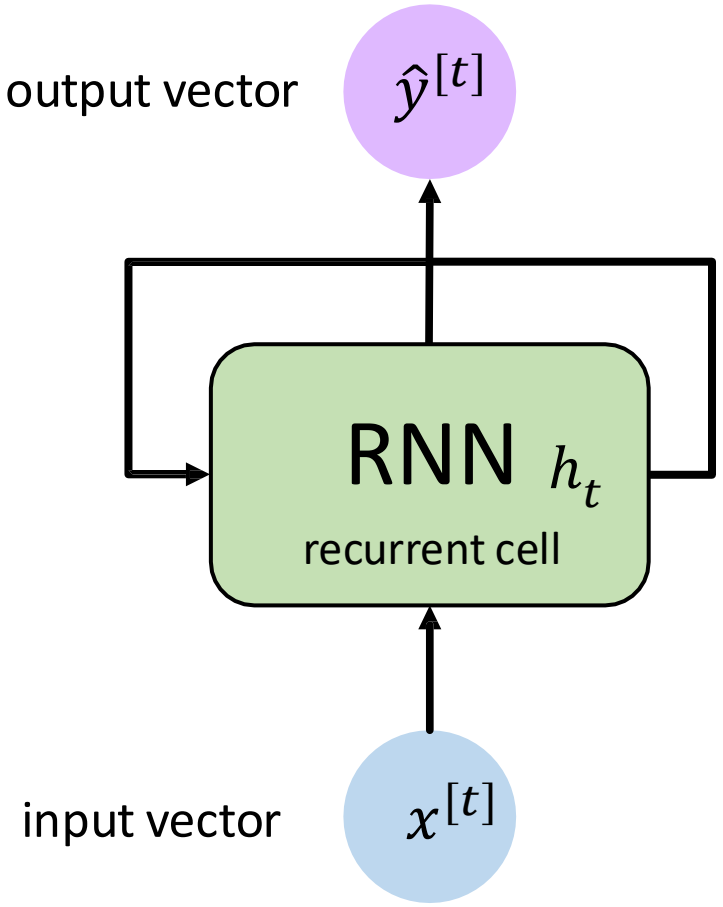


Sequence to One
Sentiment Classification

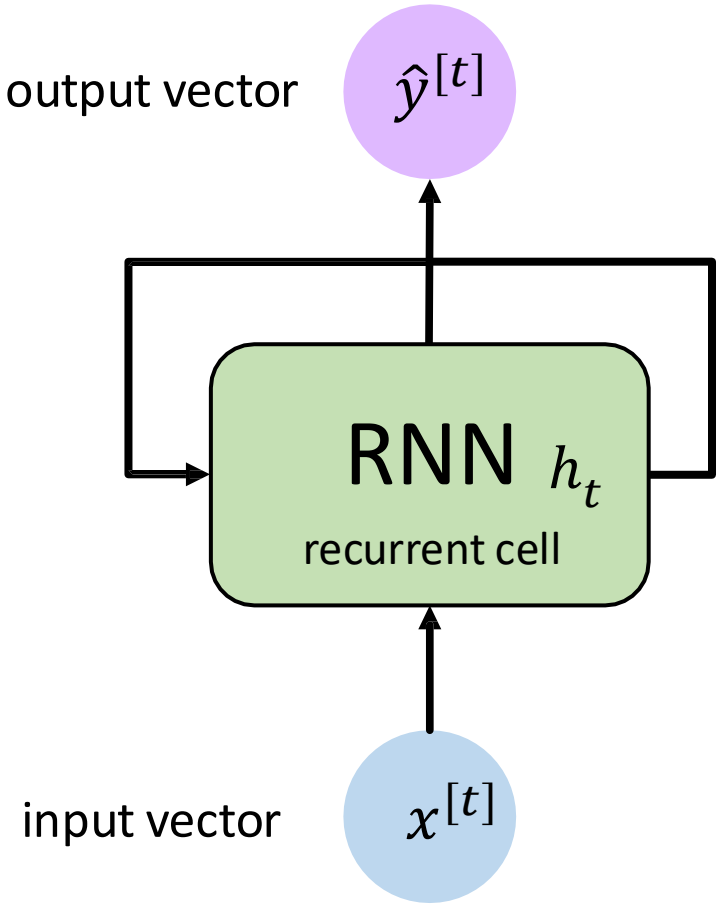


Seq to Seq
Machine Translation

A recurrent neural network (RNN)



A recurrent neural network (RNN)

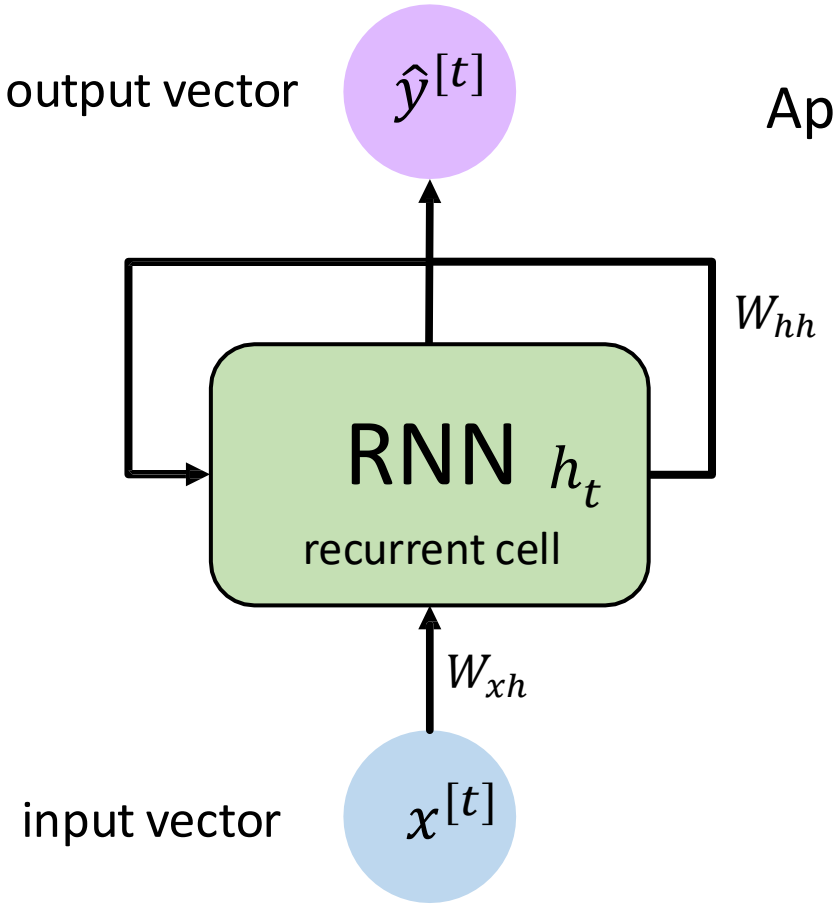


Apply a recurrence at every time step to process a sequence:

$$h_t = f_W(x^{[t]}, h_{t-1})$$

- h_t: cell state
- f_W: function parameterized by W
- $x^{[t]}$: input vector at time t
- h_{t-1}: past memory

A recurrent neural network (RNN)



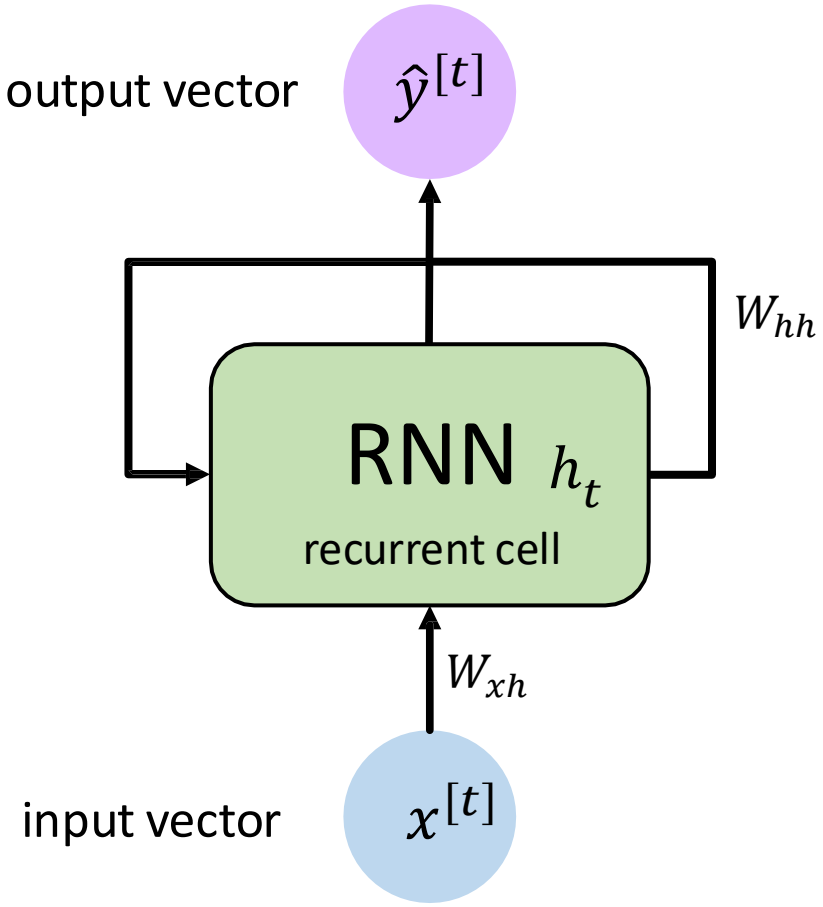
Apply a recurrence at every time step to process a sequence:

$$h_t = f_W(x^{[t]}, h_{t-1})$$

- h_t: cell state
- f_W: function parameterized by W
- $x^{[t]}$: input vector at time t
- h_{t-1}: past memory

$$W = (W_{xh}, W_{hh})$$

A recurrent neural network (RNN)



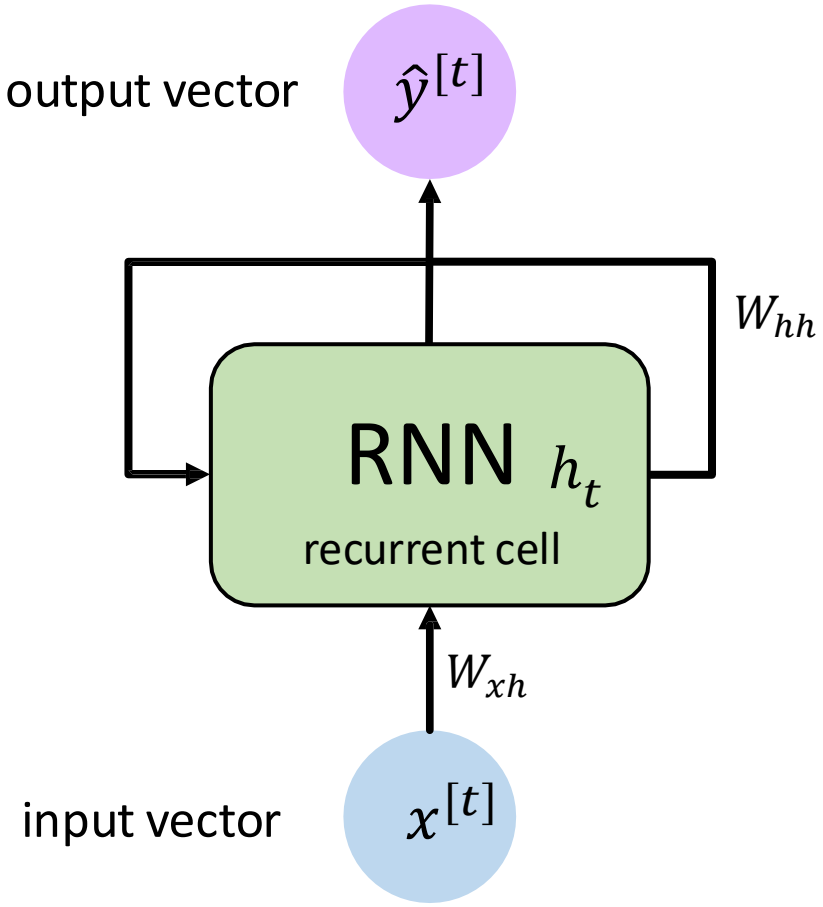
$$h_t = f_W(x^{[t]}, h_{t-1})$$

new state — function parameterized by W — input vector at time *t* — old state

$$W = (W_{xh}, W_{hh})$$

Sharing parameters: the same function and set of parameters W are used at every time step

A recurrent neural network (RNN)



$$W = (W_{xh}, W_{hh})$$

Output Vector

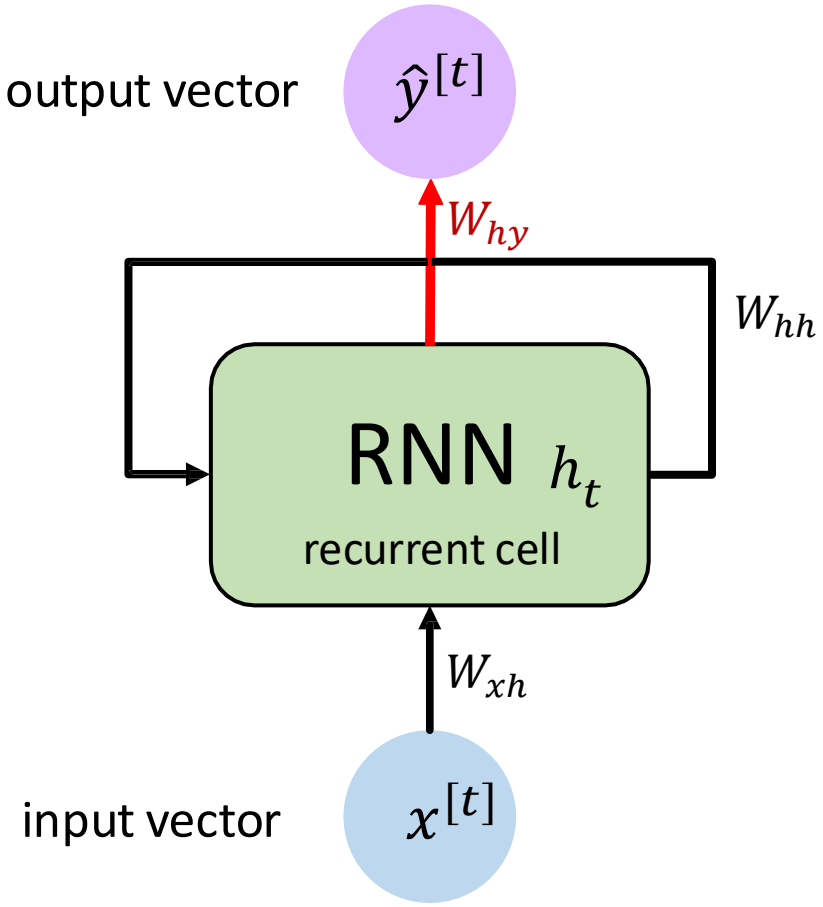
$$\hat{y}^{[t]} = h_t$$

Update Hidden State

$$h_t = f_W\left(x^{[t]}, h_{t-1}\right)$$
$$= g(W_{xh}\, x^{[t]} + W_{hh} h_{t-1})$$

Input Vector

A recurrent neural network (RNN)



$$W = (W_{xh}, W_{hh}, W_{hy})$$

Output Vector

$$\hat{y}^{[t]} = W_{hy} h_t$$

Update Hidden State

$$h_t = f_W\left(x^{[t]}, h_{t-1}\right)$$
$$= g(W_{xh}\, x^{[t]} + W_{hh} h_{t-1})$$

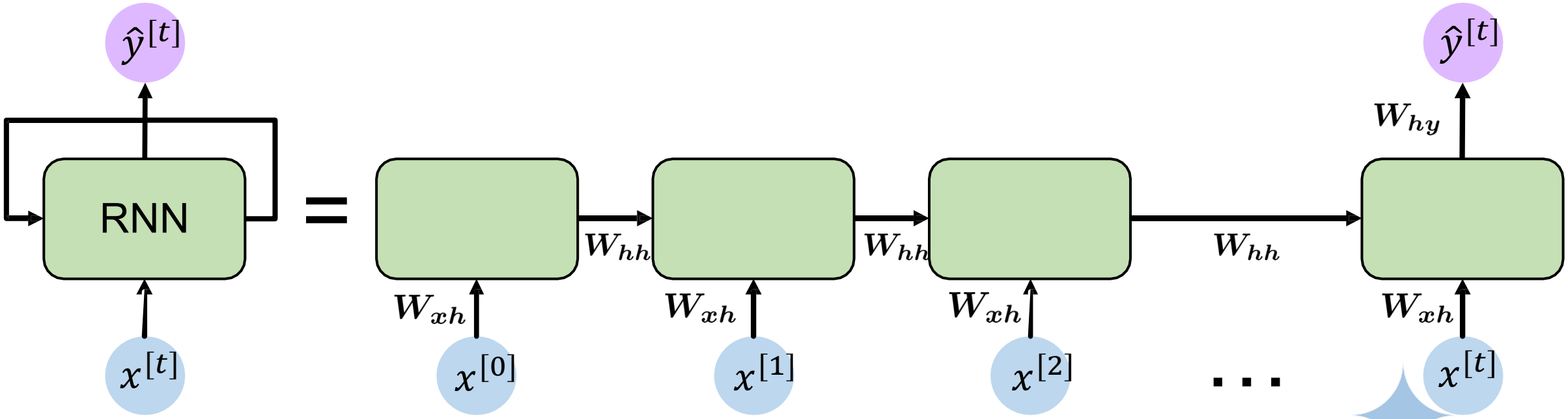
Input Vector

RNN: activation function

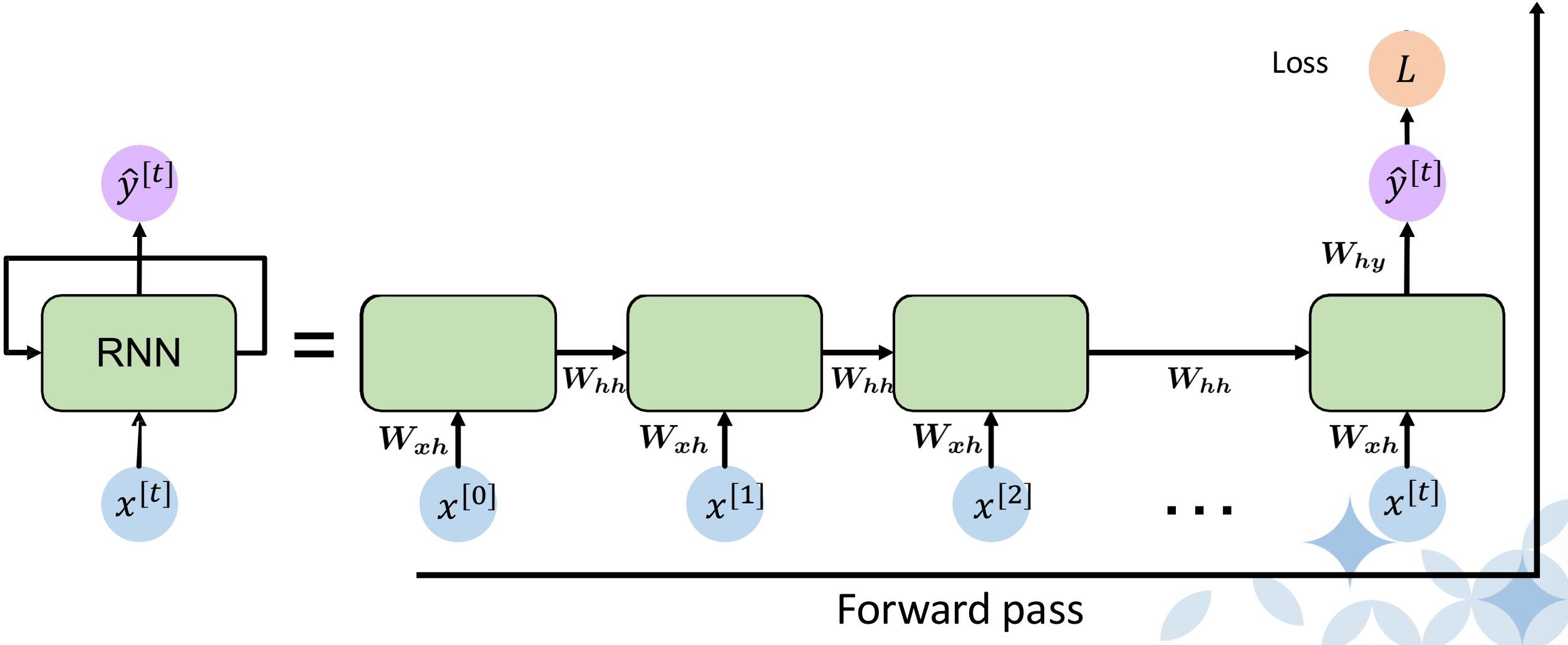
- $h_t = g(W_{xh}\, x^{[t]} + W_{hh} h_{t-1})$
  - Non-linear activation function $g(\cdot)$
  - RNN typically uses the hyperbolic function $\tanh(\cdot)$ instead of ReLU. Why?
  - RNN (1986) was invented far before ReLU (popularized in 2010). Now?
  - ReLU still not a popular choice for RNN now.
  - Before we answer this question, Let's look first at the RNN computation graph.

