Deep Learning Fundamentals

Takuma Kimura

Chapter 3
Recurrent Neural Network

1. What is RNN?

Analyzing Longitudinal Trends

I want to learn deep learning because I am…

Recurrent Neural Network (RNN)

- A neural network with a recursive structure.

  “Recursive”: A thing's definition or description contains the thing itself.

- The recursive structure enables RNNs to handle longitudinal data.

  e.g., Stock price data

  Time-lagged data in psychological research

Longitudinal Data

Employee Motivation Data

Not Longitudinal

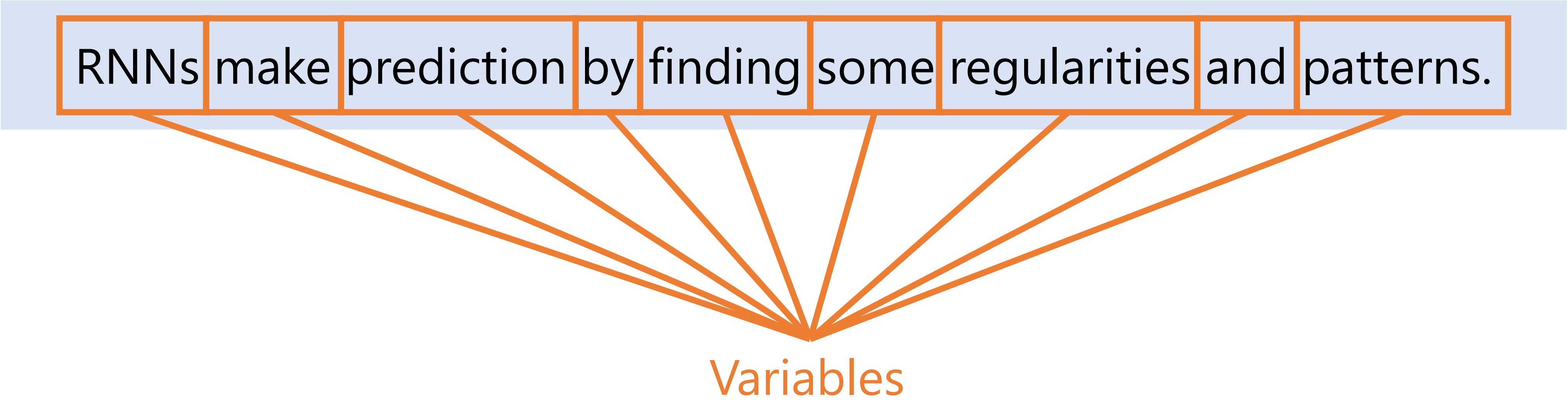
Employee ID	Motivation
Emp1	4
Emp2	5
Emp3	3
Emp4	3
Emp5	2
Emp5	5
emp6	4

Longitudinal Data

Employee ID	Time 1	Time 2	Time 3	Time 4	Time 5
Emp1	4	3	5	4	3
Emp2	5	5	4	5	4
Emp3	3	3	2	3	3
Emp4	3	2	4	3	2
Emp5	2	2	1	2	1
Emp5	5	5	4	4	5
emp6	4	3	2	3	4

Analyzing Longitudinal Data by RNN

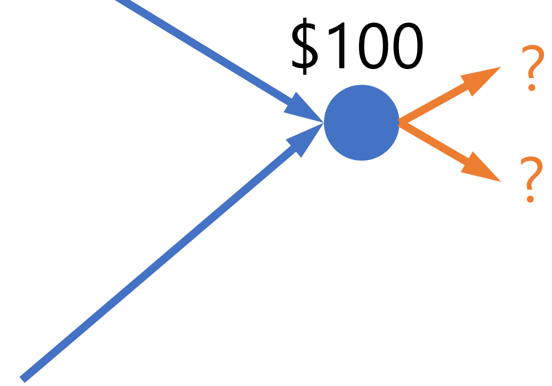
- RNNs make prediction by finding some regularities or patterns in the variation of sequenced variables.



Importance of Past State

- In a longitudinal analysis, the model needs to find regularities and pattern in a sequence.

Stock Price

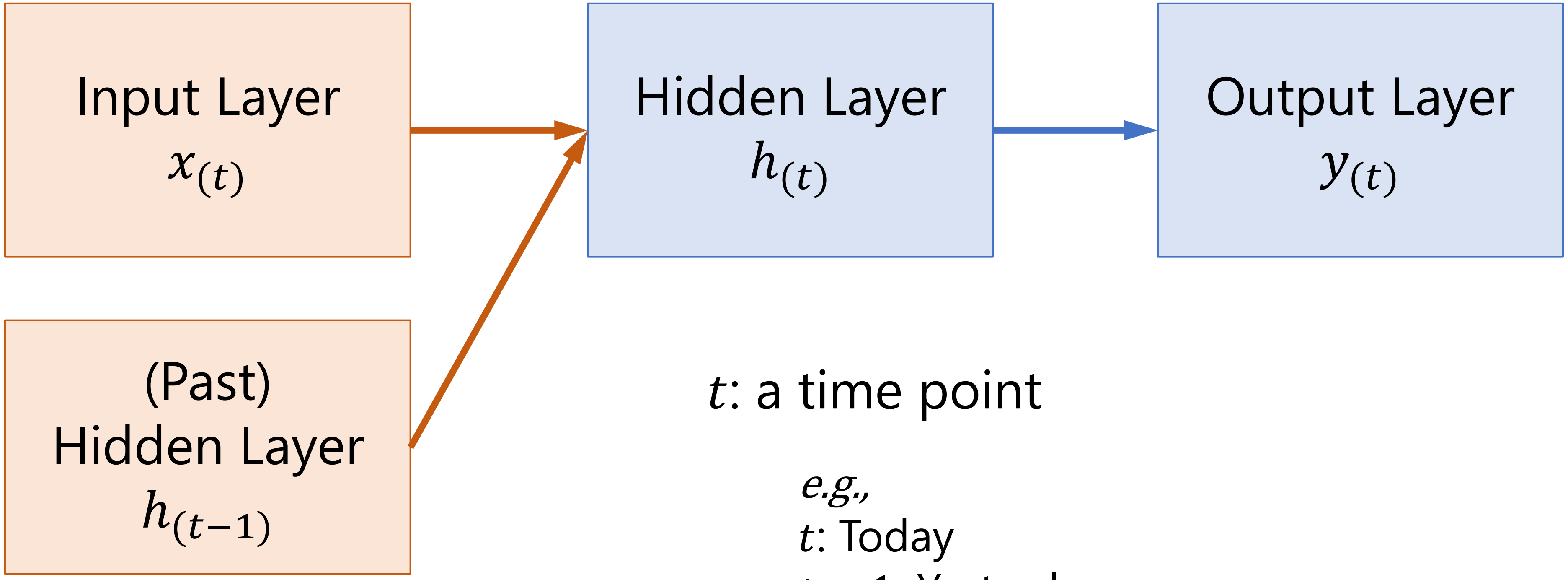


- You may do it well.
- You may well do it.

- There is a cloud in the sky.
- There is a stone in the ____.

