KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information

2022

스마트교통 빅데이터 분석

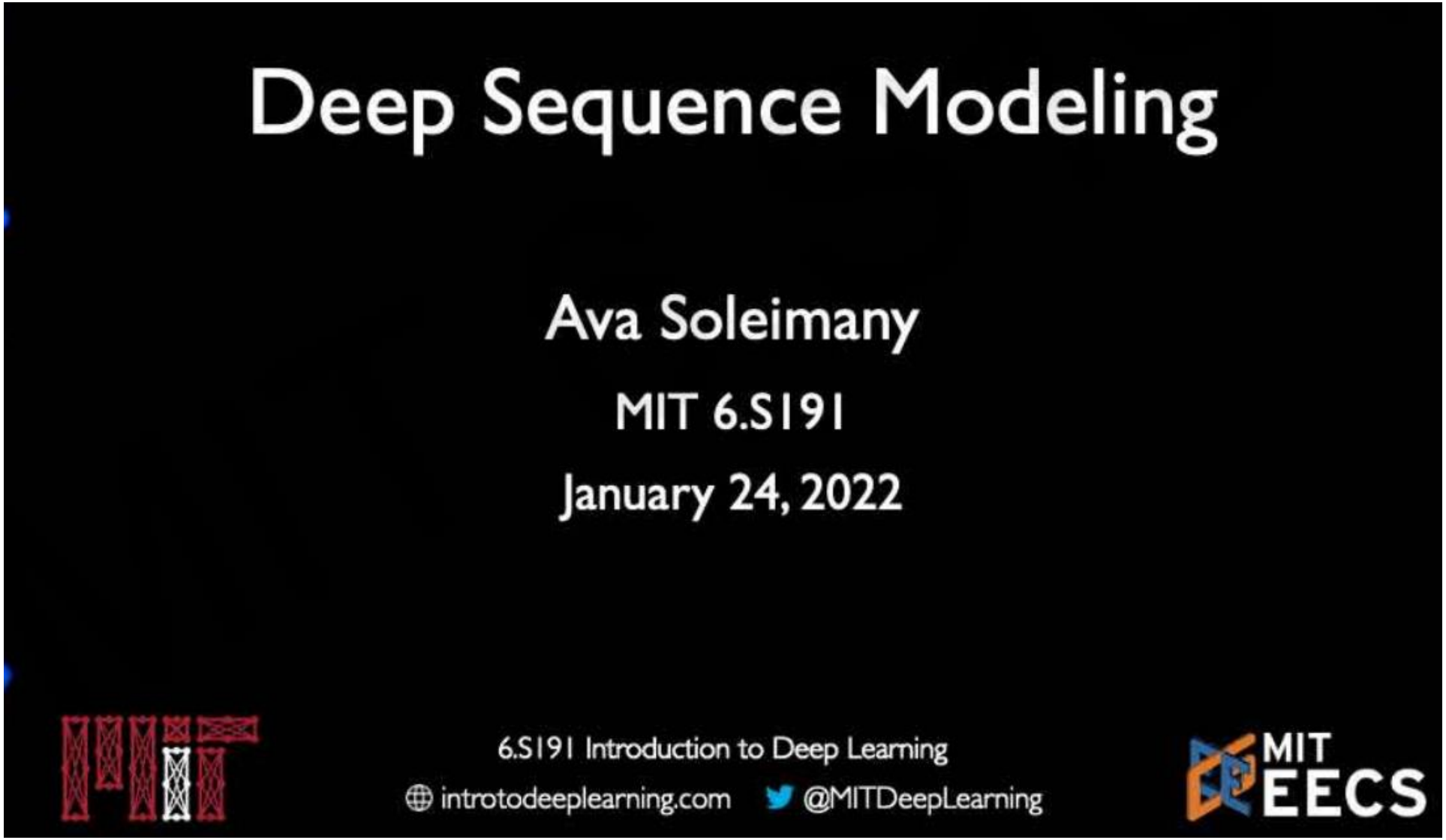
시계열 데이터 분석을 위한 RNN 모델 이해

2022.6.22.

이 홍 석 (hsyi@kisti.re.kr)

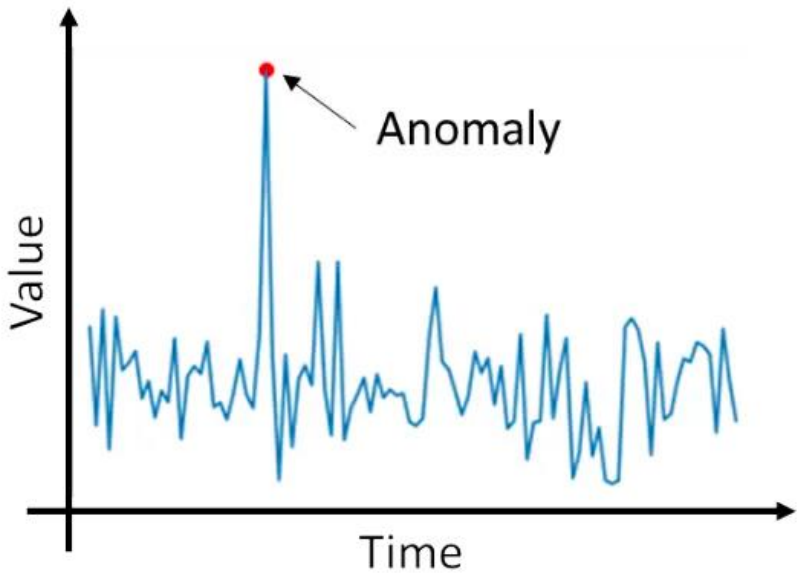
시계열 데이터 분석을 위한 RNN 이해

Reference : MIT Intorduction to Deep Learning



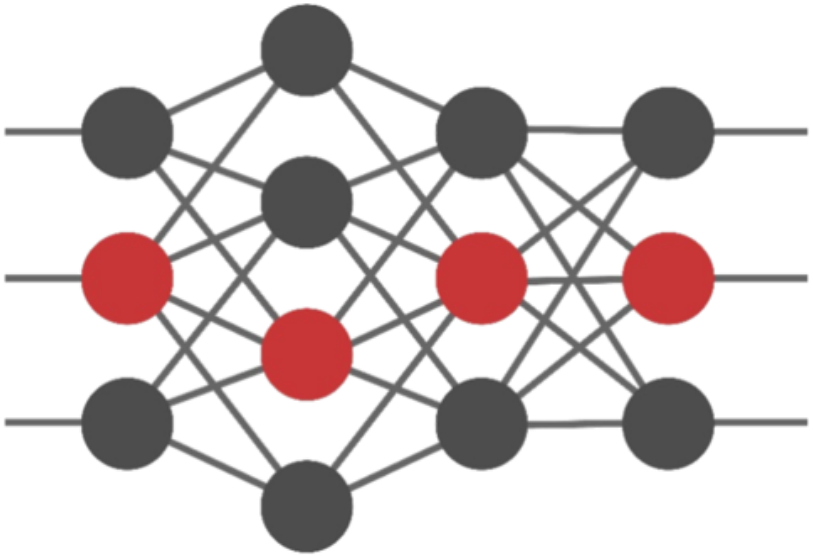
What is a time series problem?

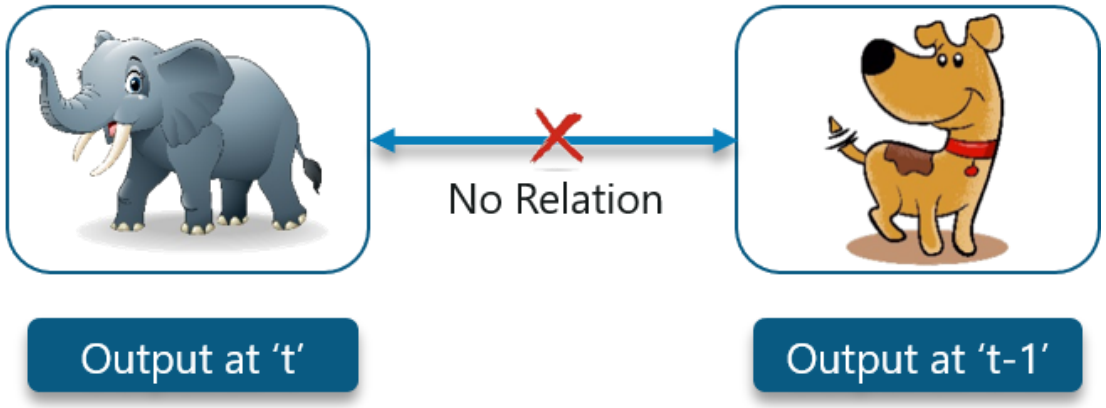
- Time series deals with data over time:
  - ✓weather (short term forecasts), climate (long term forecasts)
  - ✓sales accross different geographies for different products in the catalog
  - ✓Identifying special events in time : Anomalous weather events



Why Not Feedforward Networks

- 기존 FeedForwared Networks에서는
  - ✓입력은 개 이미지이고, 학습이 잘된 신경망은 개의 라벨을 출력한다.
  - ✓바로 다음 코끼리 이미지를 입력을 사용하면 코끼리(라벨)을 출력한다.
  - ✓하지만 코끼리와 개는 서로 관련성이 없고 독립적이다.

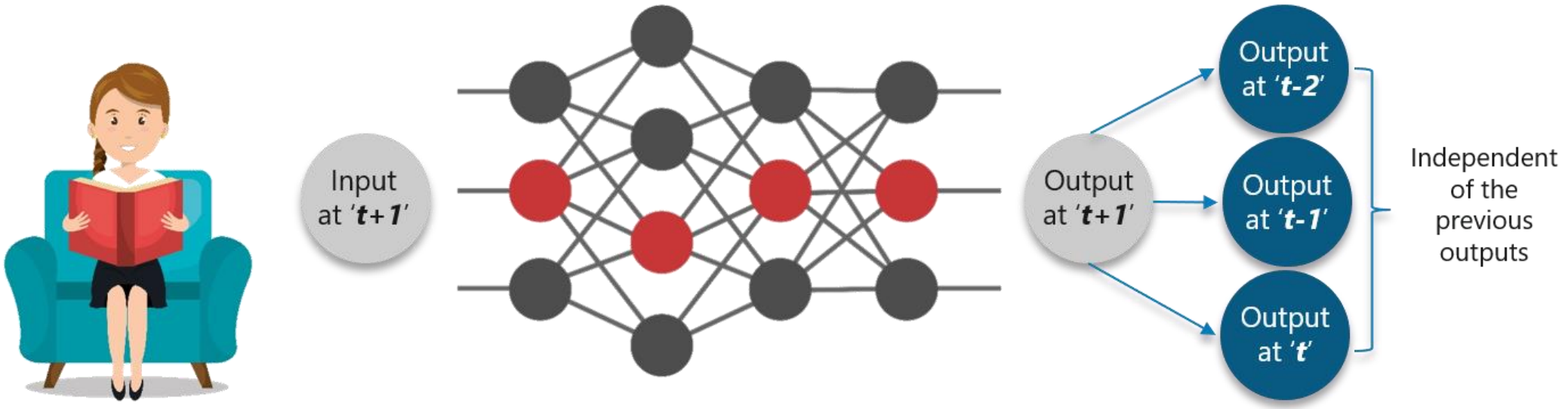




ref: https://www.edureka.co/blog/recurrent-neural-networks/

Why Not Feedforward Networks

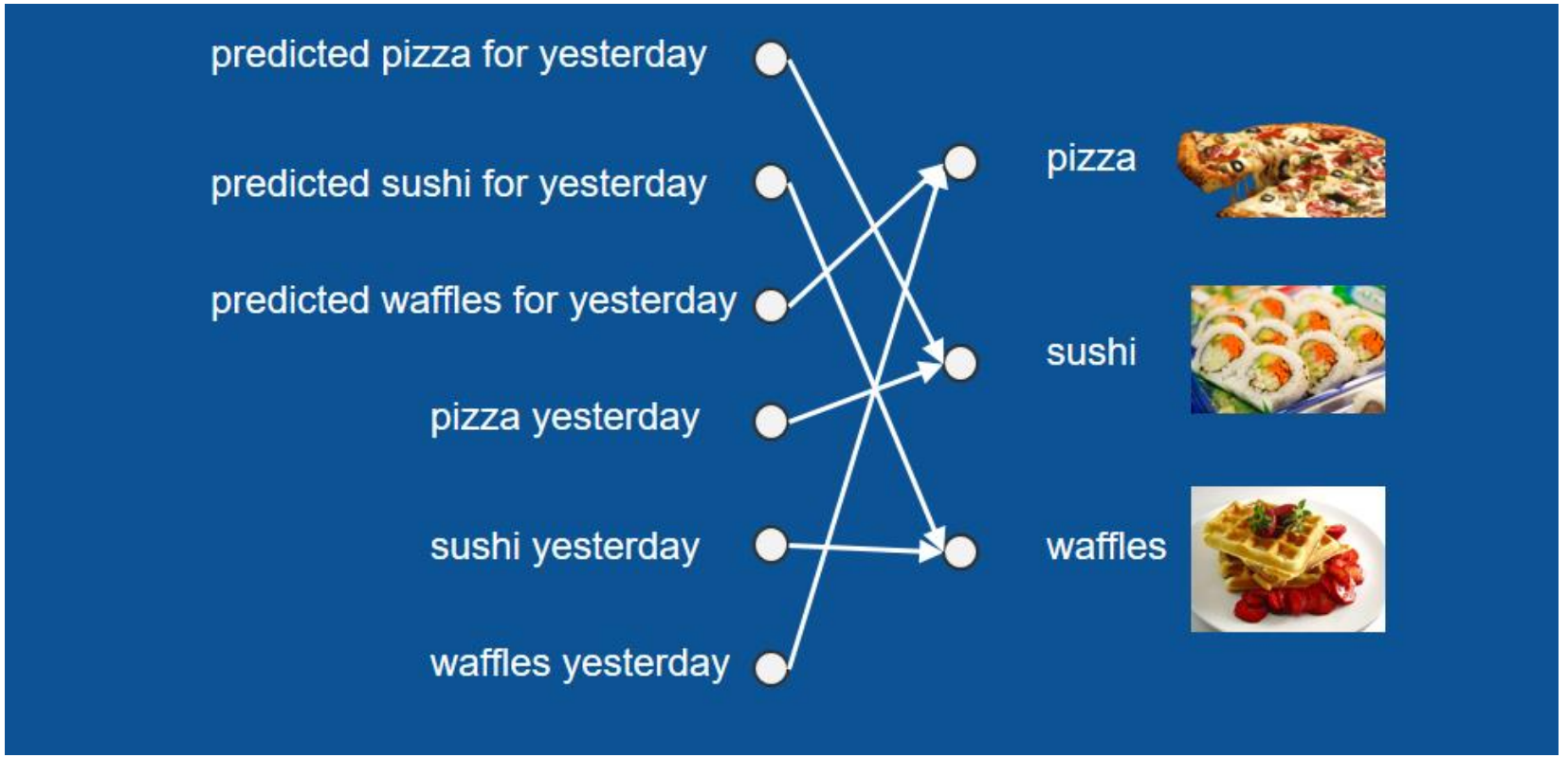
- 책을 읽고 있을 경우, 이전 페이지의 내용을 잘 알고 있어야 한다.
  - ✓책의 문맥을 이해하기 위해서는 이전의 내용 및 이전 단어를 기억하고 있어야 한다.
  - ✓그래서 Feeforward 네트워크는 이전 단어들로 다음 단어의 관계를 예측해 줄 수 없다.
  - ✓ 즉 이전의 내용(단어)를 기억해 줄 수 있는 네트워크가 필요하다.



What Are Recurrent Neural Networks?

❖ 음식을 먹을 경우도 이전 기억이 중요하다.

✓ RNNs are a **type** of **artificial neural network** designed to **recognize patterns** in sequences of data



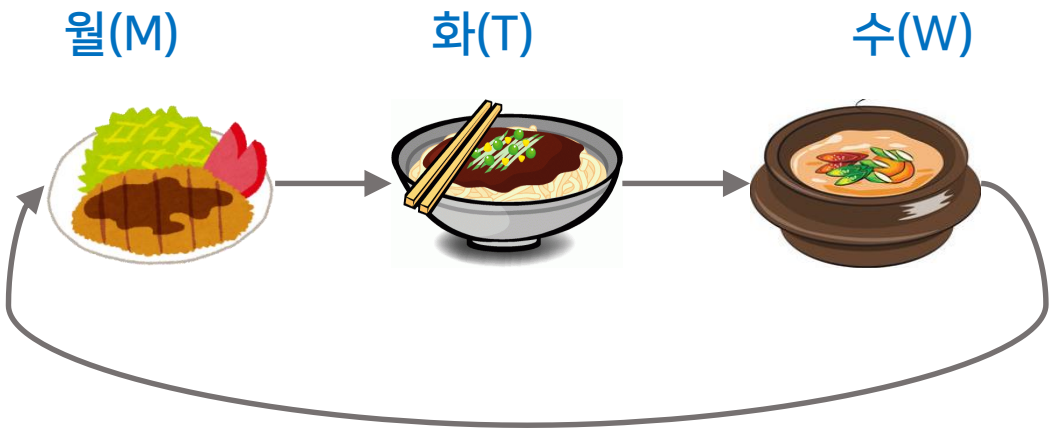
간단한 순환신경망(RNN)

❖ **함께 살고 있는 룸메이트는 요리를 하느데 3종류만 할 줄 안다.**

✓친구는 저녁을 하는데 3가지 음식만 만든다.

✓월요일은 돈까스, 화요일은 짜장면, 수요일은 찌게 등 순차적으로 3가지 음식을 반복한다.

✓룸메이트가 기억하고 있는 패턴은 단순하다.



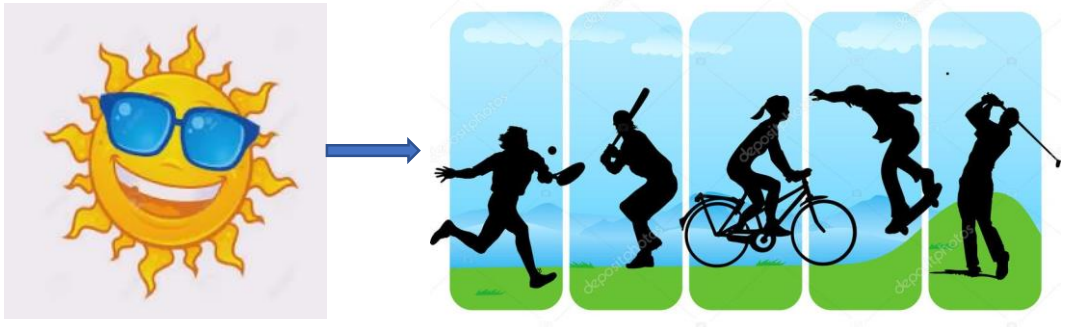
RNN = Recurrent Neural Network

간단한 순환신경망(RNN)

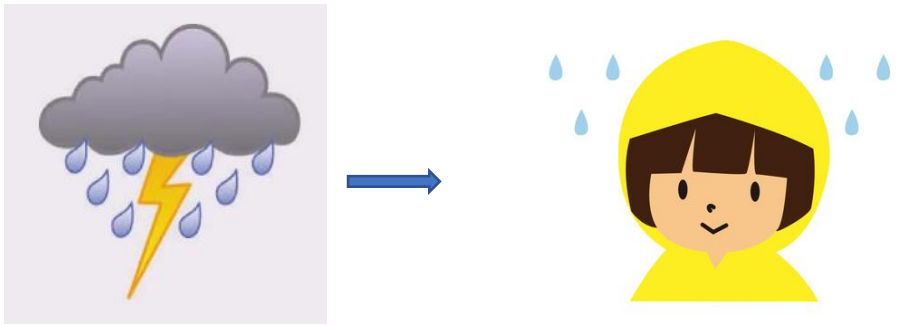
❖ 실제로 룸메이트는 날씨에 따라서 식사 메뉴에 규칙을 가지고 있다.