RNNs: computational graph

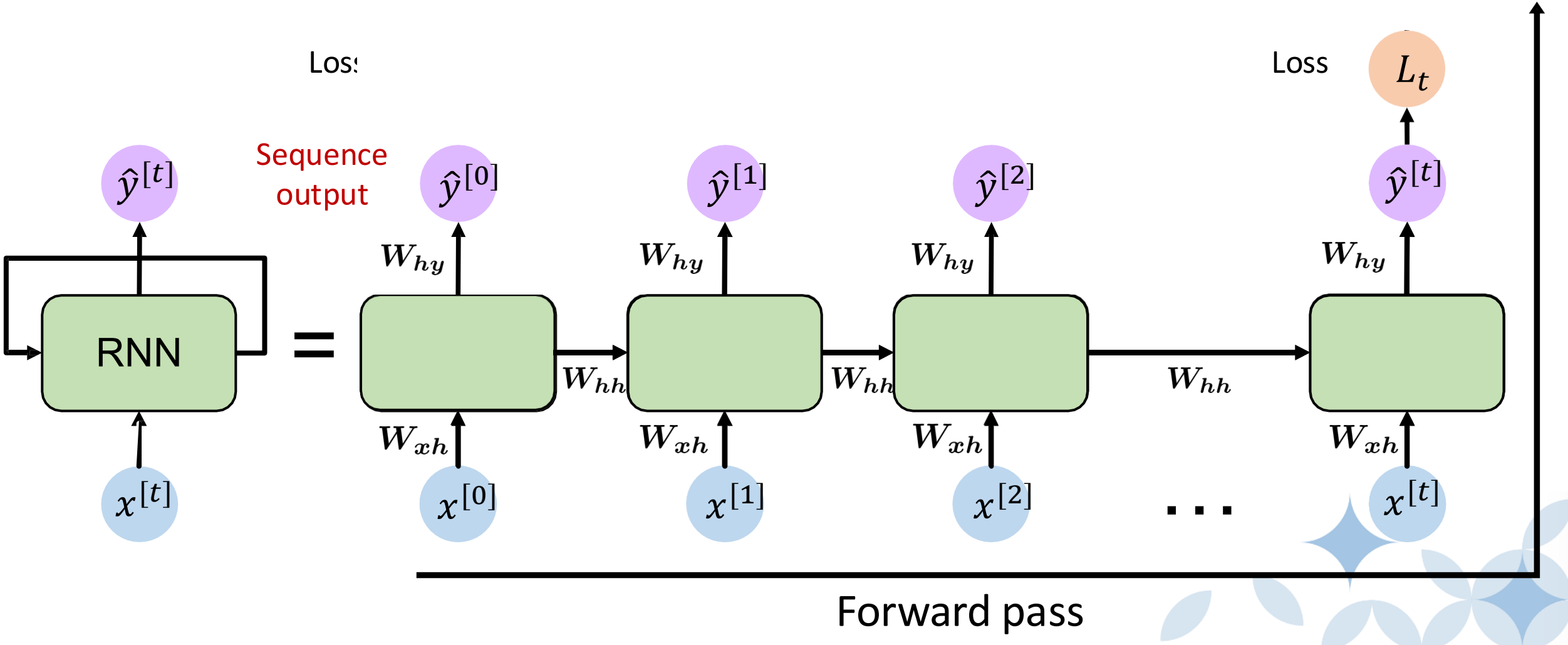
- Sharing parameters: Re-use the same weight matrices at every time step



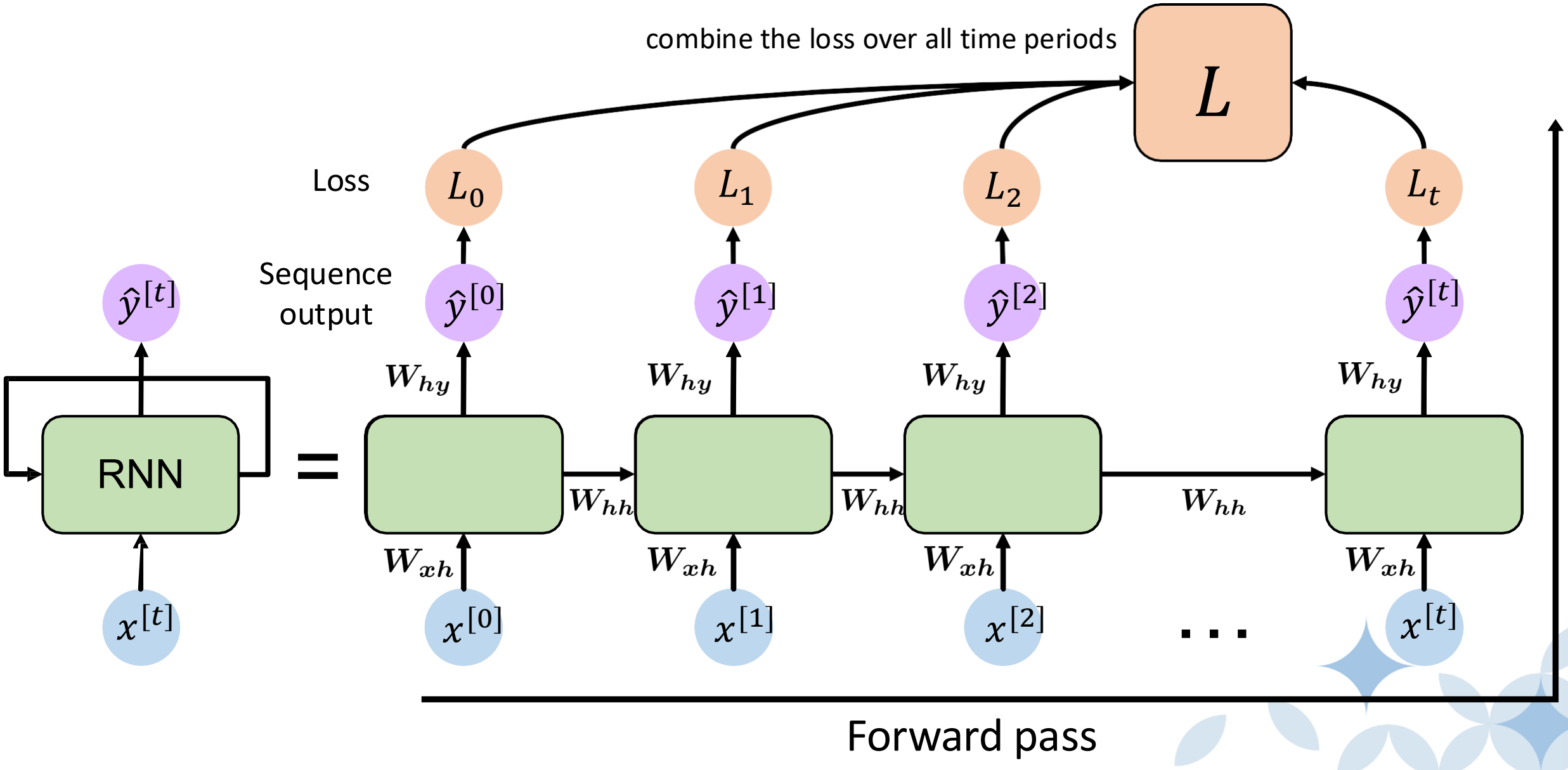
RNNs: computational graph



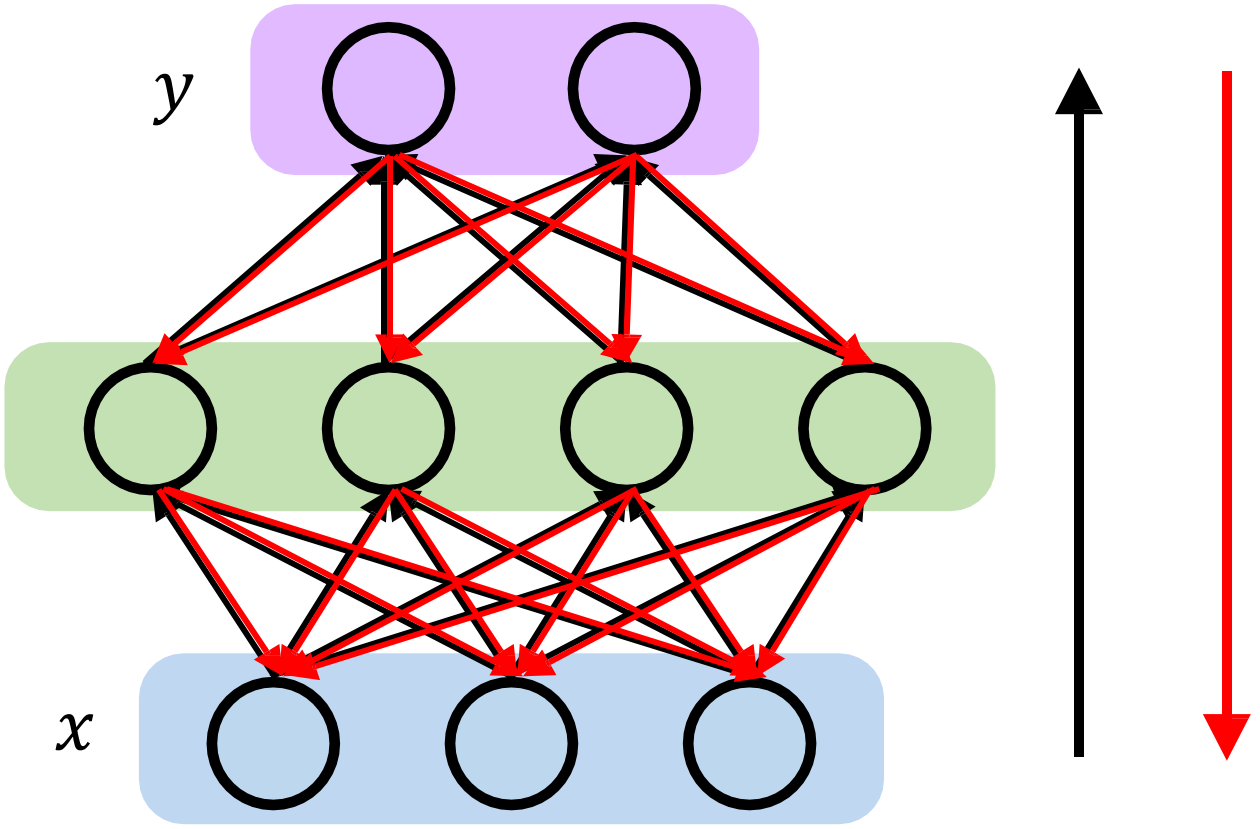
RNNs: computational graph



RNNs: computational graph



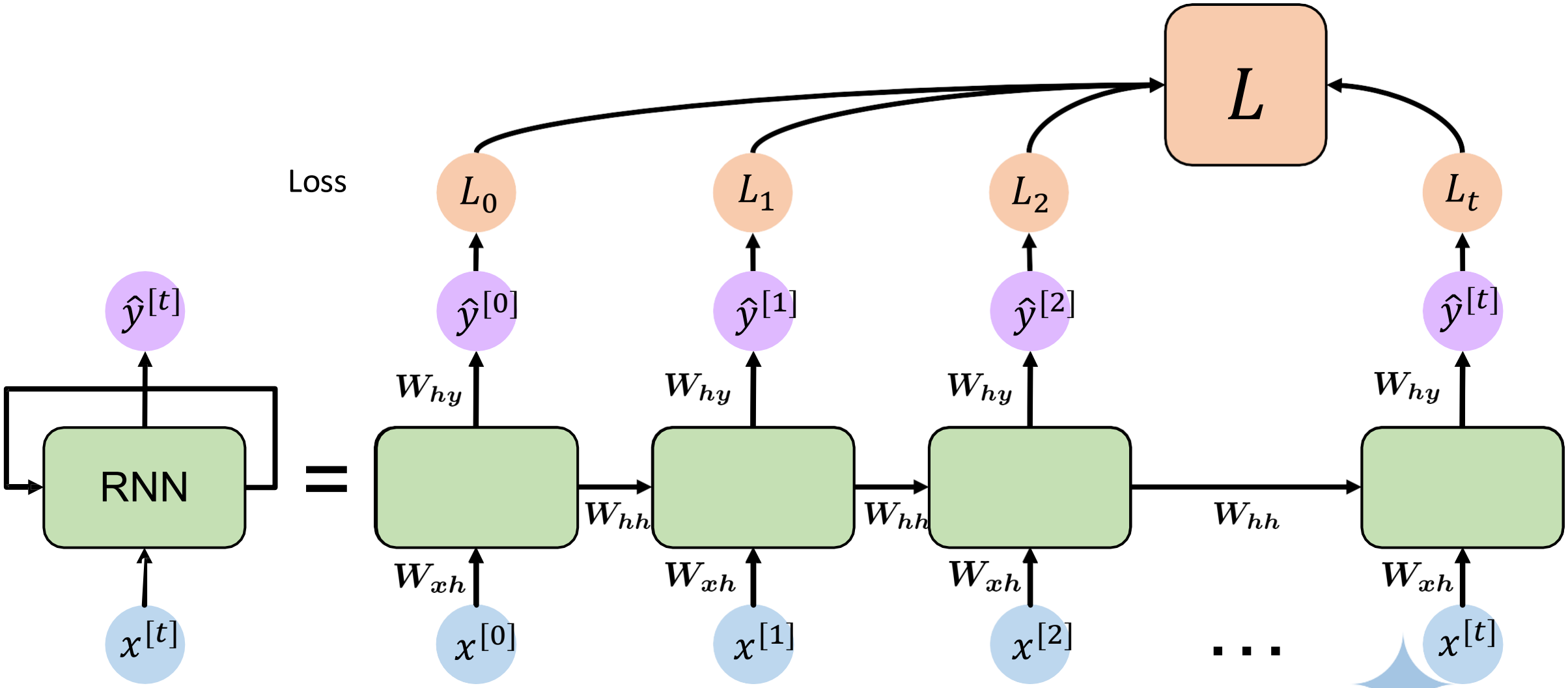
Recall: backpropagation in DNN



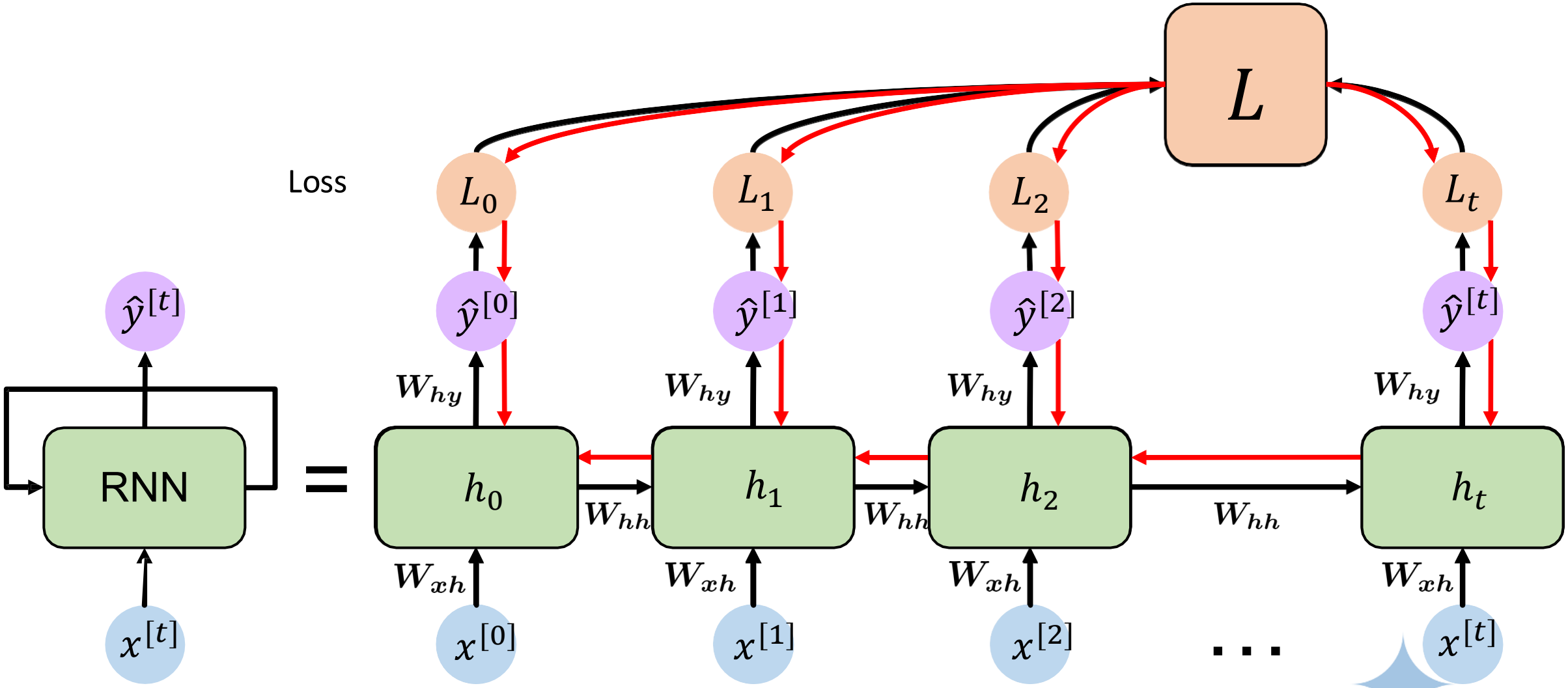
Backpropagation algorithm:

1. Take the gradient of the loss with respect to each parameter
2. Update parameters in order to minimize loss

RNNs: Backpropagation



RNNs: Backpropagation



RNNs: Backpropagation w. parameter sharing

- $\frac{\partial L}{\partial W_{hh}} = \sum_{k=0}^{t} \frac{\partial L}{\partial L_k} \frac{\partial L_k}{\partial W_{hh}}$

  - $\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial \hat{y}^{[t]}} \cdot \frac{\partial \hat{y}^{[t]}}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_{hh}}$ $\quad h_t = g(z_t)$, *where* $z_t = W_{xh}\, x^{[t]} + W_{hh} h_{t-1}$

  - $= \frac{\partial L_t}{\partial \hat{y}^{[t]}} \cdot \frac{\partial \hat{y}^{[t]}}{\partial h_t} \cdot g'(z_t) \left[\frac{\partial z_t}{\partial W_{hh}} + \frac{\partial z_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{hh}} \right]$

  - $= \frac{\partial L_t}{\partial \hat{y}^{[t]}} \cdot W_{hy} \cdot g'(z_t) \left[h_{t-1} + W_{hh} \frac{\partial h_{t-1}}{\partial W_{hh}} \right]$

  - $= \frac{\partial L_t}{\partial \hat{y}^{[t]}} \cdot W_{hy} \cdot g'(z_t) \left\{ h_{t-1} + W_{hh} \cdot g'(z_{t-1}) \left[h_{t-2} + W_{hh} \frac{\partial h_{t-2}}{\partial W_{hh}} \right] \right\}$