2. Structure of RNN?

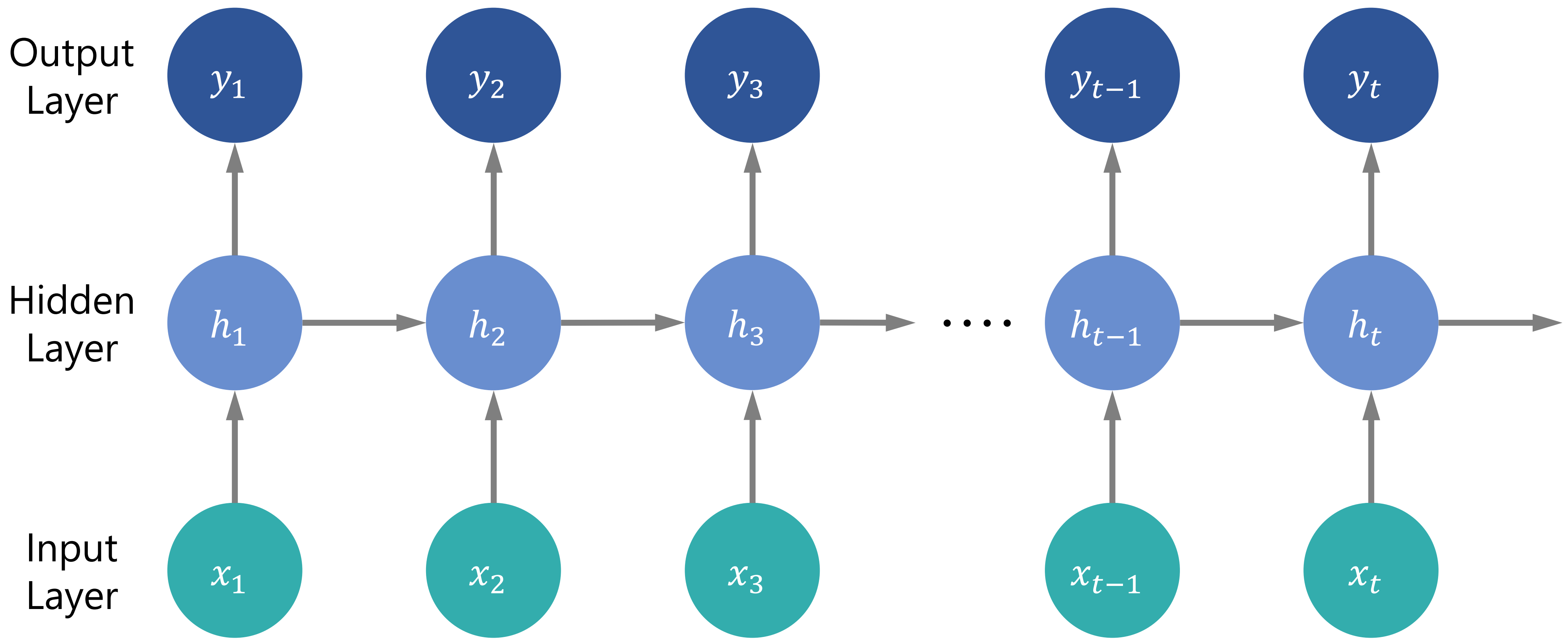
Basic Structure of RNN



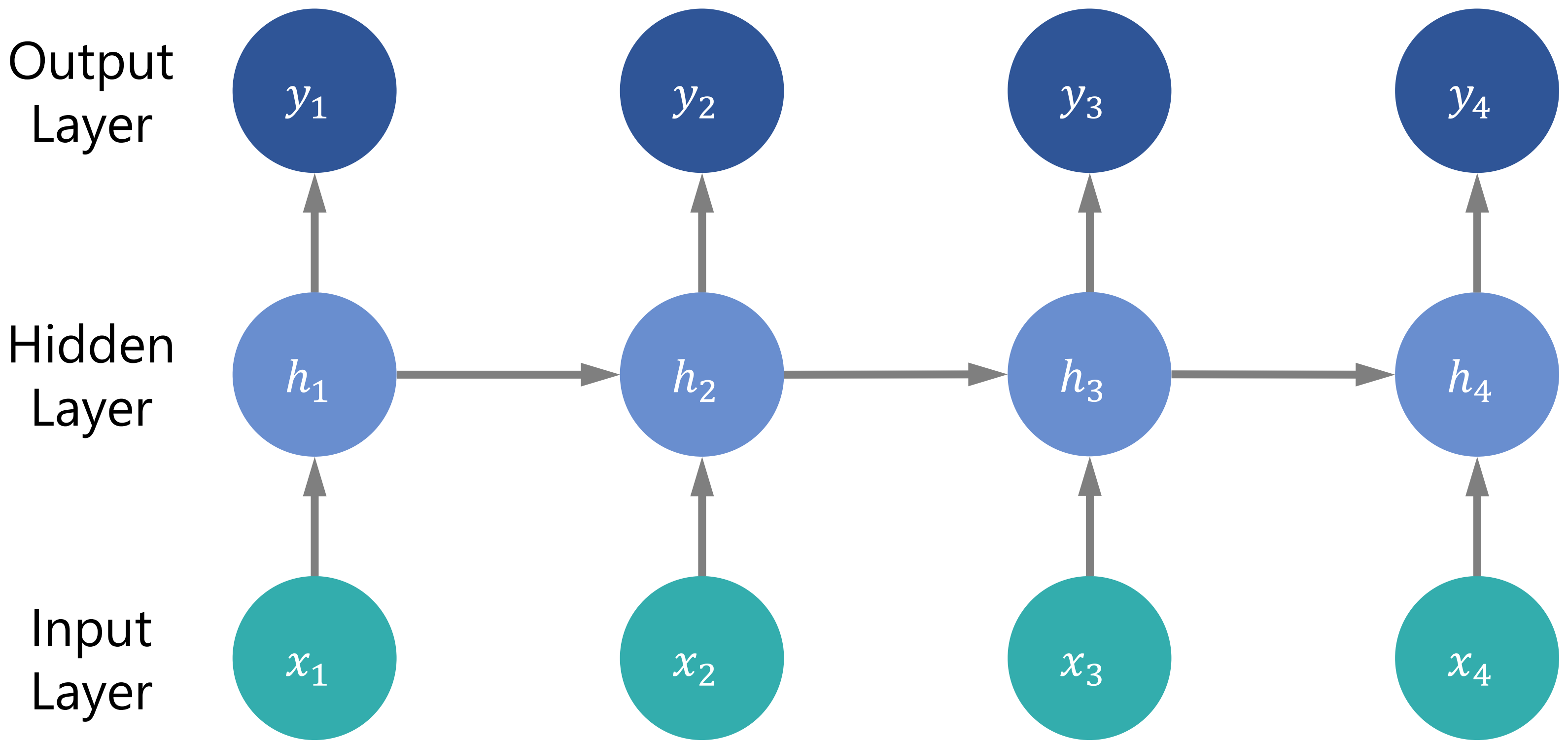
t: a time point

e.g.,
t: Today
$t-1$: Yesterday
$t+1$: Tomorrow

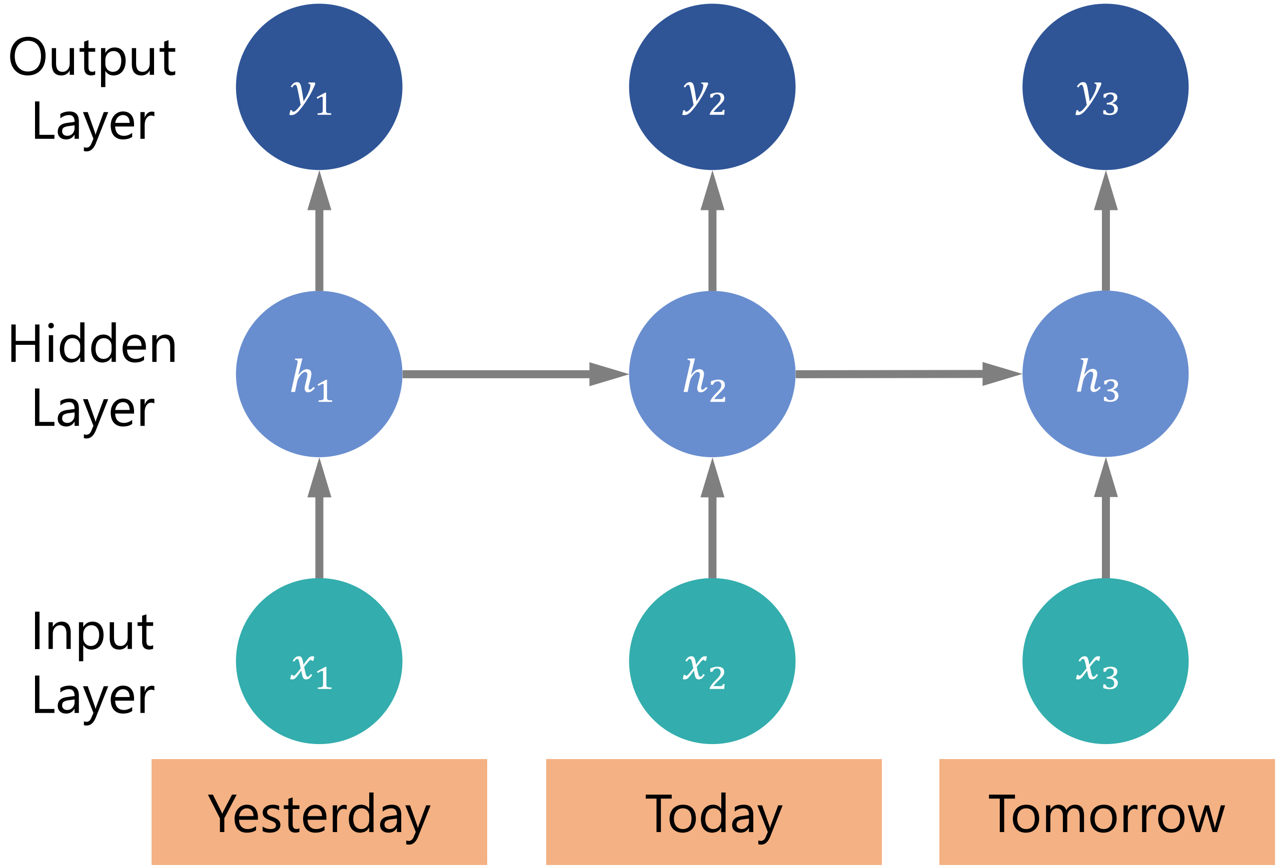
Bigger Picture of RNN



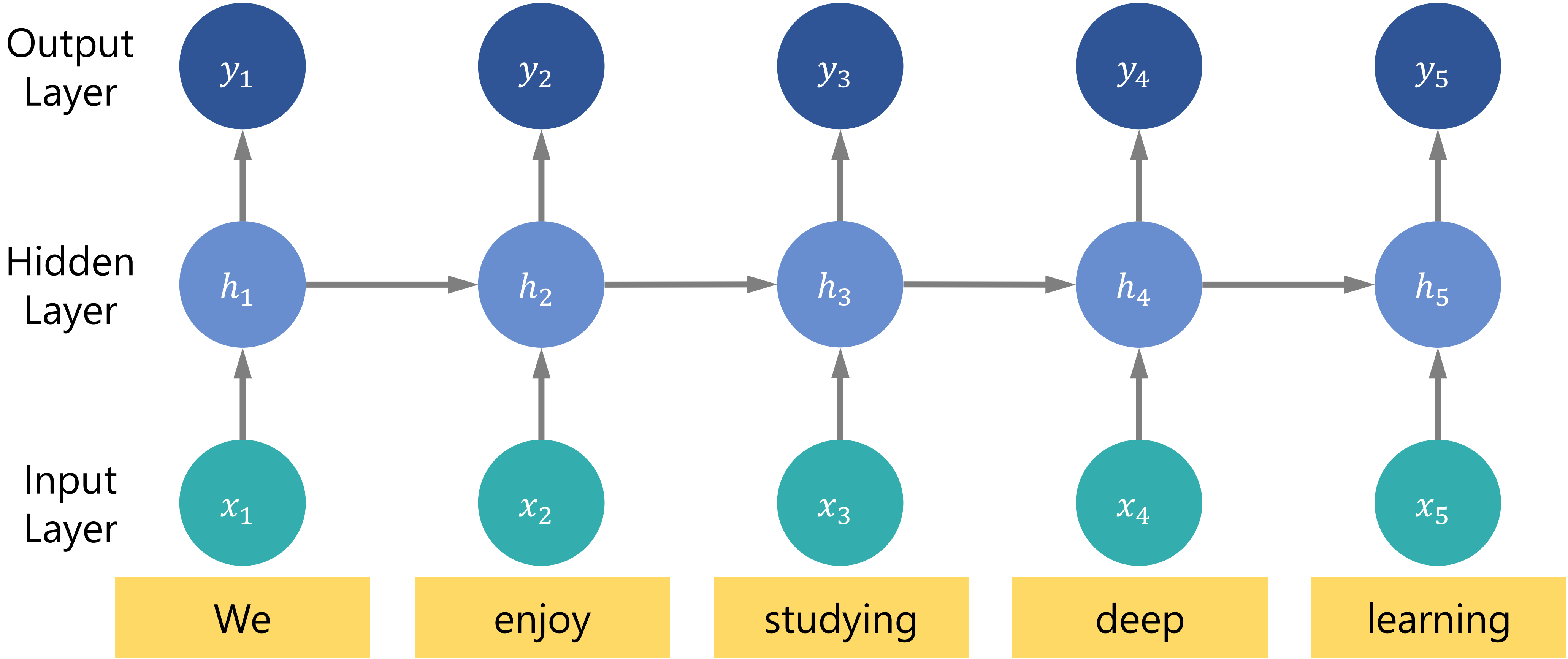
Input Data Size and Repetition



Input Data Size and Repetition

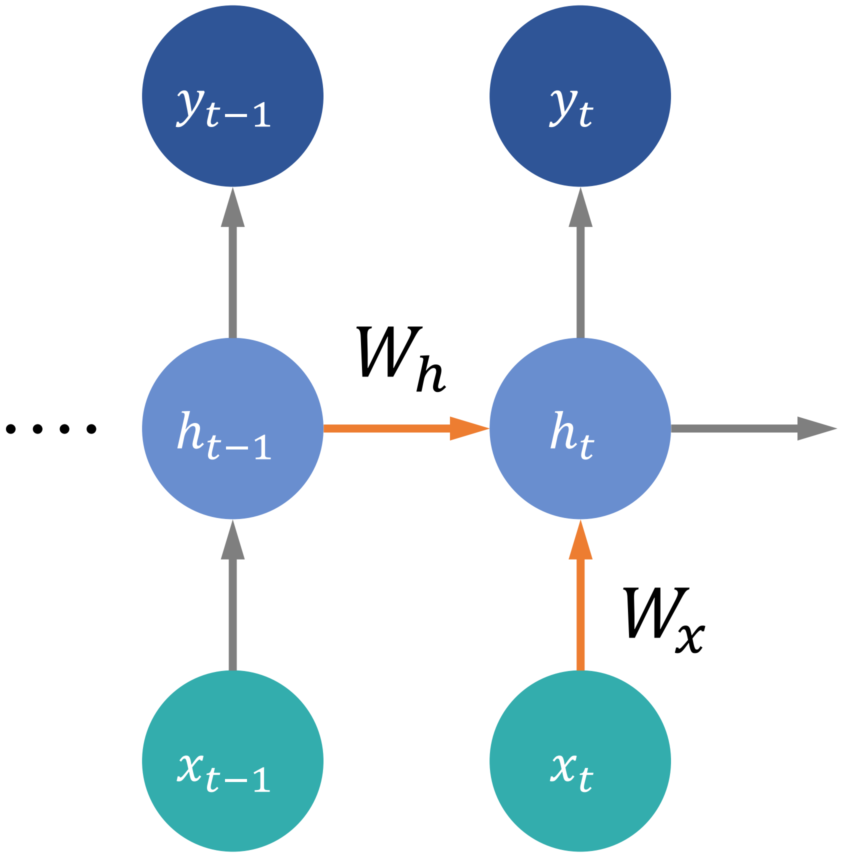


Input Data Size and Repetition

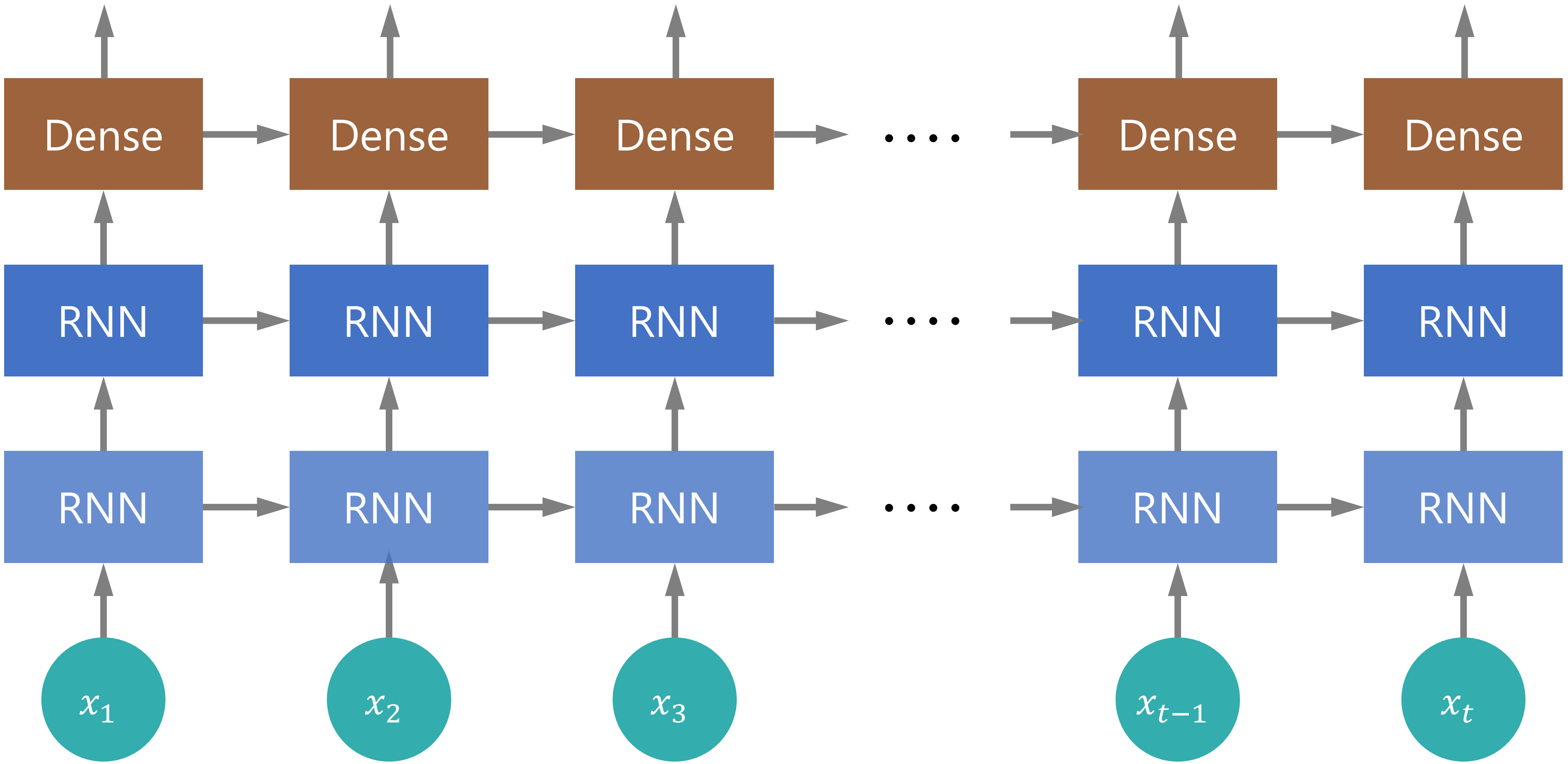


Output of the Hidden Layer

$$h_t = tanh(h_{t-1}W_h + x_tW_x + b)$$

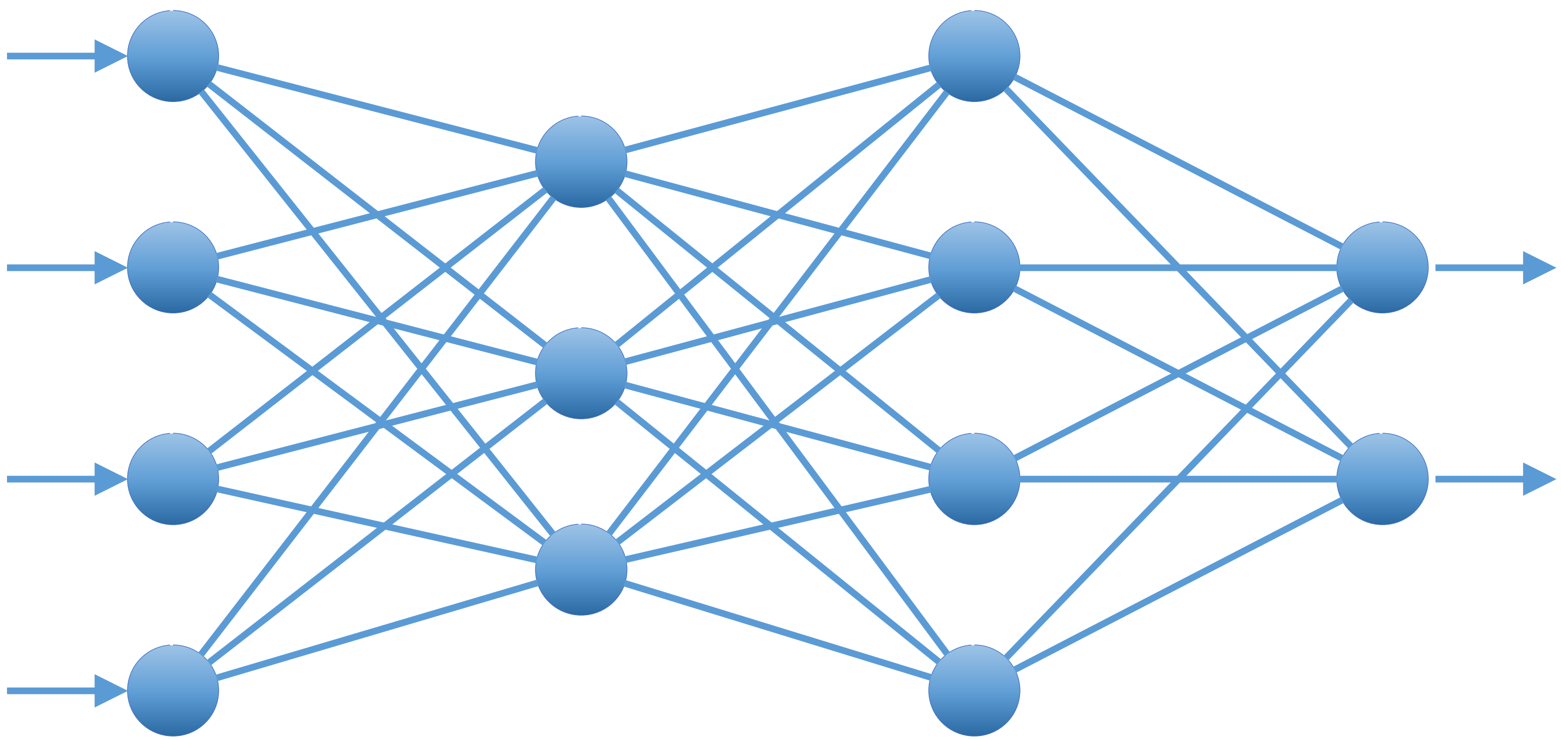


More Complicated Structure



3. Variable-Length Input

Case: Ordinary ANN



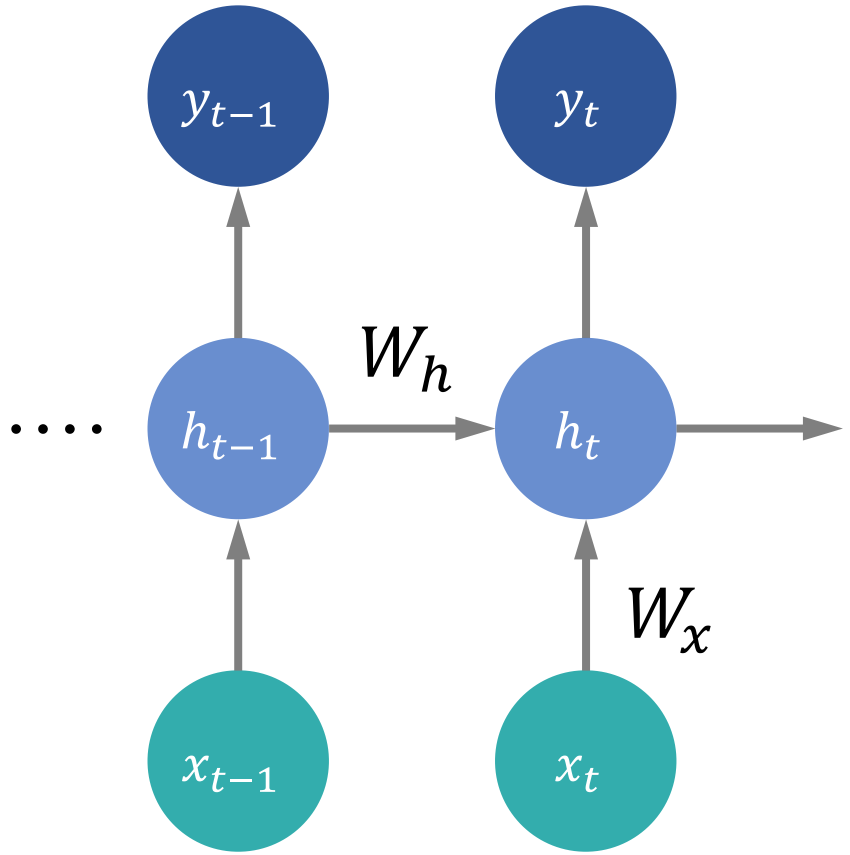
Variable-Length Input

- We can use different sized datasets for training and testing.
- RNN is effective for time-series analysis.
- RNN is also effective for analyzing text data.