✓날씨가 좋은 날은 야외활동해서 늦게 귀가하며 전날 먹은 음식을 다시 먹는다.

✓하지만 비가 오는 날은 일찍 집에 귀하하여 원래 패턴을 따라서 음식을 만든다.



어제 먹었던 음식

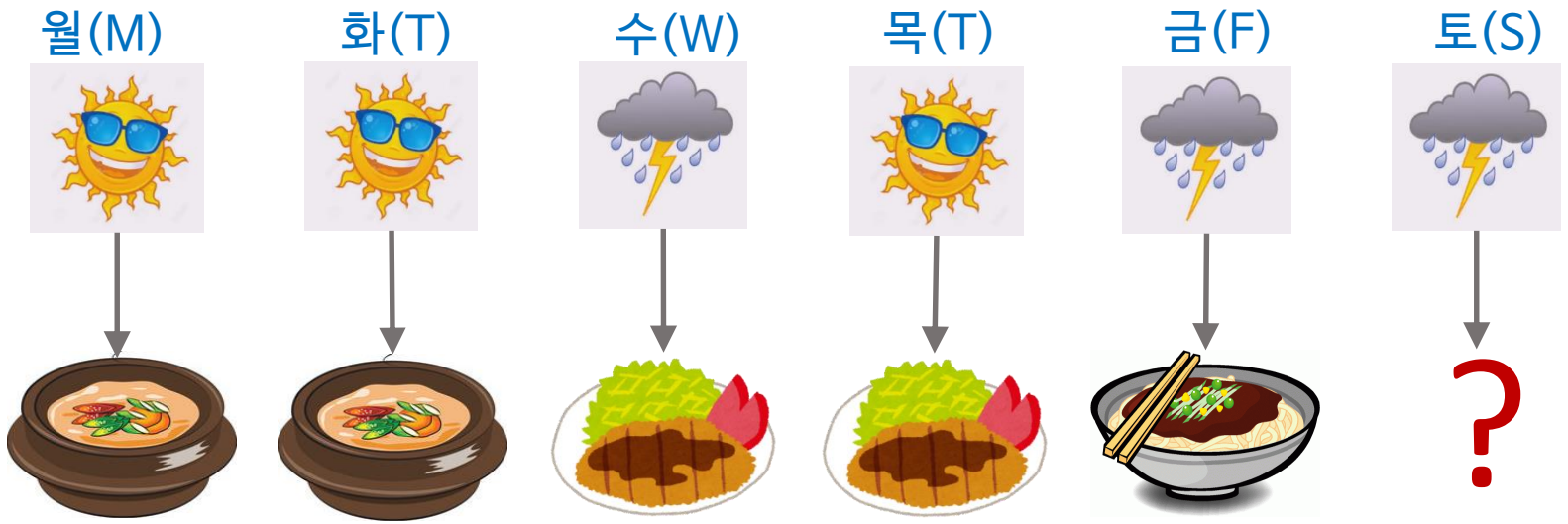


원래 RNN 패턴

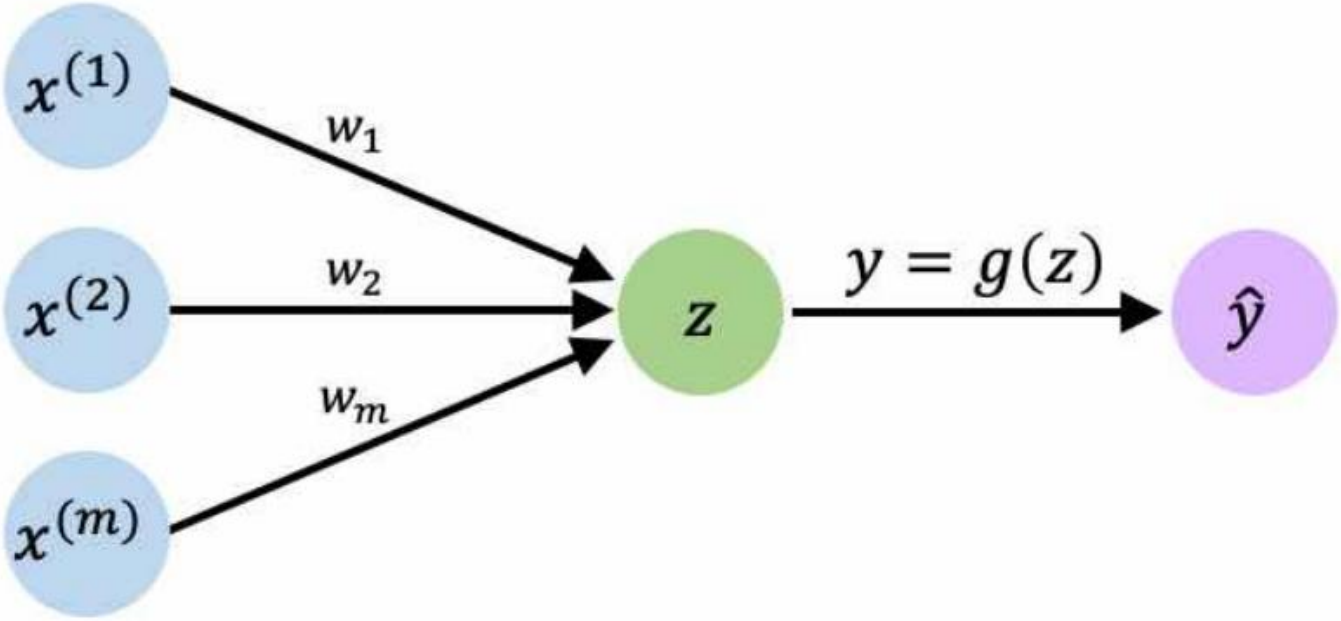
RNN depending on the weather

❖ 나는 오늘 아침 날씨에 따라서 저녁에 먹을 음식을 예측할 수 있다.

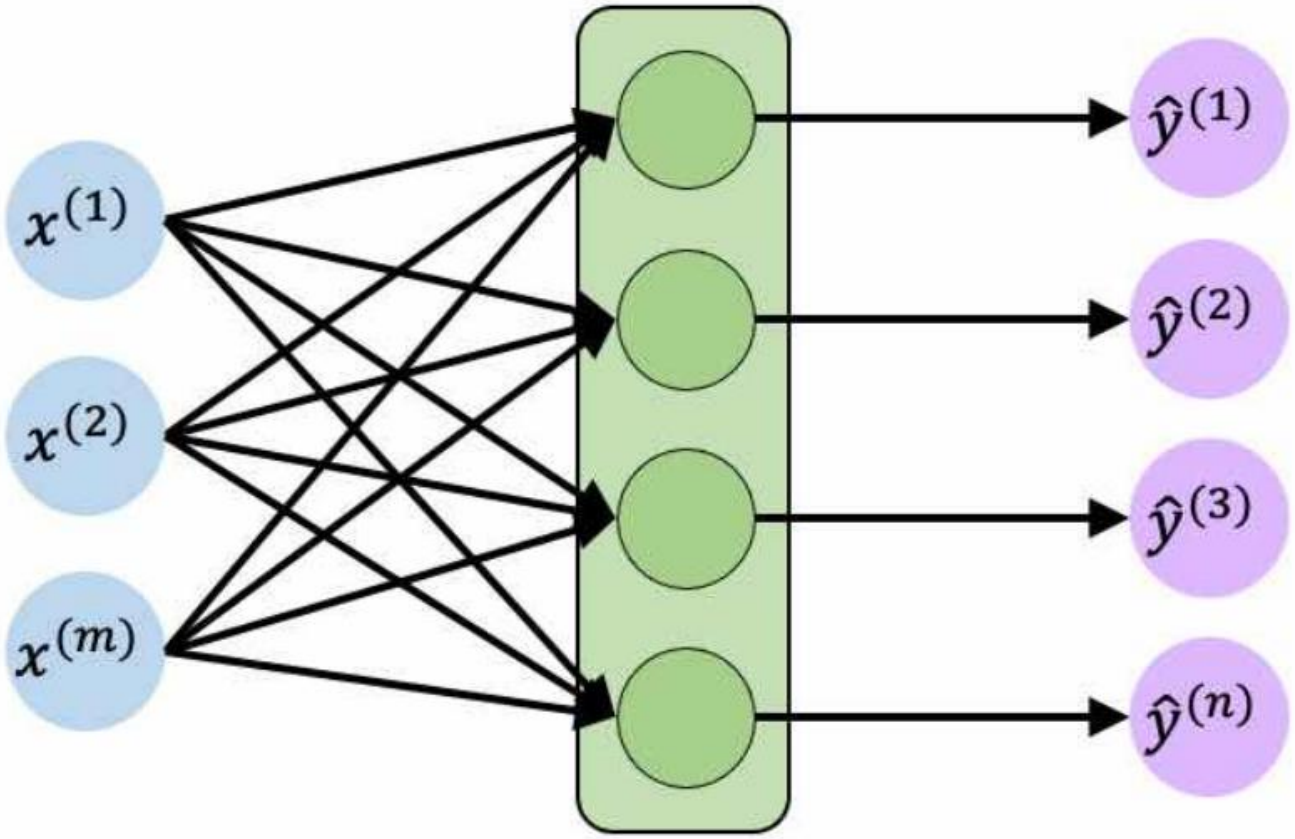
✓월요일은 저녁에 찌게를 먹었고, 화요일 아침 날씨가 좋아서 저녁에도 찌게를 먹었다.
✓수요일은 비가와서 원래 패턴인 돈까스, 목요일은 좋은 날씨라서 어제 먹은 돈까스
✓금요일은 비가와서 원래 패턴인 짜장면을 먹었다.
✓토요일에 비가 왔는데 저녁 메뉴는 무었인가?



The Perceptron Revisited

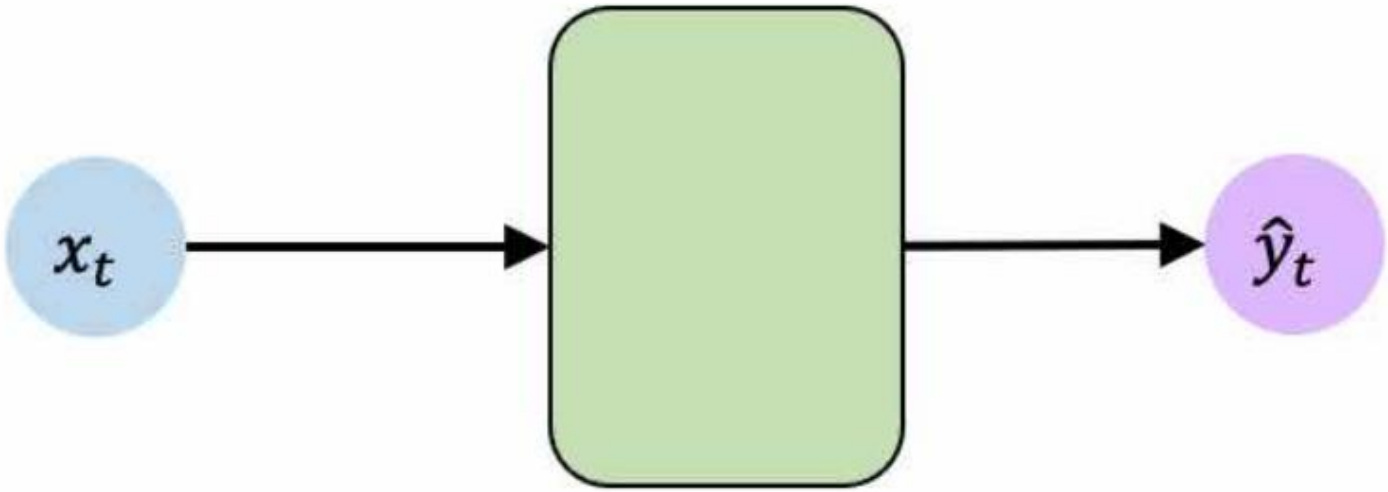


Feed-Forward Networks Revisted



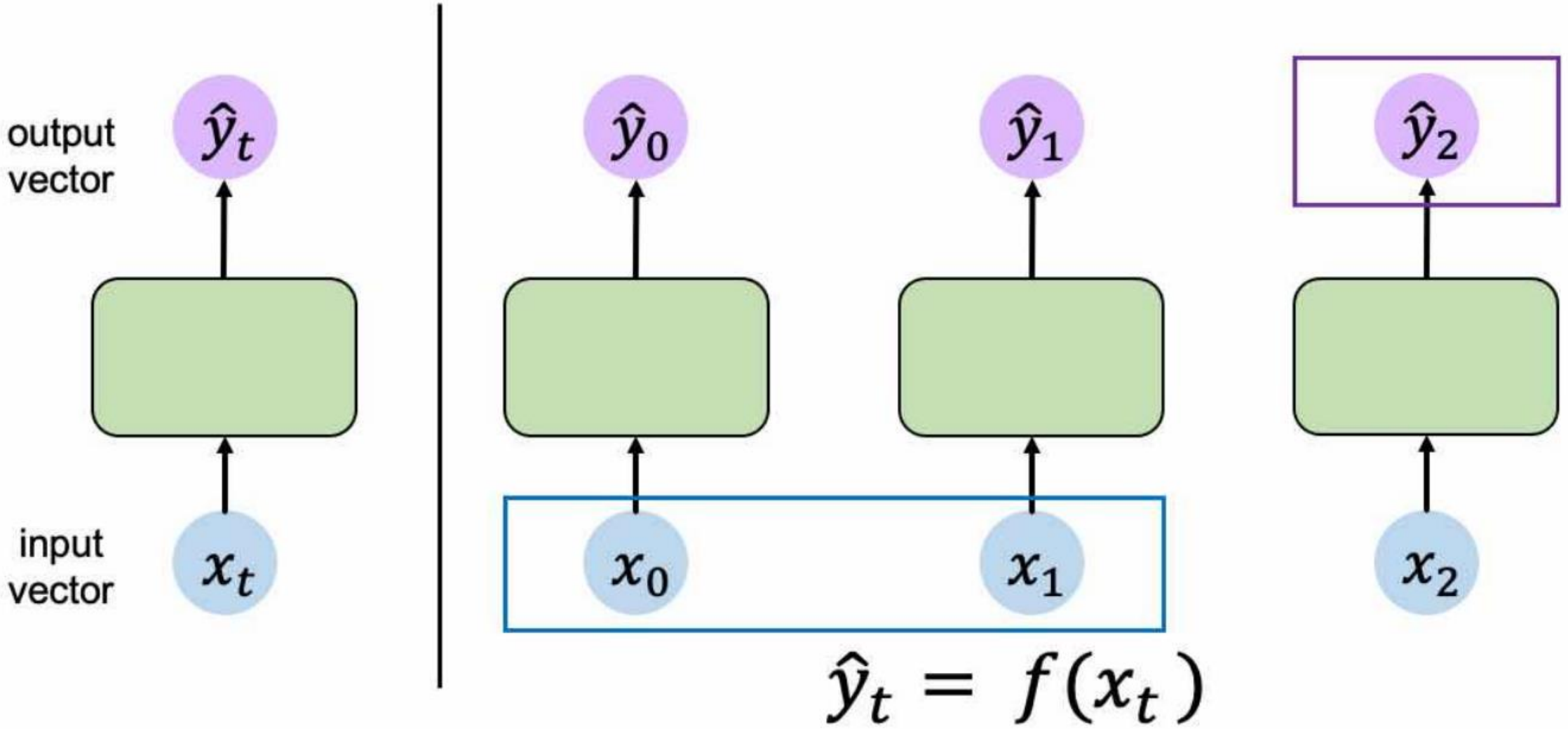
$$\boldsymbol{x} \in \mathbb{R}^m \qquad \hat{\boldsymbol{y}} \in \mathbb{R}^n$$

Feed-Forward Networks Revisted

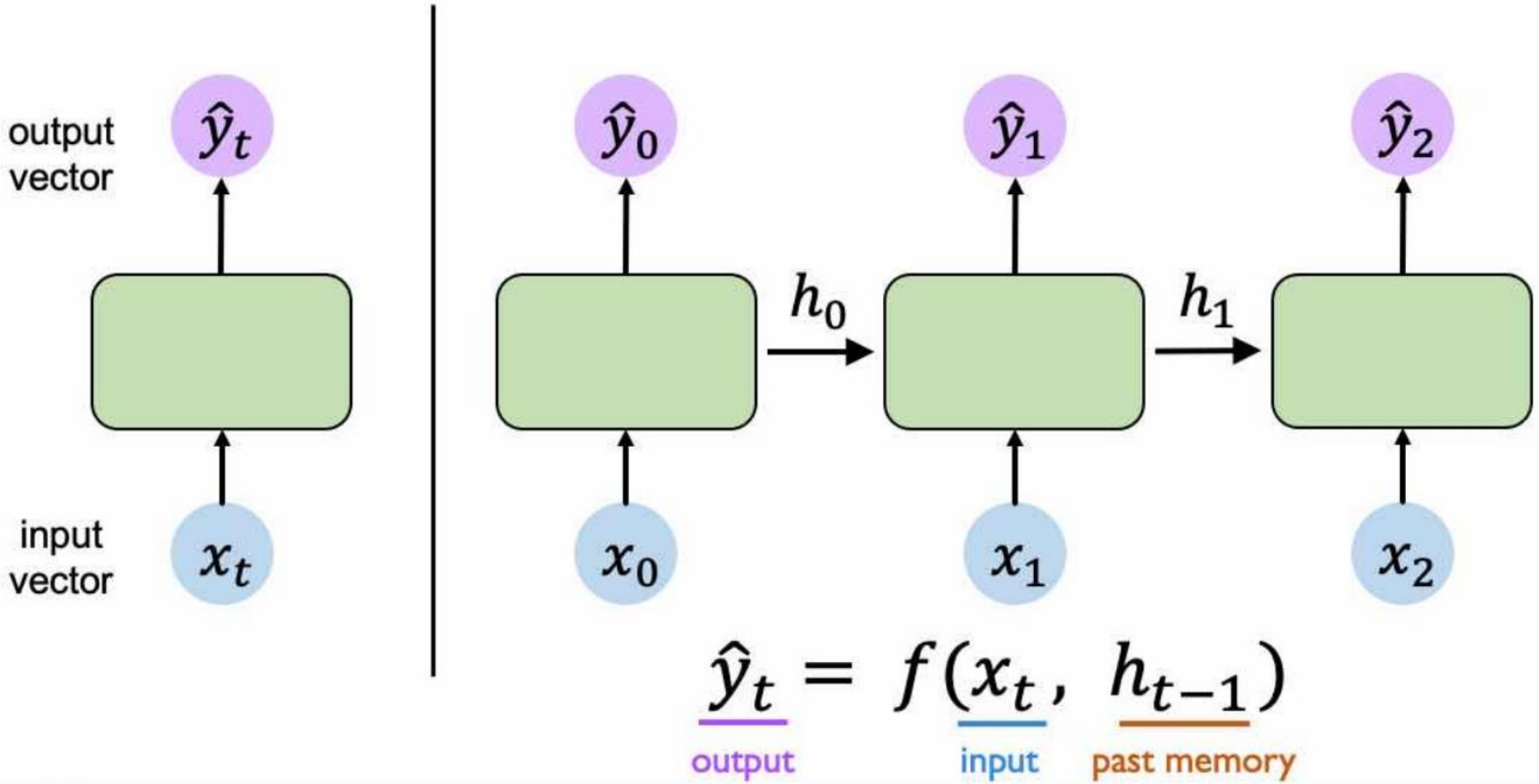


$$\boldsymbol{x}_t \in \mathbb{R}^m \qquad \hat{\boldsymbol{y}}_t \in \mathbb{R}^n$$