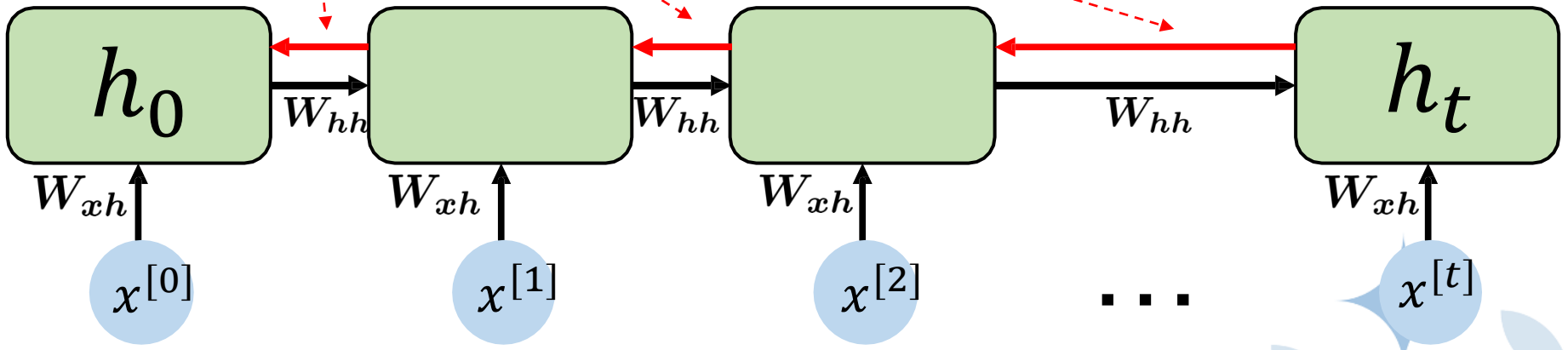
  - $= \frac{\partial L_t}{\partial \hat{y}^{[t]}} \cdot W_{hy} \cdot g'(z_t) h_{t-1} + W_{hh} \cdot g'(z_t) g'(z_{t-1}) h_{t-2} + W_{hh}^2 \cdot g'(z_t) g'(z_{t-1}) \frac{\partial h_{t-2}}{\partial W_{hh}}$

  - $= \frac{\partial L_t}{\partial \hat{y}^{[t]}} \cdot W_{hy} \cdot \sum_{i=0}^{t-1} W_{hh}^i \prod_{j=0}^{i} g'(z_{t-j}) h_{t-j-1}$

Standard RNN : gradients

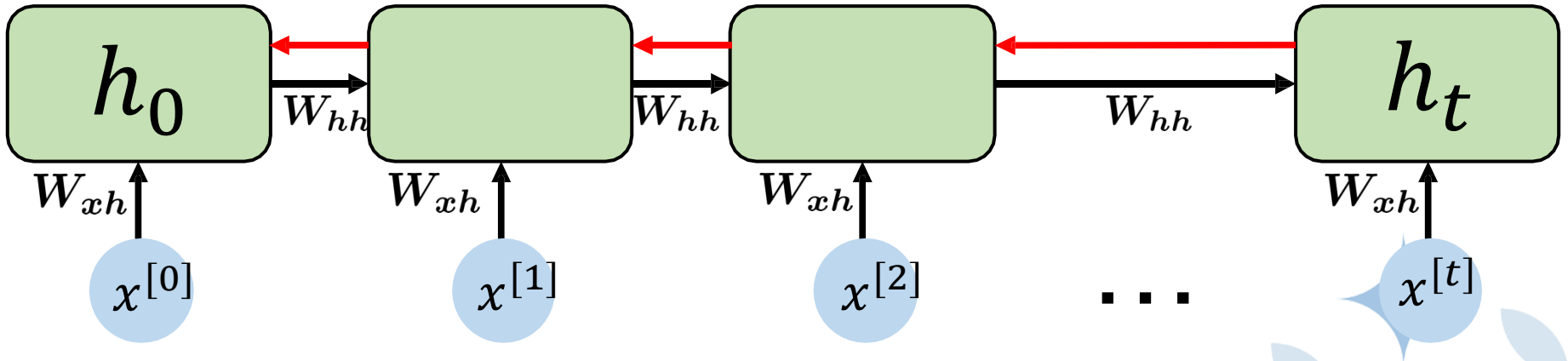
- Backpropagation

$$-\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial \hat{y}^{[t]}} \cdot W_{hy} \cdot \sum_{i=0}^{t-1} W_{hh}^{i} \prod_{j=0}^{i} g'(z_{t-j}) h_{t-j-1}$$: Involves many factors of W_{hh}



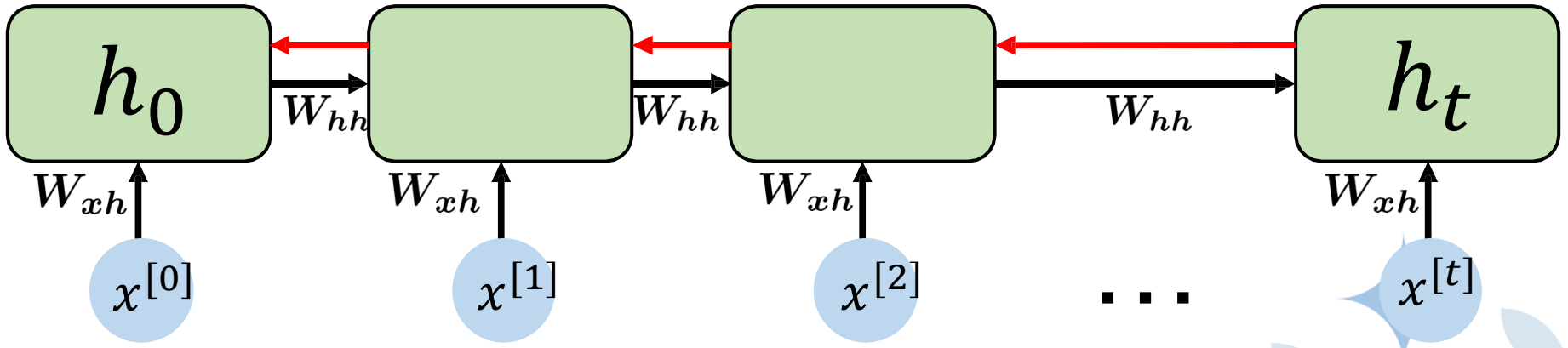
Standard RNN : gradients

- Backpropagation involves many factors of $W_{h,h}$) and repeated gradient computation over the activation!)
  - Consider we use linear activation function or ReLU
  - If $W_{h,h} > 1$: exploding gradients $1.2^{100} \approx 8.28 \times 10^{7}$
  - If $W_{h,h} < 1$: vanishing gradients $0.8^{100} \approx 2 \times 10^{-10}$



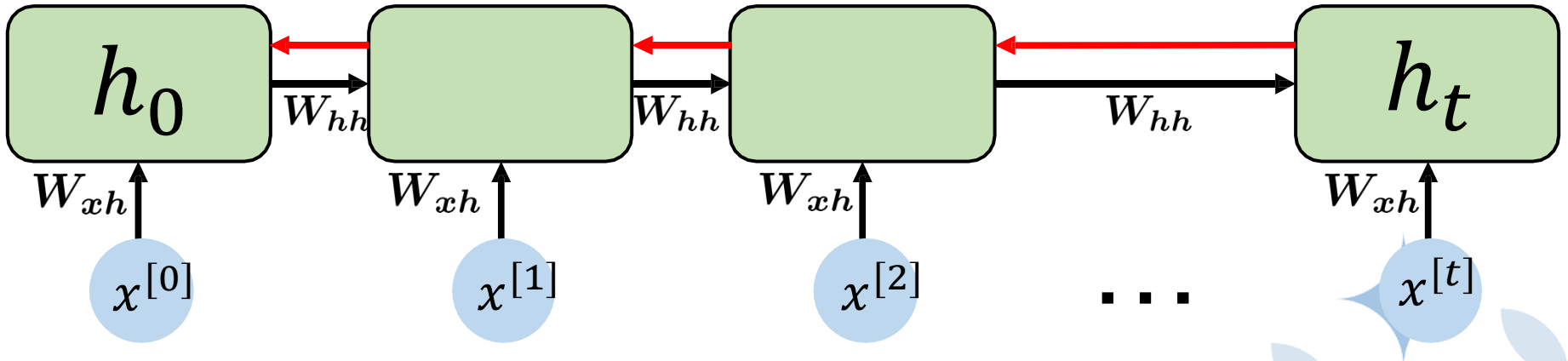
Standard RNN : gradients

- Backpropagation involves many factors of $W_{h,h}$) and repeated gradient computation over the activation!)
  - Exploding gradients. Solution: we use hyperbolic function $\tanh(\cdot)$
    - ReLU is not good idea since g' doesn't shrink: $\frac{\partial L_t}{\partial \hat{y}[t]} \cdot W_{hy} \cdot \sum_{i=0}^{t-1} W_{hh}^{i} \prod_{j=0}^{i} g'(z_{t-j}) h_{t-j-1}$
    - Use ReLU: a careful initialization of network weights to ensure that the network is stable prior to training. https://arxiv.org/abs/1504.00941



Standard RNN : gradients

- Backpropagation involves many factors of $W_{h,h}$) and repeated gradient computation over the activation!)
  - Exploding gradients. Solution: we use hyperbolic function $\tanh(\cdot)$
  - Still: vanishing gradients

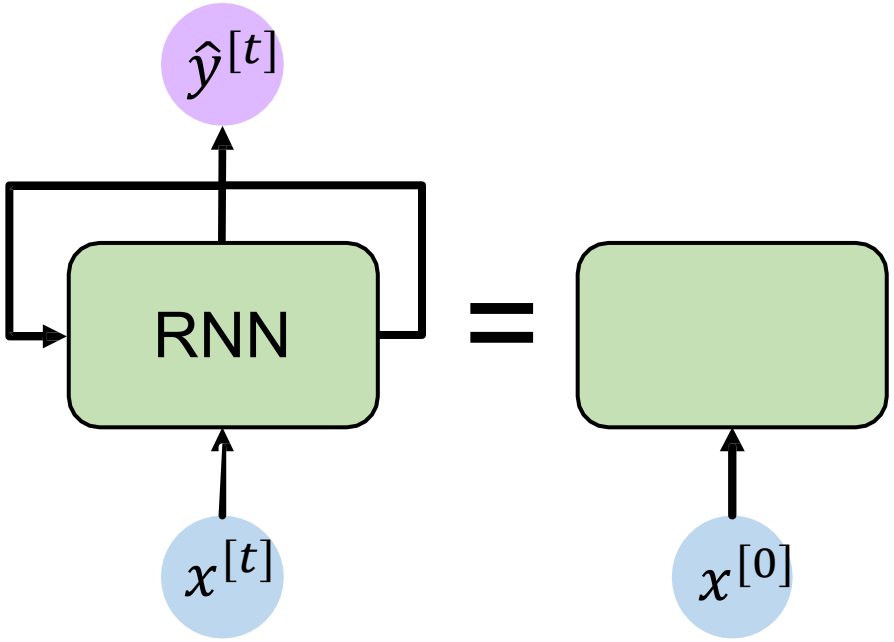


Standard RNN : vanishing gradients

- Vanishing gradients:
  - Multiplied by many small numbers together
  - Errors due to further back time steps have smaller and smaller gradients
  - Easy to forget things: hard to capture long-term memory
  - Force parameters to capture short-term dependence only
- Solution: Use a more complex recurrent unit with gates to control what information is passed through: LSTM

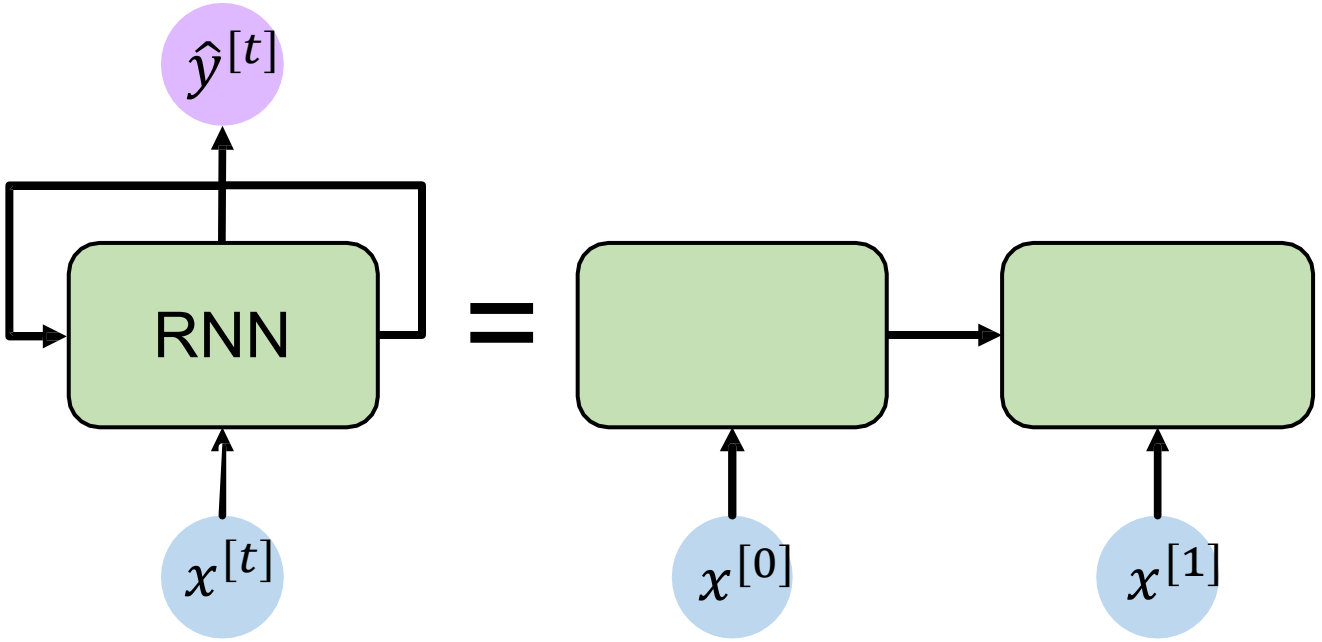
Recurrent Neural Network (RNN)

Sequence Modeling



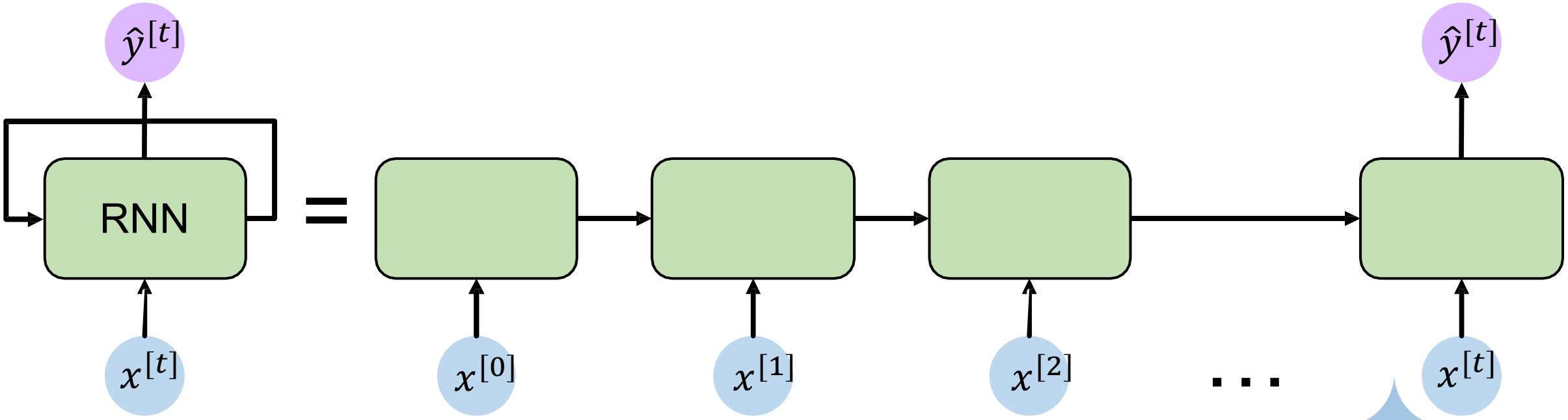
Recurrent Neural Network (RNN)

Sequence Modeling



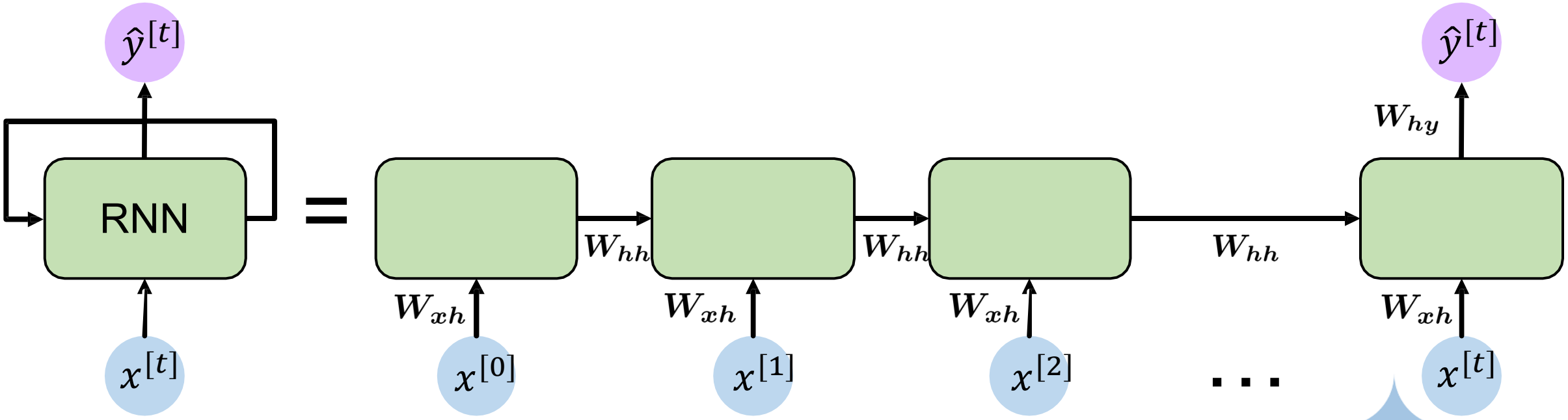
Recurrent Neural Network (RNN)

Sequence Modeling



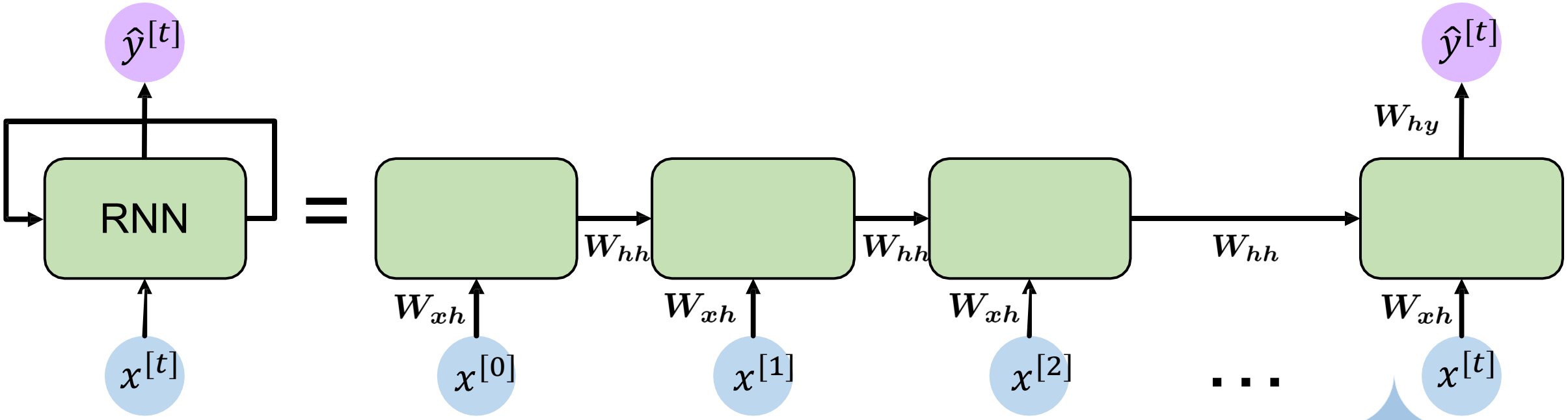
Recurrent Neural Network (RNN)