4. Weight & Bias

Recap: Structure of RNN

$$h_t = tanh(h_{t-1}W_h + x_tW_x + b)$$



Weights and Biases

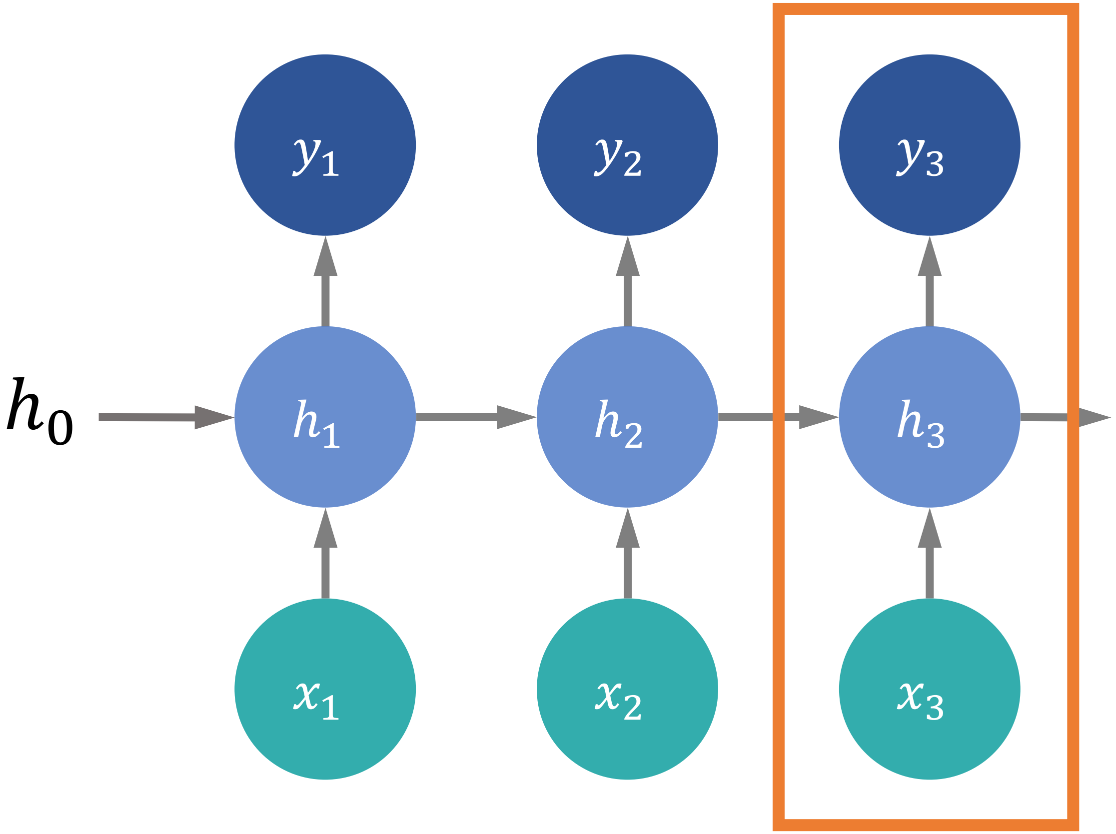
$$h_t = tanh(h_{t-1}W_h + x_tW_x + b)$$

$$h_1 = tanh(h_0W_h + x_1W_x + b)$$

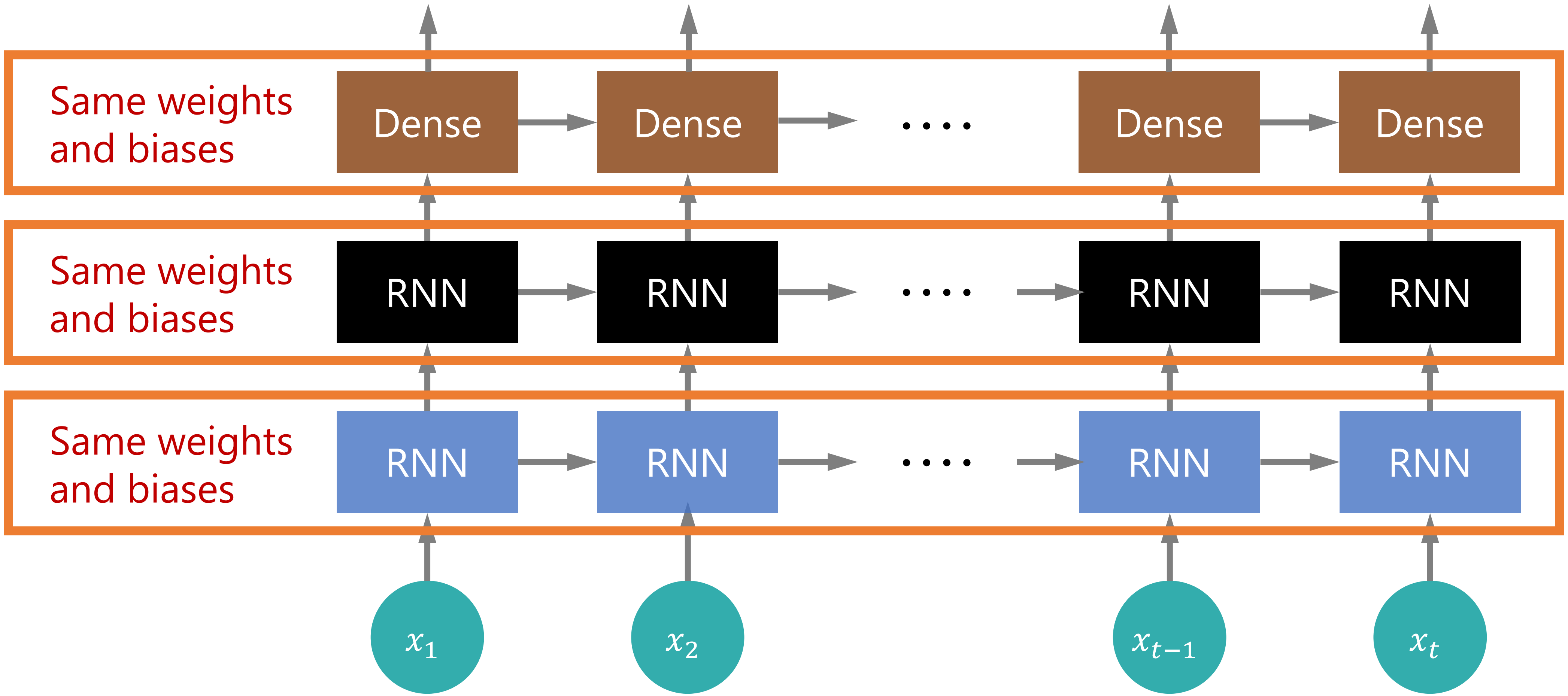
$$h_2 = tanh(h_1W_h + x_2W_x + b)$$

$$h_3 = tanh(h_2W_h + x_3W_x + b)$$

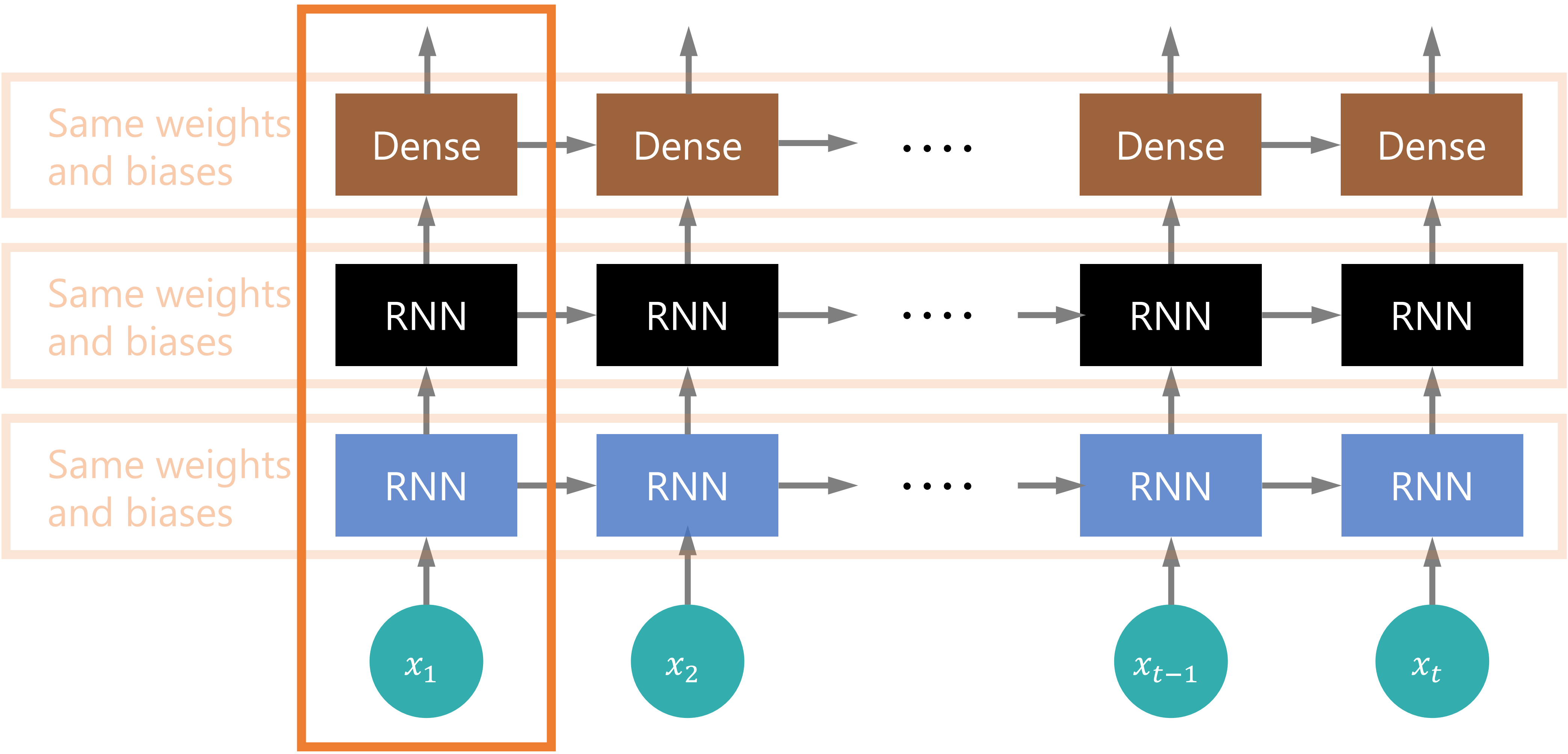
$$h_4 = tanh(h_3W_h + x_4W_x + b)$$



Weights and Biases in More Complicated Structure



Weights and Biases in More Complicated Structure



5. Types of RNN

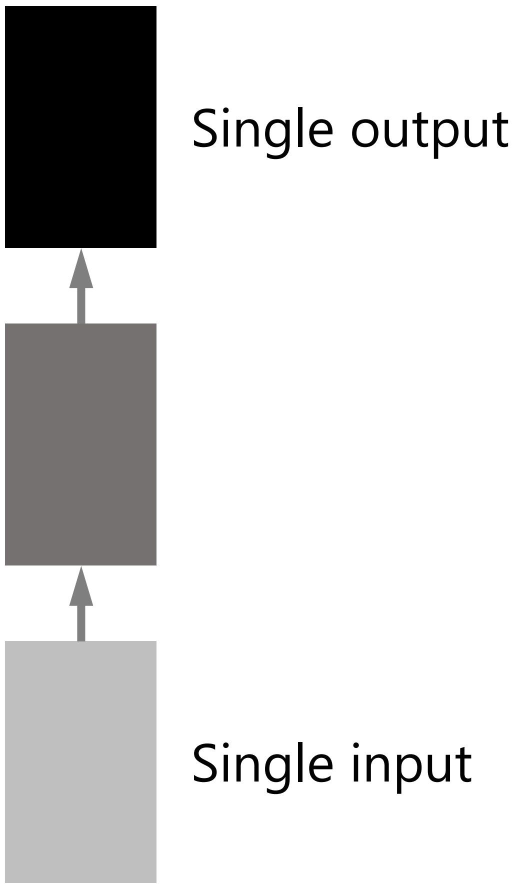
Types of RNN

We can change the input and output layers’ sizes and types.

- One to One
- One to Many
- Many to One
- Many to Many

One-to-One RNN

- General machine learning problems



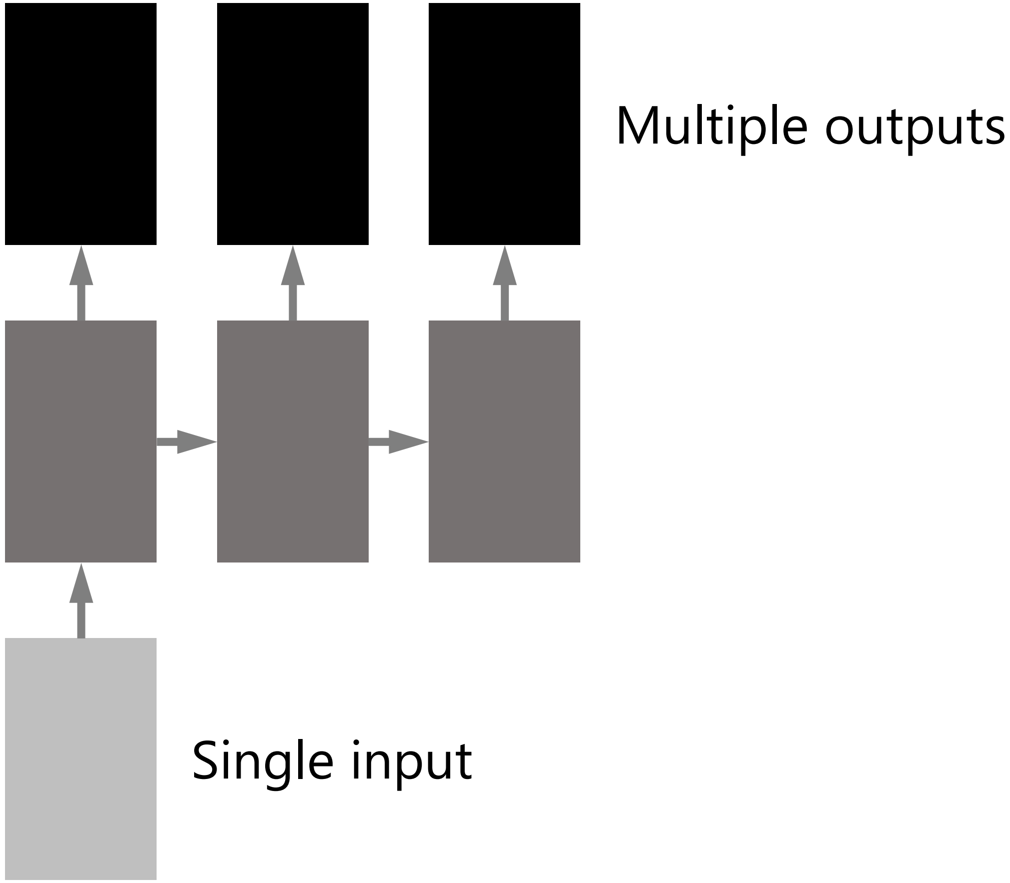
One-to-Many RNN

- The input data is not a sequence, but the output is a sequence.

  A One-to-Many RNN generates multiple outputs from a single input.

e.g., Image captioning

Boys are playing soccer.