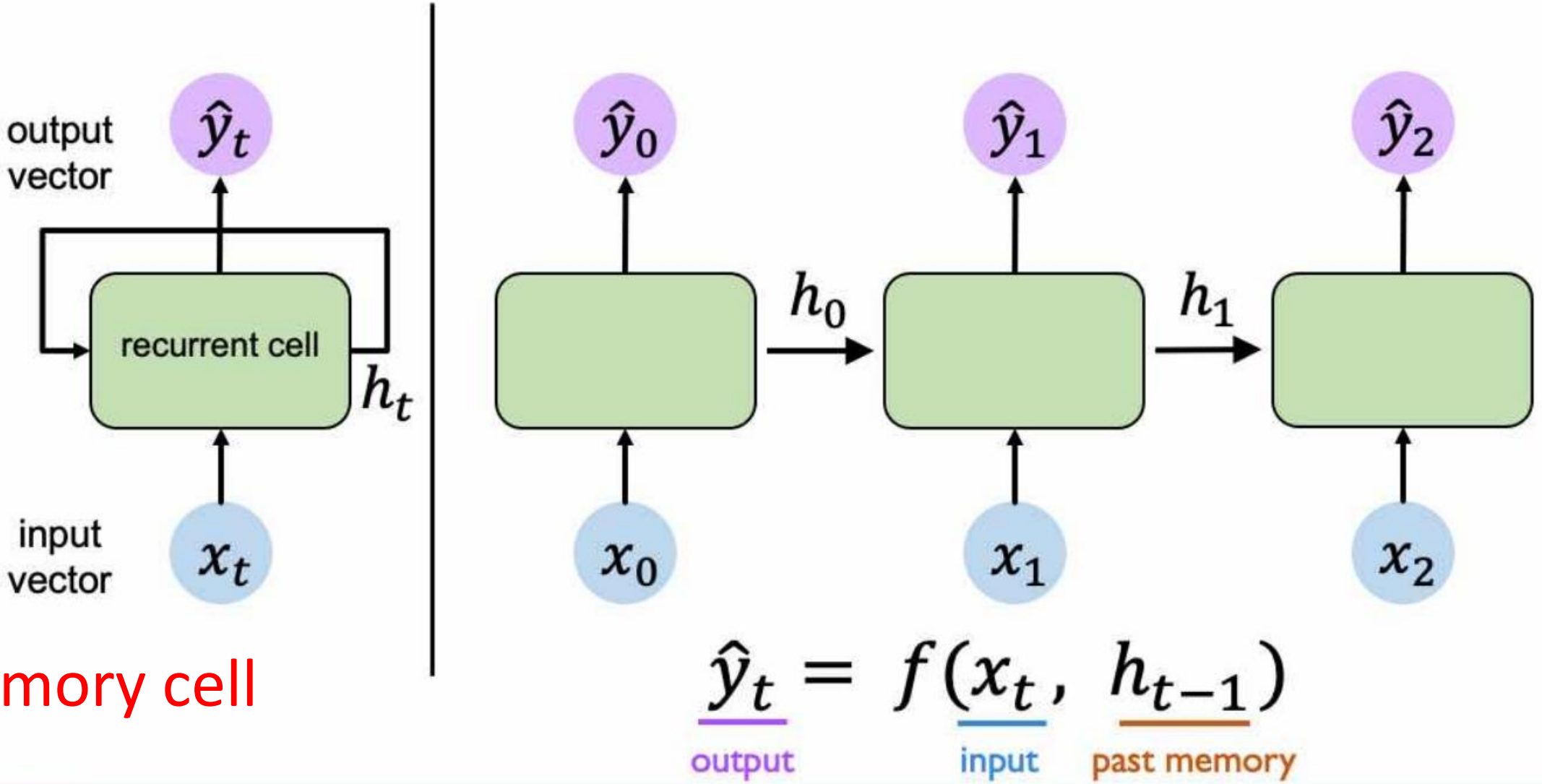
Handling Individual Time Steps



Neurons with Recurrence



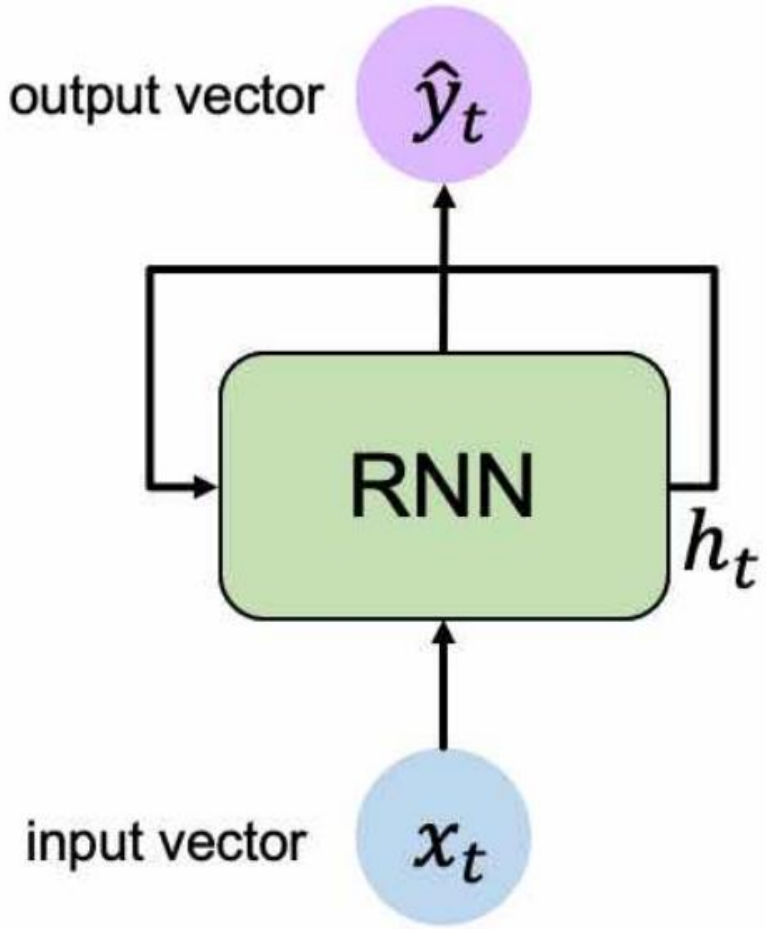
Neurons with Recurrence



memory cell

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs)



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

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

cell state

function with weights W

input

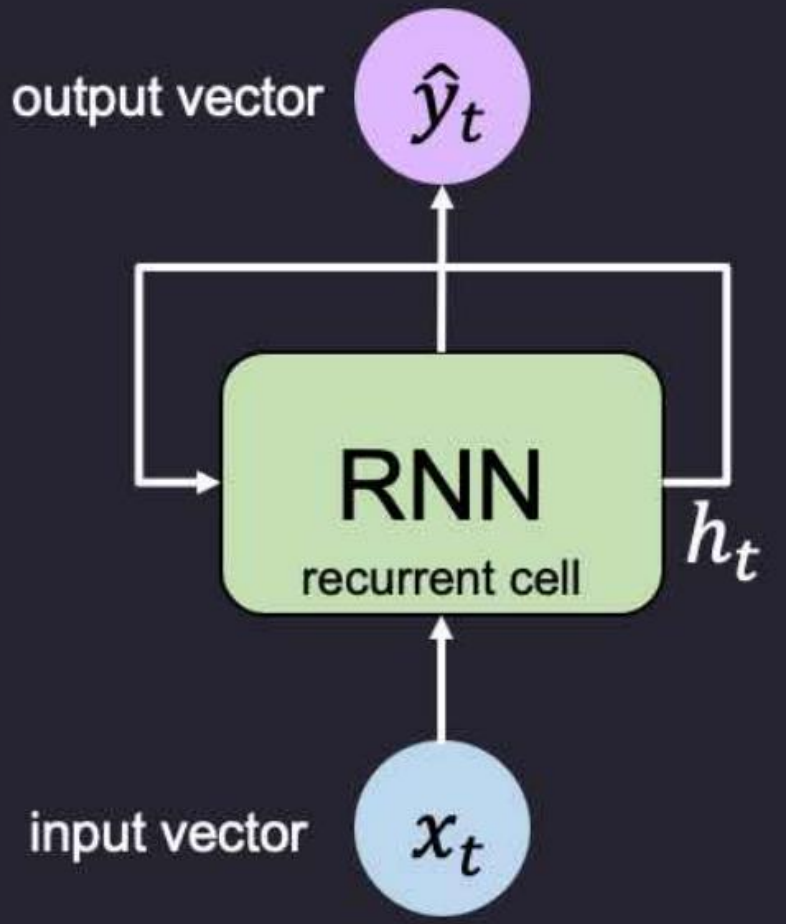
old state

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

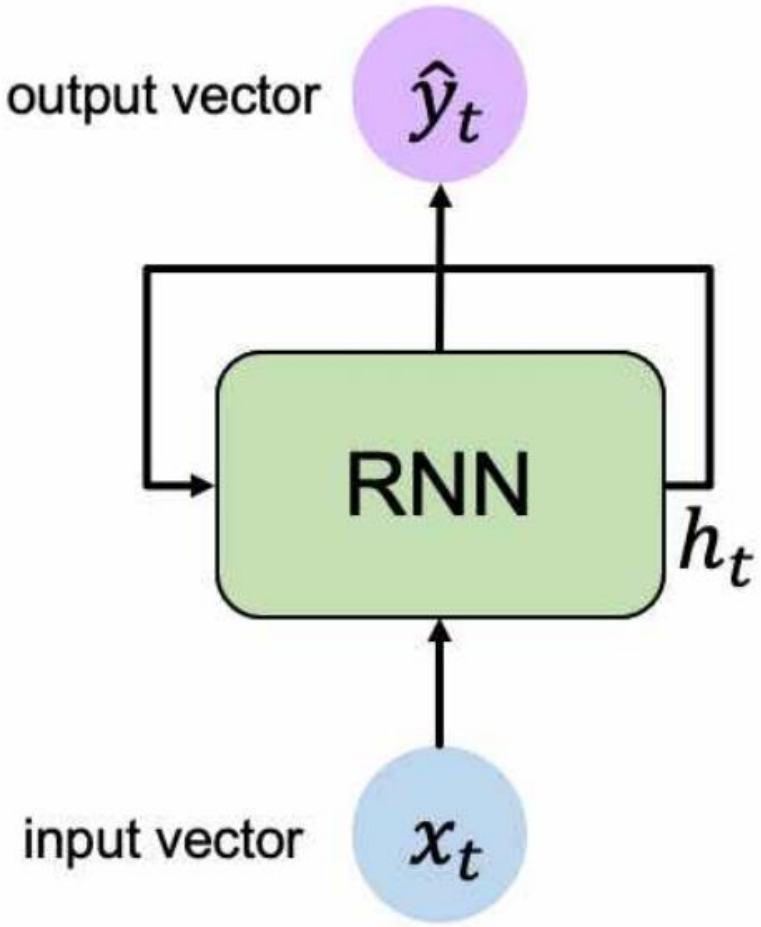
RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{hy}^{T} h_t$$

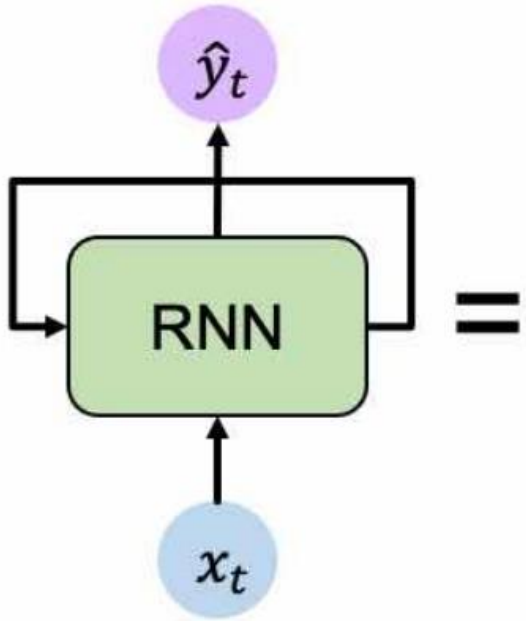
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^{T} h_{t-1} + \boldsymbol{W}_{xh}^{T} x_t)$$

Input Vector

$$x_t$$

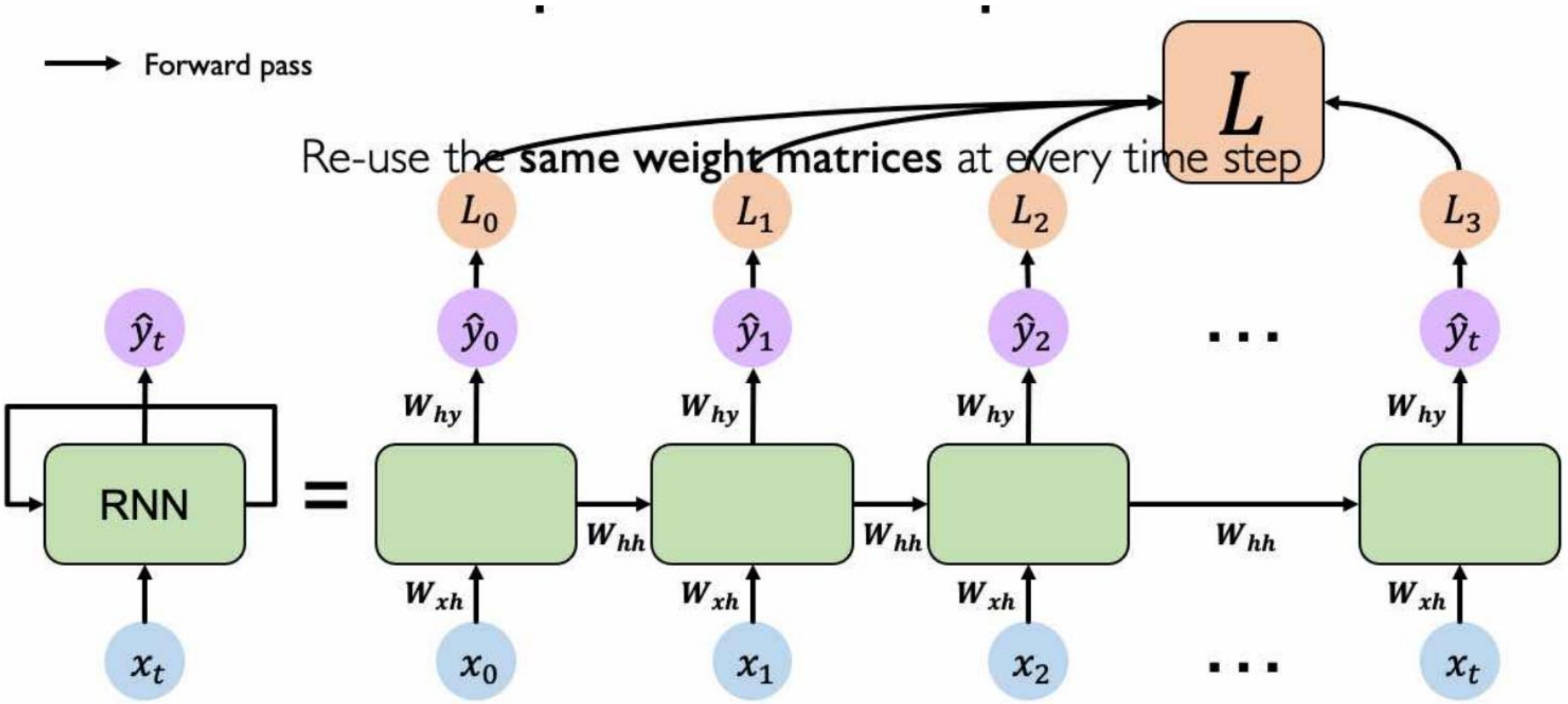
RNNs: Computational Graph Across Time



=

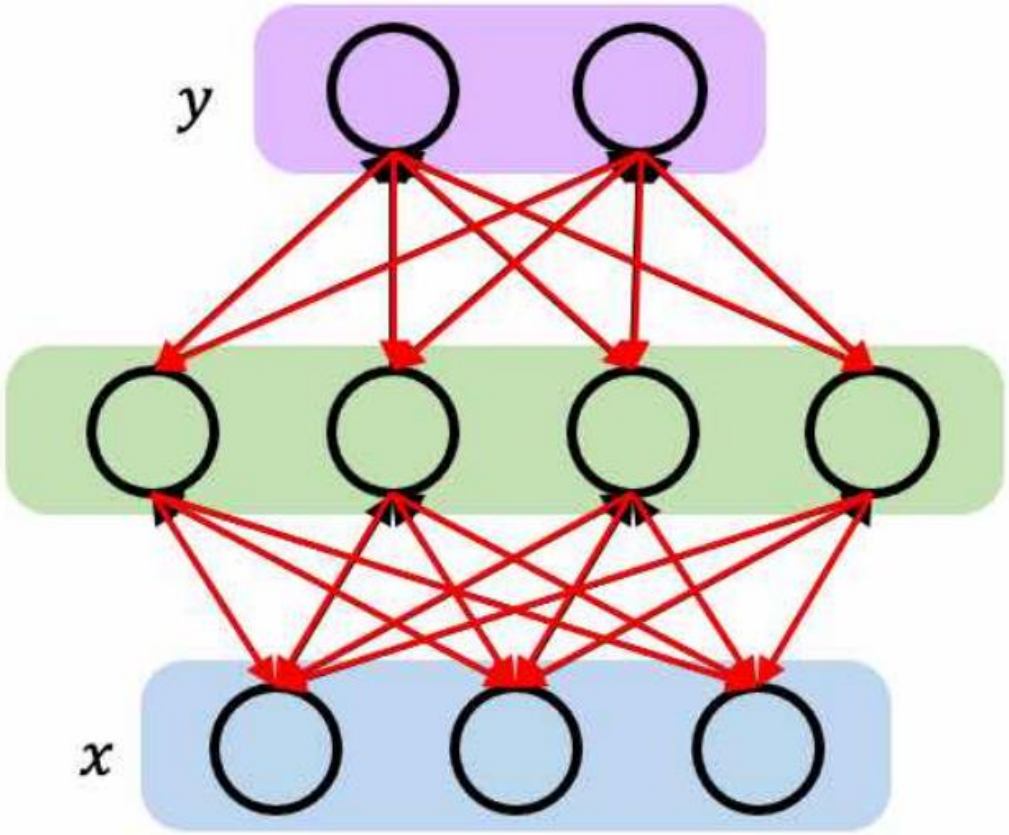
Represent as computational graph unrolled across time

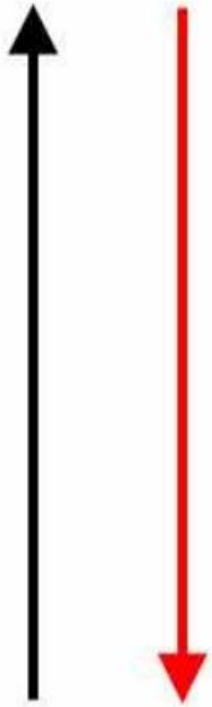
RNNs: Computational Graph Across Time



Backpropagation Through Time (BPTT)