Sequence Modeling



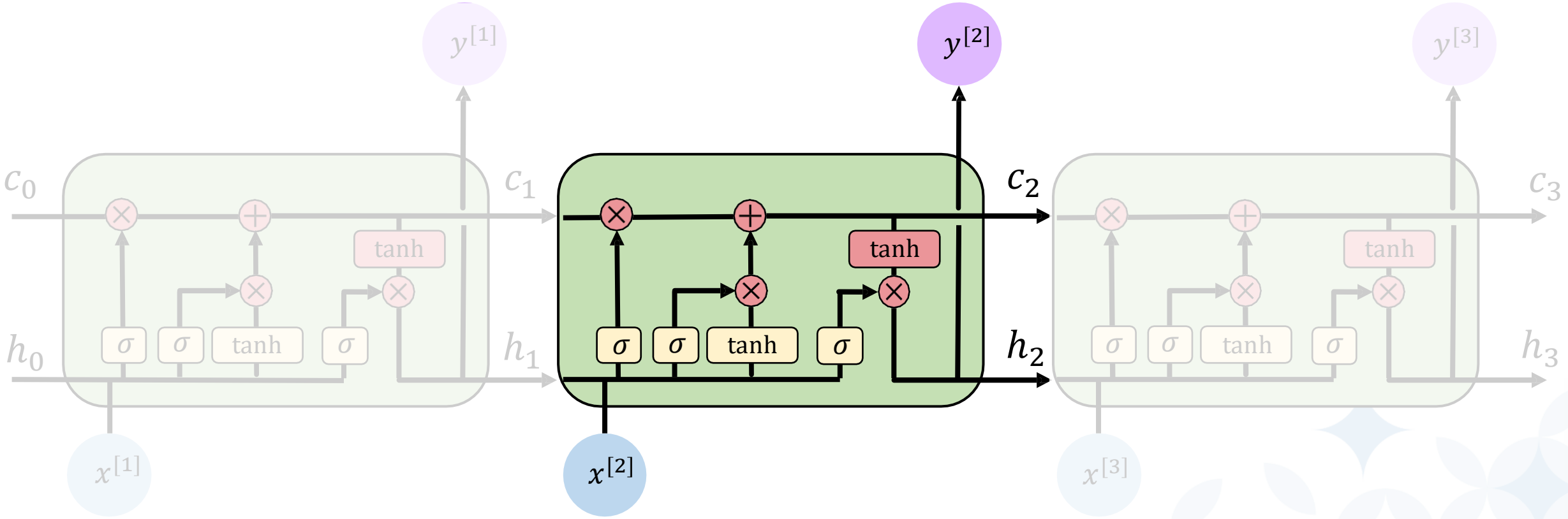
Long Short Term Memory (LSTMs)

- LSTM: each module contains multiple computational blocks for better information flow control so that it can track information throughout many timesteps



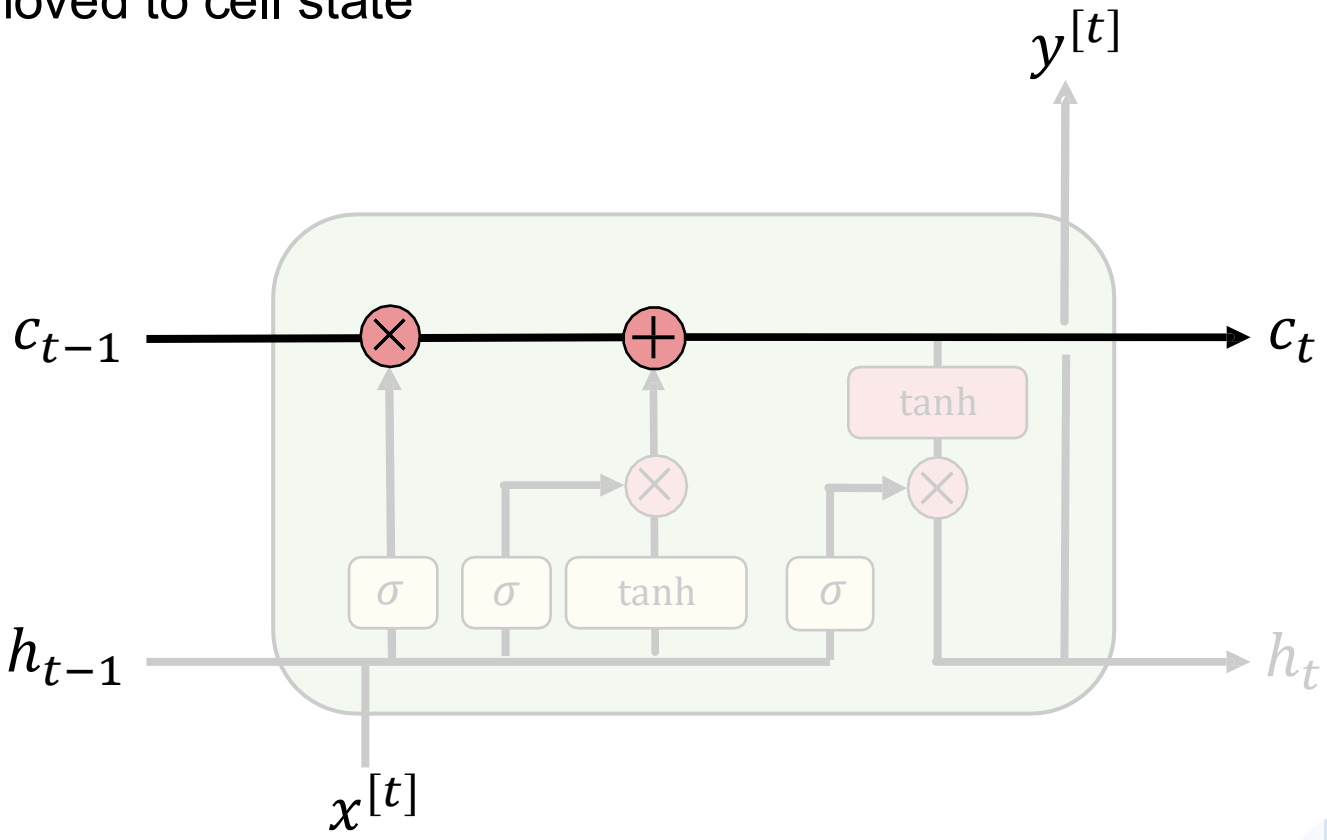
Long Short Term Memory (LSTMs)

- LSTM: each module contains multiple computational blocks for better information flow control so that it can track information throughout many timesteps



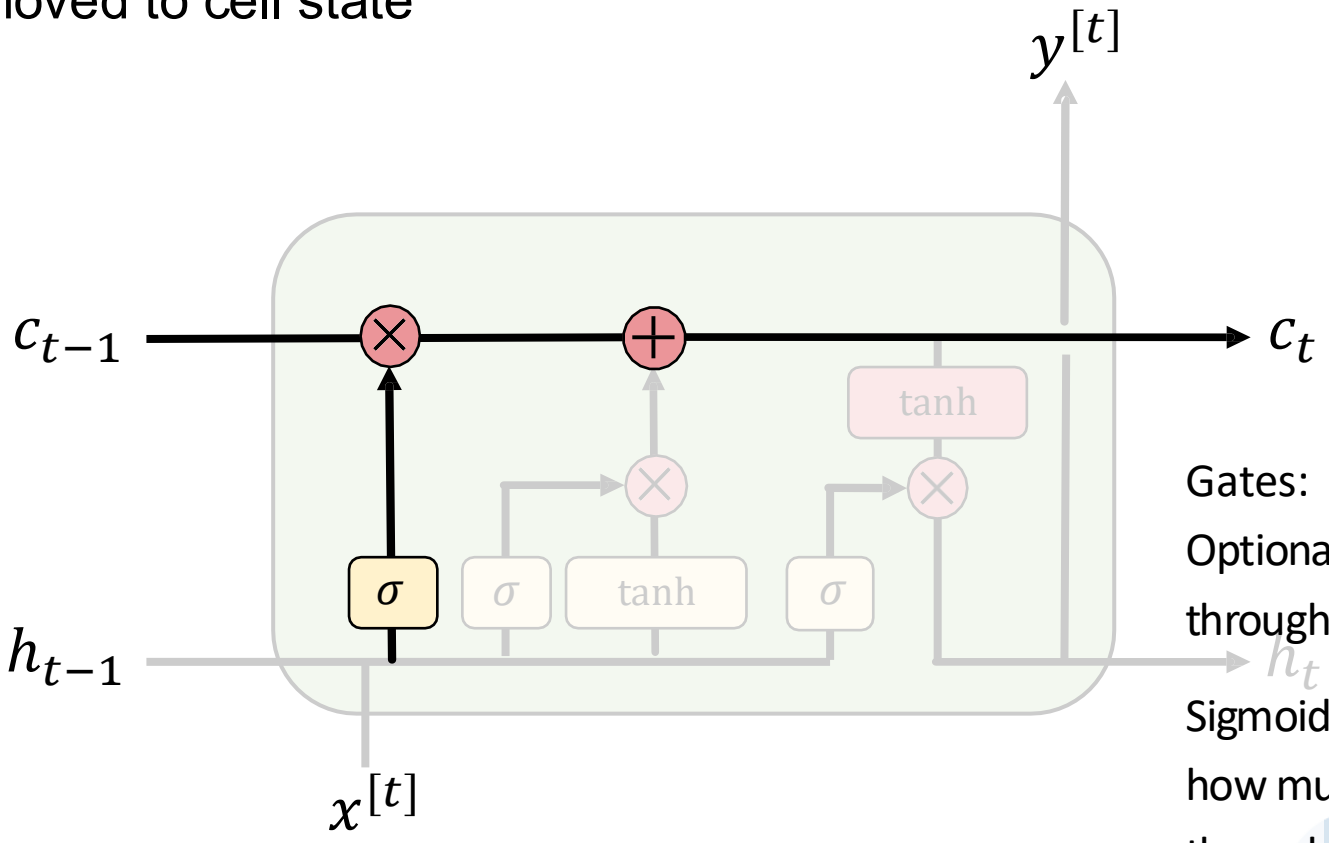
Long Short Term Memory (LSTMs)

- Maintain an additional conveyor belt (cell state) c_t to pass the information along time
  - Gates: allow information being added or removed to cell state



Long Short Term Memory (LSTMs)

- Maintain an additional conveyor belt (cell state) c_t to pass the information along time
  - Gates: allow information being added or removed to cell state

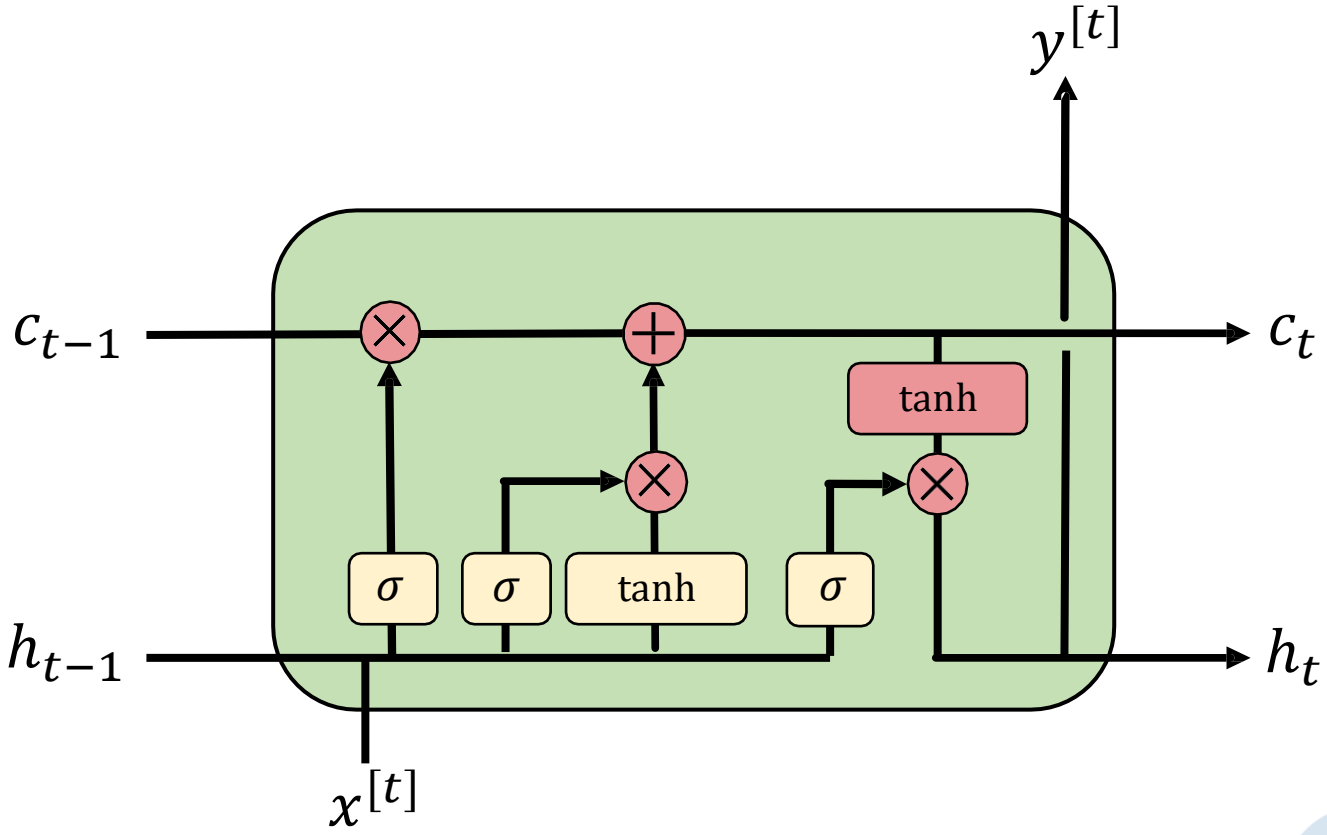


Gates:
Optionally let information through, via a sigmoid function

Sigmoid between 0,1: control how much information can flow through this gate

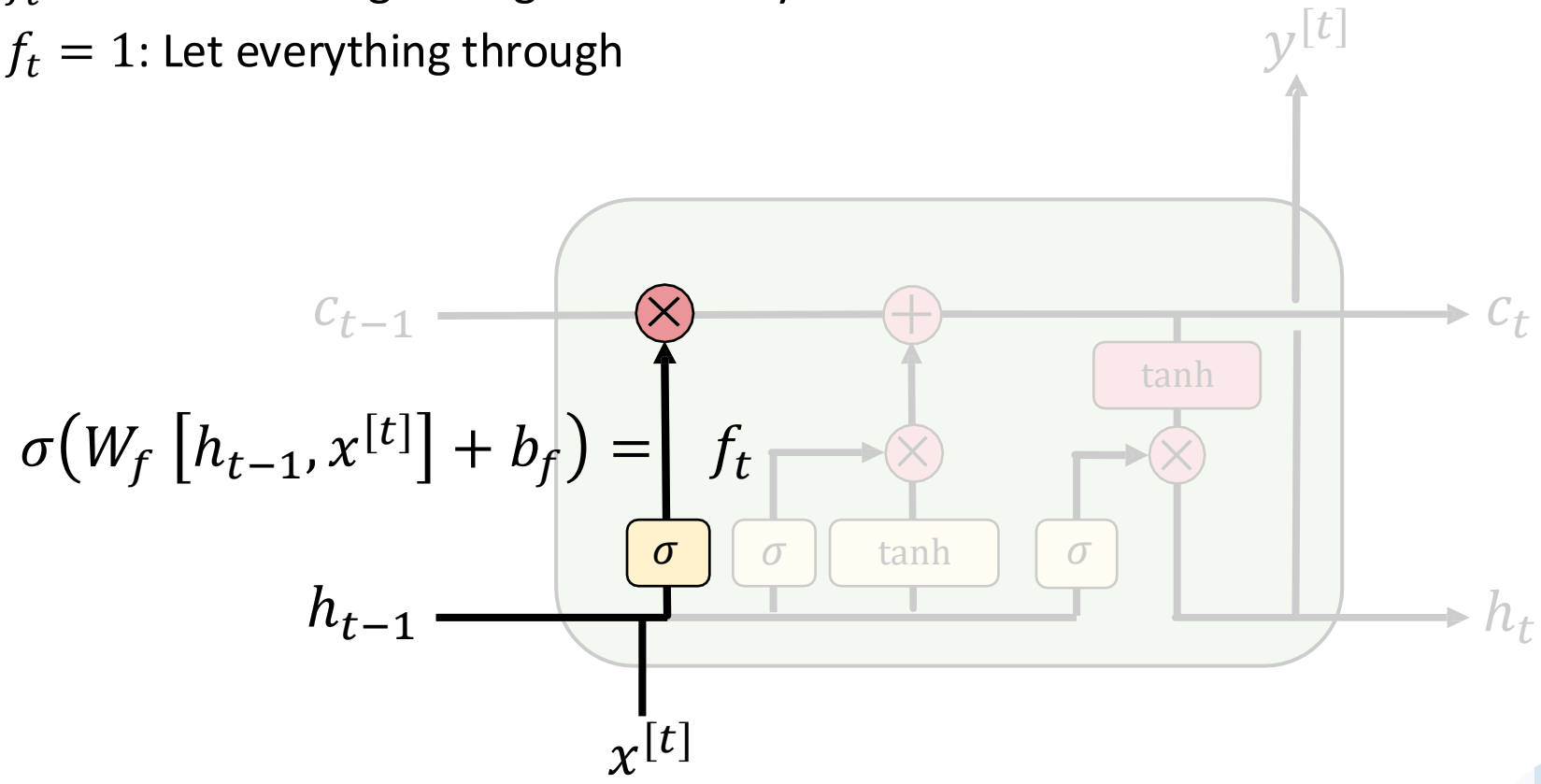
Long Short Term Memory (LSTMs)

- How do LSTM work: 4 Stages
  - (1) Forget (2) Input (3) Update (4) Output



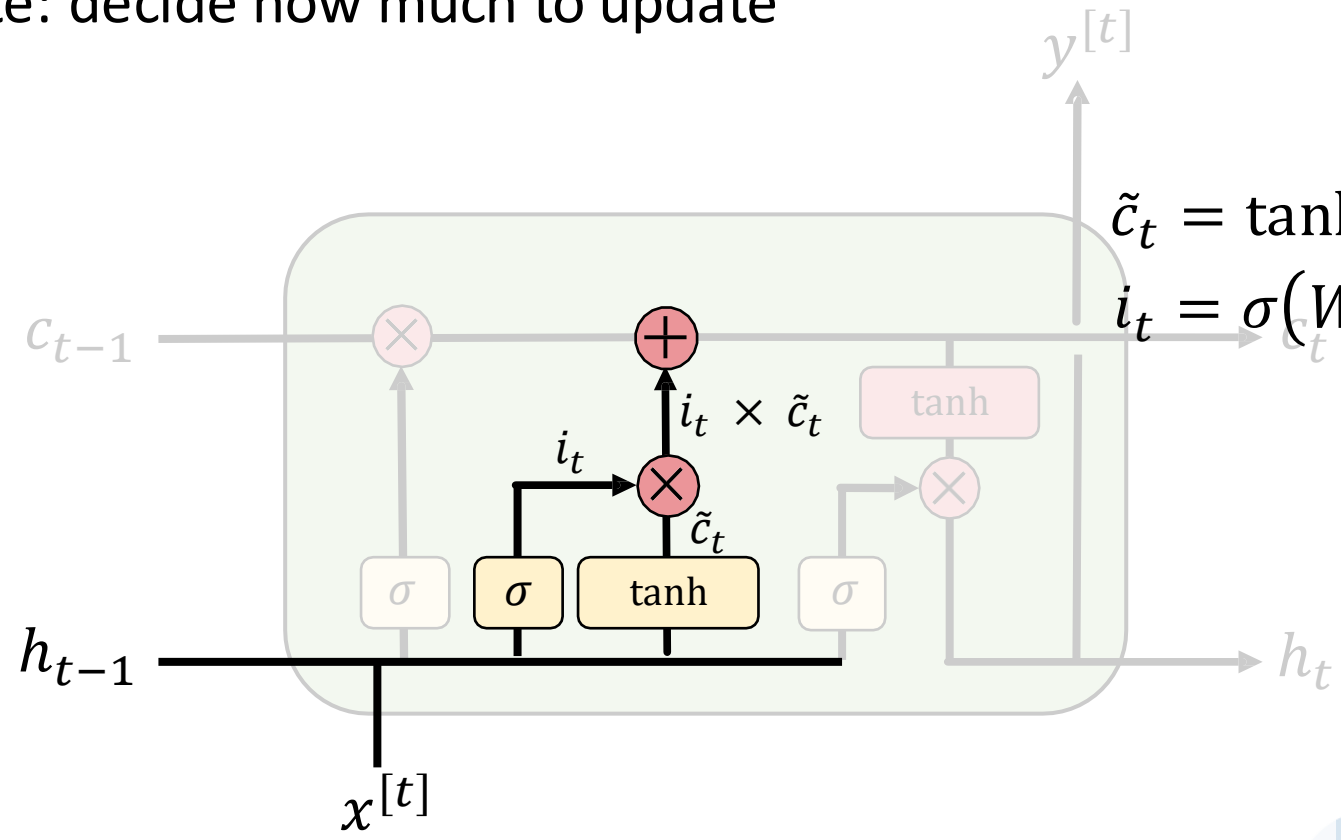
Long Short Term Memory (LSTMs): Forget

- Forget Gate f_t : optionally let information through, via a sigmoid function
- Sigmoid between 0,1: control how much information flowing through the gate
  - $f_t = 0$: Let nothing through the conveyor belt
  - $f_t = 1$: Let everything through