Many-to-One RNN

- The input data is a sequence, but the output is a single element.

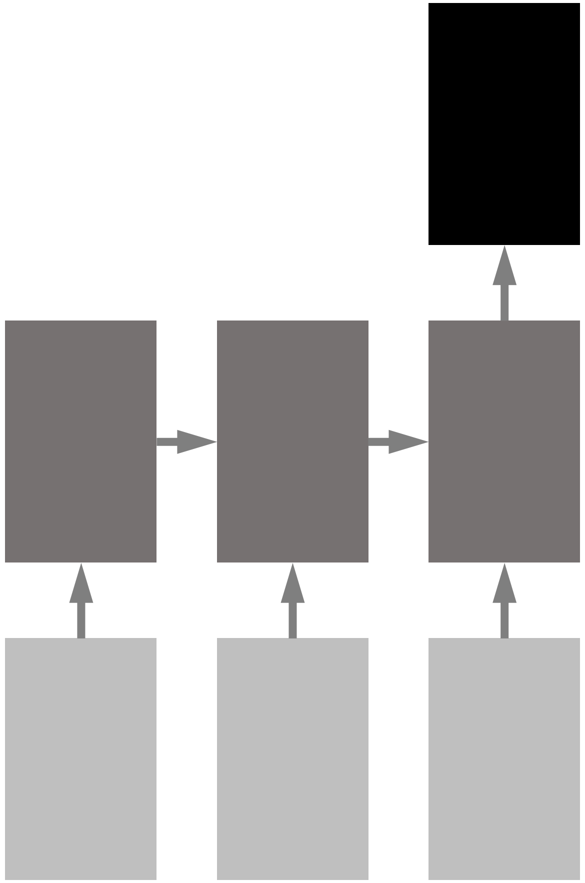
  A Many-to-One RNN uses multiple inputs to generate a single input.

e.g., Sentiment analysis

I liked this course since it provided me with a concise guide of RNN. → Score 4/5

e.g., Video classification

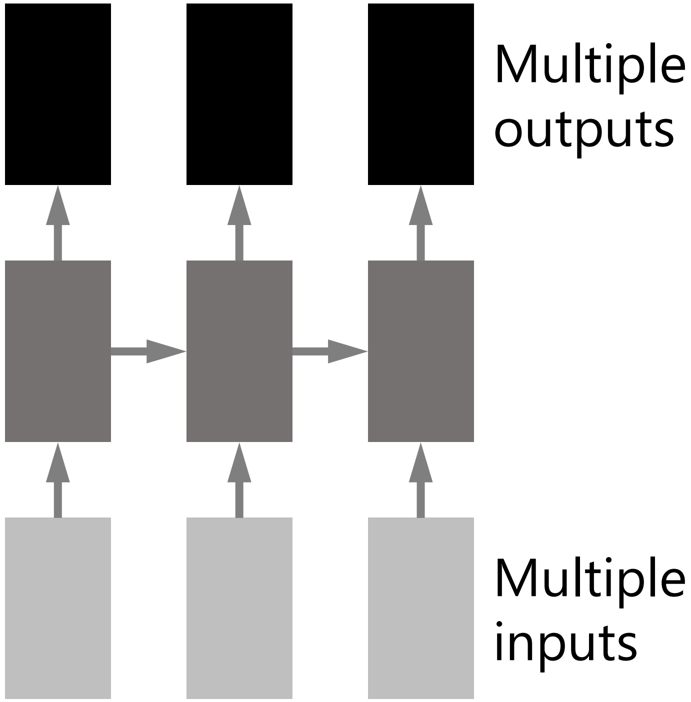
Soccer Match



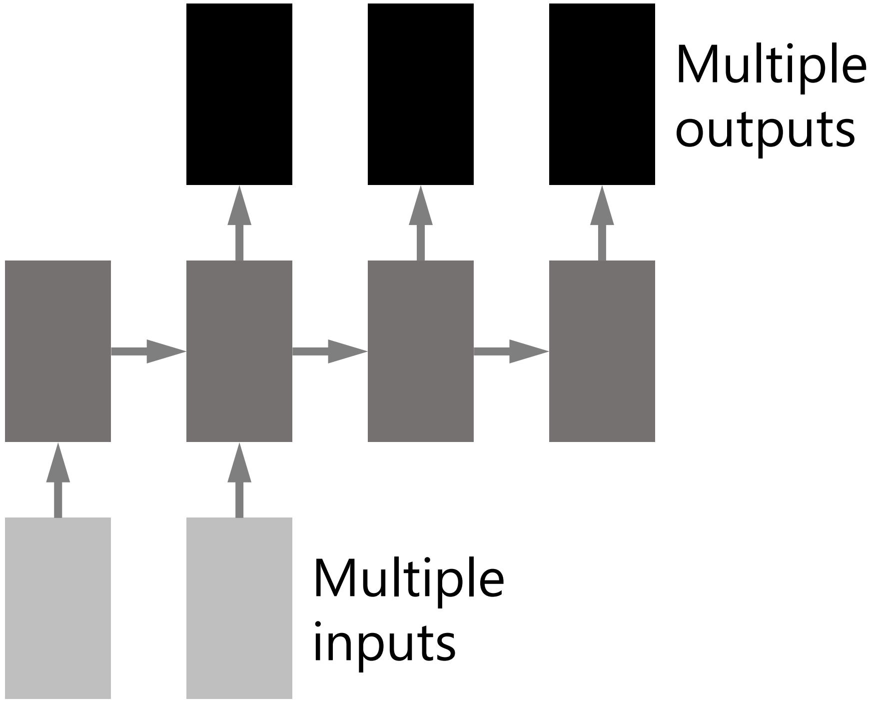
Many-to-Many RNN

- Both input and output are sequences.

  A Many-to-Many RNN uses a sequence to generate a sequence.



e.g., Video captioning

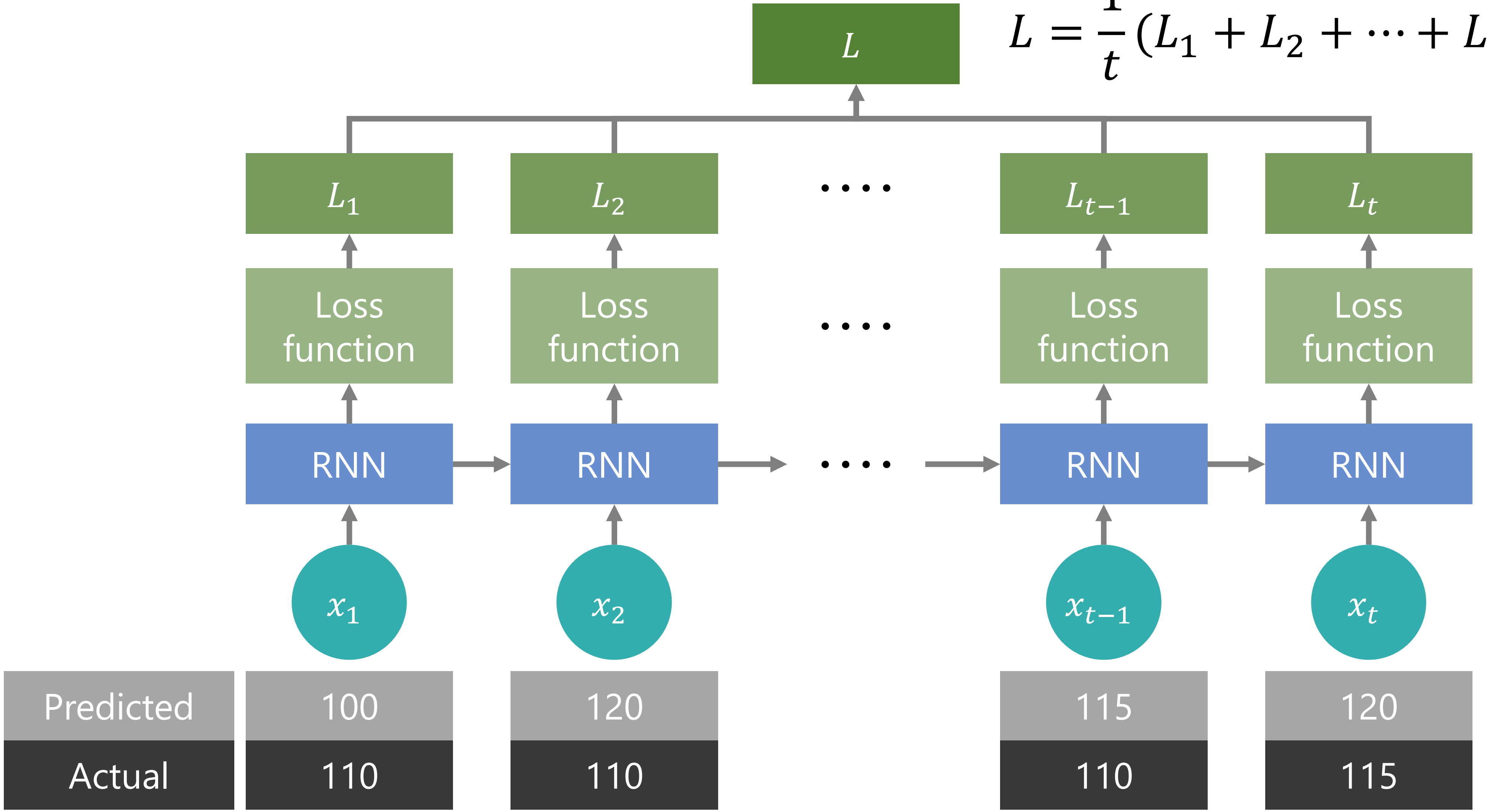


e.g., Machine translation

6. BPTT

Loss Function in RNN

$$L = \frac{1}{t}(L_1 + L_2 + \cdots + L_{t-1} + L_t)$$



BPTT

$$W = W - \eta \frac{\partial L}{\partial W} \qquad \eta : \text{ Learning rate}$$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdots \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W}$$

$$= \frac{\partial L_t}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right) \frac{\partial h_1}{\partial W}$$

BPTT (2)

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_t}{\partial h_T}\left(\prod_{t=2}^{T}\frac{\partial h_t}{\partial h_{t-1}}\right)\frac{\partial h_1}{\partial W}$$

$$h_t = tanh(h_{t-1}W_h + x_tW_x + b)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(h_{t-1}W_h + x_tW_x + b)\cdot\frac{\partial(h_{t-1}W_h + x_tW_x + b)}{\partial h_{t-1}}$$

$$= tanh'(h_{t-1}W_h + x_tW_x + b)\cdot W_h$$

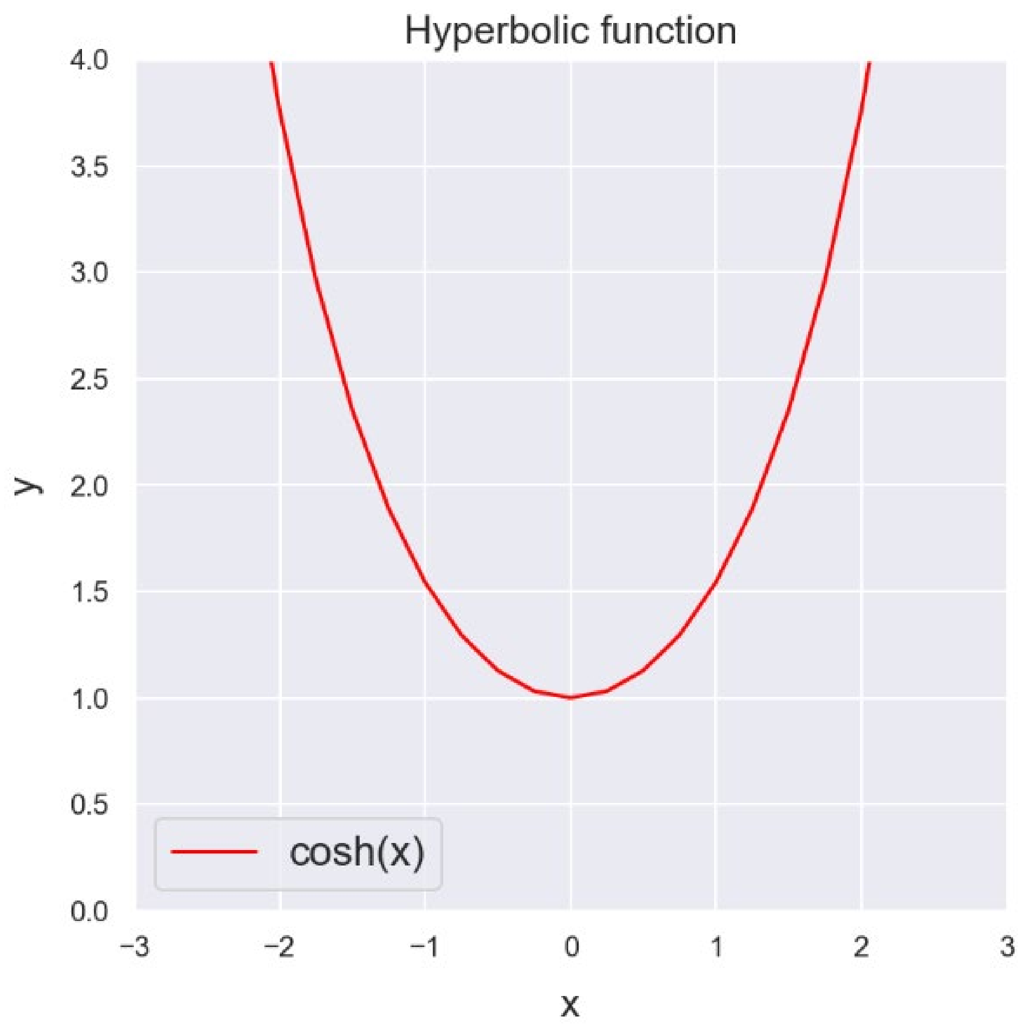
$$= \frac{\partial L_t}{\partial h_T}\left(\prod_{t=2}^{T} tanh'(h_{t-1}W_h + x_tW_x + b)\cdot W_h\right)\frac{\partial h_1}{\partial W}$$

Vanishing and Exploding Gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_t}{\partial h_T}\left(\prod_{t=2}^{T} tanh'(h_{t-1}W_h + x_tW_x + b)\cdot W_h\right)\frac{\partial h_1}{\partial W}$$

$$\tanh x' = \frac{1}{\cosh^2 x}$$

$$\cosh x = \frac{e^x + e^{-x}}{2}$$



7. LSTM

Long-Term Dependencies

Since I have lived in Japan for 10 years, I am good at speaking (   ).

Long-Term Dependencies

Since I have lived in Japan for 10 years, I am good at speaking (Japanese).

Long-Term Dependencies (Continued)

I lived in Japan when I was a high school student. In my high school days, I spent a lot of time playing baseball, and it was enjoyable. After graduating from high school, I came back to India and started working as a salesperson in a bookstore. Then, more than ten years passed. In high school, I met and played with my friends every day. However, now, there is no correspondence between us. So, now, I can no longer speak (   ).

Long-Term Dependencies (Continued)

I lived in Japan when I was a high school student. In my high school days, I spent a lot of time playing baseball, and it was enjoyable. After graduating from high school, I came back to India and started working as a salesperson in a bookstore. Then, more than ten years passed. In high school, I met and played with my friends every day. However, now, there is no correspondence between us. So, now, I can no longer speak (Japanese).

The information depends on far past information.

→ The training data must be a long sequence that RNN cannot handle.