Backpropagation in Feed Forward Models

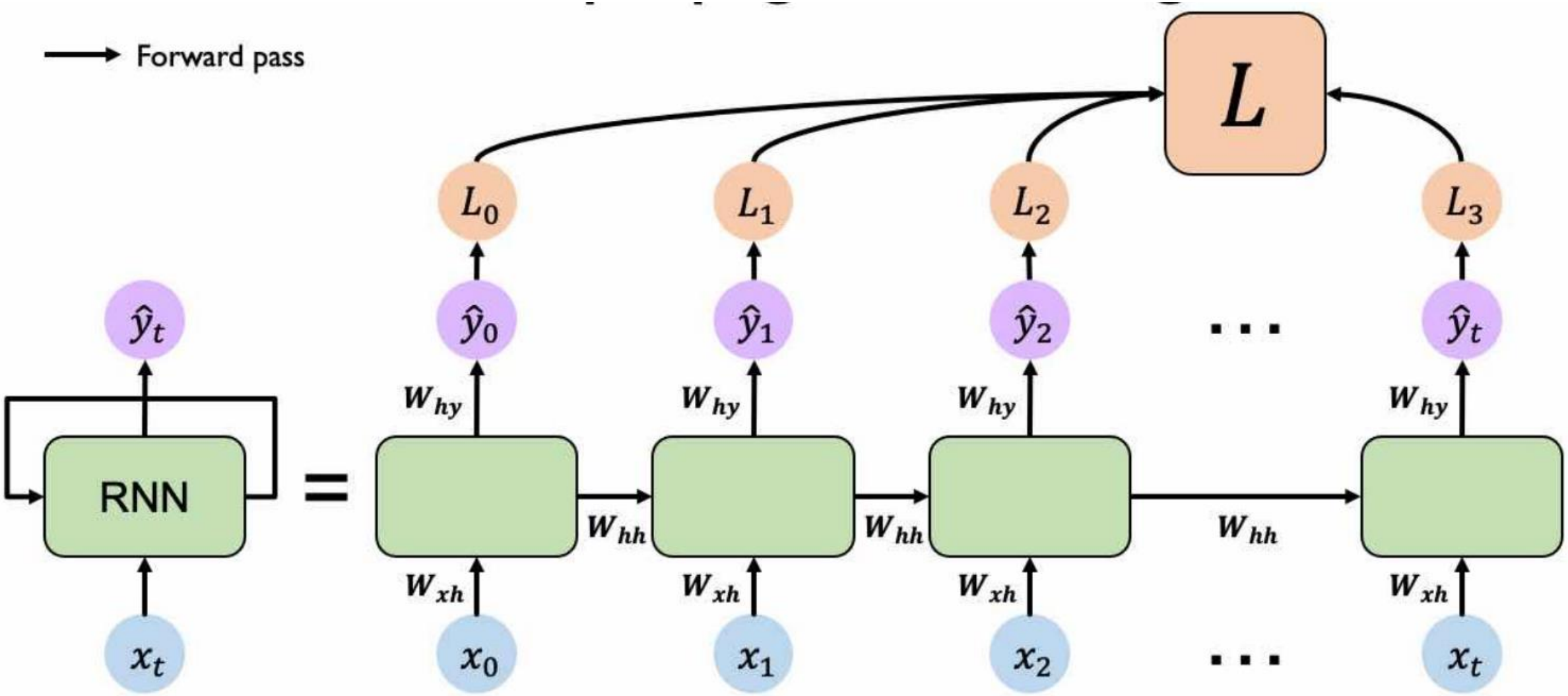




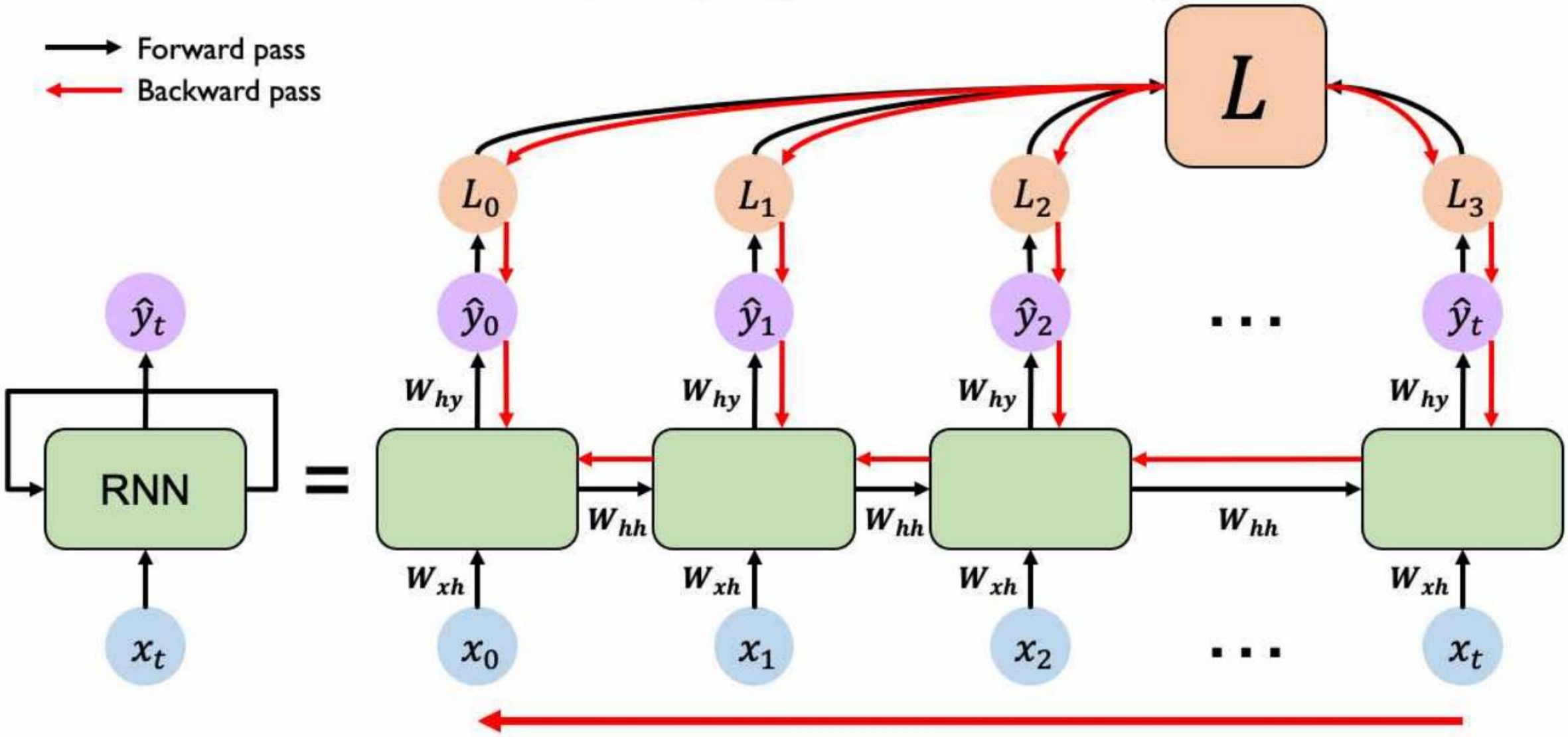
Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

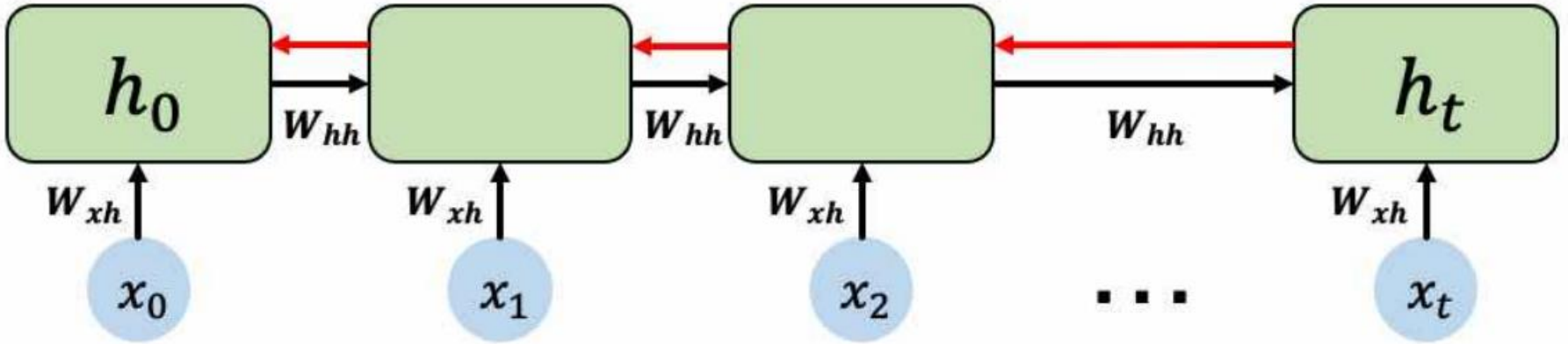
Backpropagation Through Time



Backpropagation Through Time

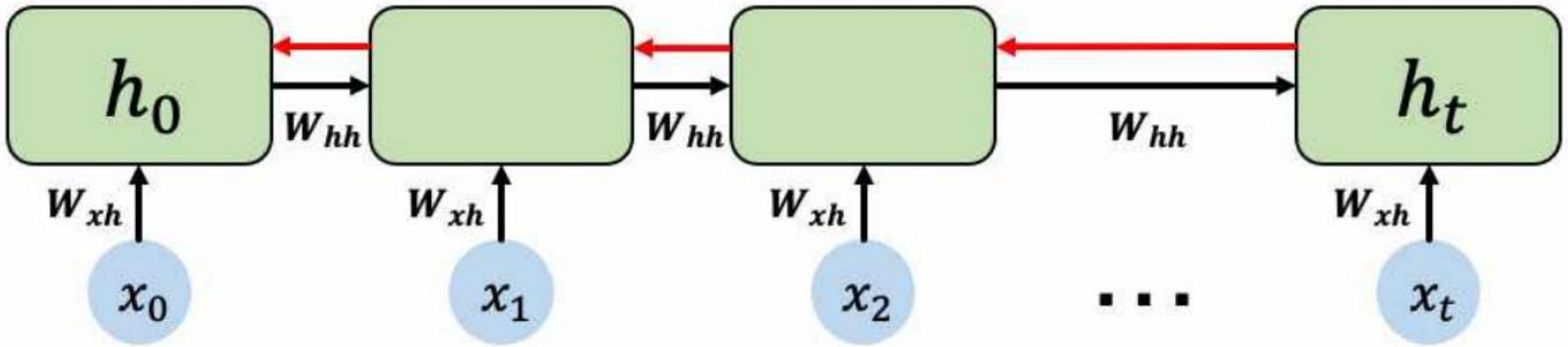


Standard RNN Gradient Flow



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Standard RNN Gradient Flow: Expoding Gradients

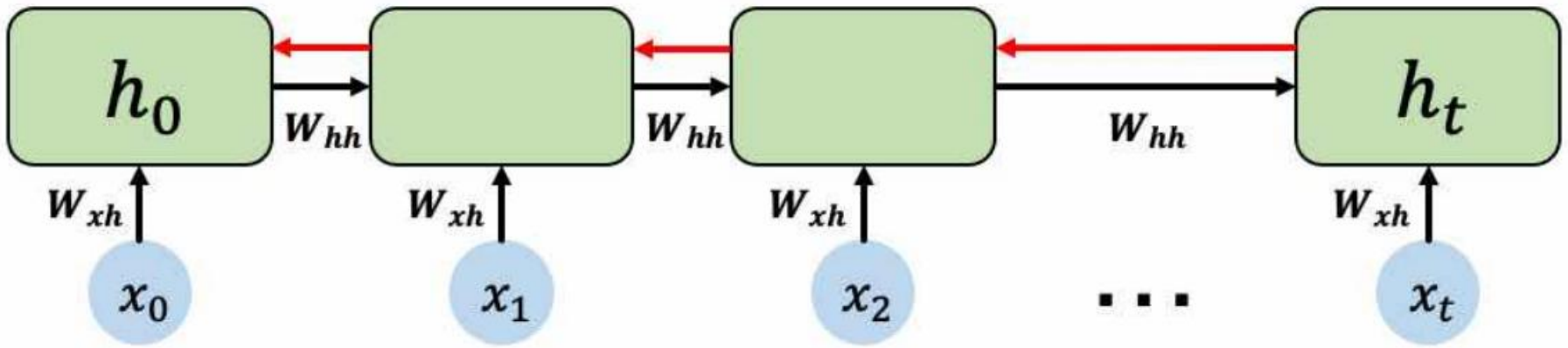


Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

Standard RNN Gradient Flow: Vanishing Gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Many values > 1:
exploding gradients

Gradient clipping to
scale big gradients

Many values < 1:
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

Backpropagation through time

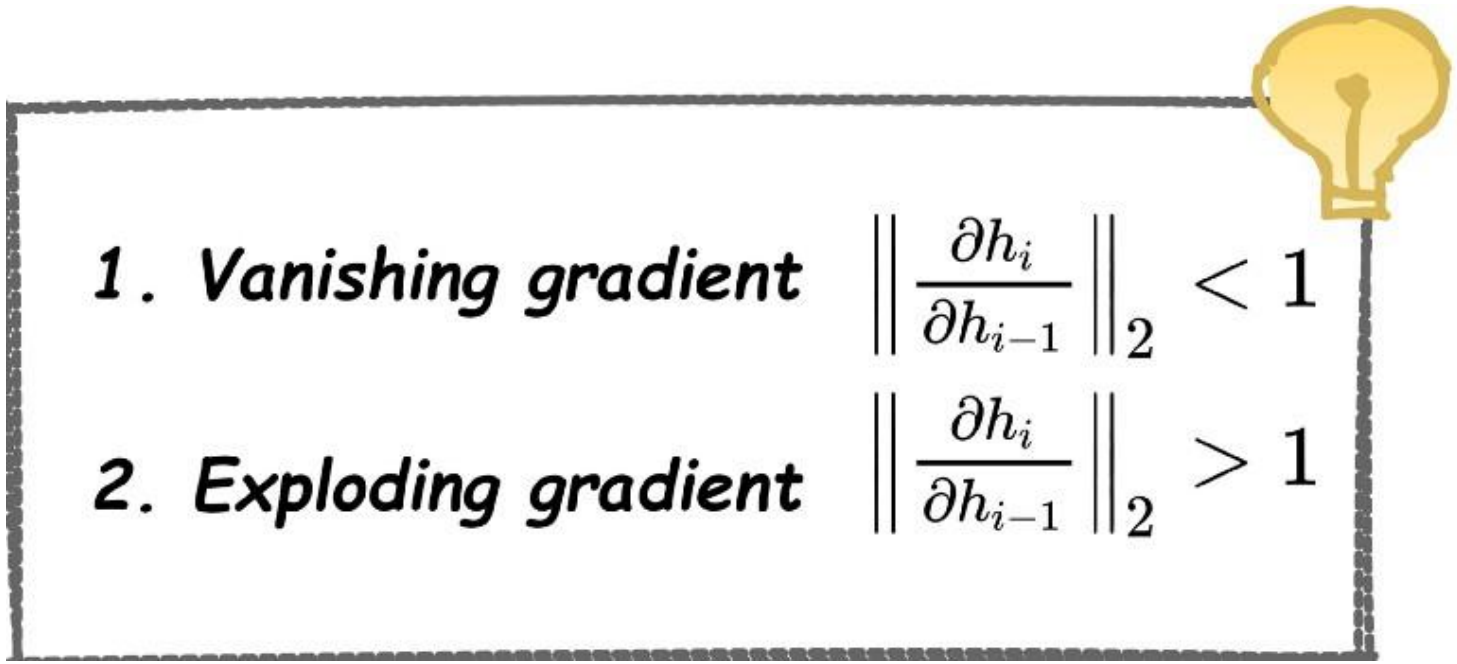
$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

BPTT suffer from the problem of VGP and EGP

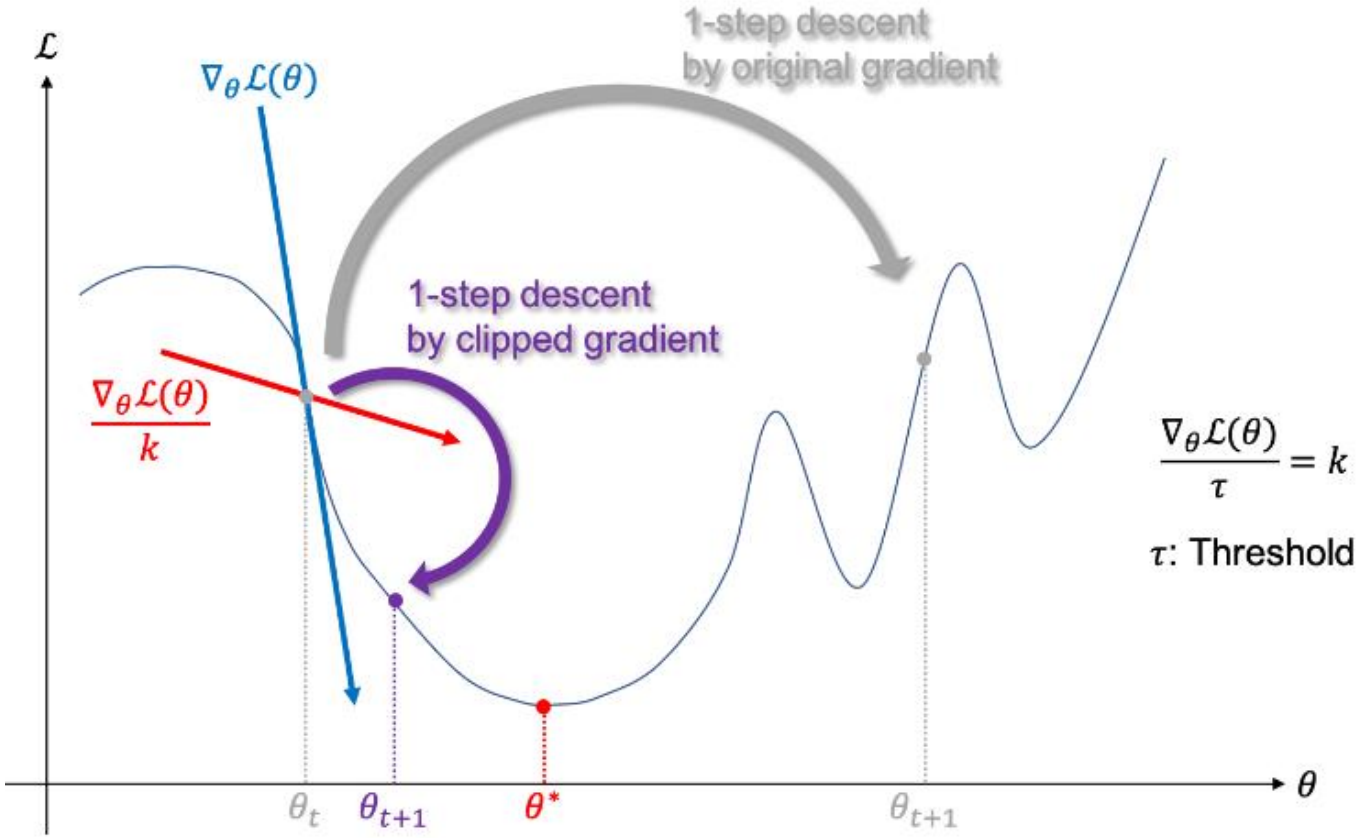
- VGP(Vanishing Gradient Problem)
  - ✓when the gradient becomes too small, the parameter updates become insignificant.
- EGP(Exploding Gradient Problem)
  - ✓ If the slope tends to grow exponentially instead of decaying

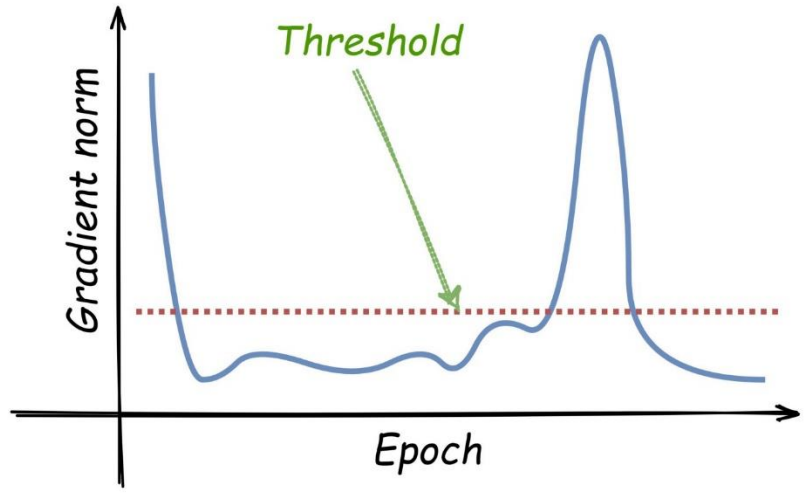


Solutions to the vanishing/exploding gradient problems

- **Truncated backpropagation through time (TBPTT)**
  - ✓ simply limits the number of time steps the signal can backpropagate after each forward pass.
- **Long short-term memory (LSTM)**
  - ✓ uses a memory cell for modeling long-range dependencies and avoid VGP
- **Gradient Clipping**
  - ✓ set a max value for gradients if they grow to large