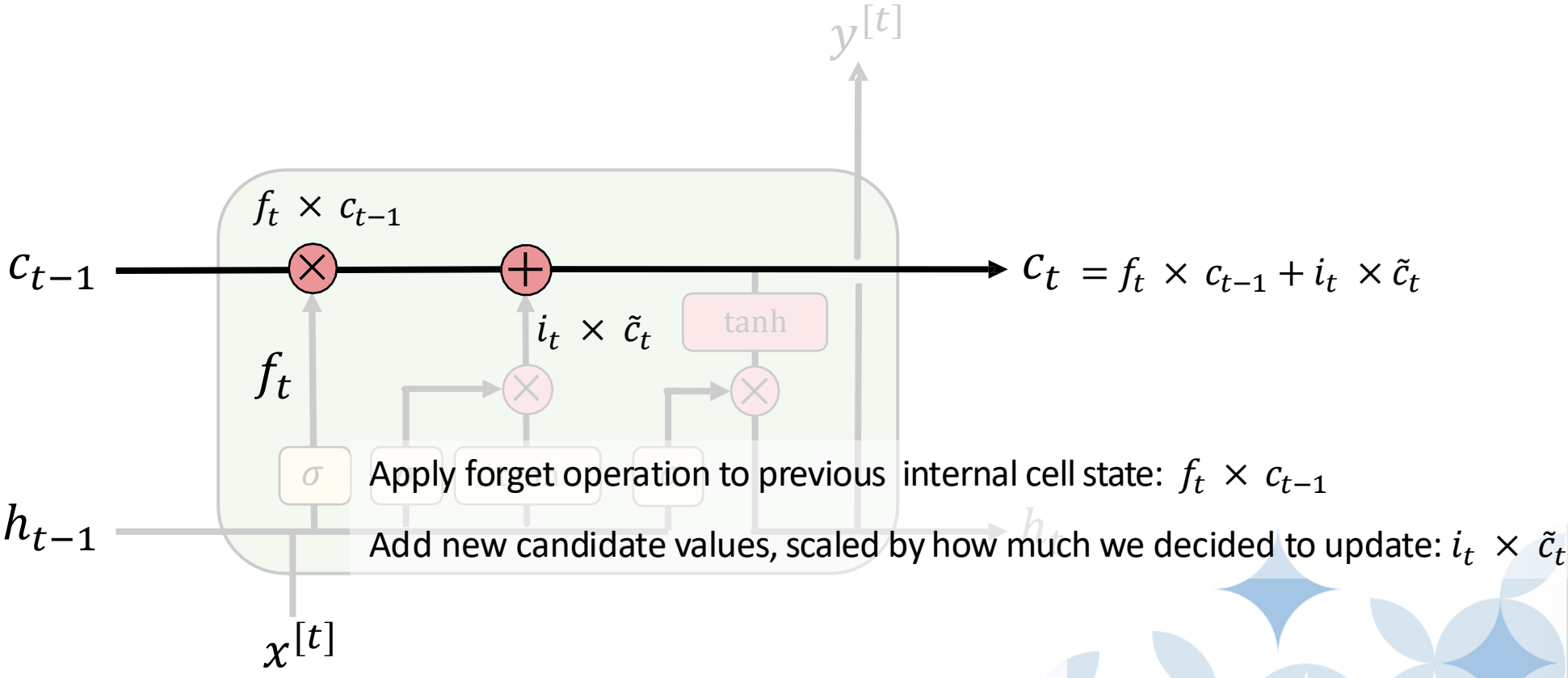
Long Short Term Memory (LSTMs): Input

- Input Stage:
  - $\tilde{c}_t$: generate new vector of "candidate values" that could be added to the state
  - i_t: input gate: decide how much to update



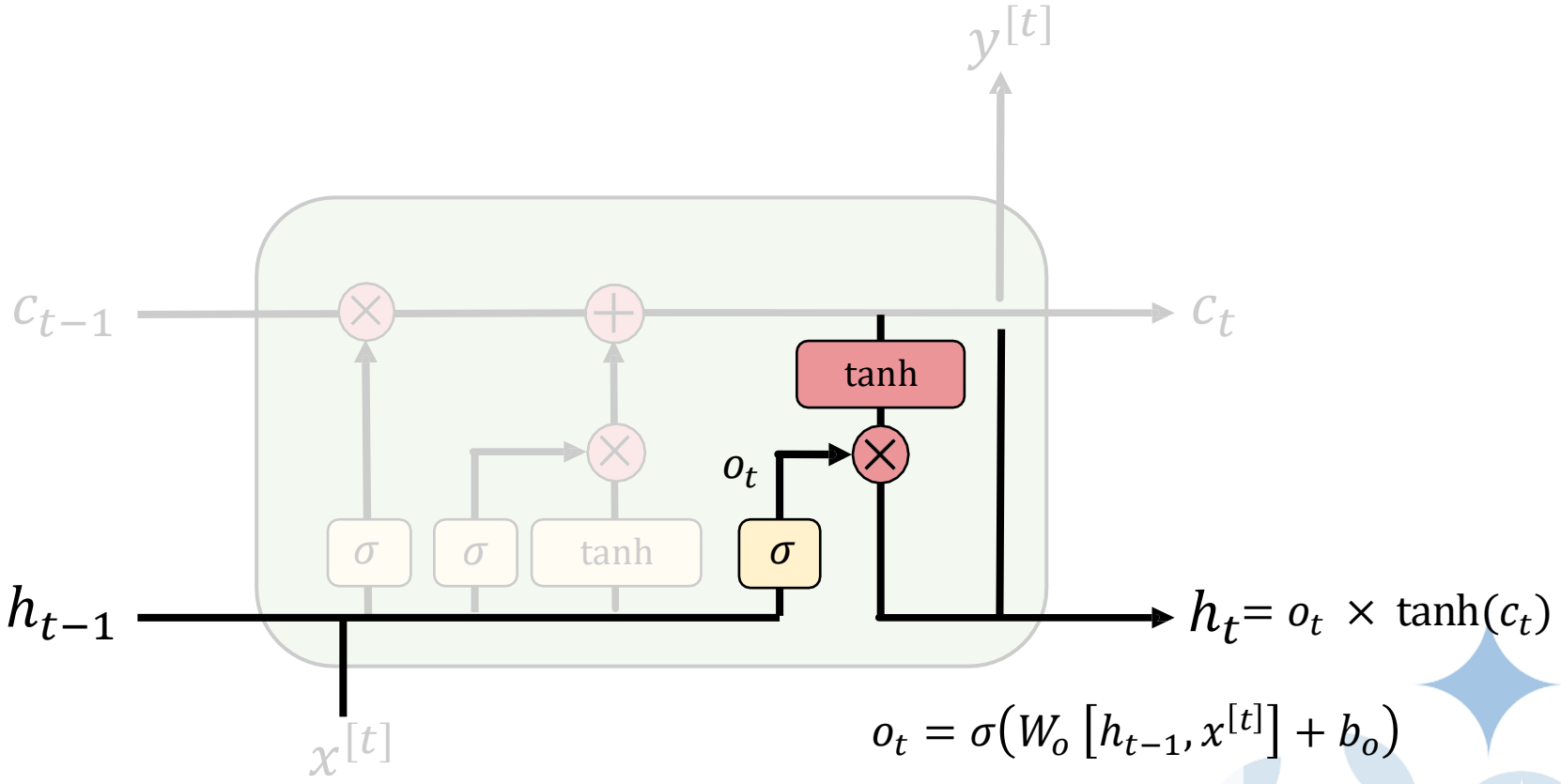
Long Short Term Memory (LSTMs): Update

- (1) Forget (2) Input (3) Update (4) Output
  - Update the cell state $c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$



Long Short Term Memory (LSTMs): Update

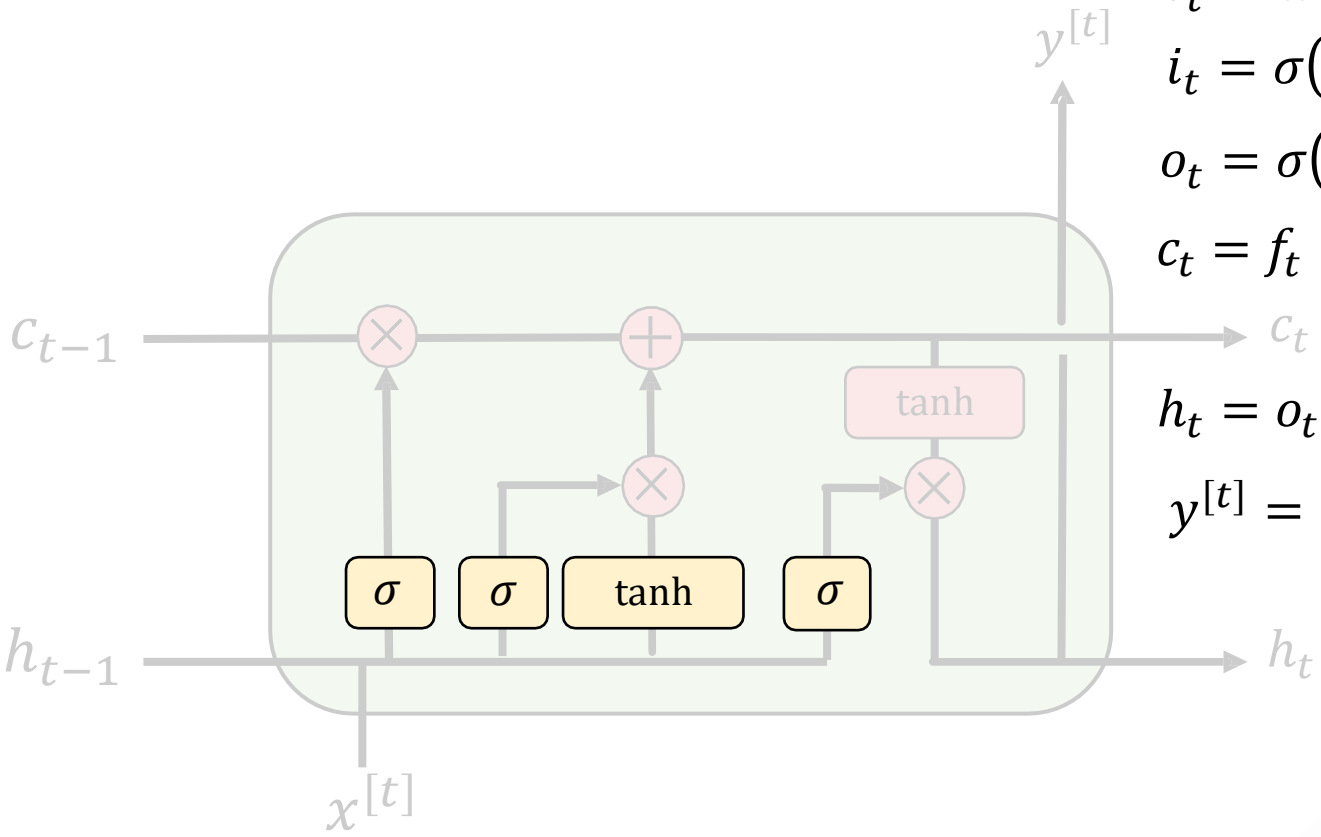
- (1) Forget (2) Input (3) Update (4) Output
  - Output a filtered version of the cell state



Long Short Term Memory (LSTMs): Output

- Model complexity
  - W_f, W_i, W_c, W_o : 4 times larger than RNN

$$f_t = \sigma\left(W_f \left[h_{t-1}, x^{[t]}\right] + b_f\right)$$
$$\tilde{c}_t = \tanh\left(W_c \left[h_{t-1}, x^{[t]}\right] + b_c\right)$$
$$i_t = \sigma\left(W_i \left[h_{t-1}, x^{[t]}\right] + b_i\right)$$
$$o_t = \sigma\left(W_o \left[h_{t-1}, x^{[t]}\right] + b_o\right)$$
$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$$
$$h_t = o_t \times \tanh(c_t)$$
$$y^{[t]} = W_{h,y} h_t$$



LSTM: Gradient Flow

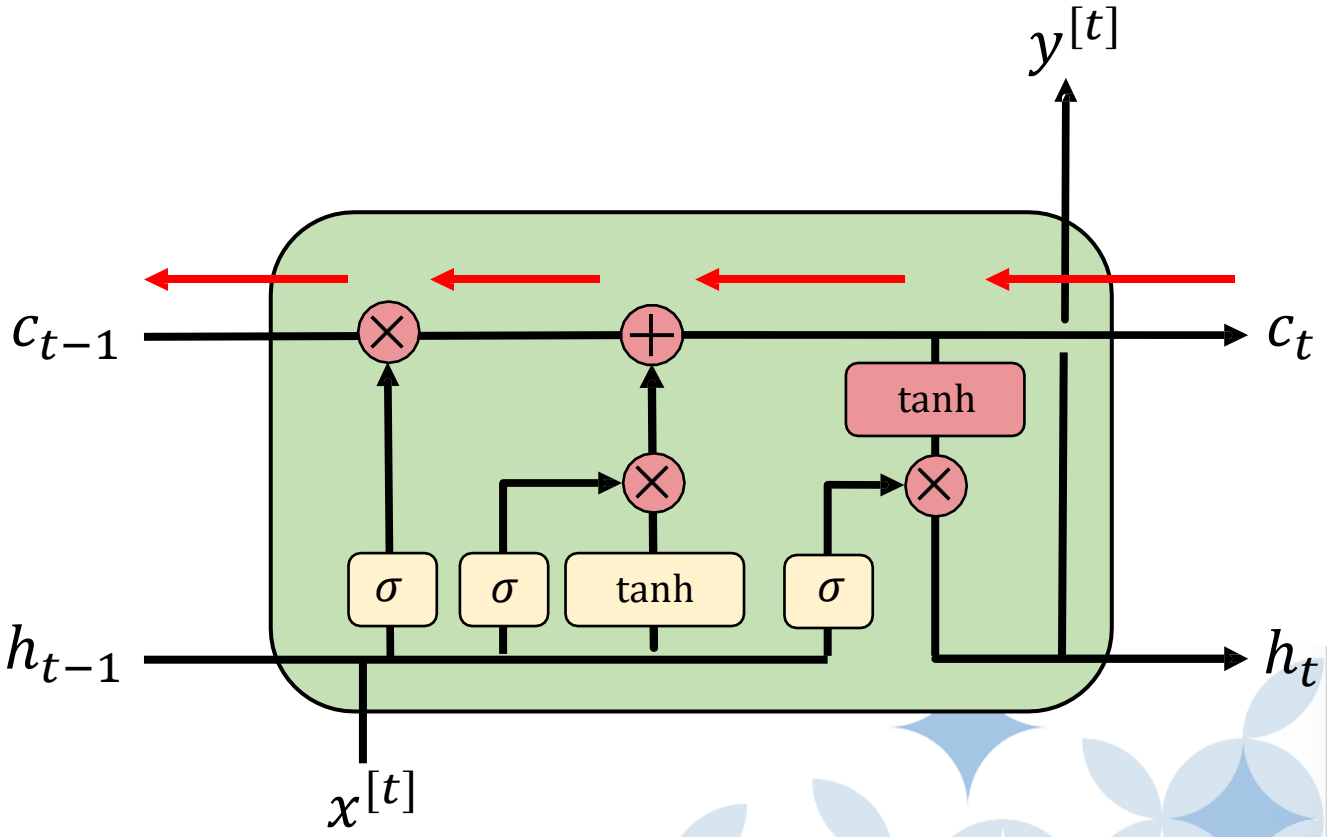
- Backpropagation from c_t to c_{t-1} : No matrix multiplication
  - Avoid vanishing gradient problem

LSTM: $c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t = \sigma\left(W_f \left[h_{t-1}, x^{[t]}\right] + b_f\right)$$

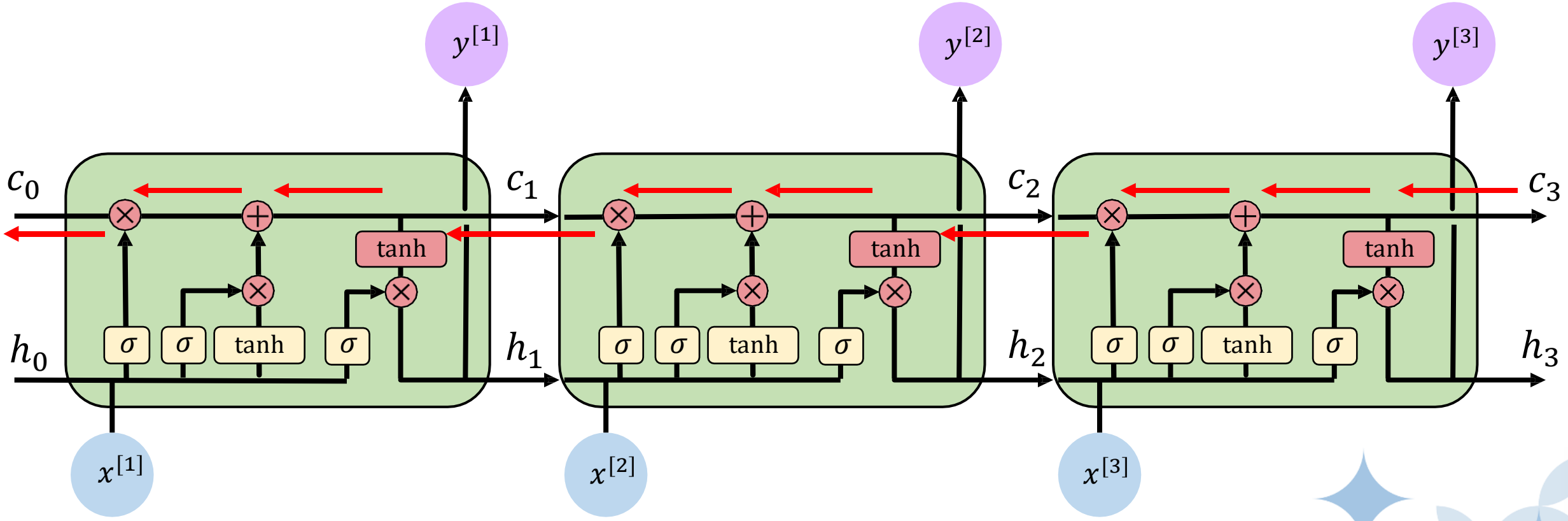
RNN: $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1})$

$$\frac{\partial h_t}{\partial h_{t-1}} = W_{hh} \tanh'(W_{xh}x_t + W_{hh}h_{t-1})$$



LSTM: Gradient Flow

- Uninterrupted gradient flow!