LSTM Networks

- Long Short-Term Memory networks

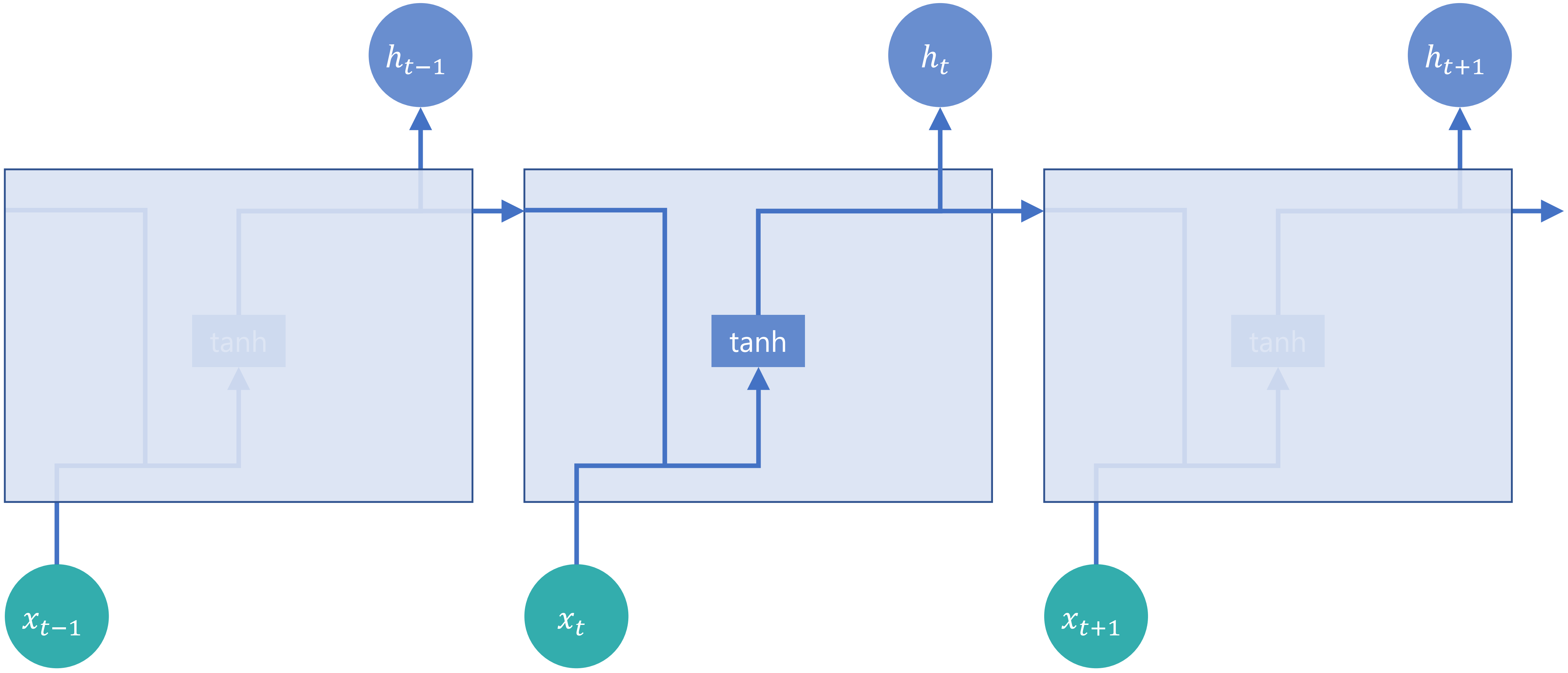
  *Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

- Central idea: "Learning to forget."

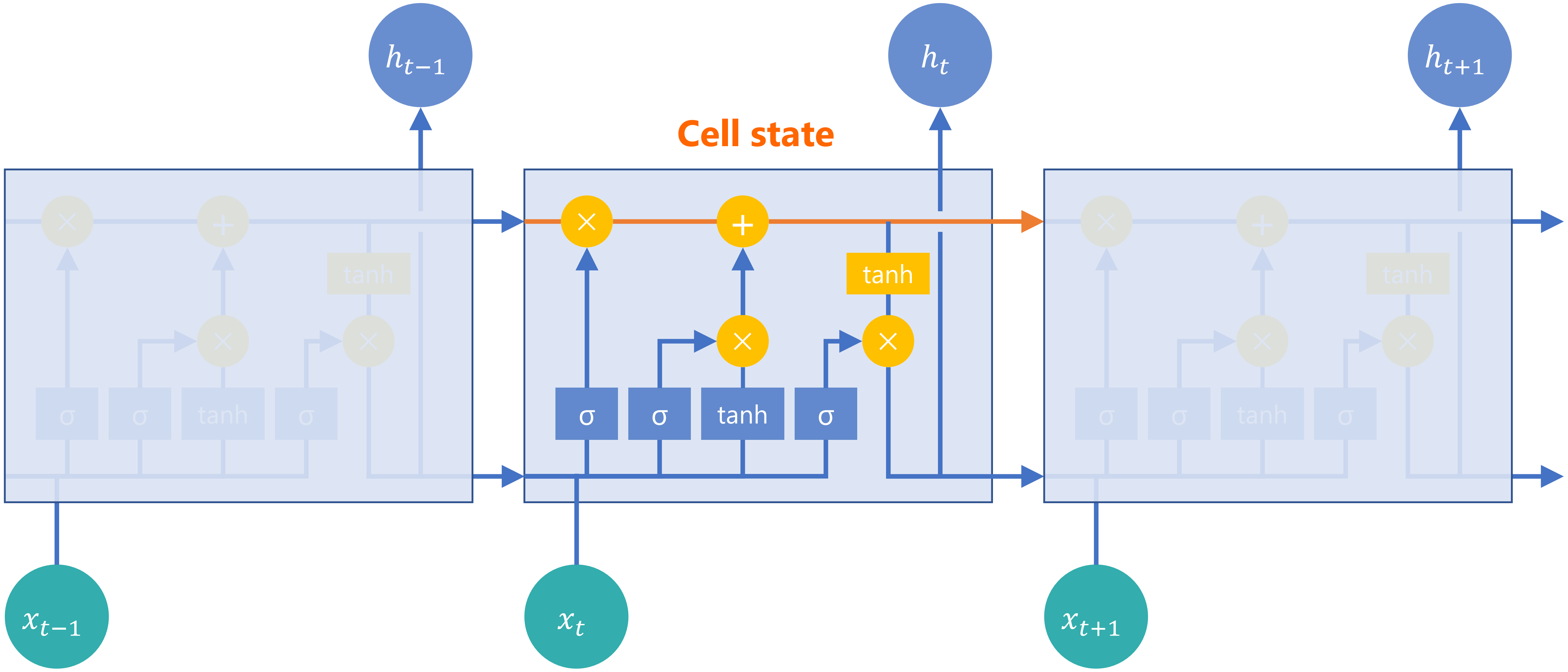
  LSTM erases unnecessary memories.

  → It can prevent the vanishing gradient problem.

Structure of RNN



LSTM Block

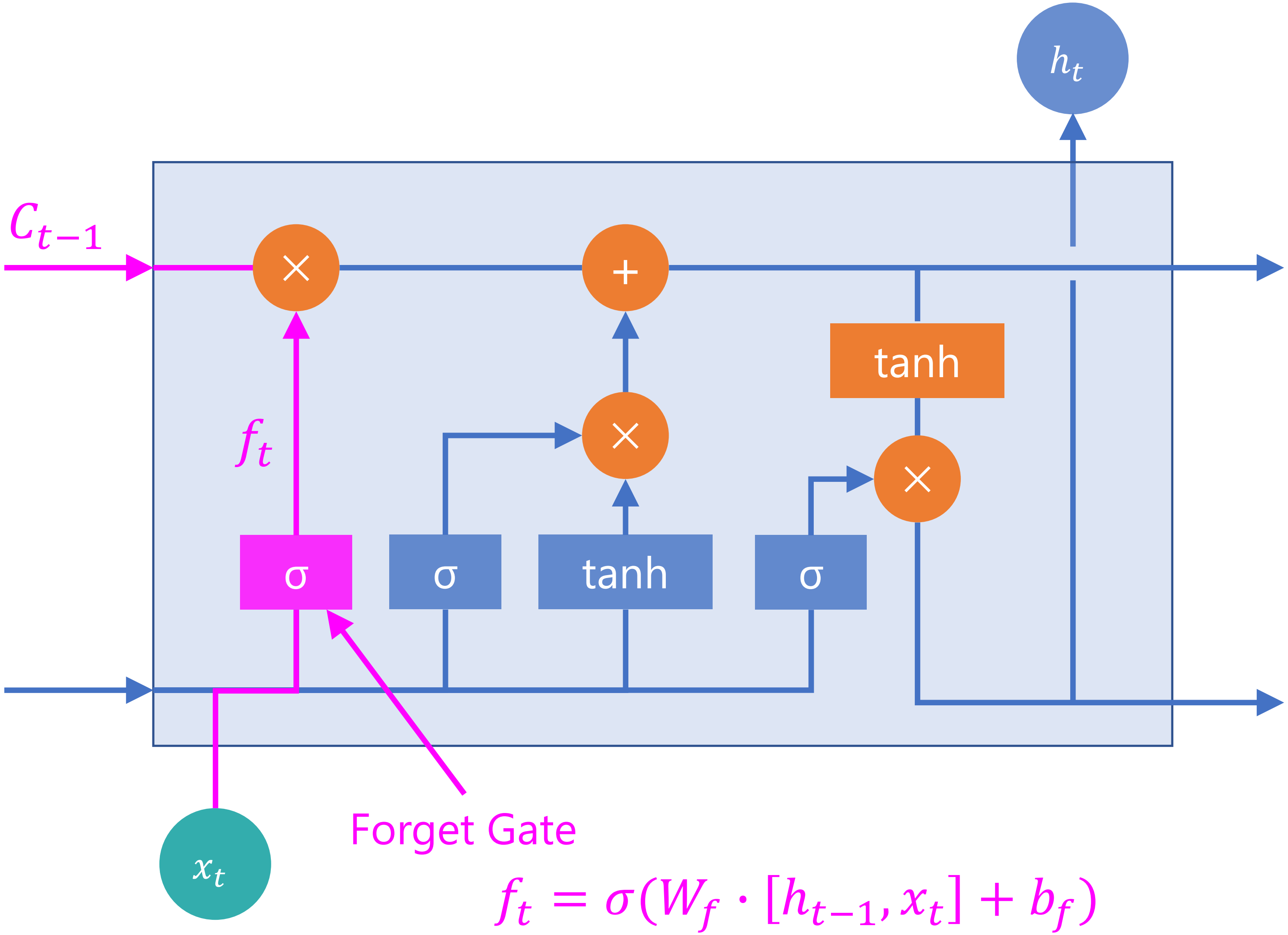


8. How does LSTM work?

Gates

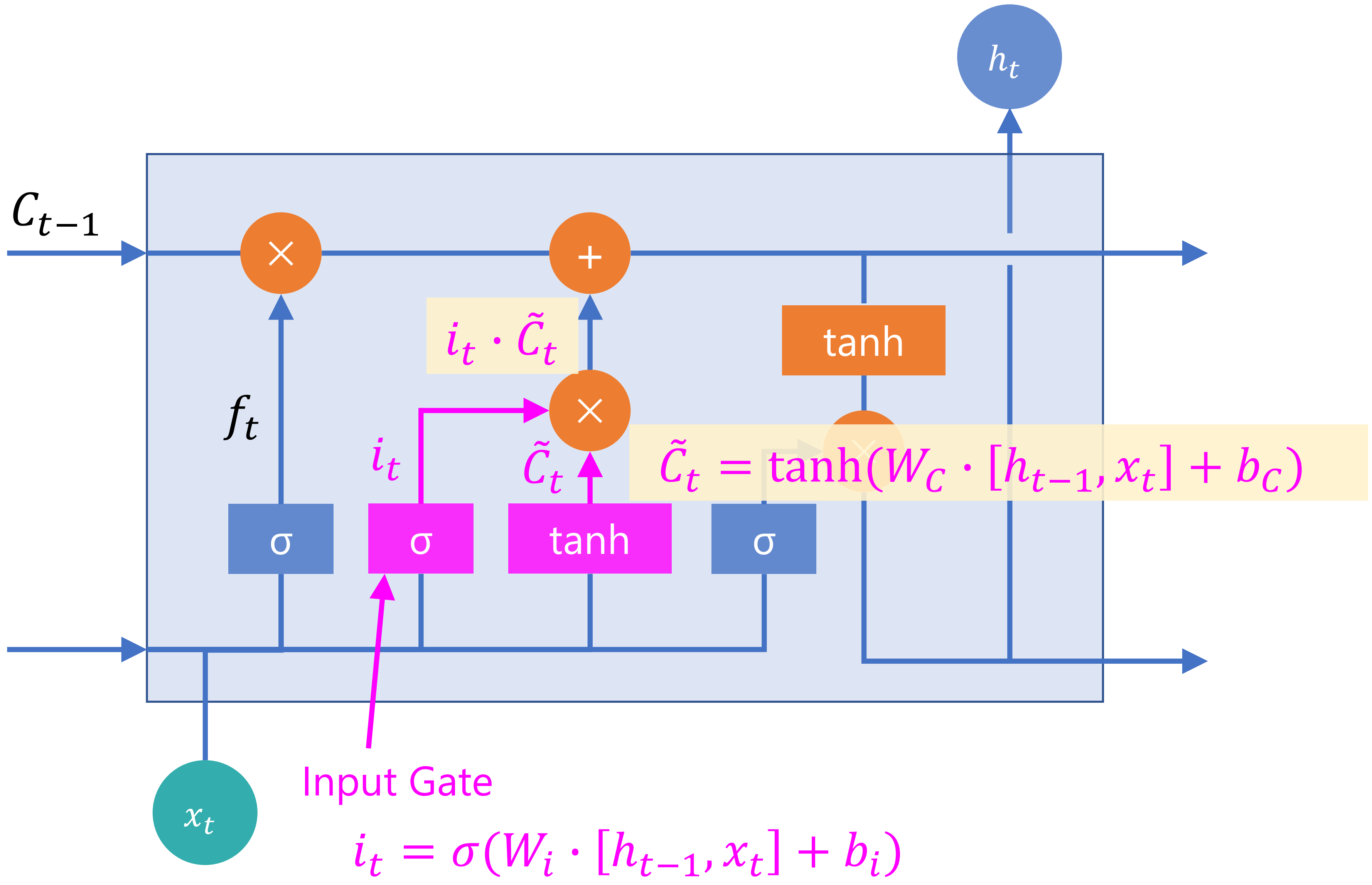
- Forget gate
- Input gate
- Output gate

Forget Gate



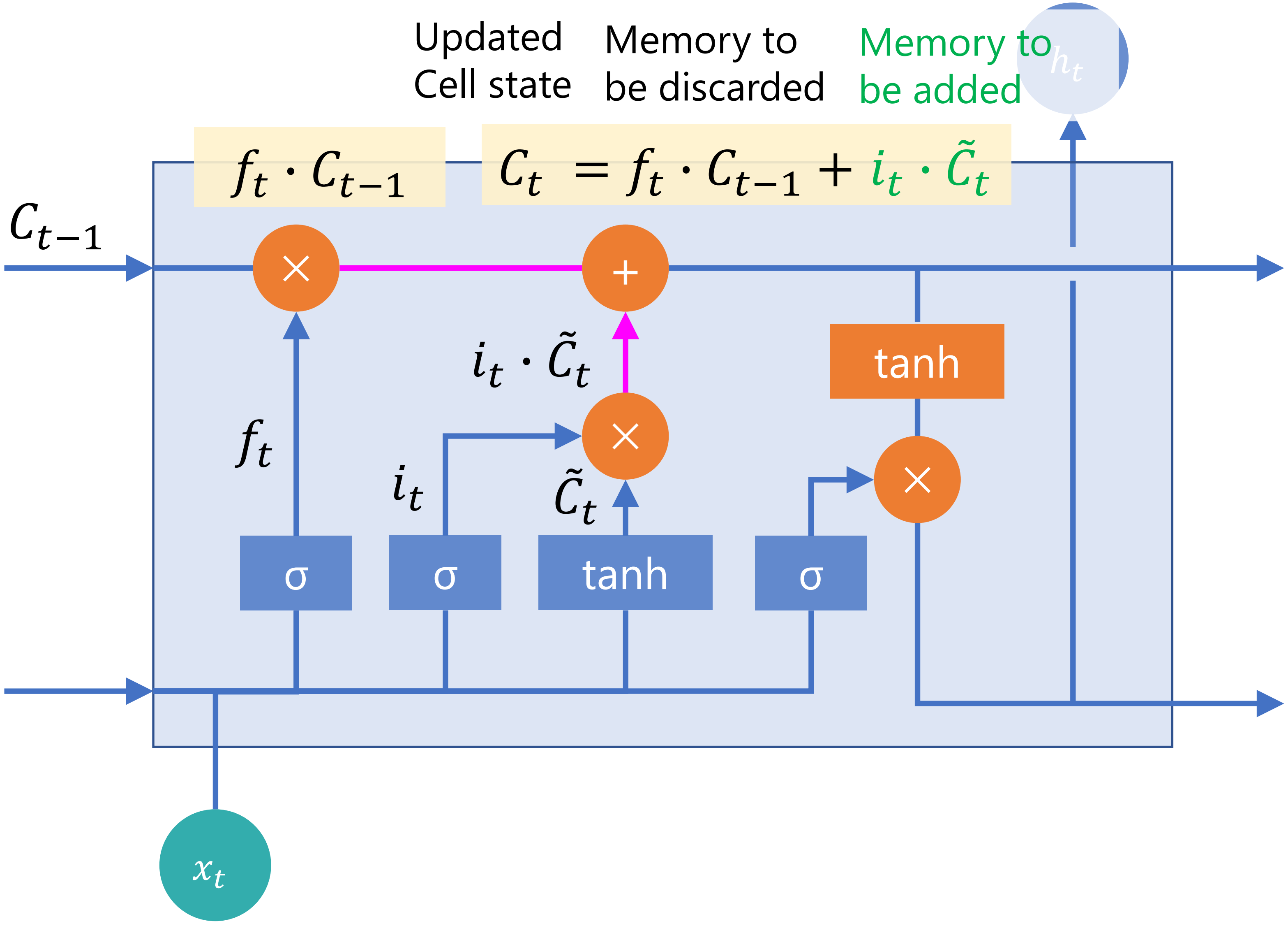
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input Gate