Gradient clipping



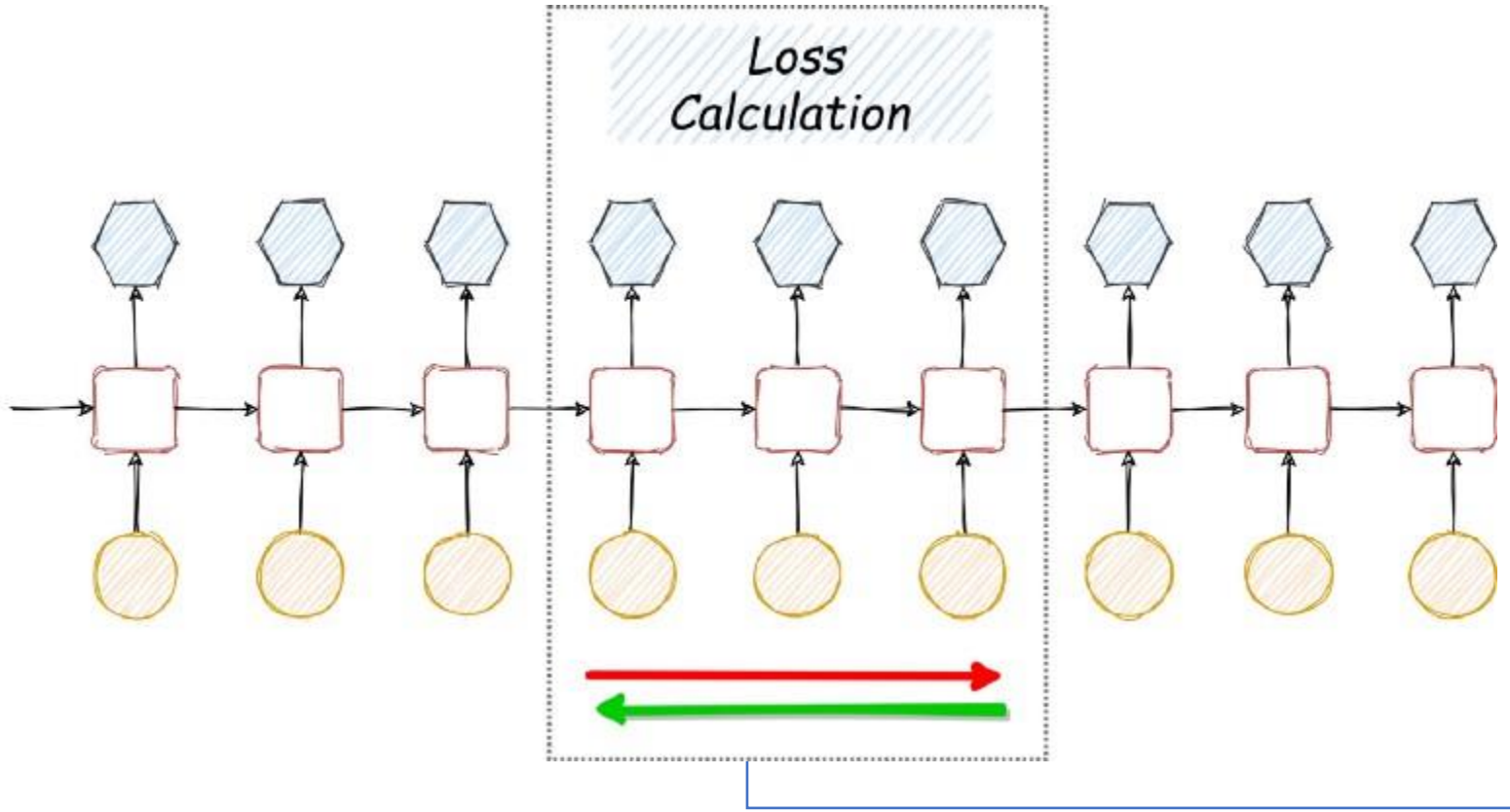


https://sanghyu.tistory.com/87

https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series

Truncated Backpropagation Through Time

Truncated BPTT trick tries to overcome the VGP by considering a moving window through the training process



Run forward and backward through chuncks of the sequence instead of whole sequence

Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps.

The problem of Long-Term Dependencies

Why are vanishing gradients a problem?

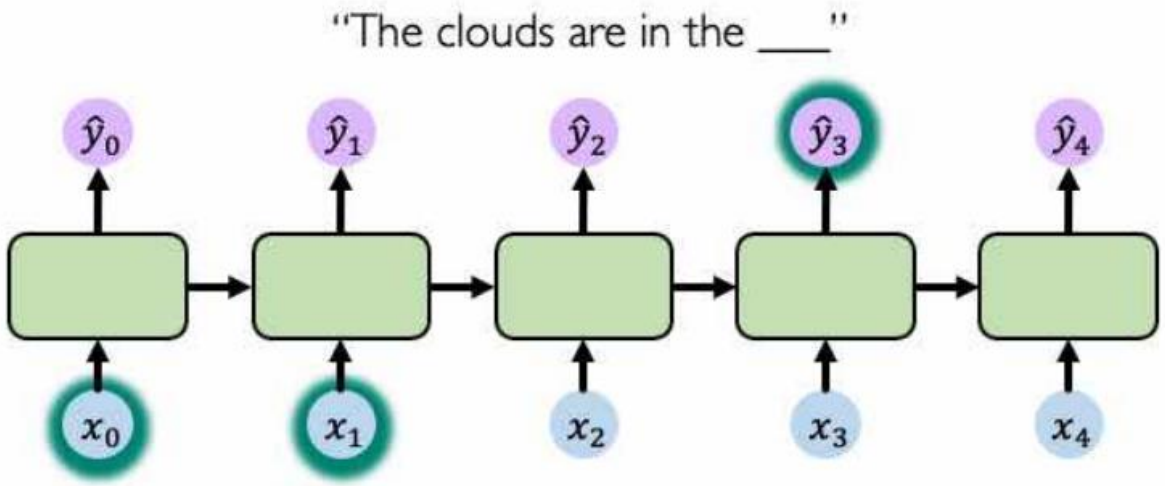
Multiply many **small numbers** together

↓

Errors due to further back time steps have smaller and smaller gradients

↓

Bias parameters to capture short-term dependencies



The problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

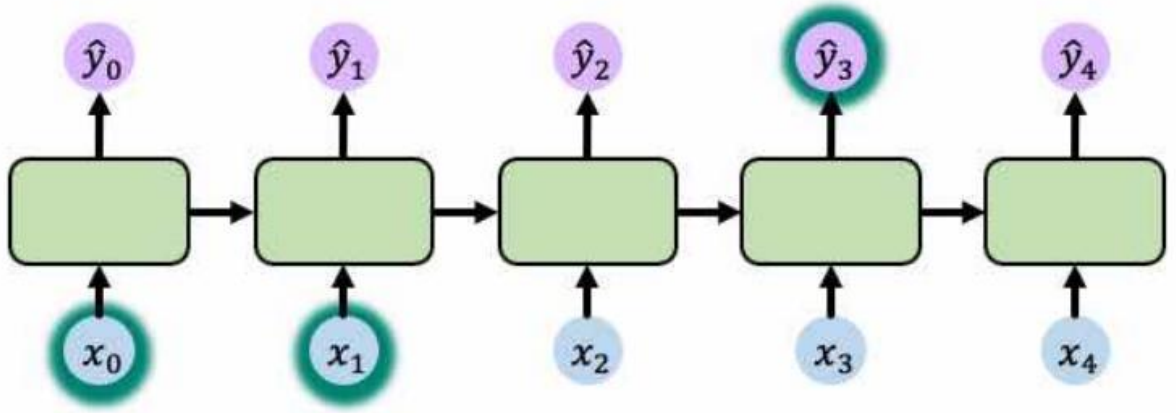
↓

Errors due to further back time steps have smaller and smaller gradients

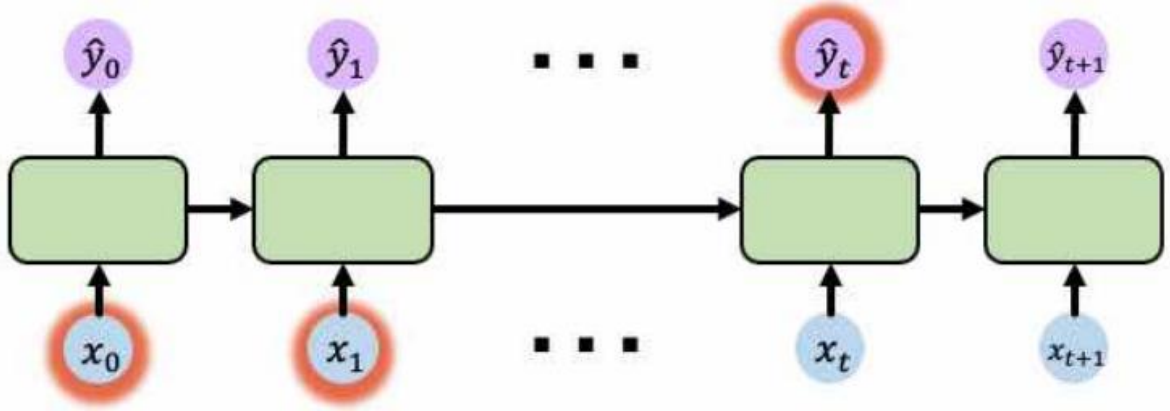
↓

Bias parameters to capture short-term dependencies

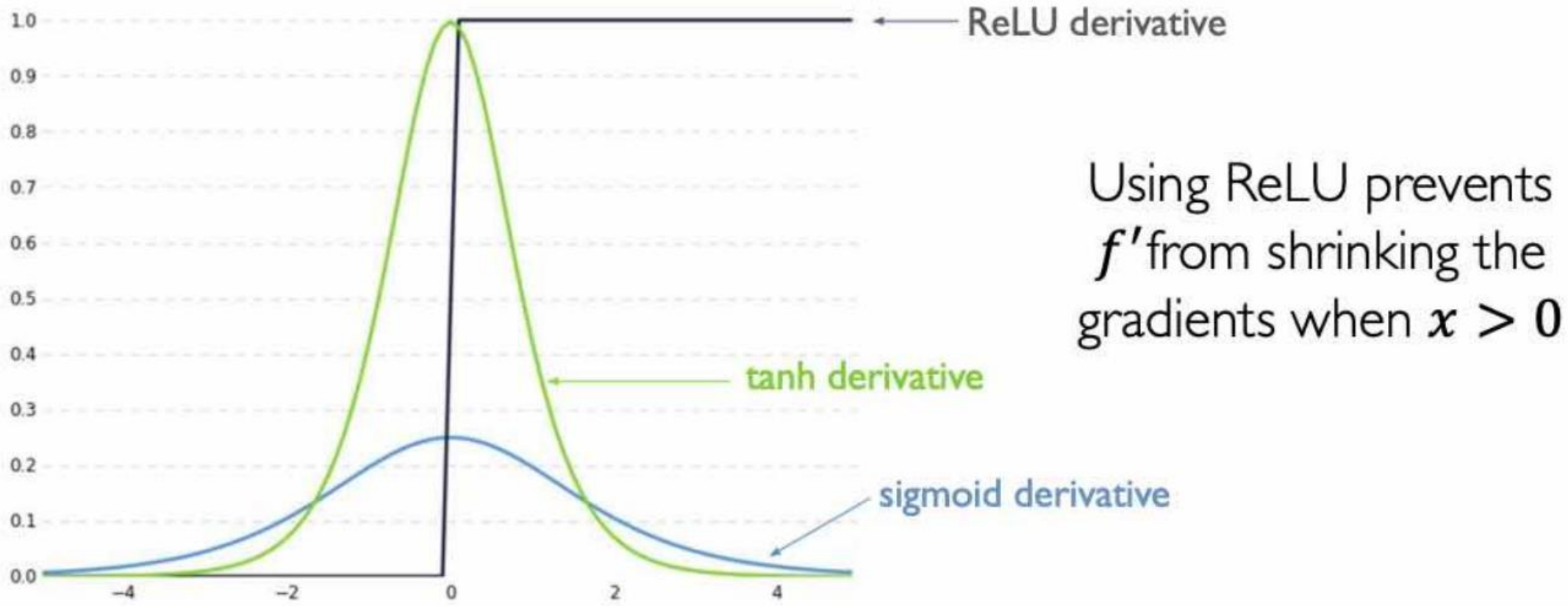
"The clouds are in the ___"



"I grew up in France, … and I speak fluent___"



Trick #1: Activation Fuctions



Using ReLU prevents f' from shrinking the gradients when $x > 0$

Trick #2: parameter Initialization

Initialize **weights** to identity matrix

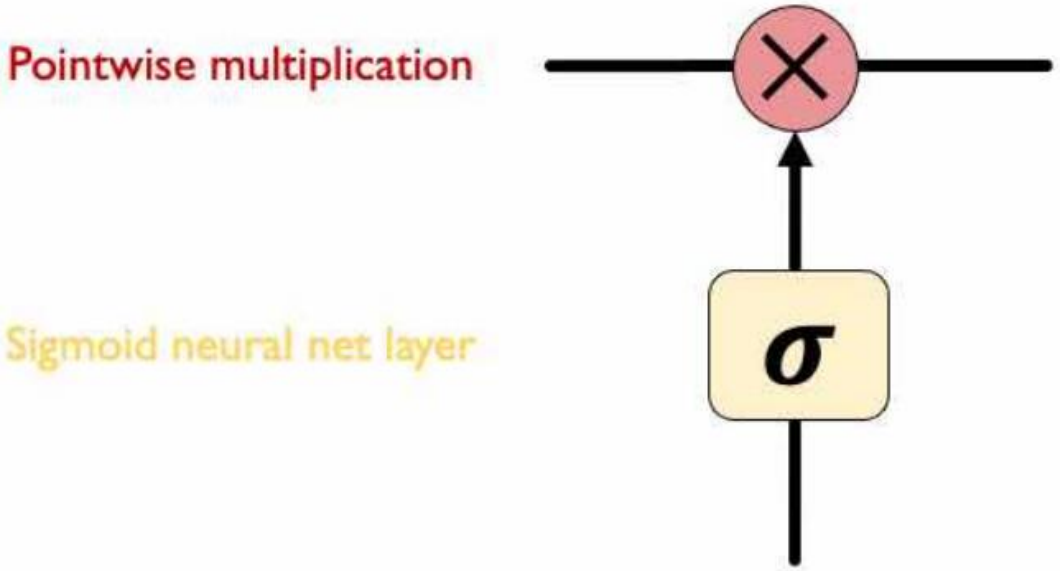
Initialize **biases** to zero

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

Trick #3: Gated Cells

Idea: use **gates** to selectively **add** or **remove** information within **each recurrent unit with**



gated cell

LSTM, GRU, etc.

Gates optionally let information through the cell

Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

RNN 실습하기

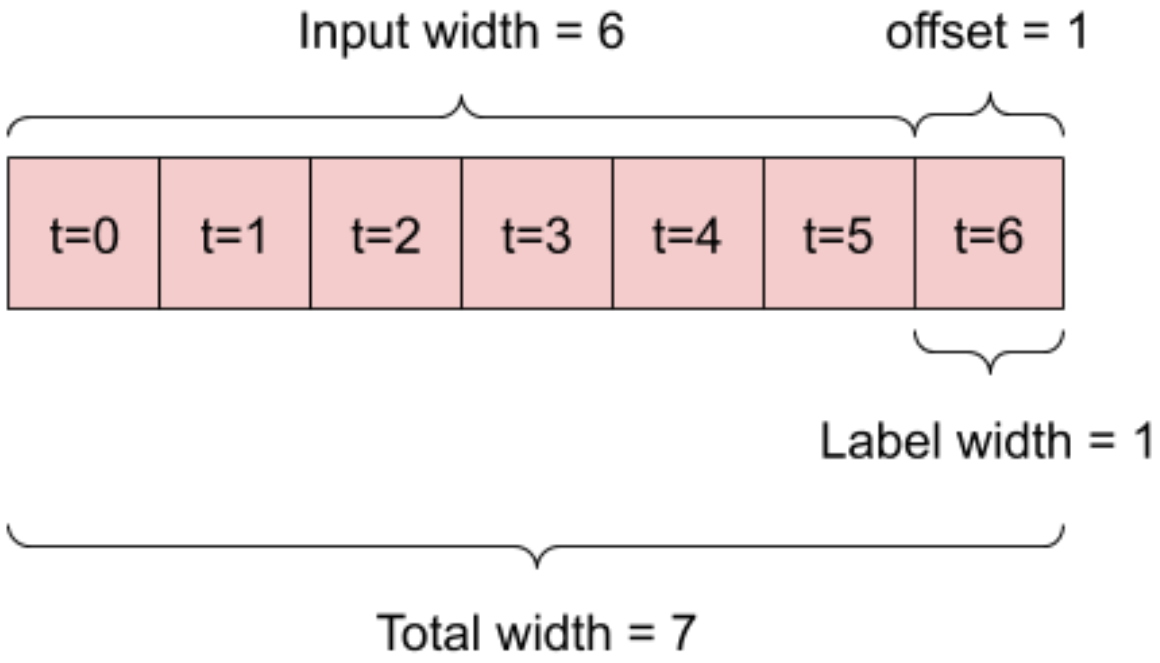
Let's Code!

Forecasting a Time Series

Data windowing

✓A model that makes a prediction one hour into the future, given six hours of history

```
w2 = WindowGenerator(input_width=6, label_width=1, shift=1, label_columns=['sinefunction'])
```



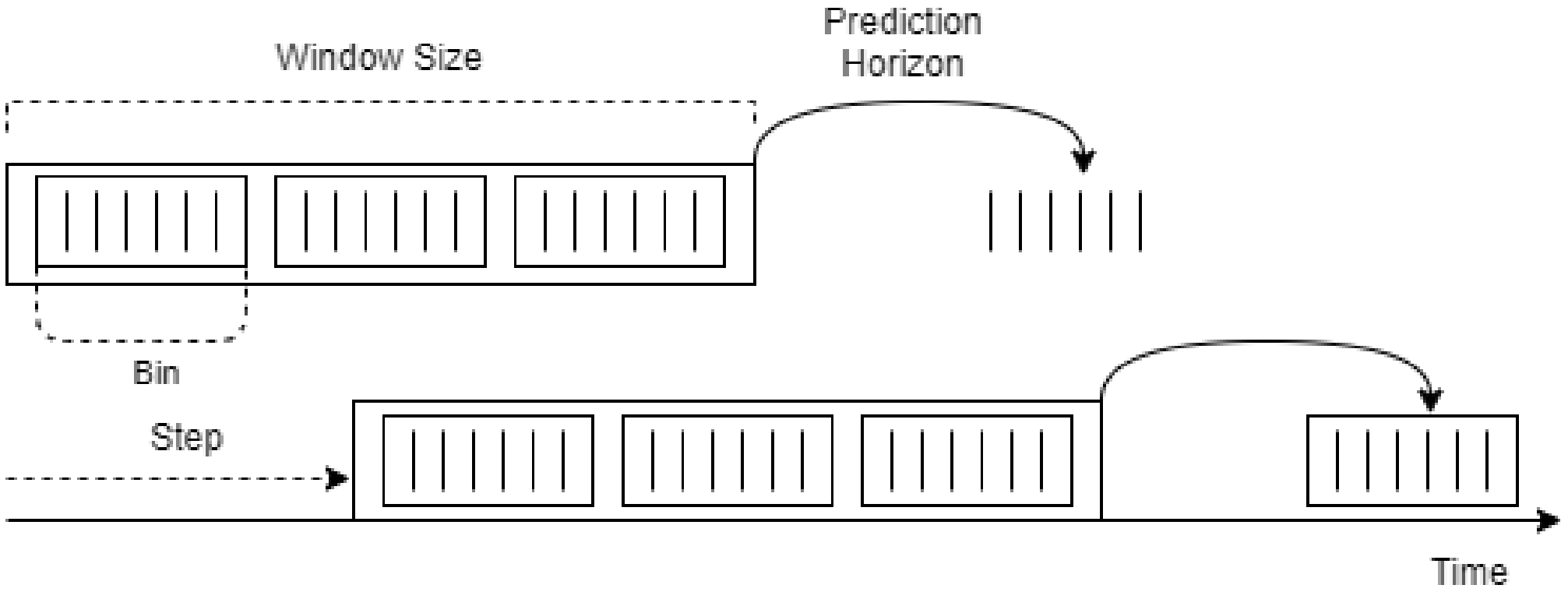
Total window size: 7
Input indices: [0 1 2 3 4 5]
Label indices: [6]
Label column name(s): ['sinefunction']

What is a time series problem?