LSTMs: keyconcepts

- Maintain a separate cell state from what is outputted

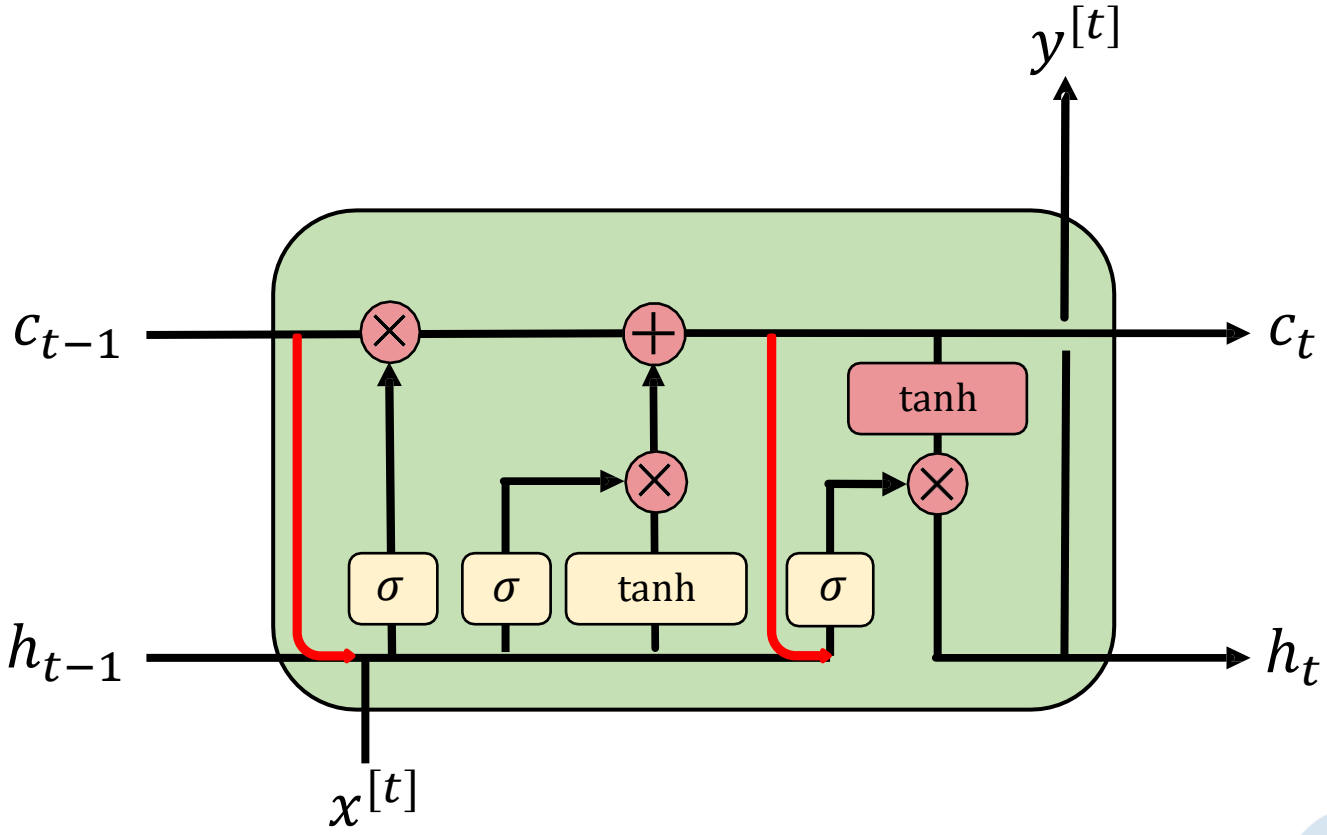
- Use gates to control the flow of information
  - Forget gate gets rid of irrelevant information
  - Input gate adds information from the new data point
  - Selectively update the cell state
  - Output gate returns a filtered version of the cell state

- Backpropagation from c_t to c_{t-1} requires only elementwise multiplication!
- Uninterrupted gradient flow! - easier way to learn long-term dependencies

- LSTM is still forgetful: impossible to represent a long sequence with a compact memory vector

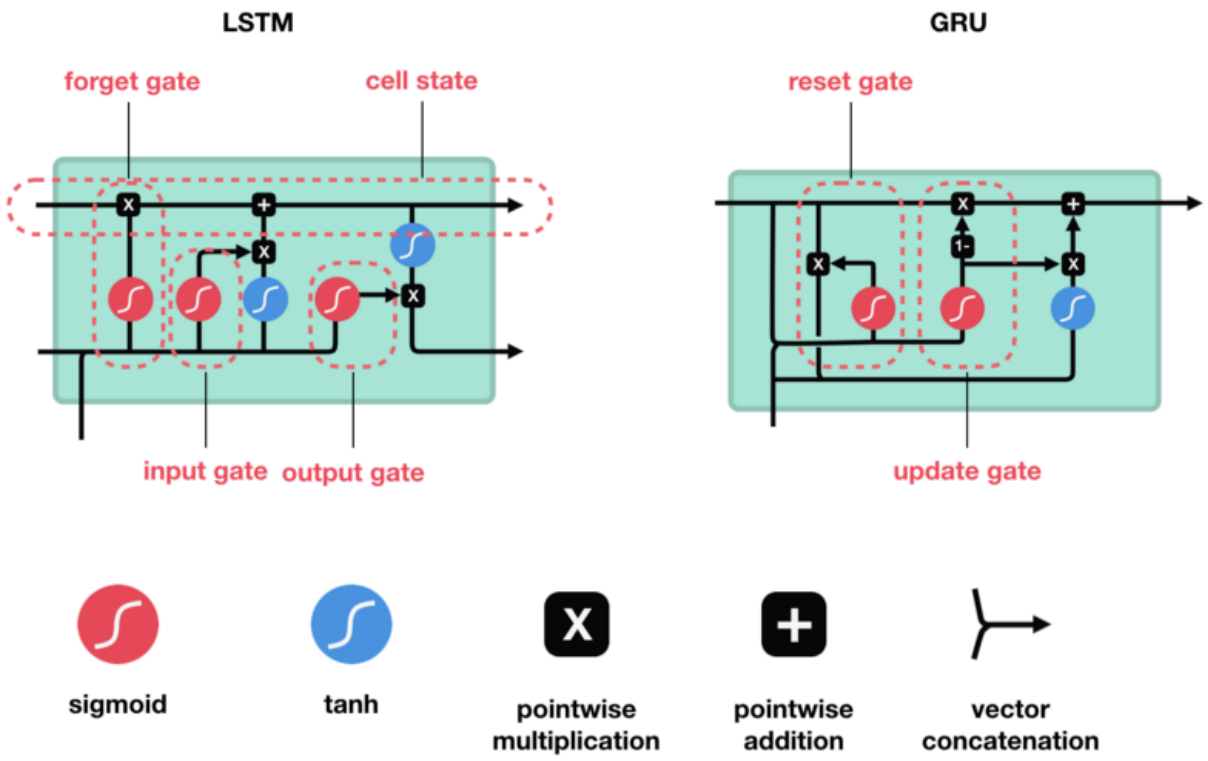
Other variants of LSTM: "Peephole" Connection

- Gers and Schmidhuber (2000)
  - Allow the gate layers look at the cell state



Other variants of LSTM:

- Gated Recurrent Unit:
  - simpler than LSTM, very popular, Cho et al, (2014):
  - combines forget and input gates
  - merges cell state & hidden state

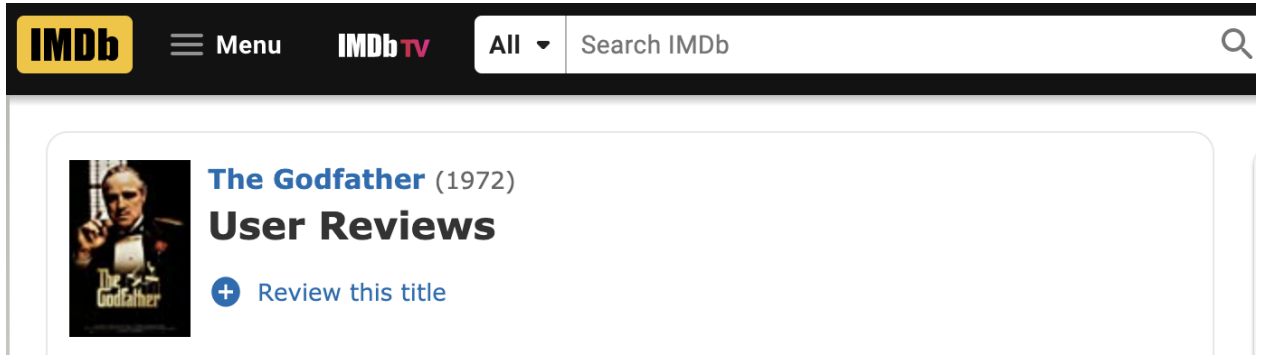


- Multiplicative LSTM (mLSTMs), Krause et al 2016
- etc.

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Practice Problem: Sentiment Analysis

IMDB Movie Review



★ 10/10

Amazing movie
danielfeerst 22 January 2020

The acting was simply amazing, what else could you say. What could be more appealing to people(even today) than watching actors like Al Pacino, Marlon Brando, James Caan, Diane Keaton, Talia Shire and Robert Duvall. This is like heaven for someone who is a fan of movies. With this movie Brando was able to bring himself back into the limelight. His performance as the godfather alone is iconic. His character has been recreated so much in films that it has almost if it has not already become a cliché. His performance though was not a cliché. His performance was subtle and breathtaking. It was so genuine and realistic that it was not just probably but definitely more genuine than Marlon Brando himself. Al Pacino was perfect for this film as well. What a way to start up your career. His character was all about depth and he displayed it perfectly. He was able to display his own inner-battles in his mind as well as the battles he had with his family, friends and

★ 1/10

Massively overrated.
jojofla 5 August 2002

I continually fail to understand why The Godfather is hailed as "The Greatest Movie of All Time". I've seen it twice--a second time just to make sure--and I have to tell you that I sat there in a stupor, bored out of my mind. And I'm not a teenager raised on MTV; I'm in my 30s and am absolutely devoted to movies--I've seen as many classics (American & foreign) that I can get my hands on. But, for me, The Godfather ranks alongside Singin' in the Rain as the most overrated films of all time.

Singin' in the Rain, at least, I get (it's just my intense dislike for Donald O'Connor that makes me dislike this film). But The Godfather? It's just a bland epic about a bunch of moronic gangsters, with Marlon Brando giving a campy performance, and riddled with repulsive violence. Give me a break. The fact that this movie is so "beloved" has had the direct result that nowadays we got absurdly worse and worse films every year, created by clueless filmmakers.

Practice Problem: Sentiment Analysis

- This is a dataset of movies reviews from IMDB
  - labeled by sentiment (positive/negative).
  - 25,000 for training and 25,000 for testing

- Built-in Dataset by Pytorch
  - https://docs.pytorch.org/text/master/datasets.html

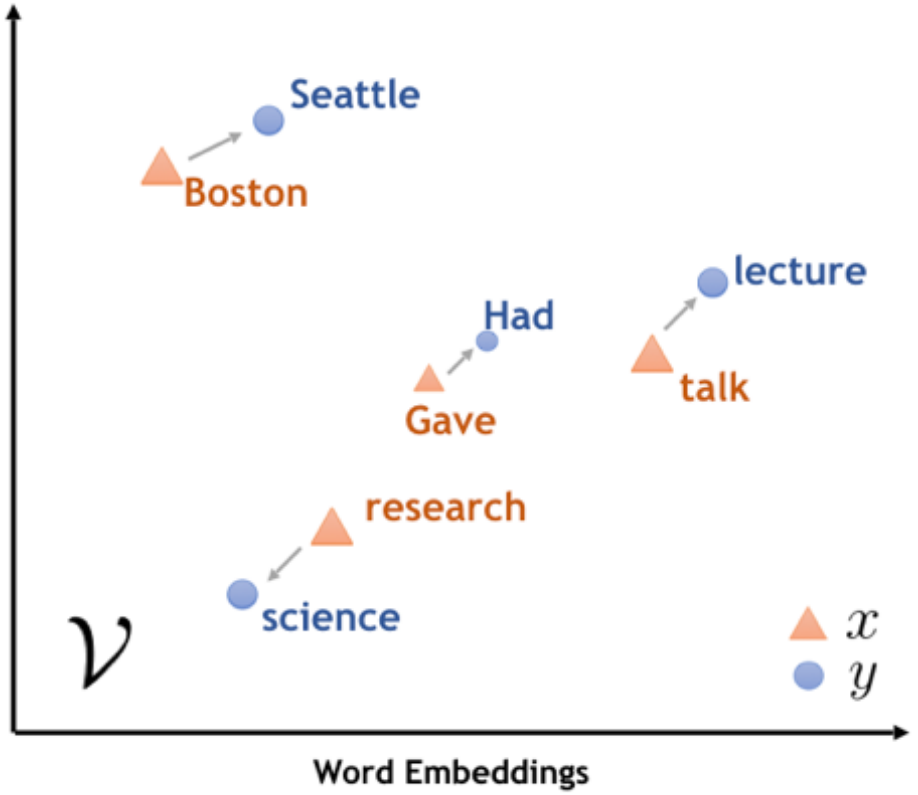
- Preprocessing: Text to Sequence
- Word Embedding: Word to Vector

Preprosessing Text Data

1. Tokenization (Text to Words)
   - Given a piece of text (string), e.g., S = "…sentiment analysis is the use of natural language processing"
   - Break the string (string) into a list of words:
     L = [..., sentiment, analysis, is, the, use, of, natural, language, processing, ...]
2. Build a dictionary (e.g. hash table) to count words' frequencies
   - We may choose to keep only the 10,000 most frequent words
3. Map each word to its index
   - L = [..., sentiment, analysis, is, the, use, of, natural, language, processing, ...]
   - L = [..., 310, 23, 6, 432, 93, 14, 359, 135, 437, ...]
   - Encode words not appearing in the dictionary to 0
4. One-Hot Encoding:
   - 6 -> [0,0,0,0,0,0,1,0,…]

Embedding Layer

- Directly feed the one-hot encoded input to RNN/LSTM?
  - If dictionary size is 10,000 and text length is 100
  - Input has dimension $= 10{,}000 \times 100 = 1\ \text{million}$
- map the one-hot vectors to a low-dimensional vectors
  - Map to a 2-dimension space, the input of RNN has dimension $= 2 \times 100 = 200$
  - Map to d-dimension space: $d \times \text{text\_length}$
  - Dimension Reduction
- Embedding in classical ML: PCA, t-SNE, IsoMap
  - NN: build embedding layers into the network and learn the weights -
  - From a dictionary to d -dim output: we have $\text{dictionary\_size} \times d$ weights to learn



Build the RNN model

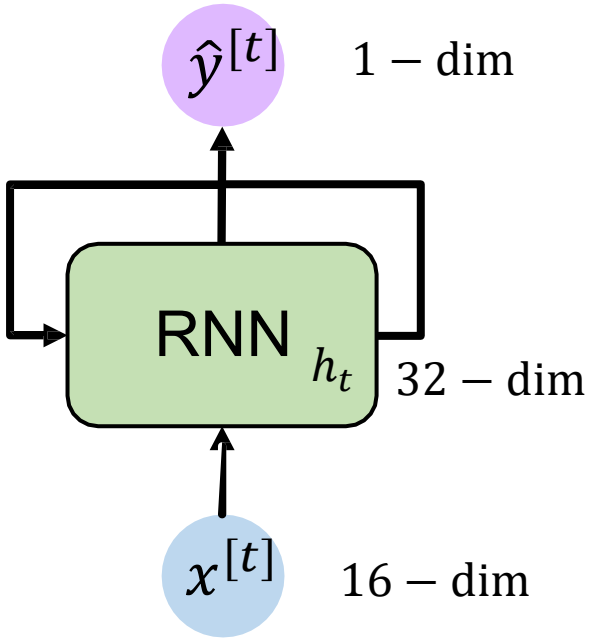
```python
class SimpleRNNBinaryClassifier(nn.Module):
    def __init__(self, vocab_size=20_000, embed_dim=16, hidden_size=32):
        super().__init__()
        self.embedding = nn.Embedding(num_embeddings=vocab_size,
                                      embedding_dim=embed_dim)
        # torch.nn.RNN
        self.rnn = nn.RNN(input_size=embed_dim,
                          hidden_size=hidden_size,
                          batch_first=True)   # input shape: (batch, seq_len, embed_dim)
        self.fc = nn.Linear(hidden_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        """
        x: LongTensor of shape (batch, 80) containing token ids in [0, vocab_size-1]
        """
        x = self.embedding(x)          # (batch, 80, 16)
        out, h_n = self.rnn(x)         # out: (batch, 80, 32), h_n: (1, batch, 32)

        # torch.nn.RNN returns the last hidden state as h_n
        last = h_n[-1]                 # (batch, 32)  (equivalently: out[:, -1, :])

        logits = self.fc(last)         # (batch, 1)
        probs = self.sigmoid(logits)   # (batch, 1)
        return probs
```

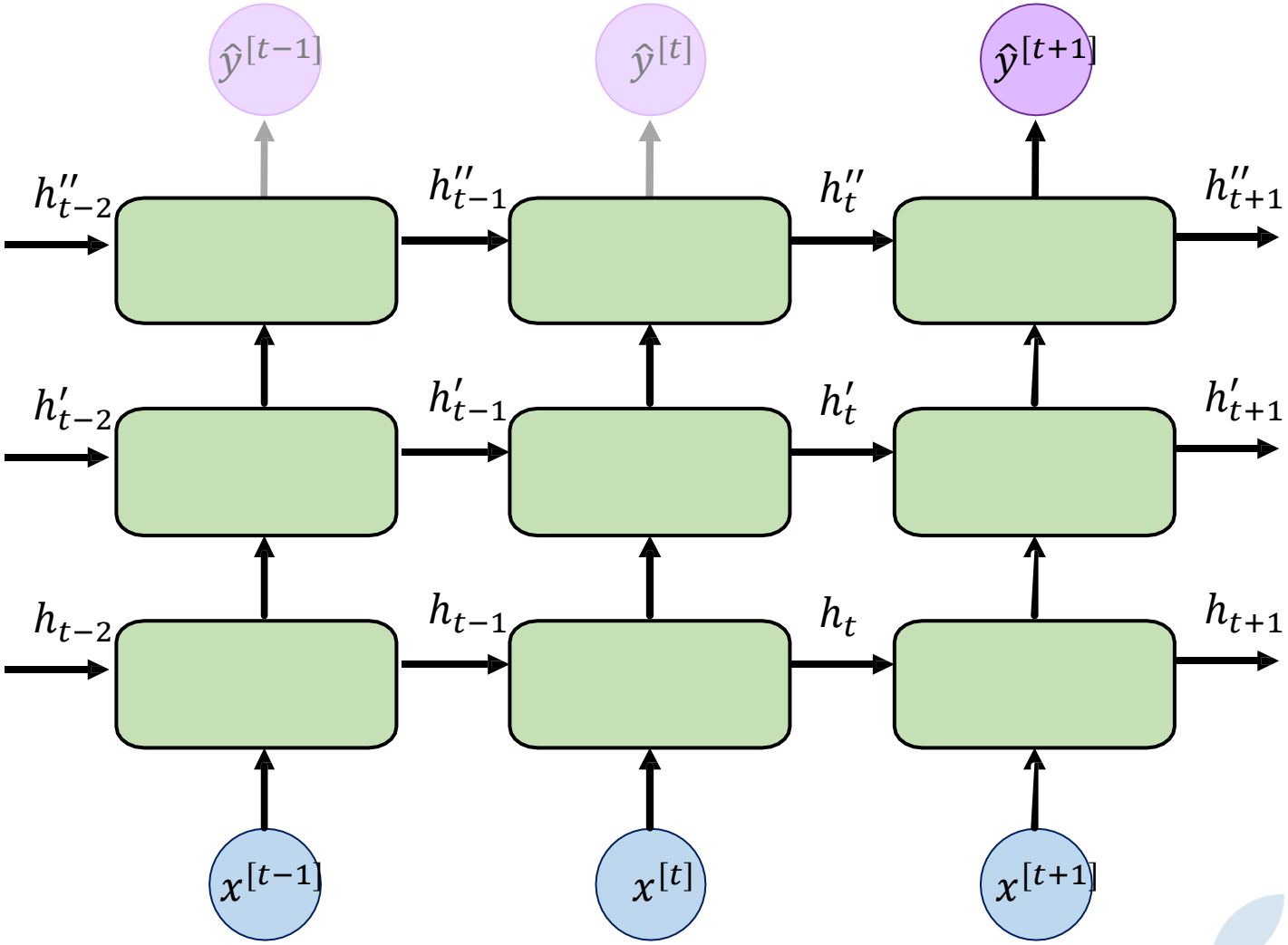
✓ 0.0s



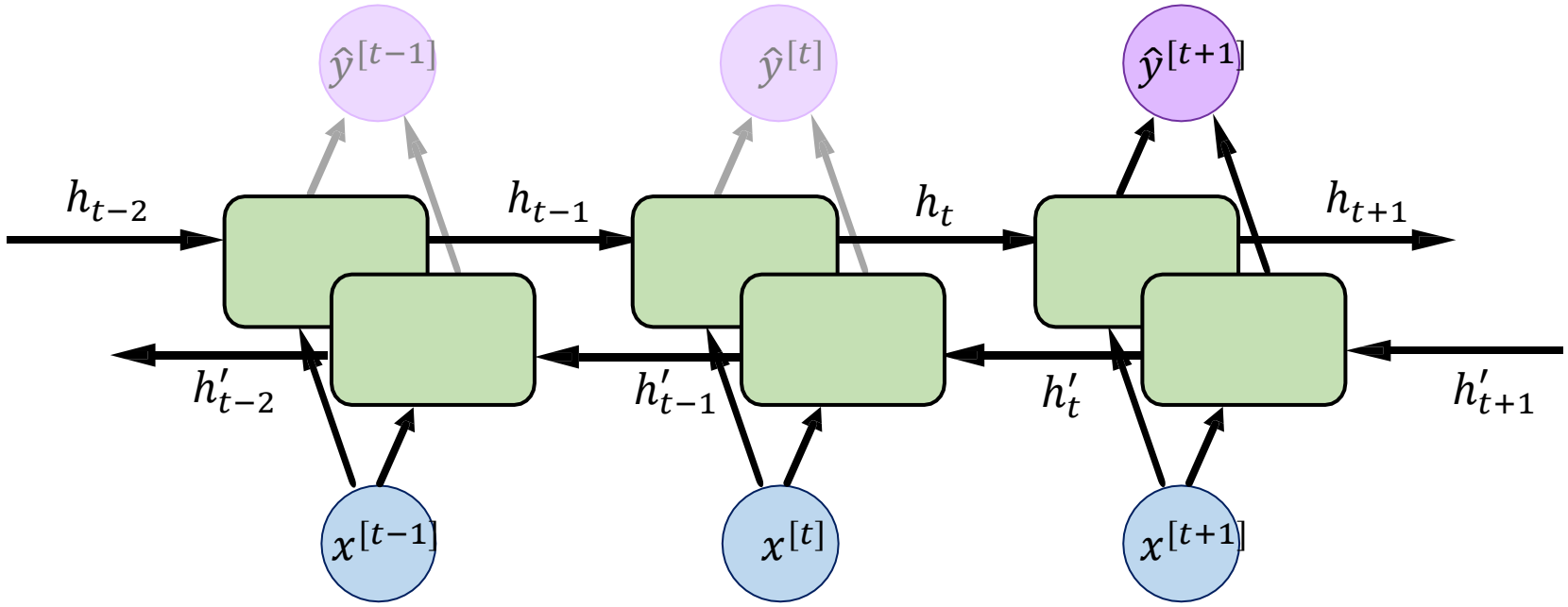
Typical Tricks of RNN/LSTM

- Stacked RNN / LSTM
- Bidirectional RNN /LSTM
- Pretraining / Multi-task Learning
- Attention (Self-Attention)