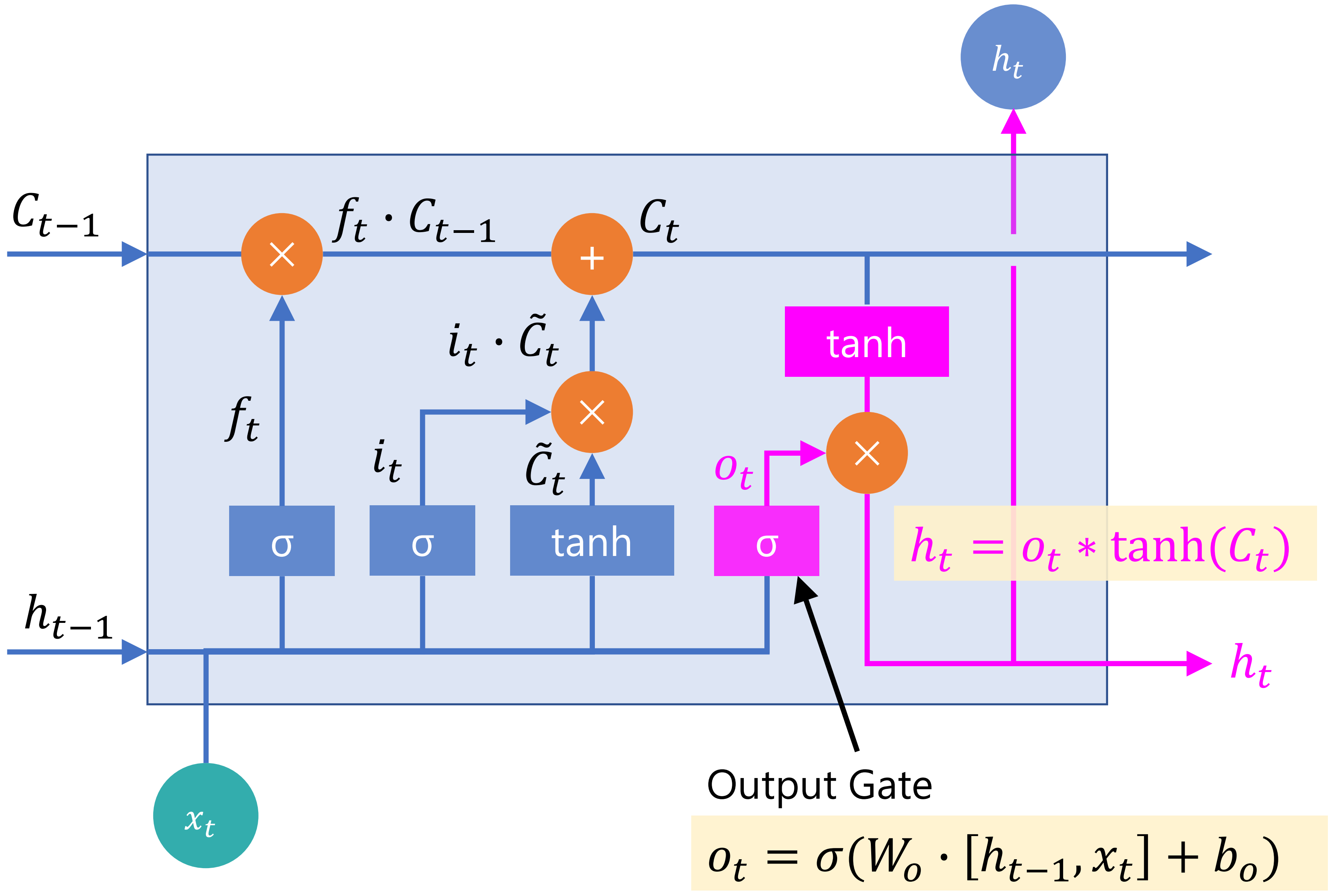


C_{t-1}
h_t
$i_t \cdot \tilde{C}_t$
tanh
f_t
i_t
$\tilde{C}_t$
$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
σ
σ
tanh
σ
x_t
Input Gate
$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Input Gate (Continued)



Output Gate

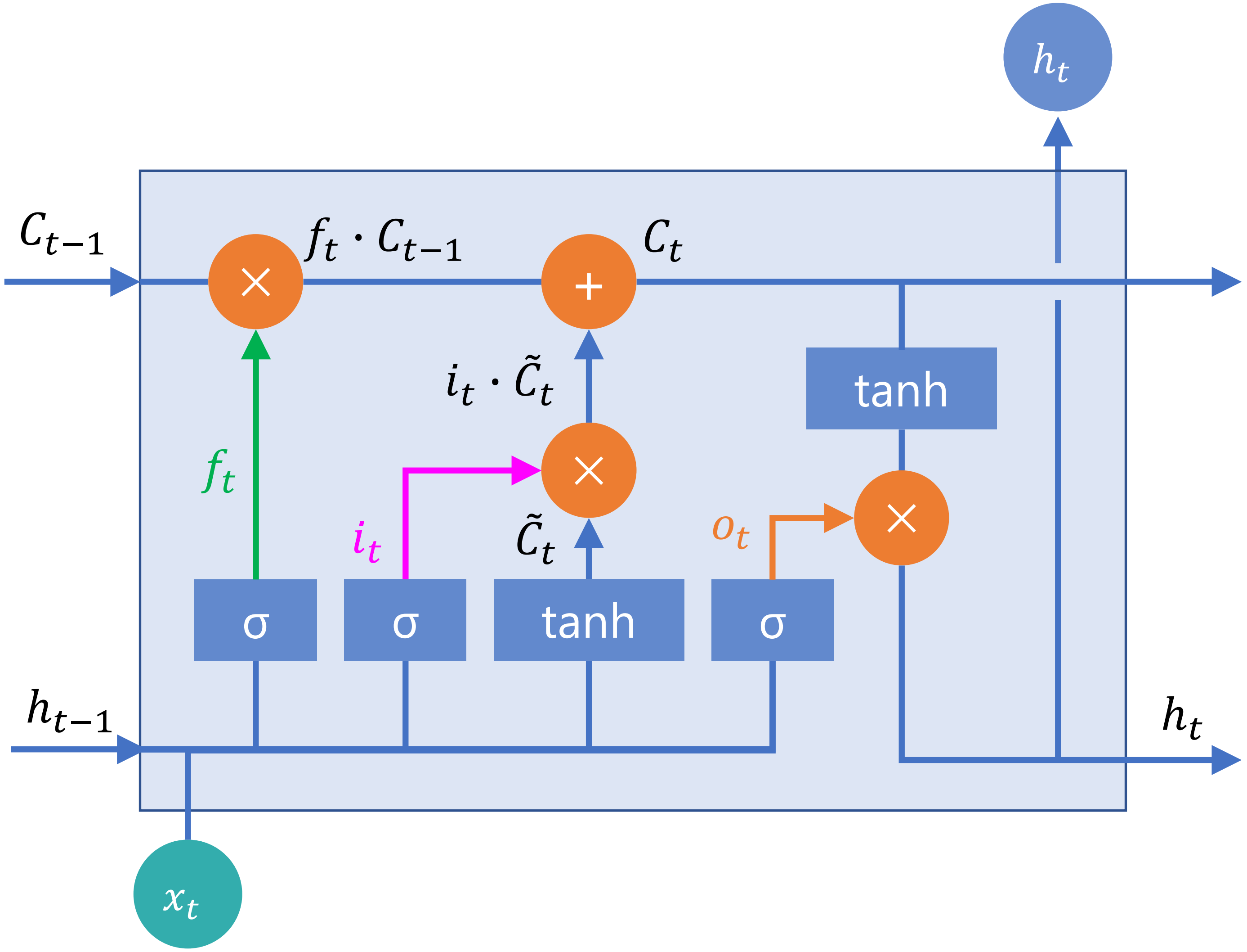

h_t
C_{t-1}
$f_t \cdot C_{t-1}$
C_t
$i_t \cdot \tilde{C}_t$
tanh
f_t
i_t
$\tilde{C}_t$
o_t
σ
σ
tanh
σ
$h_t = o_t * \tanh(C_t)$
h_{t-1}
h_t
x_t
Output Gate
$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

Peephole Architecture

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

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

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

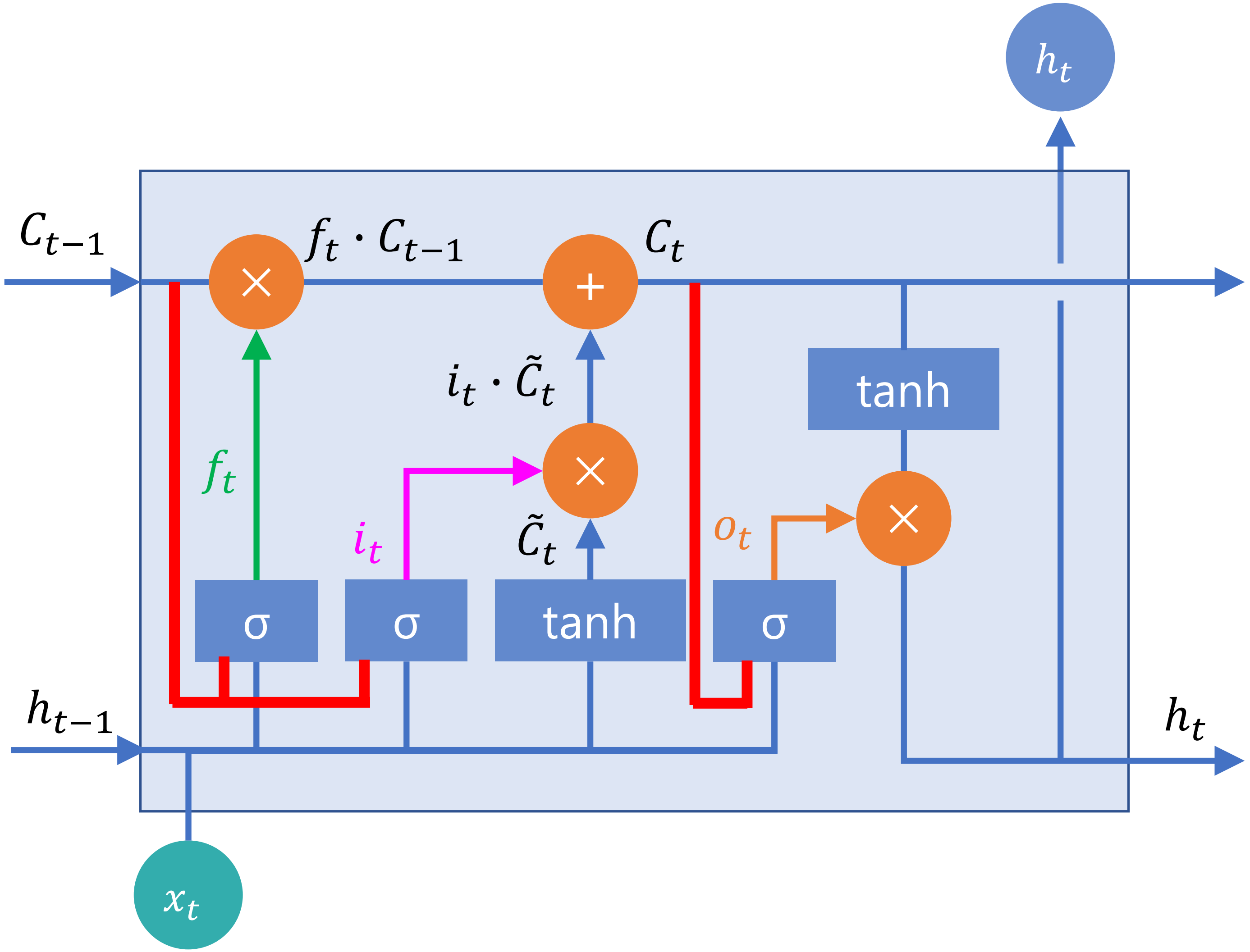


Peephole Architecture (Continued)

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

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

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



Gers, F. A., & Schmidhuber, J. (2000, July). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* (Vol. 3, pp. 189-194). IEEE.

9. BPTT in LSTM

Gradient of Loss in LSTM

- LSTM can use gradients to update the parameters for optimization.

- The gradient of the loss:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Gradient of Loss in LSTM

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\frac{\partial h_T}{\partial c_T}\frac{\partial c_T}{\partial c_{T-1}}\cdots\frac{\partial c_3}{\partial c_2}\frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

$$= \frac{\partial L_T}{\partial h_T}\frac{\partial h_T}{\partial c_T}\underline{\left(\prod_{t=2}^{T}\frac{\partial c_t}{\partial c_{t-1}}\right)}\ \frac{\partial c_1}{\partial W}$$

Cell State in LSTM

$$c_t \; = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial(f_t \cdot c_{t-1} + \; i_t \cdot \tilde{c}_t)}{\partial C_{t-1}}$$

$$= \frac{\partial(f_t \cdot c_{t-1})}{\partial c_{t-1}} + \frac{\partial(i_t \cdot \tilde{c}_t)}{\partial c_{t-1}}$$

$$= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t$$

Derivative of Cell State

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\frac{\partial h_T}{\partial c_T}\left(\prod_{t=2}^{T}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W}$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t$$

$$\prod_{t=2}^{T}\left(\frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t\right)$$

$$\prod_{t=2}^{T}(0.10 + 0.15 + 0.20 + 0.25)$$

$$\prod_{t=2}^{T}(0.10 \times 0.15 \times 0.20 \times 0.25)$$

10. GRU
(Gated Recurrent Unit)

Weakness of LSTM

- LSTM consists of several gates, and thus, contains many parameters.