- The problems can be broadly categorized into two categories:

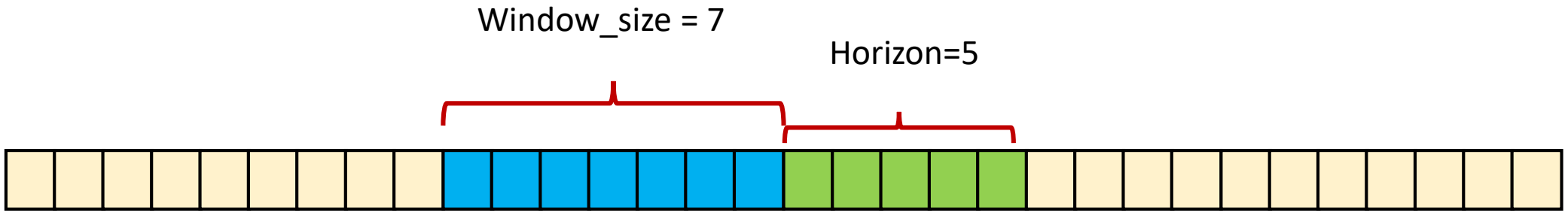
Problem Type	Examples	Output
Classification	Anomaly detection, time series identification (where did this time series come from?)	Discrete (a label)
Forecasting	Predicting stock market prices, forecasting future demand for a product, stocking inventory requirements	Continuous (a number)

- Two terms are really important in the type of forecasting model :
  - ✓**Window Size** :The number of timesteps we take to predict into the future
  - ✓**Horizon** : The number of timesteps ahead into the future we predict.

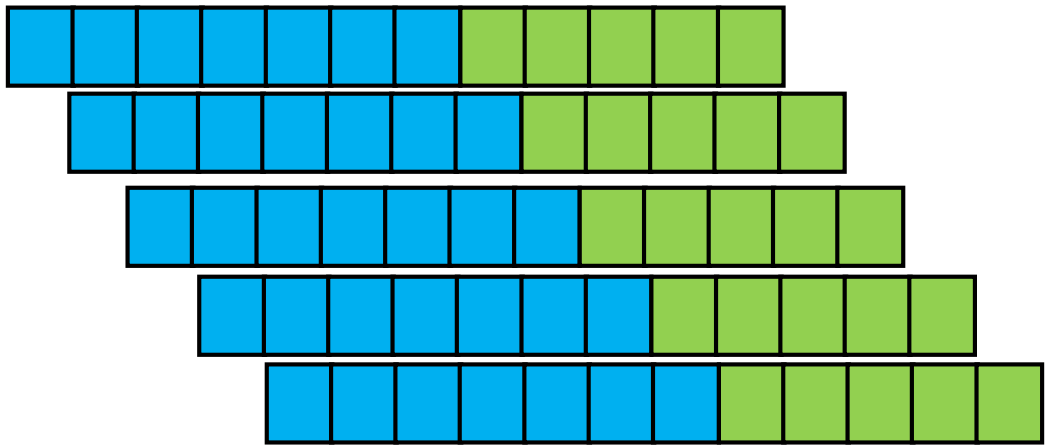


Many-to-One RNN Data Structure

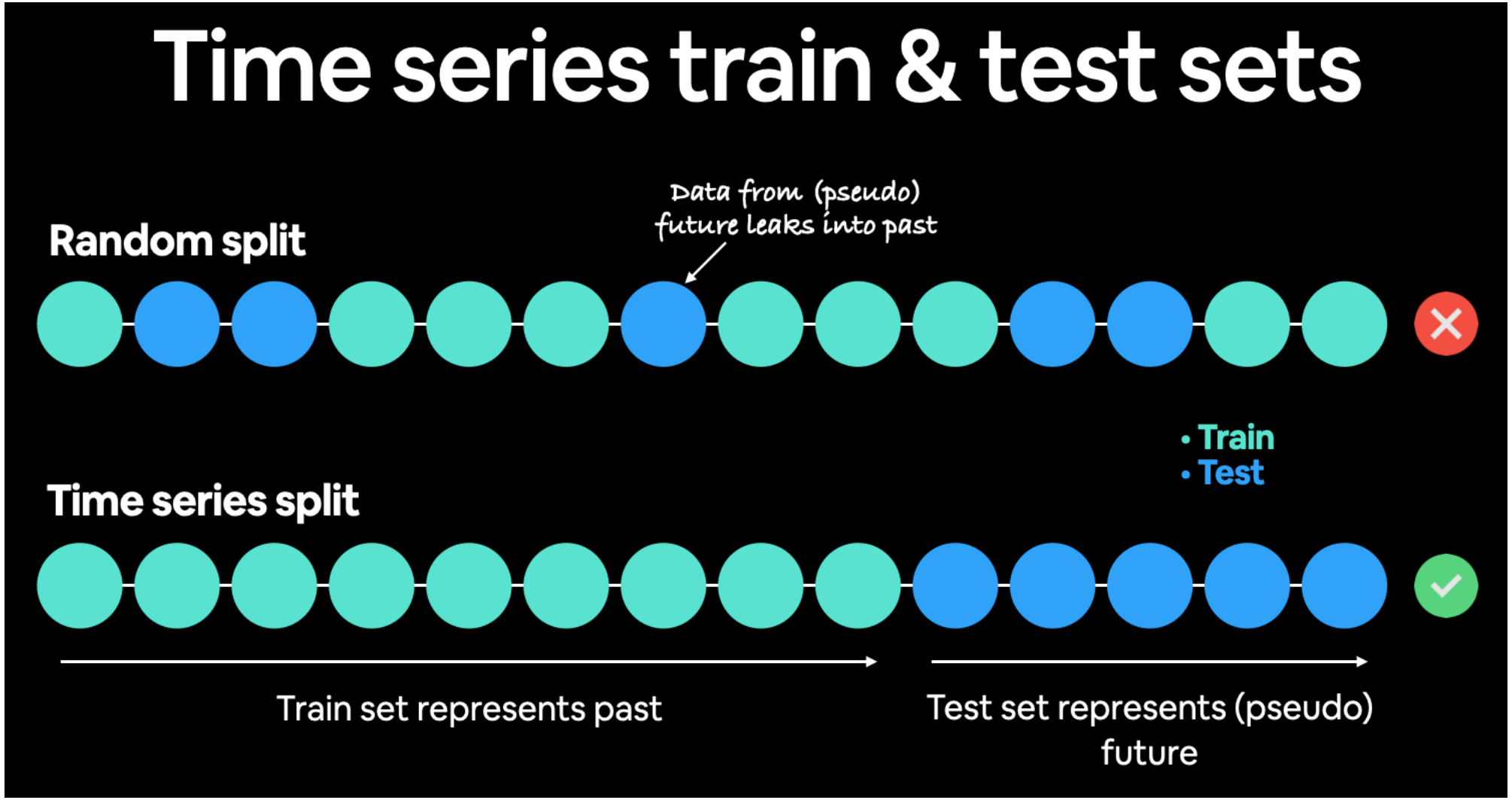
Step1: set the number of window_size, horizon



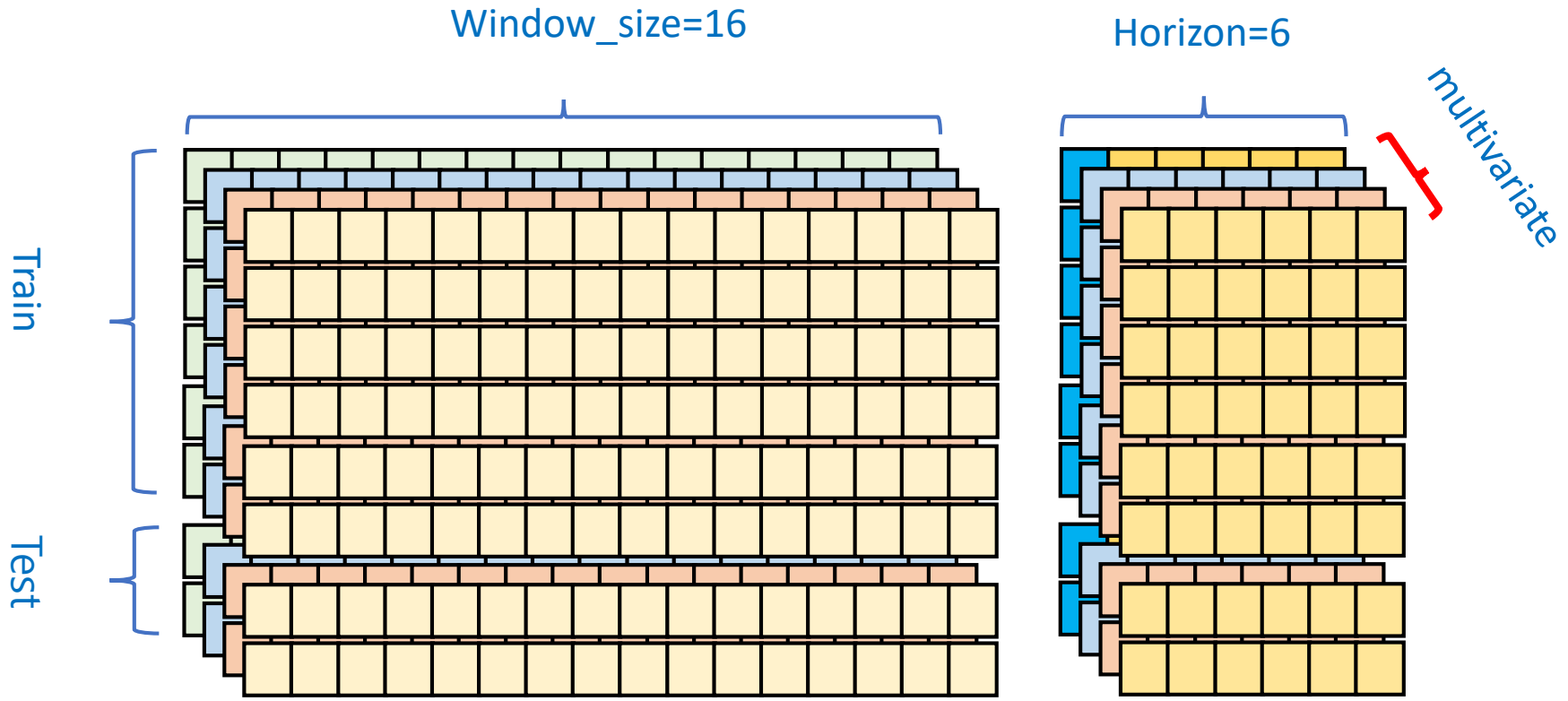
Step2: set the number of window_size, horizon



Create train and test sets the right way

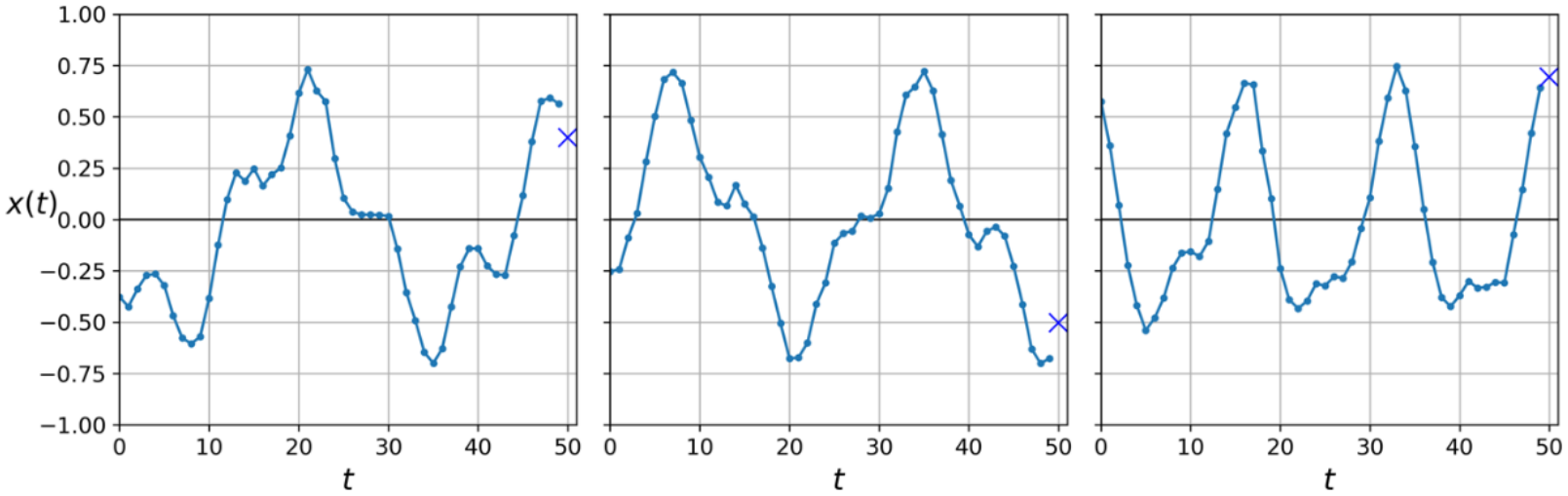


RNN Input-Output Data Structure



Forecasting a Time Series

- There is a single value per time step : *univariate time series*
  - ✓ A typical task is to predict future values, which is called *forecasting*.
- For example, figures shows 3 univariate time series
  - ✓ each of them 50 time steps long, and the goal here is to forecast the value at the next time step (represented by the X) for each of them.



Time series generated by the function

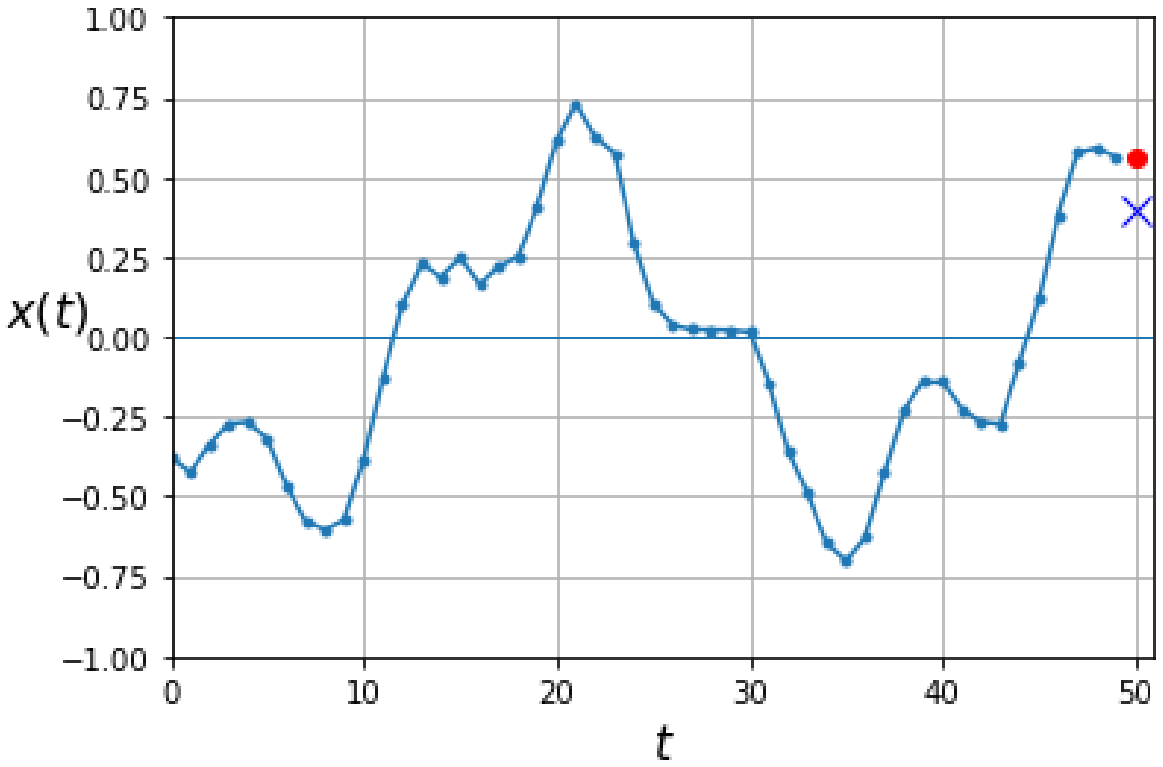
Generate the Dataset ¶

In [3]:
```
def generate_time_series(batch_size, n_steps):
    freq1, freq2, offsets1, offsets2 = np.random.rand(4, batch_size, 1)
    time = np.linspace(0, 1, n_steps)
    series = 0.5 * np.sin((time - offsets1) * (freq1 * 10 + 10))  #   wave 1
    series += 0.2 * np.sin((time - offsets2) * (freq2 * 20 + 20)) # + wave 2
    series += 0.1 * (np.random.rand(batch_size, n_steps) - 0.5)   # + noise
    return series[..., np.newaxis].astype(np.float32)
```

The function returns a NumPy array of shape [*batch size*, *time steps*, 1],

where each series is the sum of two sine waves of fixed amplitudes but random frequencies and phases, plus a bit of noise.