Stacked RNN

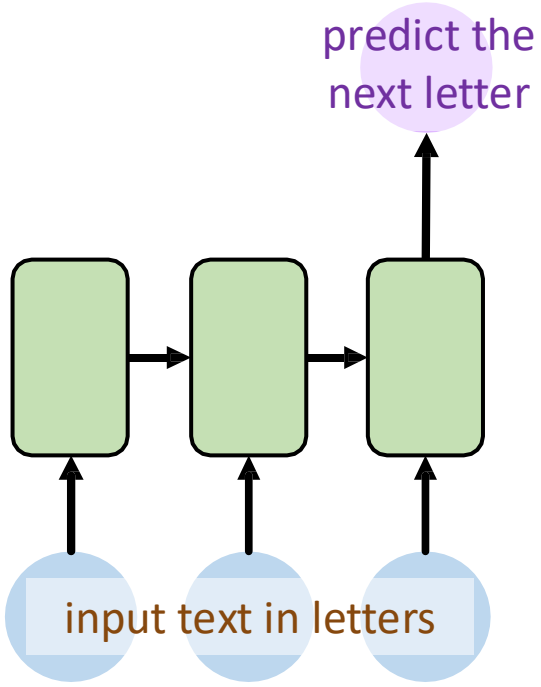

$\hat{y}^{[t-1]}$
$\hat{y}^{[t]}$
$\hat{y}^{[t+1]}$
h''_{t-2}
h''_{t-1}
h''_{t}
h''_{t+1}
h'_{t-2}
h'_{t-1}
h'_{t}
h'_{t+1}
h_{t-2}
h_{t-1}
h_{t}
h_{t+1}
$x^{[t-1]}$
$x^{[t]}$
$x^{[t+1]}$

Bidirectional RNN



Keras: model.add(layers.Bidirectional())

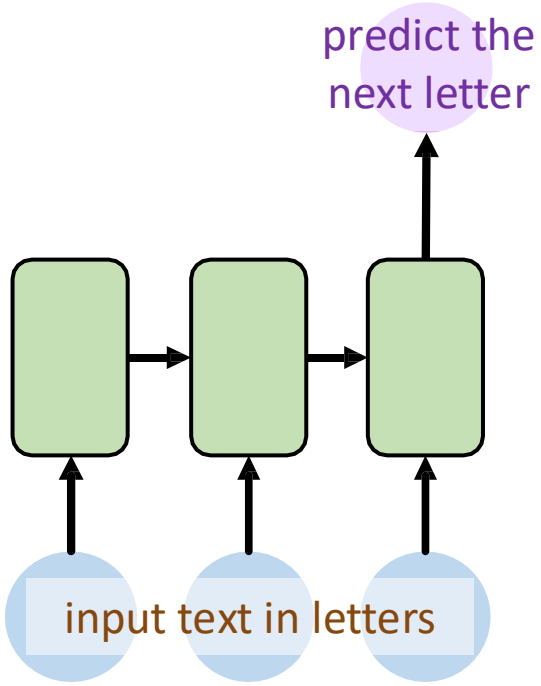
Practice Problem: Text Generation



Sequence to One
Text Generation

- Example Input: " `the cat sat on the ma` "
- Predict the next character

Practice Problem: Text Generation

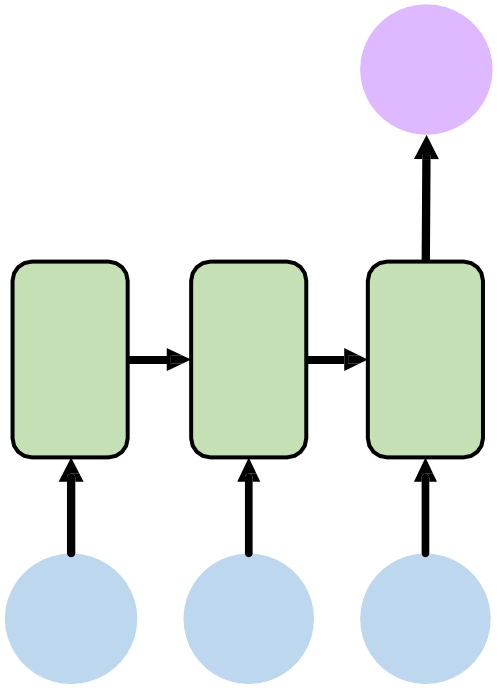


Sequence to One
Text Generation

- Input: " `the cat sat on the ma` "
- Predict the next character

- Tokenization (Text to Characters, char-level)
  - Text generation uses char-level
  - Sentiment analysis uses word-level (not uses char-level, but text generation can also use word-level)
- In word-level, dictionary contains all words appears frequently in data
- In char-level, dictionary is just those characters appeared in data
  - Letters: Aa,Bb,Cc...,Zz
  - Numbers, 1,2,3,…
  - Marks: dot(.), comma(,), question mark(?), space( ), …
  - The dictionary is much smaller.

Practice Problem: Text Generation



Sequence to One
Text Generation

- Predict a sequence of text one by one
  "Natural language generation is a software proc"
- Predict the next character
  - "proce"
- Recurrently, predict the next character
  - "proces"
  - "process"

- If the RNN is trained on Nietzsche's books, then the generated text is Nietzsche's style.

Practice Problem: Text Generation

- Generate the training data: cut the text in overlapping segments of maxlen=40 characters with a jump size of 3

```
Recurrent neural networks are artificial neural networks that use sequential data.
```

```
Recurrent neural networks are artificial neural networks that use sequential data.
```

```
Recurrent neural networks are artificial neural networks that use sequential data.
```

Practice Problem: Text Generation

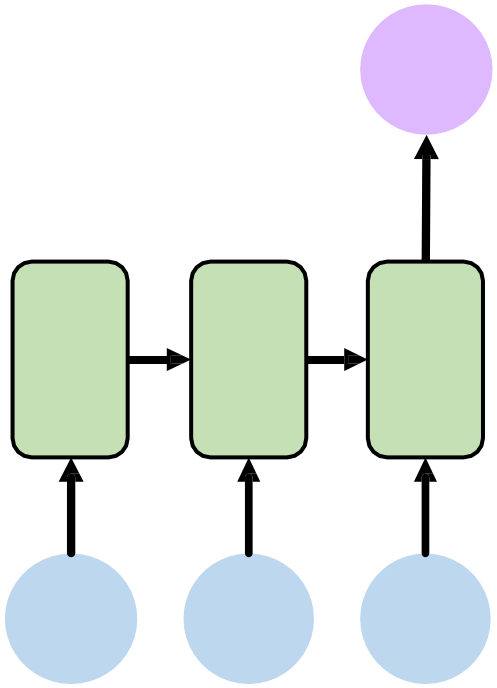
- Generate the training data: cut the text in overlapping segments of maxlen=40 characters with a jump size of 3

```
Recurrent neural networks are artificial neural networks that use sequential data.
Recurrent neural networks are artificial neural networks that use sequential data.
Recurrent neural networks are artificial neural networks that use sequential data.
```

- A segment is used as an input text.
- Its next character is used as label.

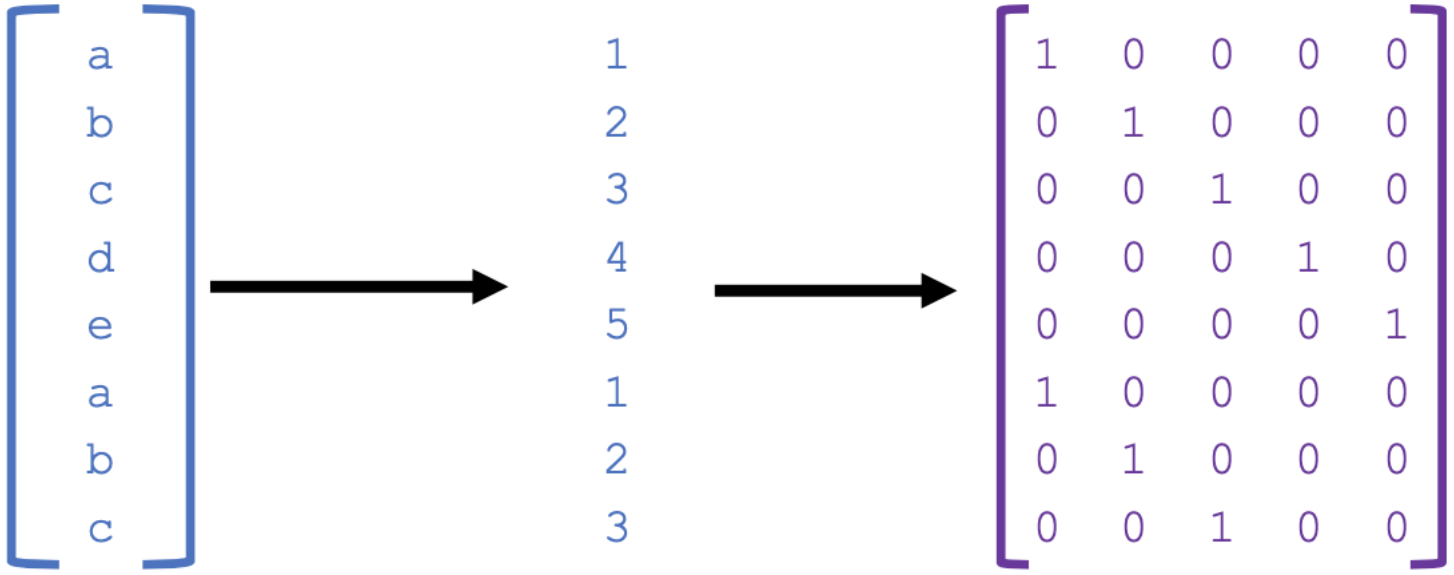
Training data: (**segment**, **next_char**) pairs

Practice Problem: Text Generation

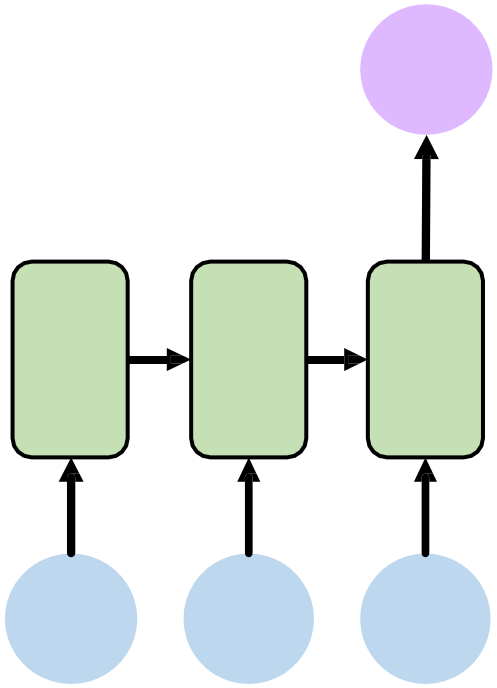


Sequence to One
Text Generation

- Training data: (segment, next_char) pairs
- This is a Multi-class classification problem.
  - #class = #unique chars.
  - One-hot Encoding



Practice Problem: Text Generation



Sequence to One
Text Generation

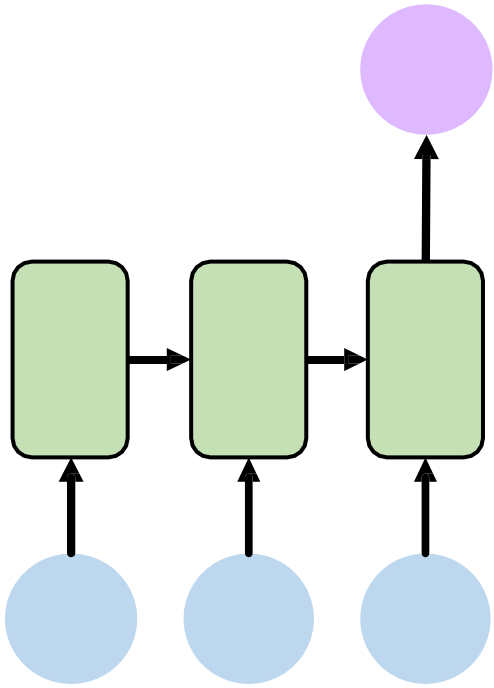
- Training data: (segment, next_char) pairs
- This is a Multi-class classification problem.
  - #class = #unique chars.
- Build an LSTM model with

You don't need an embedding layer!

  - Input layer shape with `(maxlen, 57)`
  - 128 nodes in the LSTM layer
  - Dense layer to 57 nodes in output (57 unique characters)

```
_________________________________________________________________
Layer (type)                 Output Shape
=================================================================
lstm (LSTM)                  (None, 128)
_________________________________________________________________
dense (Dense)                (None, 57)
_________________________________________________________________
```

Practice Problem: Text Generation



Sequence to One
Text Generation

Initial input (seed)

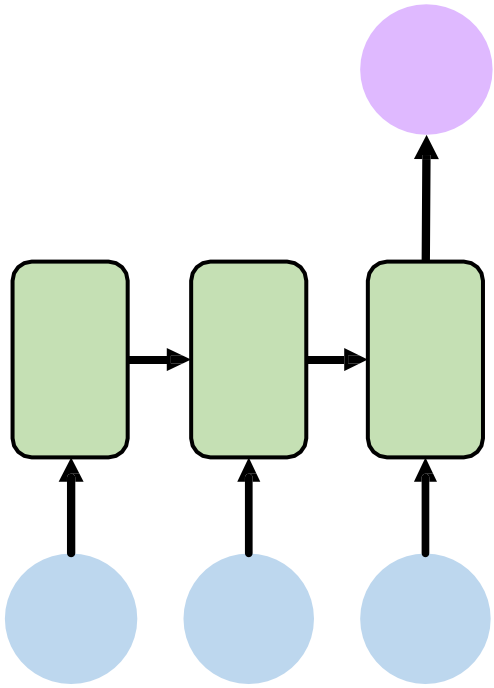
"immediately disclose what it really is--namely, a will to the"

Generated text after 1 epoch

"immediately disclose what it really is--namely, a will to the the believest constive the art of the persision the says of a gan himself need not religion a consting be naturing and suld the pretendents as the constion. the are the good teach that the most and find of endent and the self it of instinct a mean constition of can the constive in this sour from the bad and all the philosophy to condividuation men and the goence the condines the all as most strat"

credit: Shusen Wang

Practice Problem: Text Generation



Sequence to One
Text Generation

Initial input (seed)

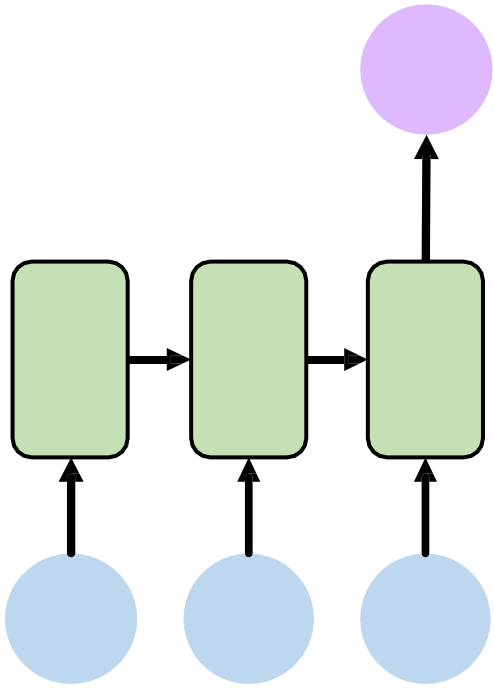
"immediately disclose what it really is--namely, a will to the"

Generated text after 20 epochs

"immediately disclose what it really is--namely, a will to the self-religious superitial self-partial religion himself of the superstition of the contrast in the accomant in the person of the assertion, at the valuations and consequences and things has no longer and how only an man of the soul and manifest of the disciplides and an whom all to the constance of its own power, in the constraining in the truth of the strength of life of the see to remain in the"

credit: Shusen Wang

Practice Problem: Text Generation

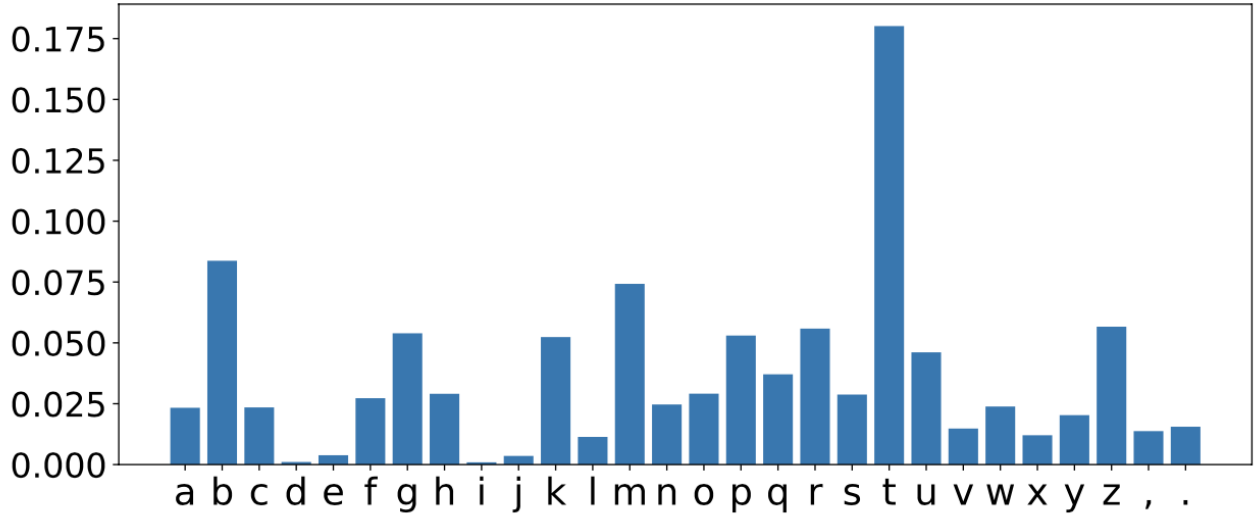


Sequence to One
Text Generation

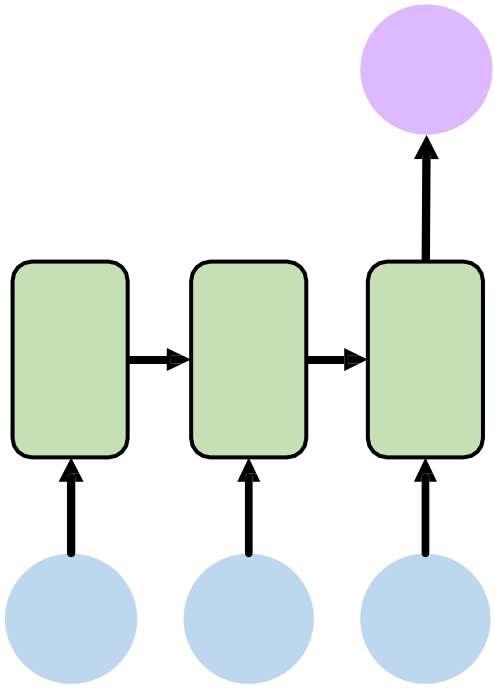
Layer (type)	Output Shape
lstm (LSTM)	(None, 128)
dense (Dense)	(None, 57)

```
pred = model.predict(x_input)[0]
```