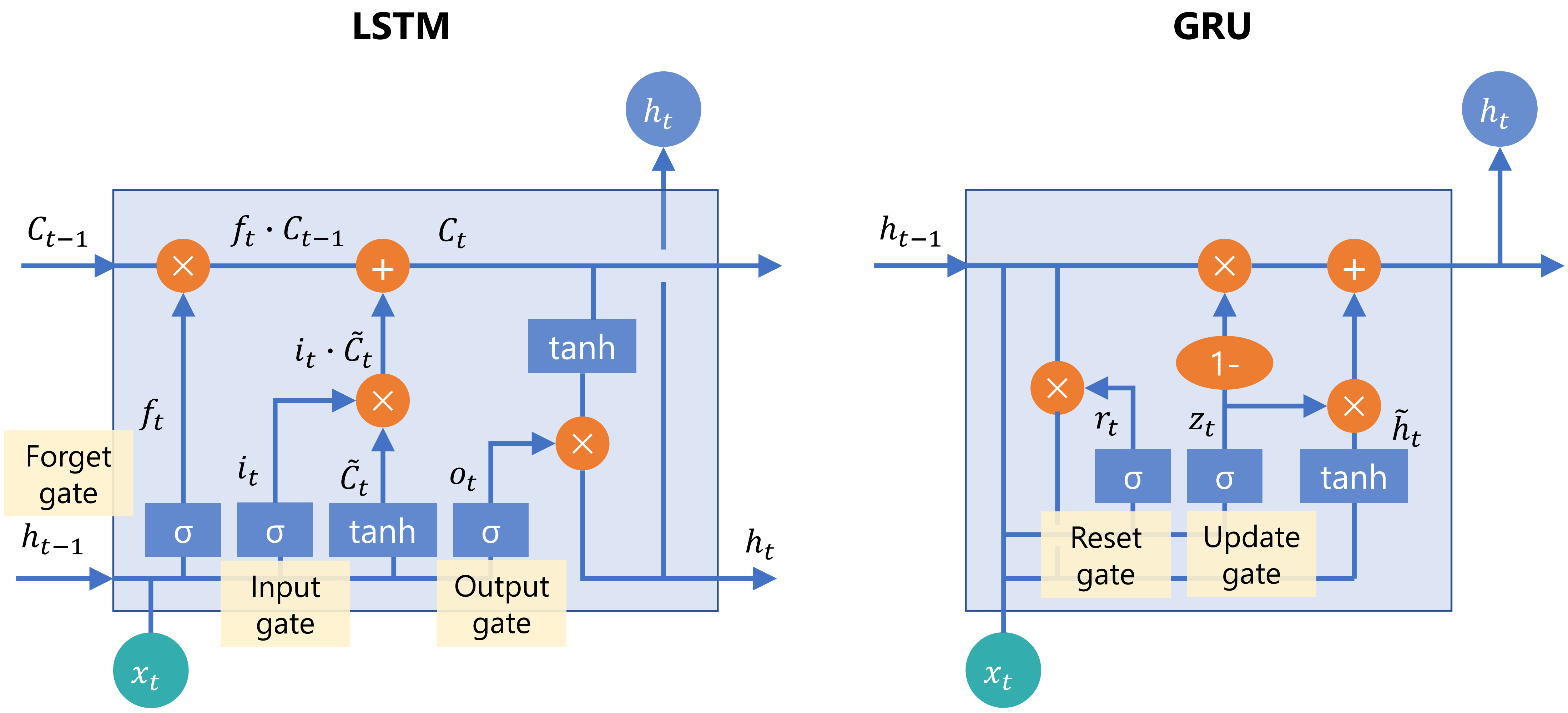
  → Its computational cost is high.

  → LSTM takes much time for model training.

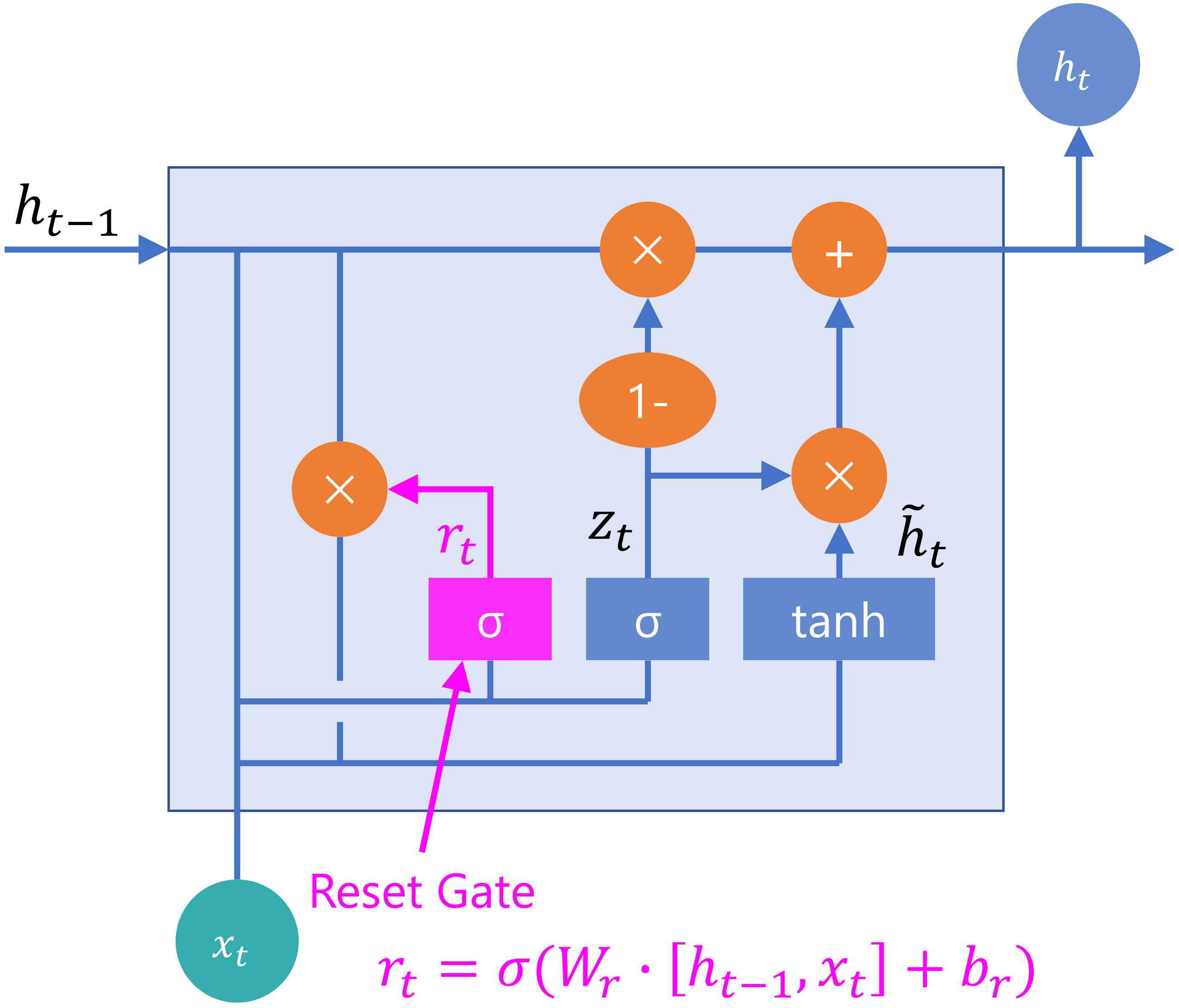
- GRU (gated recurrent unit) was introduced to address this problem (Cho et al. 2014).

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

GRU

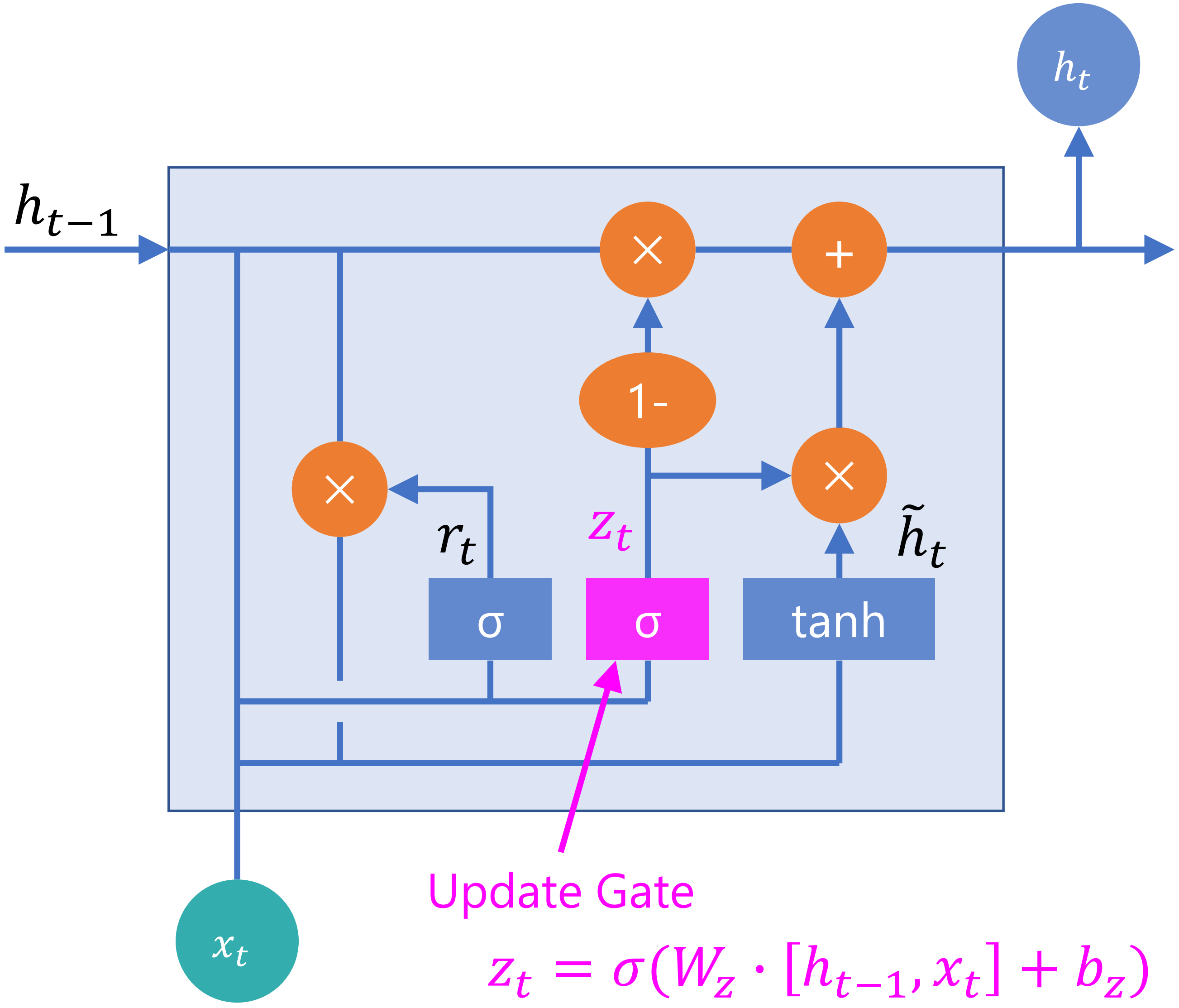

LSTM
GRU
h_t
C_{t-1}
$f_t \cdot C_{t-1}$
C_t
$i_t \cdot \tilde{C}_t$
tanh
f_t
Forget gate
i_t
$\tilde{C}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
Input gate
Output gate
x_t
h_t
h_{t-1}
1-
r_t
z_t
$\tilde{h}_t$
σ
σ
tanh
Reset gate
Update gate
x_t

Reset Gate



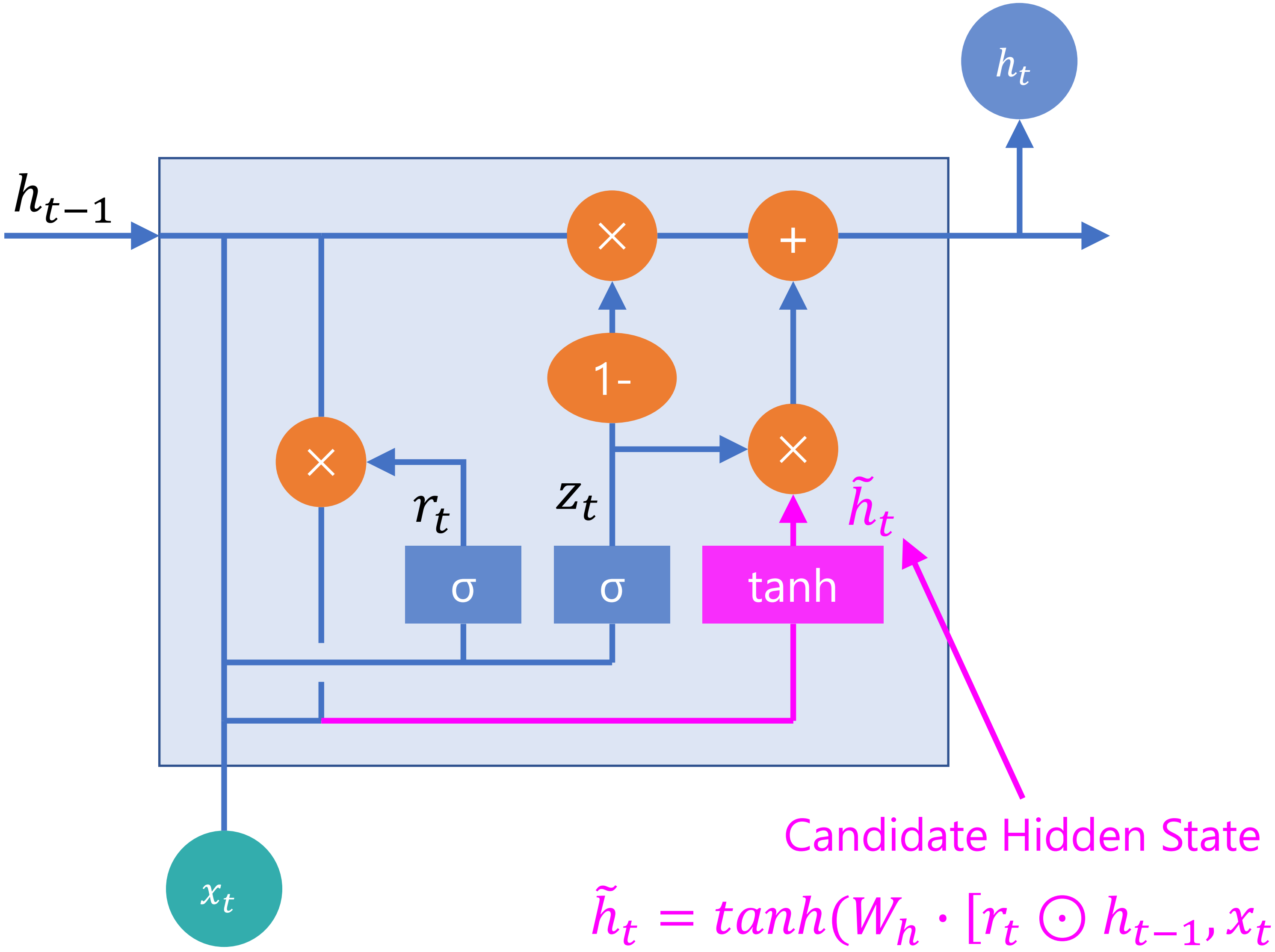
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

Update Gate



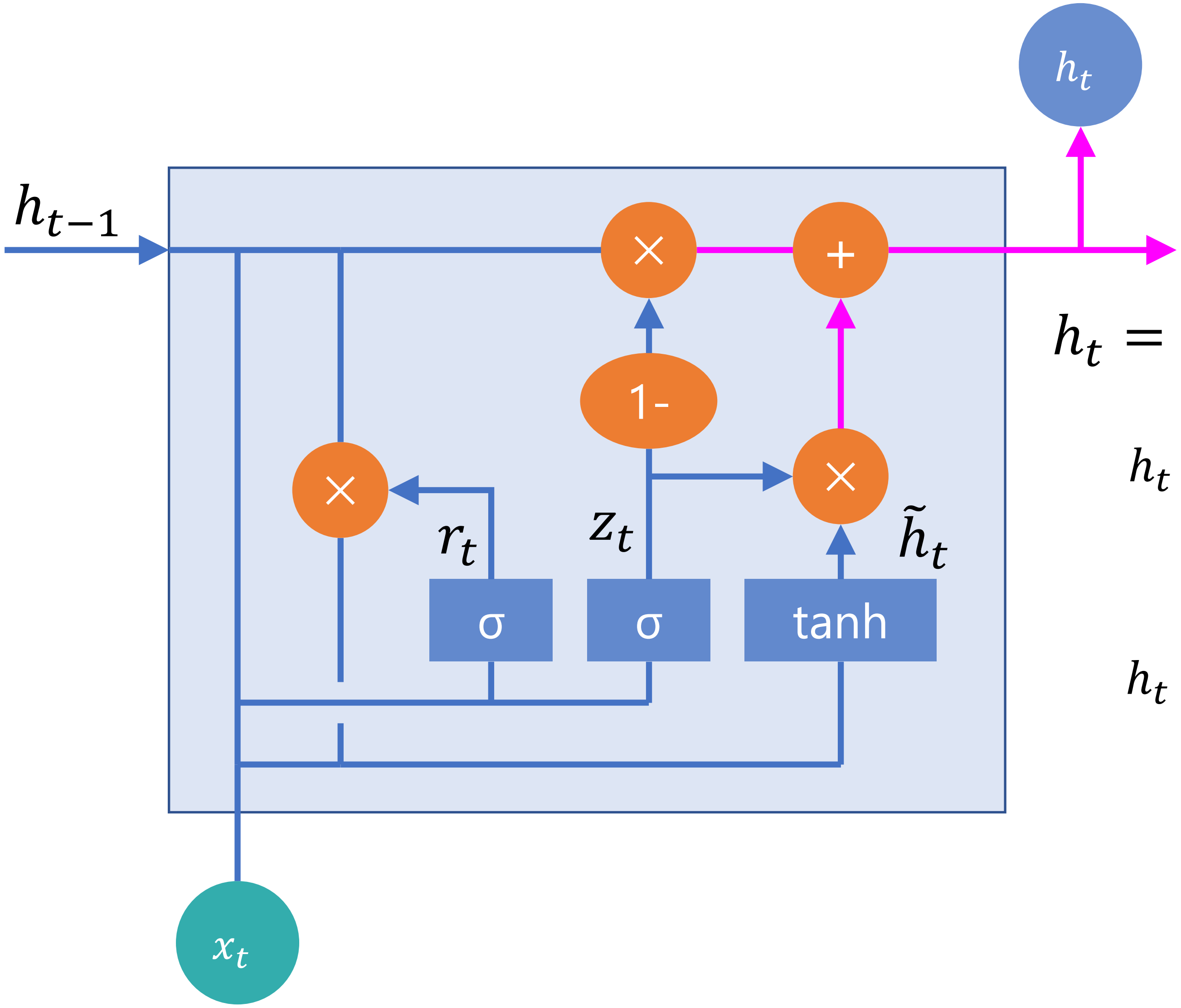
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

Candidate Hidden State



$$\tilde{h}_t = tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

Hidden State



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

$$h_t = (1 - 0) \odot h_{t-1} + 0 \odot \tilde{h}_t$$
$$= h_{t-1}$$

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

Summary: GRU

- Simpler structure than LSTM
  - Reset gate: Short-term dependencies
  - Update gate: Long-term dependencies

- A simpler structure enables GRU to learn the long-term dependencies with a smaller number of parameters.