- The function returns a NumPy array of shape : [*batch size*, *time steps*, 1]
  - ✓ where each series is the sum of two sine waves of fixed amplitudes but random frequencies and phases, plus a bit of noise.



bx : for y_values at 51
ro : for y_prediction at 51

plot series

```
def plot_series(series, y=None, y_pred=None, x_label="$t$", y_label="$x(t)$"):
    plt.plot(series, ".-")
    if y is not None:
        plt.plot(n_steps, y, "rx", markersize=10)
    if y_pred is not None:
        plt.plot(n_steps, y_pred, "ro")
    plt.grid(True)
    if x_label:
        plt.xlabel(x_label, fontsize=16)
    if y_label:
        plt.ylabel(y_label, fontsize=16, rotation=0)
    plt.hlines(0, 0, 100, linewidth=1)
    plt.axis([0, n_steps + 1, -1, 1])

fig, axes = plt.subplots(nrows=1, ncols=3, sharey=True, figsize=(12, 4))
for col in range(3):
    plt.sca(axes[col])
    plot_series(X_valid[col, :, 0], y_valid[col, 0],
                y_label=("$x(t)$" if col==0 else None))

plt.show()
```

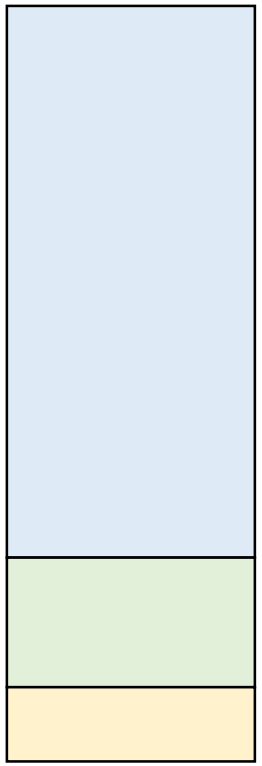
Training, Valid, Test Data

Now let's create a training set, a validation set, and a test set

X_train: 0 ~ 6999

X_valid: 7000~8999

X_test: 9000~9999



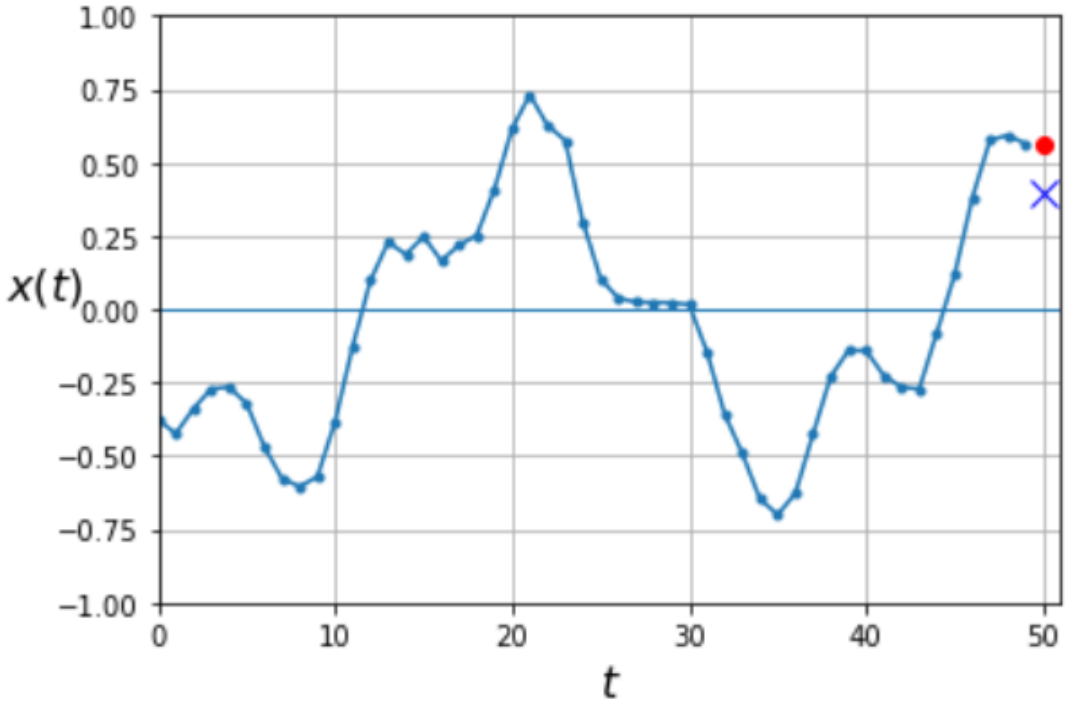
```
np.random.seed(42)

n_steps = 50

series = generate_time_series(10000, n_steps + 1)


X_train, y_train = series[:7000, :n_steps], series[:7000, -1]

X_valid, y_valid = series[7000:9000, :n_steps], series[7000:9000, -1]

X_test, y_test = series[9000:, :n_steps], series[9000:, -1]


X_train.shape, y_train.shape, X_valid.shape
```

```
((7000, 50, 1), (7000, 1), (2000, 50, 1))
```

model1: Computing Some Baselines

- Naive predictions (just predict the last observed value):
  - ✓ to predict the last value in each series
- In this case, it gives us a mean squared error of about 0.020211:

```
y_pred = X_valid[:, -1]
np.mean(keras.losses.mean_squared
_error(y_valid, y_pred))
```



model 2: Linear Predictions

- Another simple approach is to use a fully connected network.
  - ✓Since it expects a flat list of features for each input, we need to add a Flattern layer.

```
m2 = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[50, 1]),
    keras.layers.Dense(1)
])
```

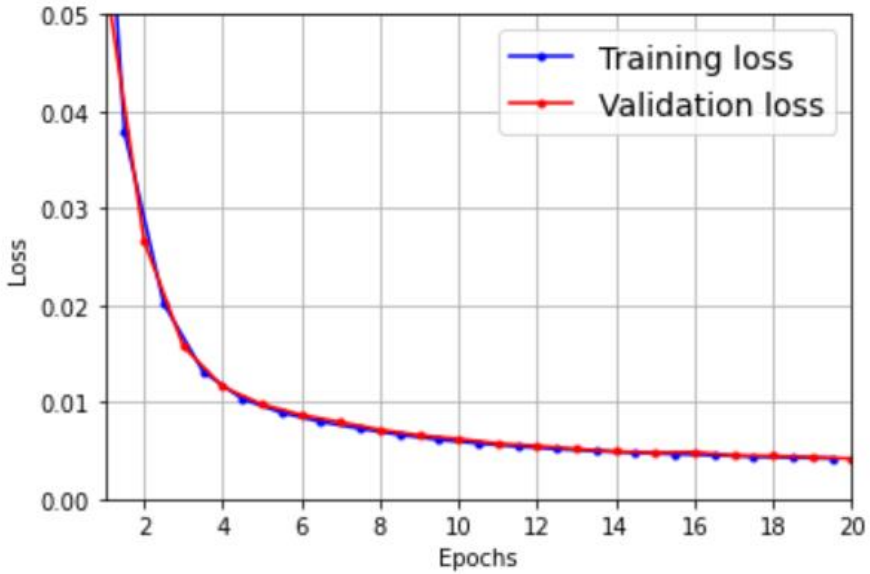
```
m2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
_________________________________________________________________
```

- Another simple approach is to use a fully connected network.
  - ✓Since it expects a flat list of features for each input, we need to add a Flattern layer.
  - ✓Let's just use a simple Linear Regression model so that each prediction will be a linear combination of the values in the time series:

```
m2 = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[50, 1]),
    keras.layers.Dense(1)
])

m2.compile(loss="mse", optimizer="adam")
history = m_lin.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
m2.evaluate(X_valid, y_valid)
```

```
Out[9]:  0.004168087150901556
```

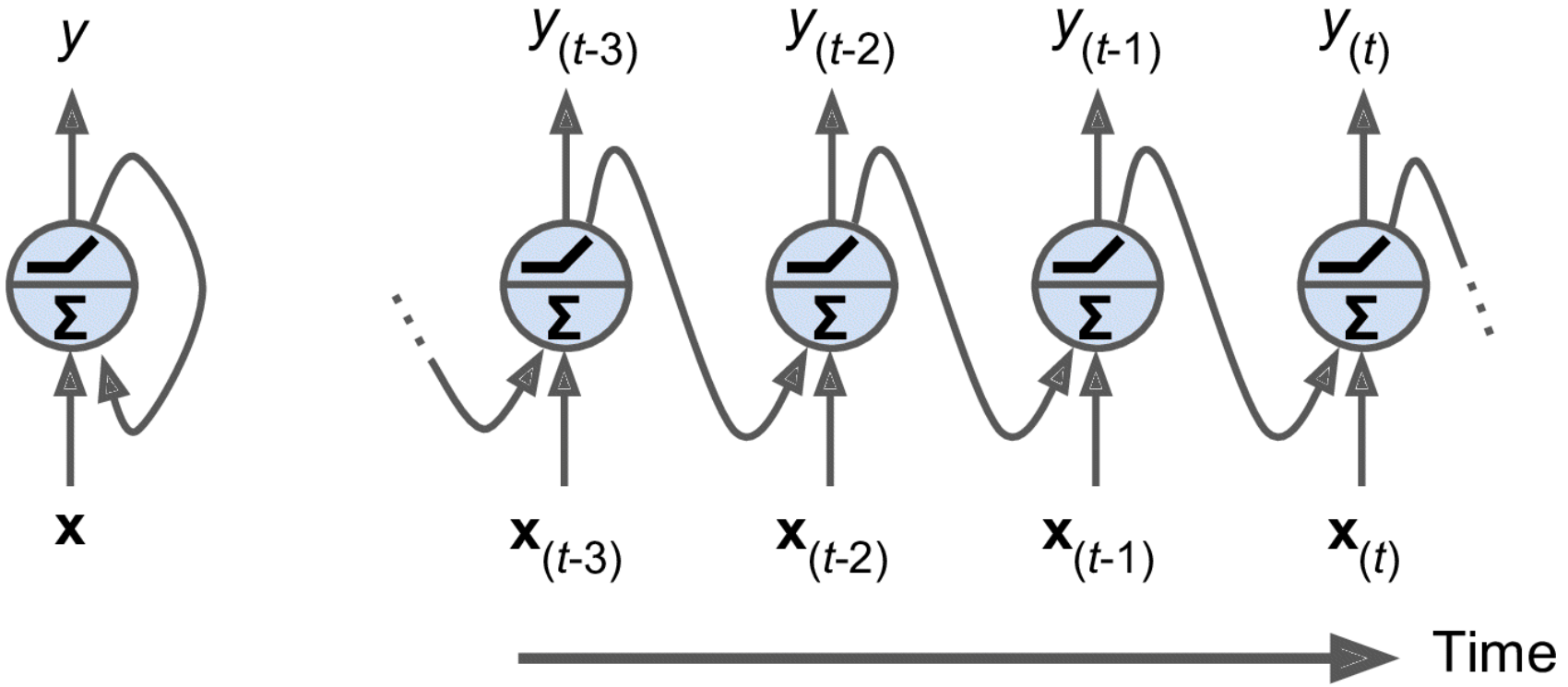


Learning Curves function and plot

```
def plot_learning_curves(loss, val_loss):
    plt.plot(np.arange(len(loss)) + 0.5, loss, "b.-", label="Training loss")
    plt.plot(np.arange(len(val_loss)) + 1, val_loss, "r.-", label="Validation loss")
    plt.gca().xaxis.set_major_locator(mpl.ticker.MaxNLocator(integer=True))
    plt.axis([1, 20, 0, 0.05])
    plt.legend(fontsize=14)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.grid(True)

plot_learning_curves(history.history["loss"], history.history["val_loss"])
plt.show()
```

❖ Simple RNN : don't need to specify the length of the RNN input sequence

✓ We do not need to specify the length of the input sequences (unlike in the previous model), since a recurrent neural network can process any number of time steps



03. Simple RNN

```
m3 = keras.models.Sequential([
    keras.layers.SimpleRNN(1, input_shape=[None, 1])
])

optimizer = keras.optimizers.Adam(lr=0.005)
m3.compile(loss="mse", optimizer=optimizer)
history = m3.fit(X_train, y_train, epochs=20,
                 validation_data=(X_valid, y_valid))
```

We do not need to specify the length of the input sequences (unlike in the previous model), since a recurrent neural network can process any number of time steps

SimpleRNN layer uses the hyperbolic tangent activation function

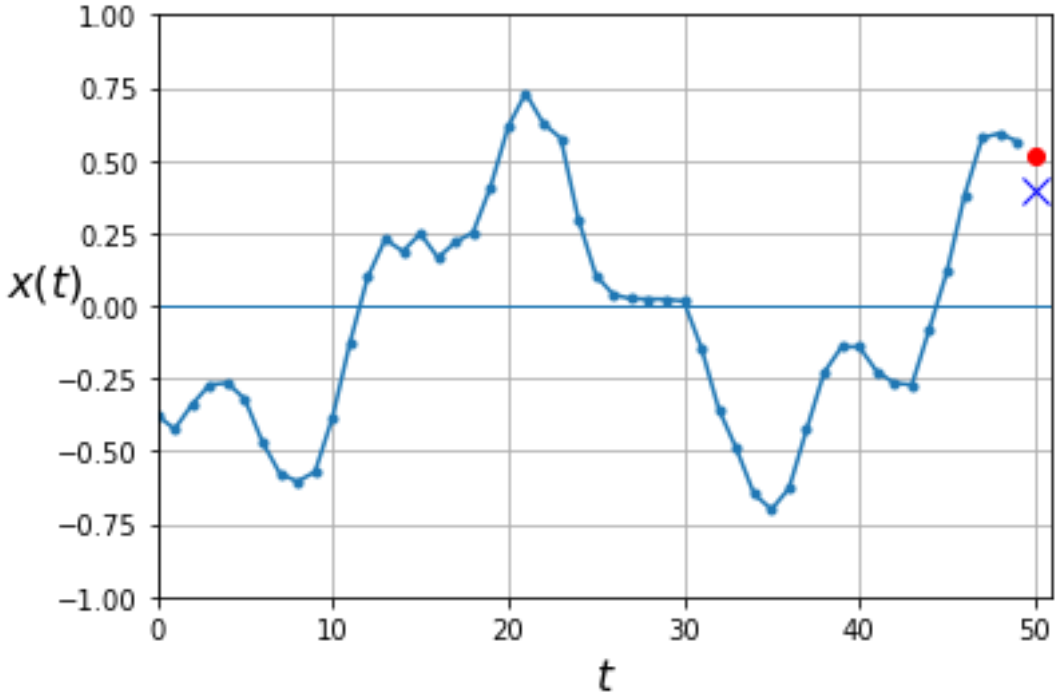
```
m3.summary()
```

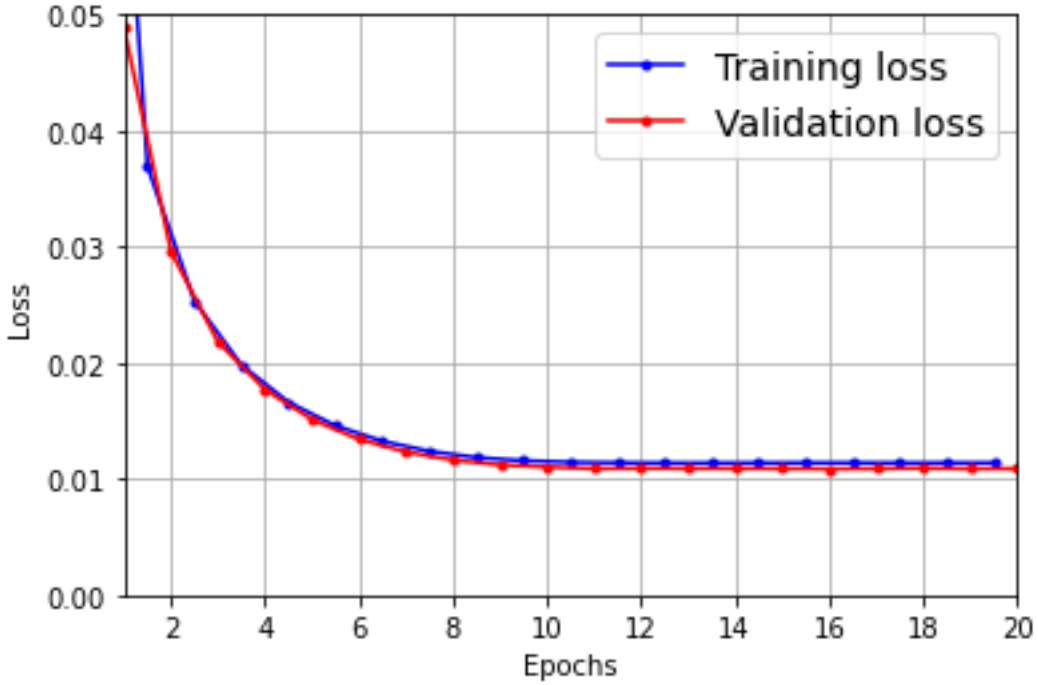
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

```
Out[31]: 0.010881561785936356
```

Evaluating a time series model

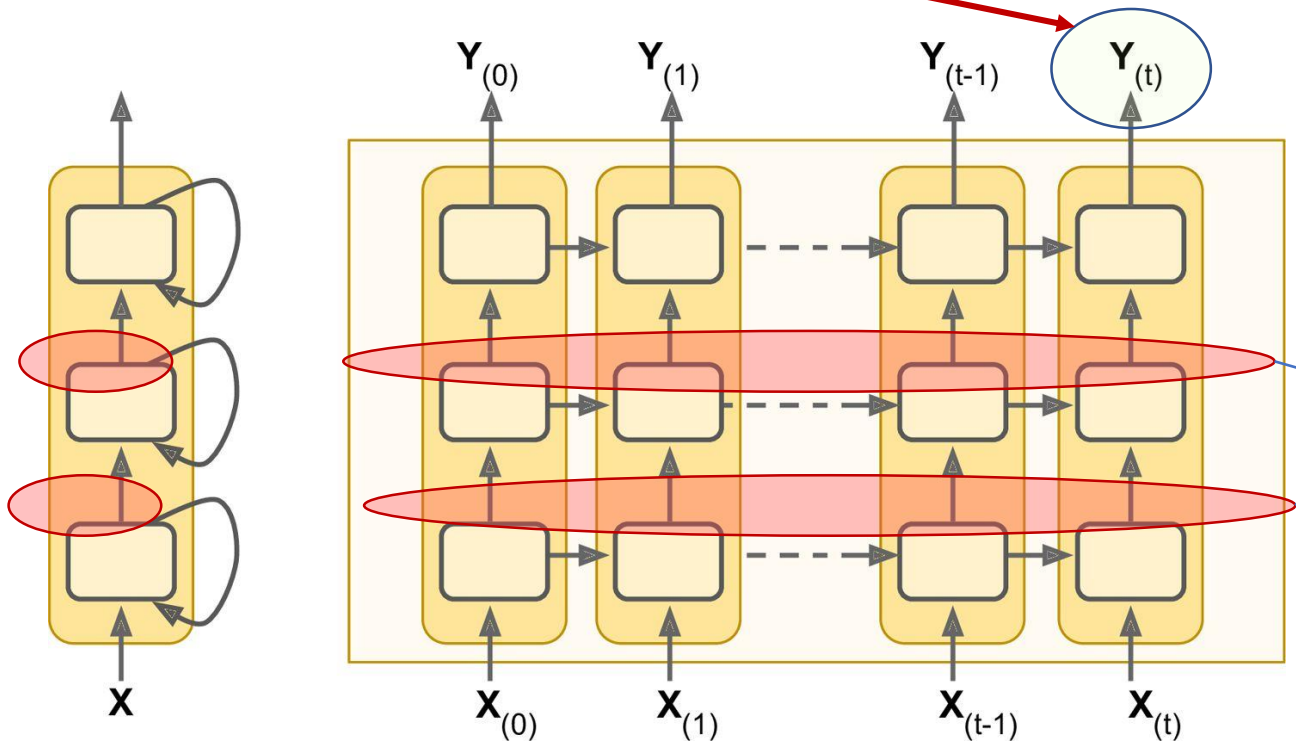
- These are metrics which can be used to compare time series values and forecasts that are on the same scale.
  - ✓ Scale-dependent errors
    - MAE (mean absolute error)
    - RMSE (root mean square error)
  - ✓ Percentage errors
    - MAPE (mean absolute percentage error)
    - sMAPE (scaled mean absolute percentage error)
  - ✓ Scaled errors
    - MASE (mean absolute scaled error)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x(t)
0
10
20
30
40
50
t


0.05
0.04
0.03
0.02
0.01
0.00
Loss
Training loss
Validation loss
2
4
6
8
10
12
14
16
18
20
Epochs

Deep RNN with return_sequences=True

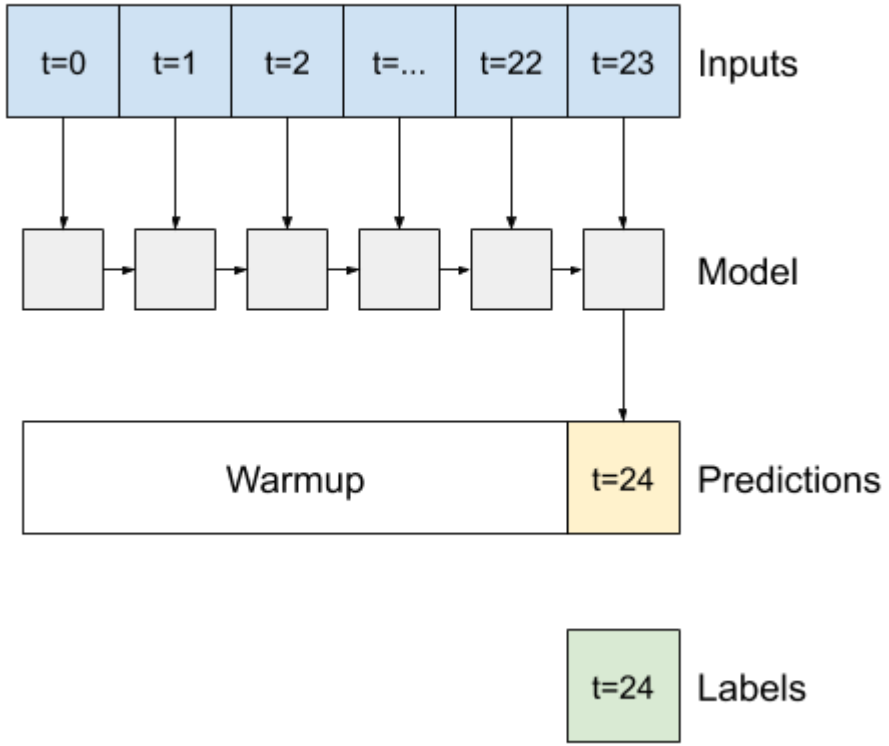
```
m4 = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
```