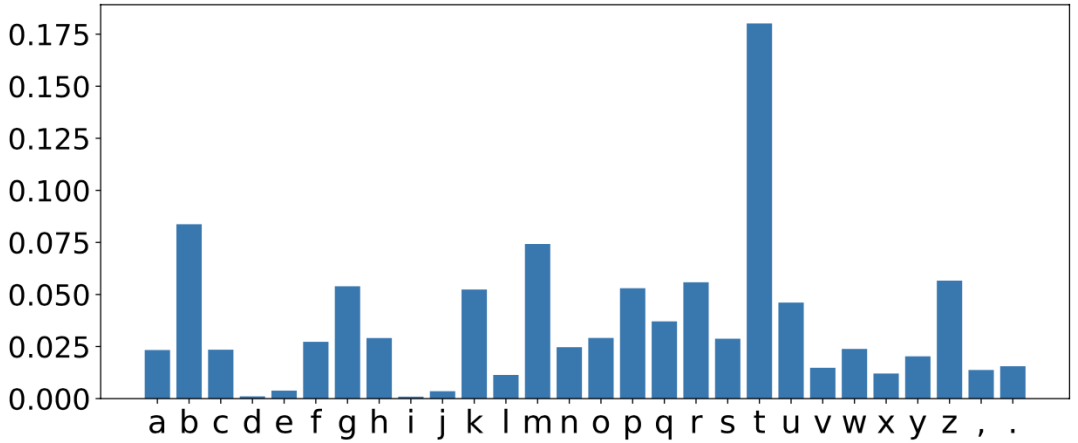
- How to predict the next character



Practice Problem: Text Generation

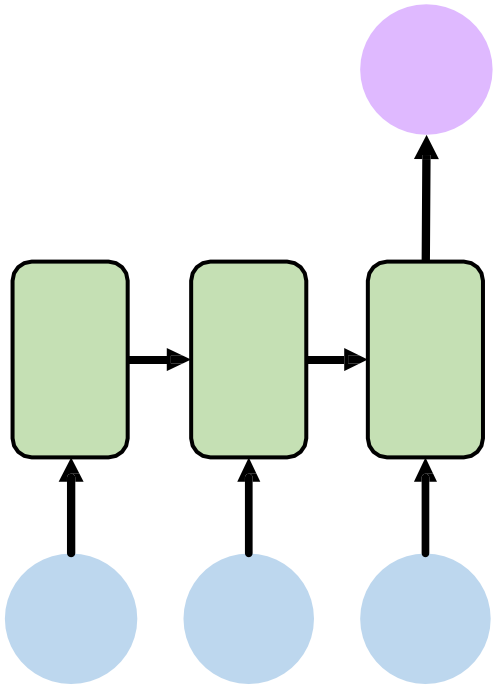


Sequence to One
Text Generation



- How to predict the next character
  - Deterministic selection — deterministic, performance bad
    - `np.argmax(pred)`
  - Sampling from the multinomial distribution — Too random, also bad
    - `np.argmax(np.random.multinomial(1,pred,1))`
  - Adjusting the multinomial distribution
    - `pred=pred**(1/temperature)` — power operator
    - `pred=pred/np.sum(pred)`
    - `np.argmax(np.random.multinomial(1,pred,1))` — `temperature` controls tradeoff

Practice Problem: Text Generation



Sequence to One
Text Generation

- If the RNN is trained on Nietzsche's books, then the generated text is Nietzsche's style.
- Homework

Build the model - fill in this box

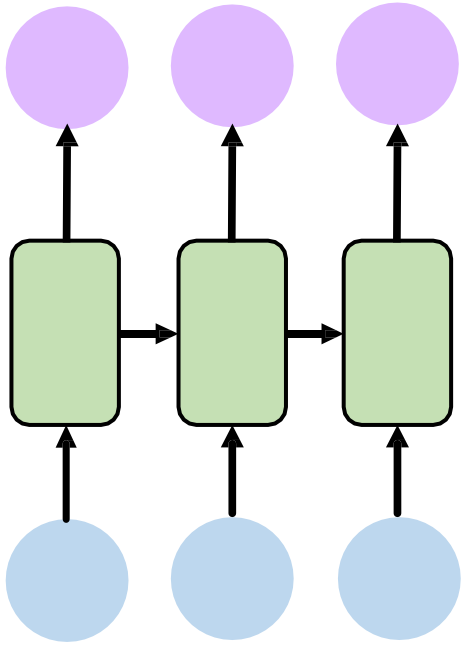
we need a recurrent layer with input shape (maxlen, len(chars)) and a dense layer with output size len(chars)

```
[ ] Start coding or generate with AI.
```

- Data is preprocessed, `model.fit` is also provided

You don't have to run all epochs for training (slow). You can stop the code after 1 epoch and submit the notebook.

Application: Neural Machine Translation



Seq to Seq

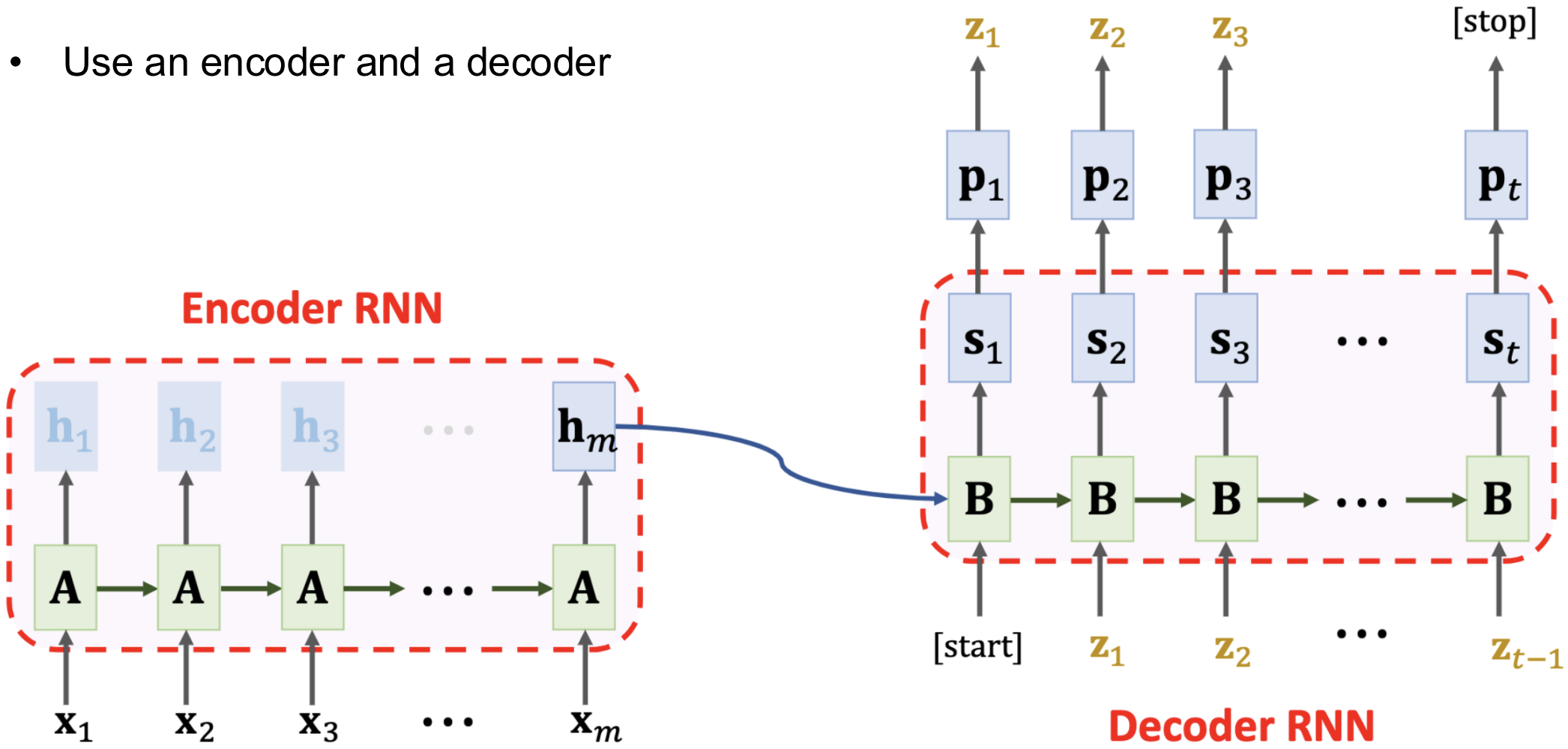
Machine Translation

English	German
Hello World	Hallo Welt
Machine Learning	Maschinelles Lernen
Neural Network	Neurales Netzwerk
What are you doing	Was machst du

The input and output are both sequences (longer than one)

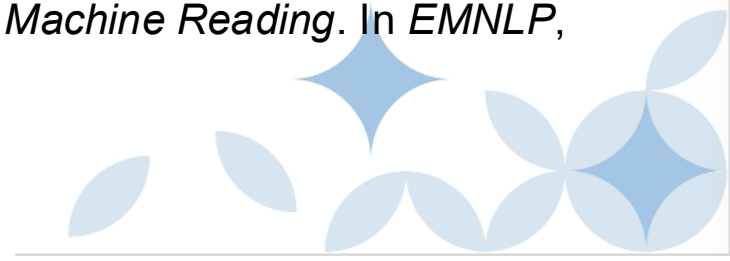
Application: Neural Machine Translation

- Use an encoder and a decoder



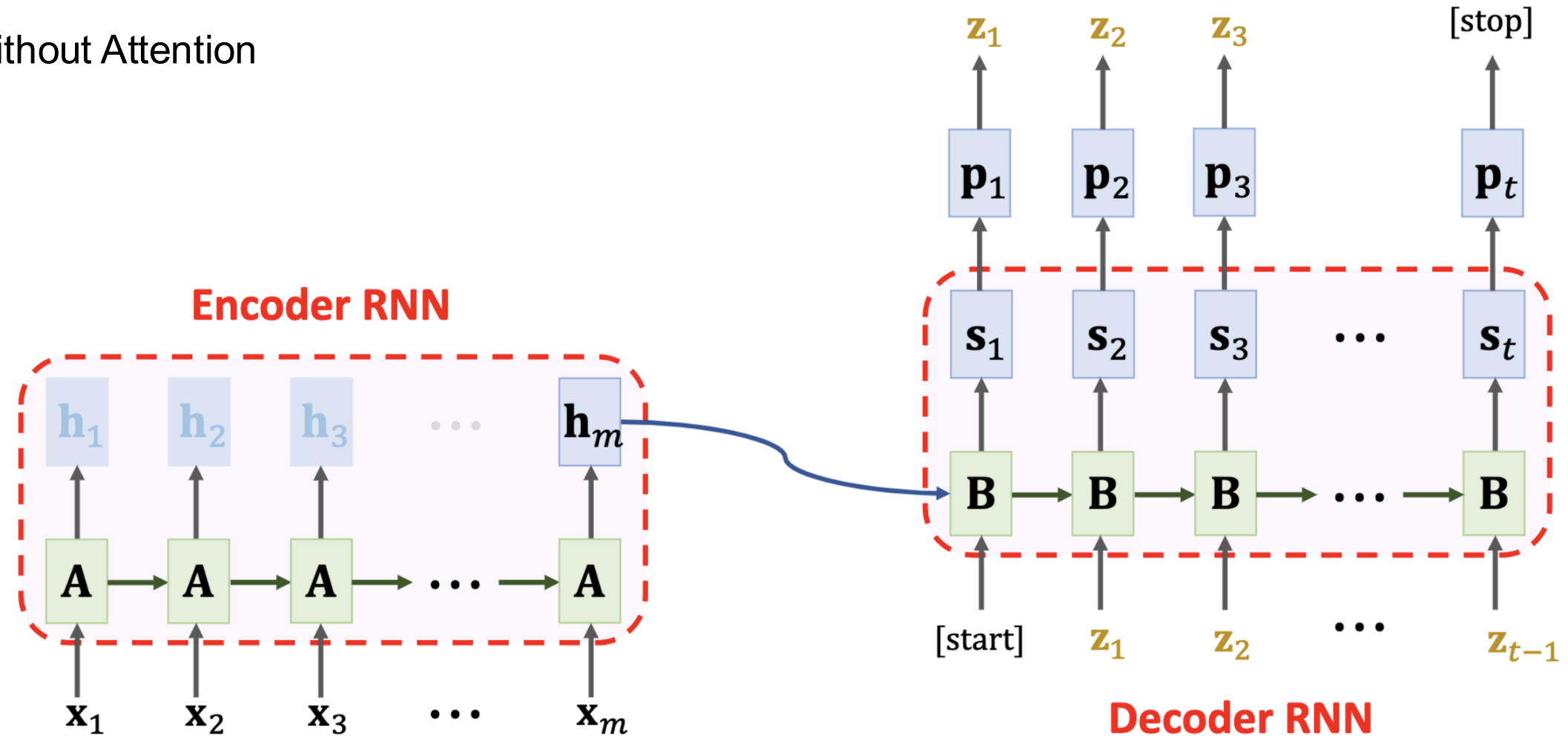
Attention / Self-Attention

- Attention
  - In a Seq2Seq model, the final state in the encoder cannot remember a long sequence
    - The decoder looks at only the current state of the encoder
  - Attention: decoder looks at all the states of the encoder
    - largely improves Seq2Seq model but requires more computation
  - Bahdanau, Cho, & Bengio. *Neural machine translation by jointly learning to align and translate*. In *ICLR*, 2015.
  - Self-Attention, Cheng, Dong, & Lapata. *Long Short-Term Memory-Networks for Machine Reading*. In *EMNLP*, 2016.
  - Transformer, Vaswani et al, *Attention is all you need*. In *NIPS*, 2017



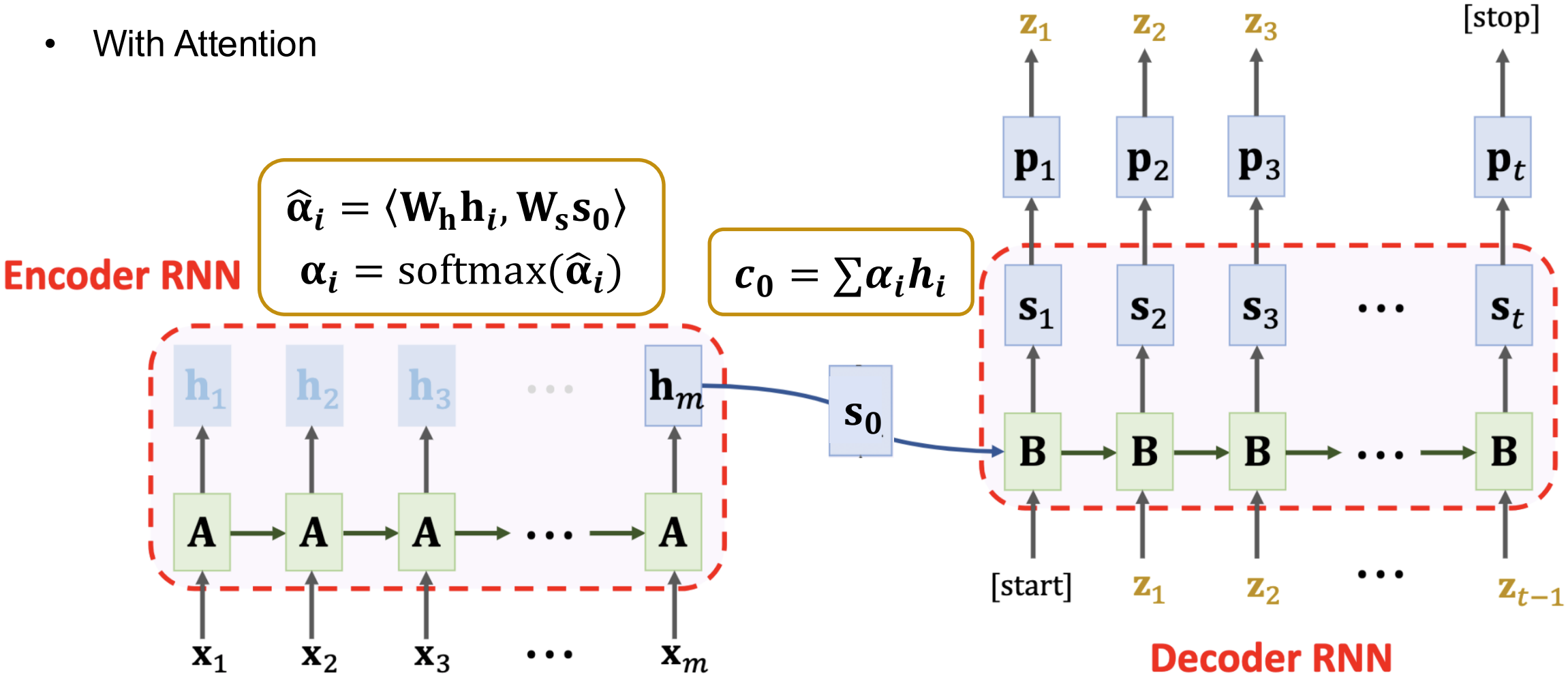
Attention

- Without Attention



Attention

- With Attention



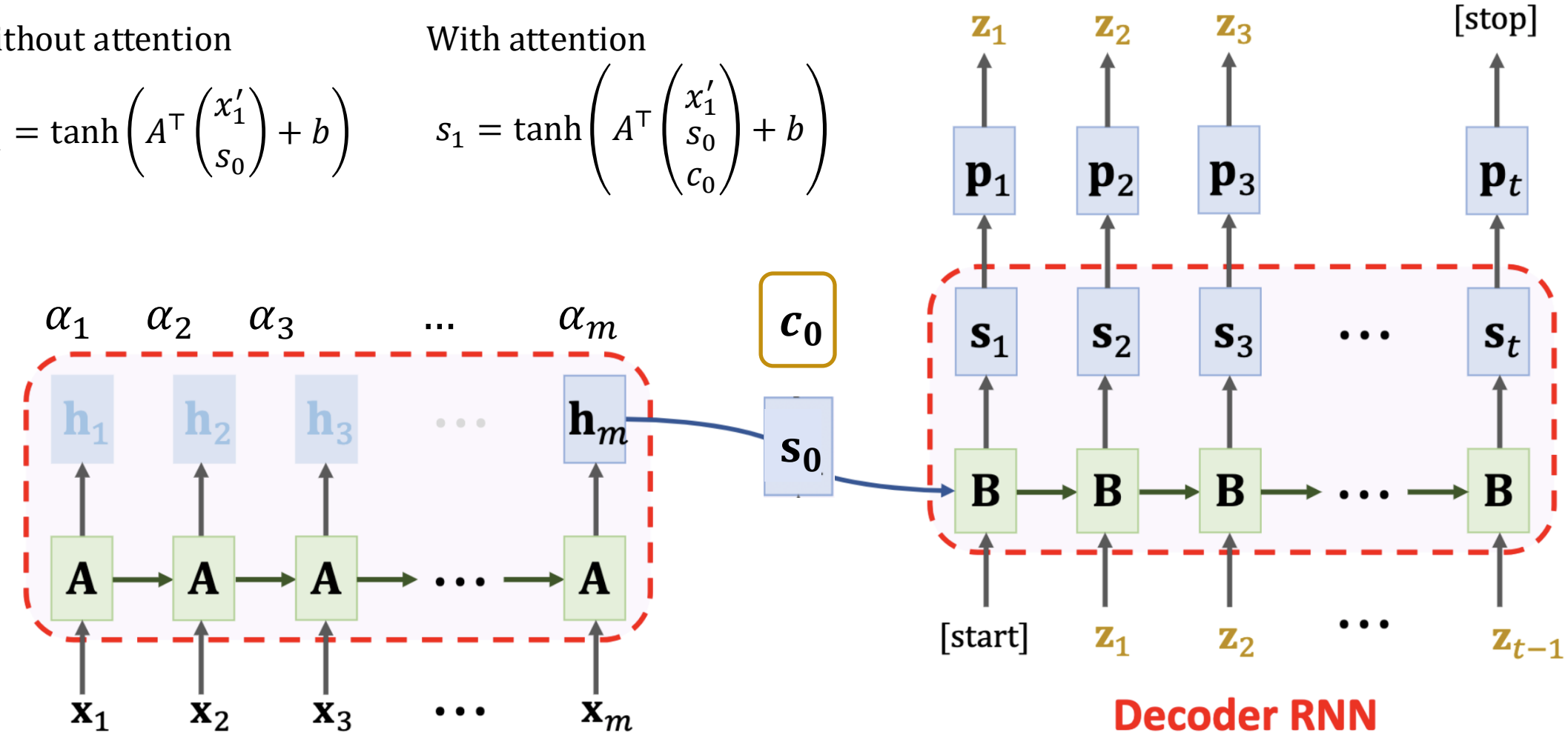
Attention

Without attention

$$s_1 = \tanh\left(A^\top \begin{pmatrix} x_1' \\ s_0 \end{pmatrix} + b\right)$$

With attention

$$s_1 = \tanh\left(A^\top \begin{pmatrix} x_1' \\ s_0 \\ c_0 \end{pmatrix} + b\right)$$



Questions?