  → Lower computational cost.

11. RNN, LSTM and GRU with Python

Data: Nikkei Stock Average (Nikkei 225)

- Stock market index of the First Section of the Tokyo Stock Exchange (TSE), Japan.
- Nikkei Stock Average consists of 225 representative companies.
- Data source: Yahoo! finance.

Import Libraries

```
# Import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.preprocessing import MinMaxScaler
import math

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.layers import SimpleRNN, LSTM, GRU
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

Data Preparation

```
# Import and show dataset
data = pd.read_csv("nikkei_stock_price.csv")
print("Shape of Data: ", df.shape)
data.head()
```

```
Shape of Data:  (244, 2)
```

	date	closed_price
0	2021/5/6	29,331.37
1	2021/5/7	29,357.82
2	2021/5/10	29,518.34
3	2021/5/11	28,608.59
4	2021/5/12	28,147.51

```
# Data information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 2 columns):
date            244 non-null object
closed_price    244 non-null object
dtypes: object(2)
memory usage: 3.9+ KB
```

Data Preprocessing (1)

```
# Delete the commas from closed price
data['closed_price'] = data['closed_price'].str.replace(',' , '')
data.head()
```

	date	closed_price
0	2021/5/6	29331.37
1	2021/5/7	29357.82
2	2021/5/10	29518.34
3	2021/5/11	28608.59
4	2021/5/12	28147.51

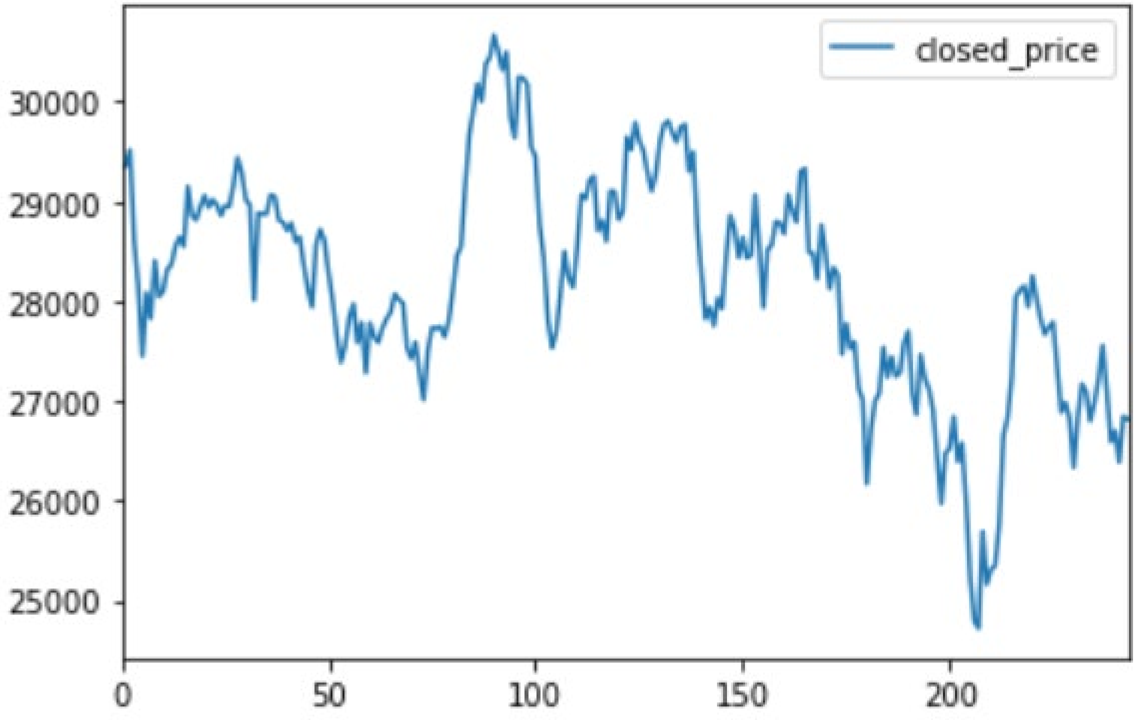
Data Preprocessing (2)

```
# Convert closed price to numerical data.
data['closed_price'] = pd.to_numeric(data['closed_price'], errors = 'coerce')
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 2 columns):
date            244 non-null object
closed_price    244 non-null float64
dtypes: float64(1), object(1)
memory usage: 3.9+ KB
None
```

Data Visualization

```
# Plot data
data.plot()
```



Data Preprocessing (3)

```
# Create a dataframe with only closed price
df=data.filter(['closed_price'])

# Convert the dataframe to a NumPy array
df=df.values
print(df[:5])
```

```
[[29331.37]
 [29357.82]
 [29518.34]
 [28608.59]
 [28147.51]]
```

Data Preprocessing (4)

```
# Normalize the data to make it applicable for RN
scaler=MinMaxScaler(feature_range=(0,1))
df_scaled=scaler.fit_transform(df)
df_scaled[:5]
```

```
array([[0.7751005 ],
       [0.77954396],
       [0.80651047],
       [0.65367732],
       [0.57621834]])
```

Data Preprocessing (5)

```
# Split data into predictors and outcomes

# Predict the present stock price by the past 5 days' stock prices
X=[]
y=[]
sequence=5

for i in range(len(df_scaled) - sequence):
    X.append(df_scaled[i:(i + sequence), 0])
    y.append(df_scaled[i + sequence, 0])

X, y = np.array(X), np.array(y)
```

Data Preprocessing (6)

```
# Show the predictors and outcomes
print("Predictors")
print(X[:5])
print("Outcomes")
print(y[:5])
```

```
Predictors
[[0.7751005  0.77954396 0.80651047 0.65367732 0.57621834]
 [0.77954396 0.80651047 0.65367732 0.57621834 0.45870607]
 [0.80651047 0.65367732 0.57621834 0.45870607 0.56562796]
 [0.65367732 0.57621834 0.45870607 0.56562796 0.52200982]
 [0.57621834 0.45870607 0.56562796 0.52200982 0.6197844 ]]
Outcomes
[0.45870607 0.56562796 0.52200982 0.6197844  0.55890481]
```

Data Preprocessing (7)

```
# Reshape the predictor
# So that it can be handled by RNN
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
X.shape
```

```
(239, 5, 1)
```

Data Preprocessing (8)

```
# Split data into training and test sets

# Set the size of training and test data
# Use 75% of the data for training
train_size = math.ceil(len(X) * 0.75)
```

Data Preprocessing (8)

```
# Split data into training and test sets

# Set the size of training and test data
# Use 75% of the data for training
train_size = math.ceil(len(X) * 0.75)

# Split X and y into training and test sets
X_train = X[:train_size, :]
y_train = y[:train_size]

X_test = X[train_size:len(X), :]
y_test = y[train_size:len(y)]
```

```
# Show the size of training and test sets
print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test : ", X_test.shape)
print("y_test : ", y_test.shape)
```

```
X_train:  (180, 5, 1)
y_train:  (180,)
X_test :  (59, 5, 1)
y_test :  (59,)
```

Model Building (1) –RNN model

```
# Build Simple RNN model
rnn=Sequential()
rnn.add(SimpleRNN(units=32, return_sequences=True, input_shape=(X_train.shape[1],1)))
rnn.add(SimpleRNN(units=32, return_sequences=True))
rnn.add(SimpleRNN(units=32, return_sequences=True))
rnn.add(SimpleRNN(units=32))
rnn.add(Dense(units=1))

rnn.compile(optimizer='adam',
            loss='mean_squared_error')

rnn.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 5, 32)             1088
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 5, 32)             2080
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 5, 32)             2080
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 7,361
Trainable params: 7,361
Non-trainable params: 0
_________________________________________________________________
```

Model Building (2) –LSTM model

```
# Build LSTM model
lstm=Sequential()
lstm.add(LSTM(units=32, return_sequences=True, input_shape=(X_train.shape[1],1)))
lstm.add(LSTM(units=32, return_sequences=True))
lstm.add(LSTM(units=32, return_sequences=True))
lstm.add(LSTM(units=32))
lstm.add(Dense(units=1))

lstm.compile(optimizer='adam',
                   loss='mean_squared_error')

lstm.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 5, 32)             4352
_________________________________________________________________
lstm_1 (LSTM)                (None, 5, 32)             8320
_________________________________________________________________
lstm_2 (LSTM)                (None, 5, 32)             8320
_________________________________________________________________
lstm_3 (LSTM)                (None, 32)                8320
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
```

Model Building (3) –GRU model

```
# Build GRU model
gru=Sequential()
gru.add(GRU(units=32, return_sequences=True, input_shape=(X_train.shape[1],1)))
gru.add(GRU(units=32, return_sequences=True))
gru.add(GRU(units=32, return_sequences=True))
gru.add(GRU(units=32))
gru.add(Dense(units=1))

gru.compile(optimizer='adam',
            loss='mean_squared_error')

gru.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 5, 32)             3360
_________________________________________________________________
gru_1 (GRU)                  (None, 5, 32)             6336
_________________________________________________________________
gru_2 (GRU)                  (None, 5, 32)             6336
_________________________________________________________________
gru_3 (GRU)                  (None, 32)                6336
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 22,401
Trainable params: 22,401
Non-trainable params: 0
_________________________________________________________________
```

Set Early Stopping

```
# Set Early Stopping
early_stop = EarlyStopping(monitor='val_loss', patience=40)
```

Model Training

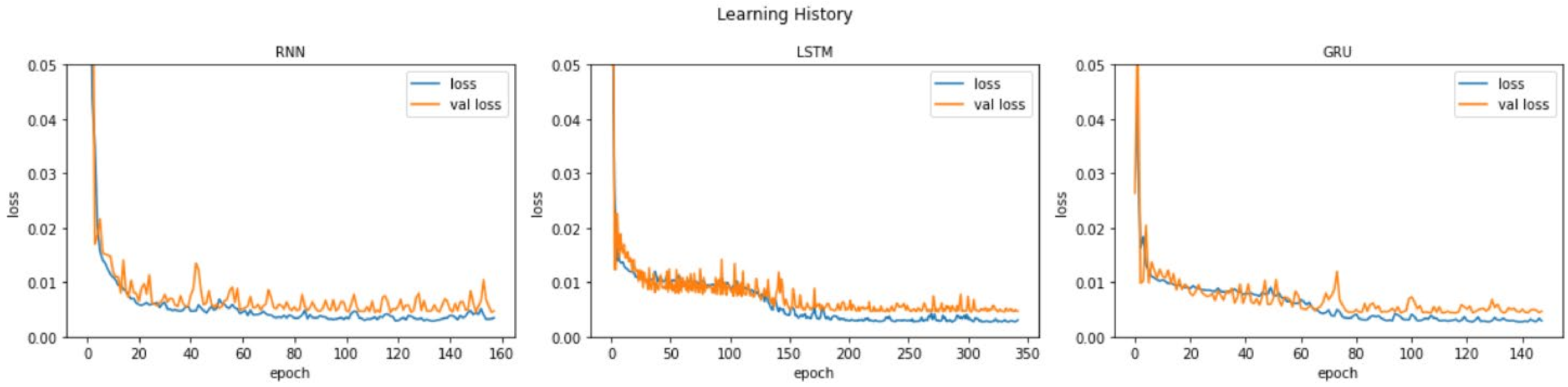
```
# Fit RNN model and store the history
rnn_history = rnn.fit(X_train, y_train,
                      batch_size=16,
                      epochs=1000,
                      validation_split=0.2,
                      callbacks=[early_stop],
                      verbose=1)

# Fit LSTM model and store the history
lstm_history = lstm.fit(X_train, y_train,
                        batch_size=16,
                        epochs=1000,
                        validation_split=0.2,
                        callbacks=[early_stop],
                        verbose=1)

# Fit GRU model and store the history
gru_history = gru.fit(X_train, y_train,
                      batch_size=16,
                      epochs=1000,
                      validation_split=0.2,
                      callbacks=[early_stop],
                      verbose=1)
```

Visualize Training Progress

```
# Visualize Training Progress

# Create a list of subplot subtitles
titles = ['RNN', 'LSTM', 'GRU’]

# Create a list of prediction models
models = [rnn_history, lstm_history, gru_history]

# Set the plot area
fig, ax = plt.subplots(1, 3, figsize=(16,4),
                       tight_layout=True)
```

```
# Plot learning progress
plt.suptitle('Learning History’)
for i in range(0, 3):
    plt.subplot(1, 3, i+1)
    plt.title(titles[i], fontsize=10)
    plt.plot(models[i].history['loss’], label='loss')
    plt.plot(models[i].history['val_loss’], label='val loss')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(loc='best')
    plt.ylim([0,0.05])
```

Visualize Training Progress



Make Predictions

```
# Make predictions and reverse the predicted values to actual values

# Predict by RNN model
rnn_y_pred=rnn.predict(X_test)
rnn_y_pred=scaler.inverse_transform(rnn_y_pred)

# Predict by LSTM model
lstm_y_pred=lstm.predict(X_test)
lstm_y_pred=scaler.inverse_transform(lstm_y_pred)

# Predict by GRU model
gru_y_pred=gru.predict(X_test)
gru_y_pred=scaler.inverse_transform(gru_y_pred)

# Reverse test data to actual values
y_test=y_test.reshape(y_test.shape[0],1)
y_test=scaler.inverse_transform(y_test)
```

Visualize Prediction Results

```
# Visualize Prediction Results

# Set subplot subtitles
titles = ['RNN', 'LSTM', 'GRU']

# Create a list of prediction models
models = [rnn_y_pred, lstm_y_pred, gru_y_pred]

# Set the plot area
fig, ax = plt.subplots(1, 3, figsize=(16,4),
                       tight_layout=True)

# Set the title
plt.suptitle('Japan Stock Price')

# Create and show subplots
for i in range(0, 3):
    plt.subplot(1, 3,i+1)
    plt.title(titles[i], fontsize=10)
    plt.plot(y_test, label='Real Stock Price')
    plt.plot(models[i], label=titles[i]+' Prediction')
    plt.legend()
    plt.xlabel('Time')
    plt.ylabel('Stock Price')
```

Visualize Prediction Results