Make sure return_sequence=True for all recurrent layers (except the last one, if you only care about the last output)

❖ Return_sequences=False, the default

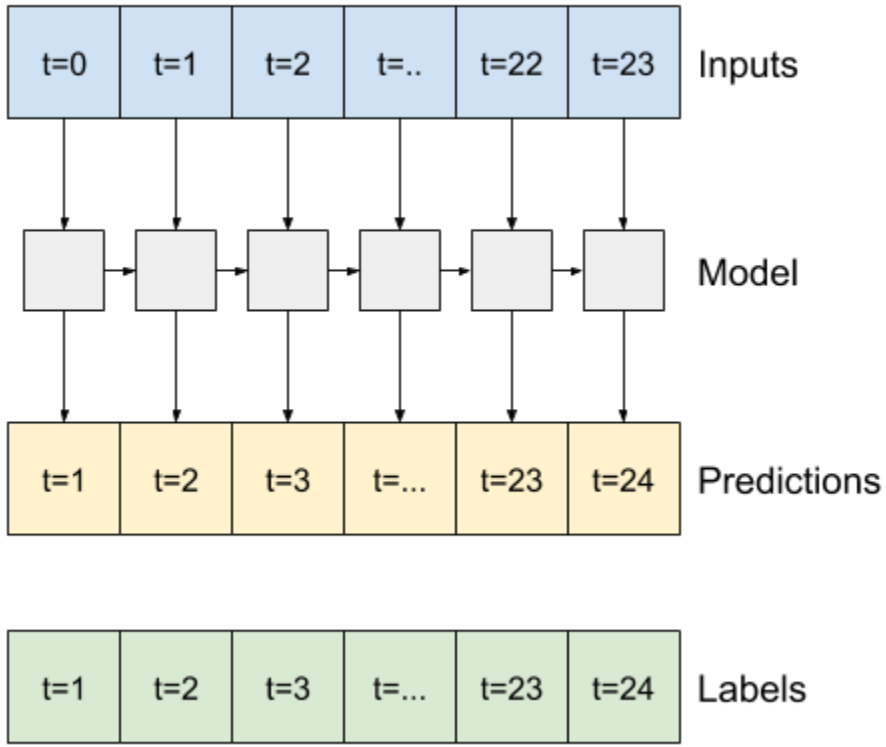
✓ the layer only returns the output of the final time step, giving the model time to warm up its internal state before making a single prediction:



return_sequences

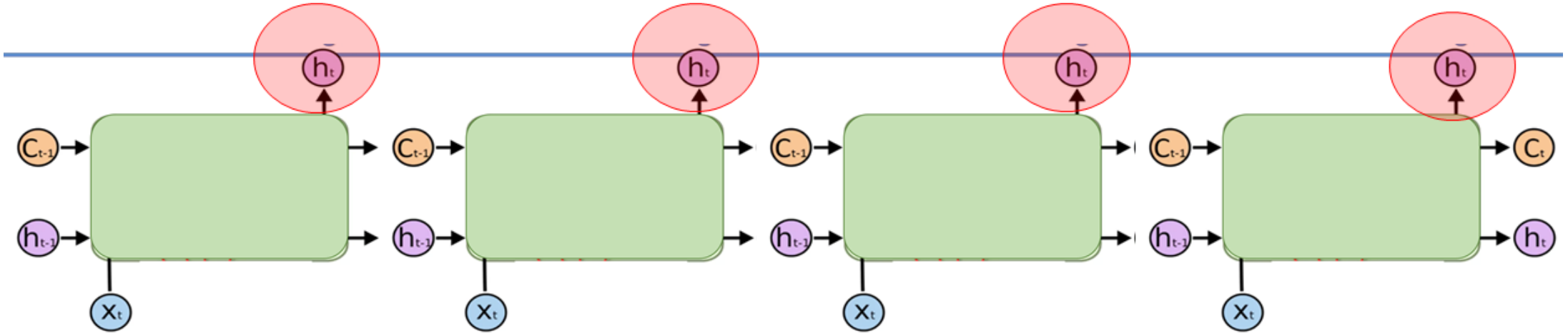
❖ Return_sequences=True

- ✓the layer returns an output for each input. This is useful for:Stacking RNN layers.
- ✓Training a model on multiple time steps simultaneously



return_sequences=True

All Hidden States (Hidden State of ALL the time steps)



Deep RNN return_sequences = True

```
m4.evaluate(X_valid, y_valid)
```

```
63/63 [==============================] - 1s 15ms/step - loss: 0.0029

0.00291056395508348 94
```

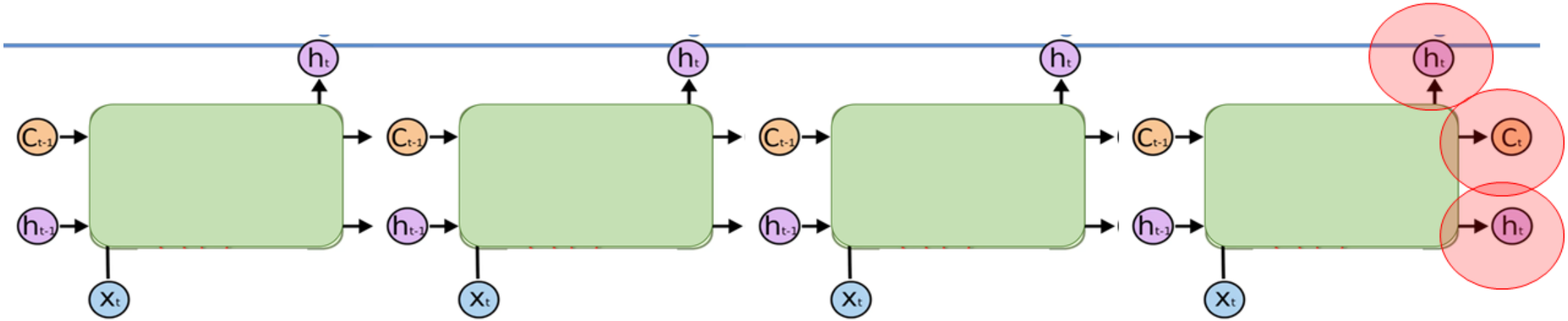
```
m4 = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
```

LSTM has 3 important parameters

- units: Positive integer
  - ✓dimensionality of the output space
- return_sequences: Boolean
  - ✓whether to return the last output. in the output sequence, or the full sequence.
    - Default: False.
- return_state: Boolean
  - ✓whether to return the last state in addition to the output.
  - ✓Default: False.

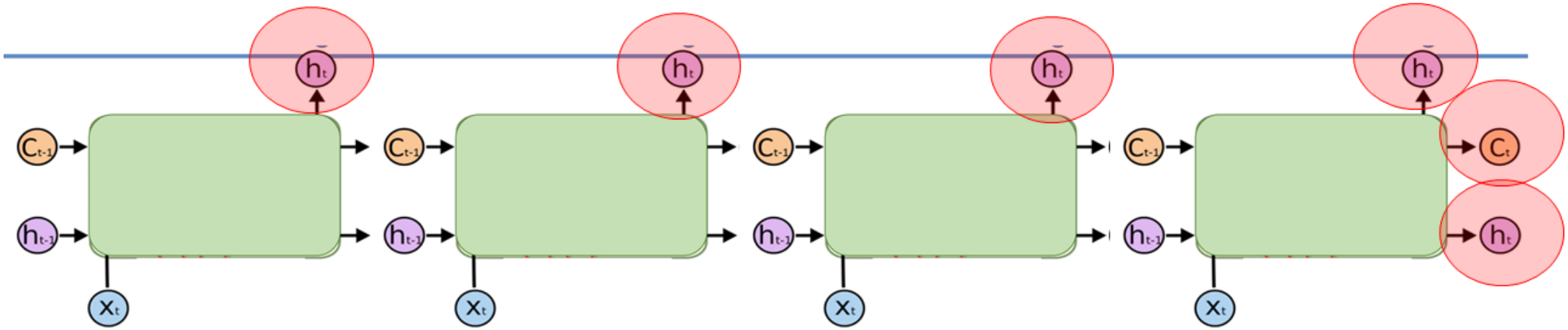
return_sequences and return_state

- return_state=True :
  - Last Hidden State+ Last Hidden State (again!) + Last Cell State (Cell State of the last time step)

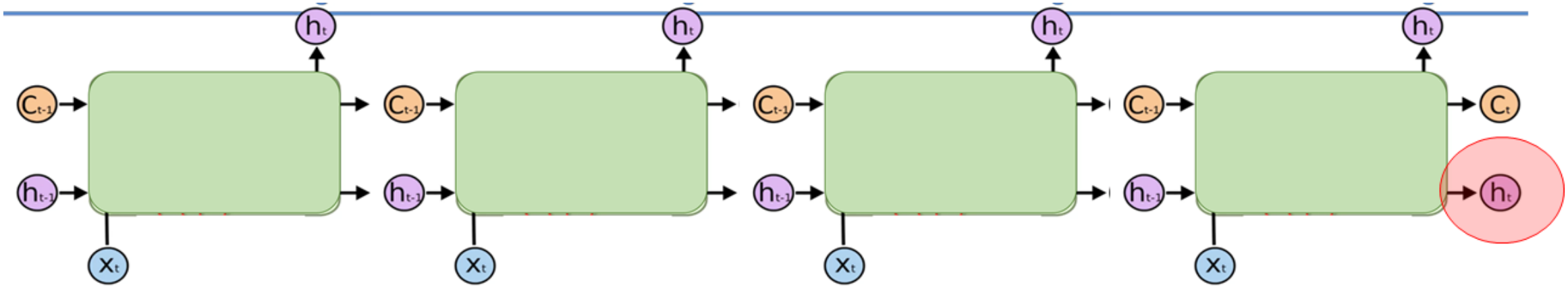


return_sequences and return_state

- return_sequences=True + return_state=True:
  - All Hidden States (Hidden State of ALL the time steps) + Last Hidden State + Last Cell State (Cell State of the last time step)



① Default: Last Hidden State (Hidden State of the last time step)



05. Deep RNN with only single output (unit)

- As a example, Use Dense layer at the last layer
  - ✓ it might be preferable to replace the output layer with a Dense layer
- remove return_sequences = True
  - ✓ it must have a single unit because we want to forecast a univariate time series, and this means we must have a single output value per time step.
  - ✓ However, having a single unit means that the hidden state is just a single number.

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None,
1]),
    keras.layers.SimpleRNN(20),   return_sequences = False
    keras.layers.Dense(1)
])
```

05. Deep RNN with only single output (unit)

```
m5 = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])
```

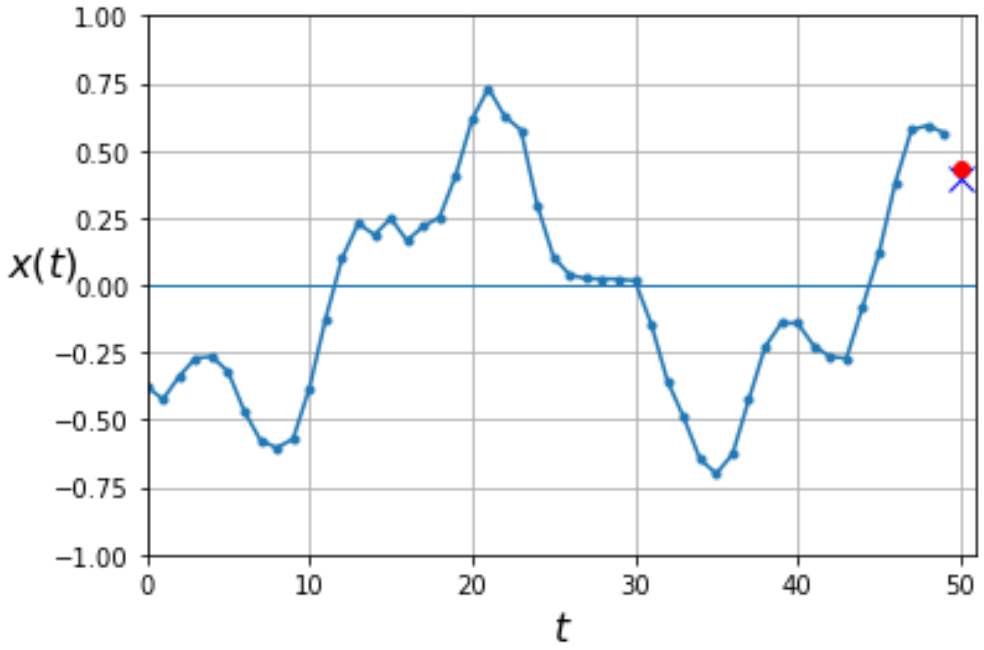
```
m5.summary()
```

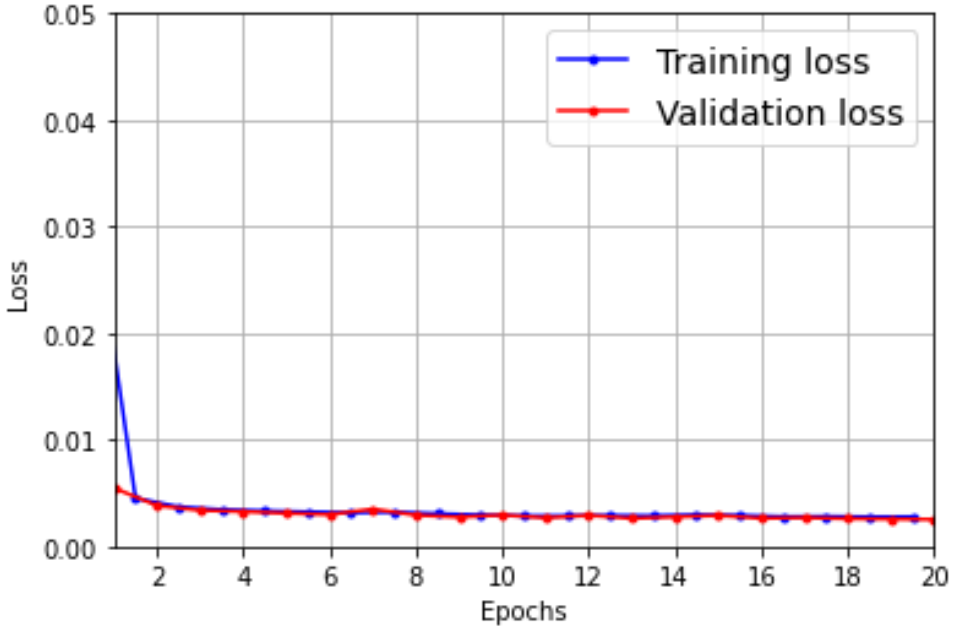
```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_7 (SimpleRNN)     (None, None, 20)          440
_________________________________________________________________
simple_rnn_8 (SimpleRNN)     (None, 20)                820
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 21
=================================================================
Total params: 1,281
Trainable params: 1,281
Non-trainable params: 0
_________________________________________________________________
```

05. Deep RNN with only single output (unit)

- If you train this model, you will see that it converges faster and performs just as well. Plus, you could change the output activation function if you wanted.

```
Out[45]: 0.0025271244812756777
```

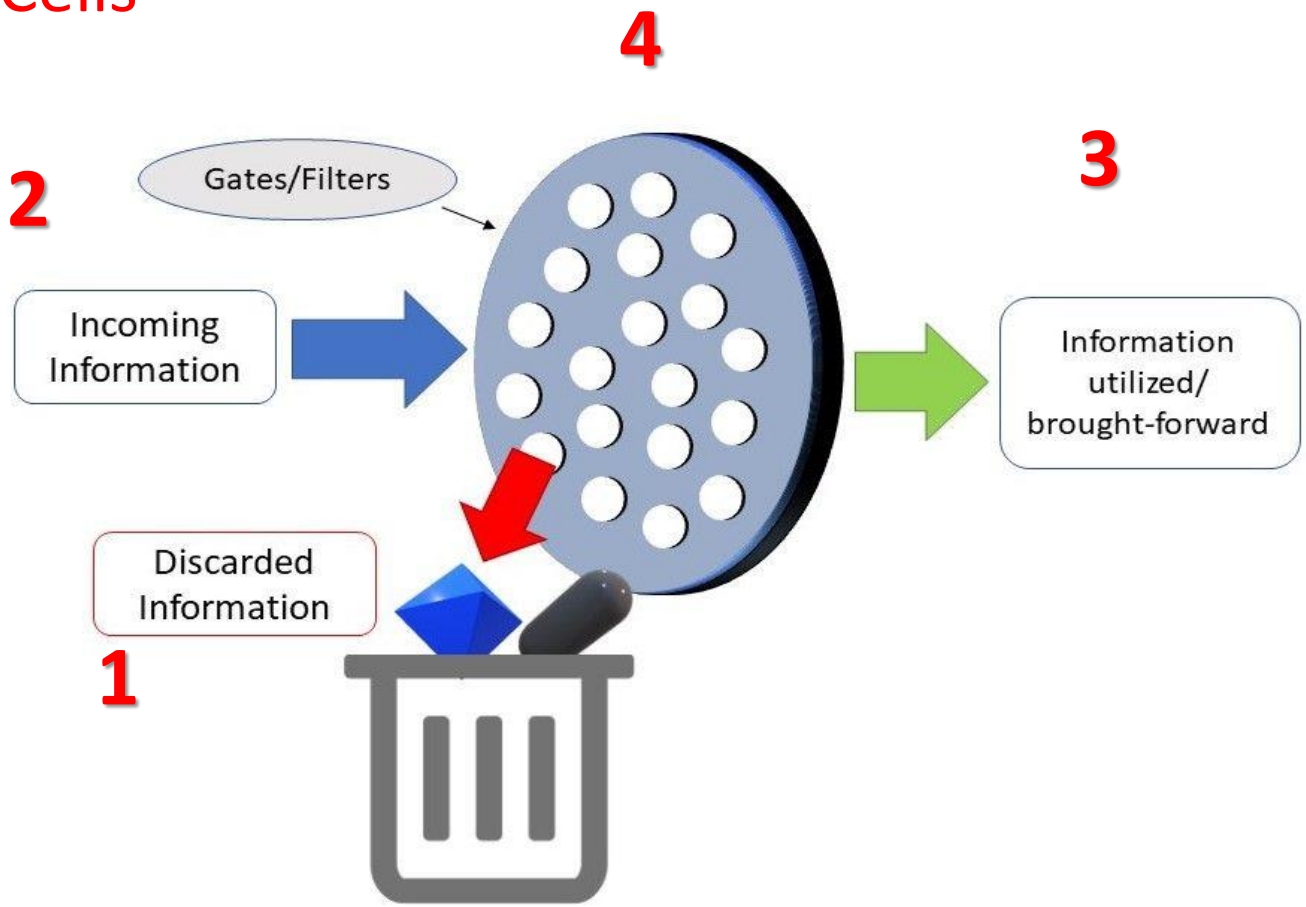




LSTM 이해하기

Gates control the flow of information to/from the memory

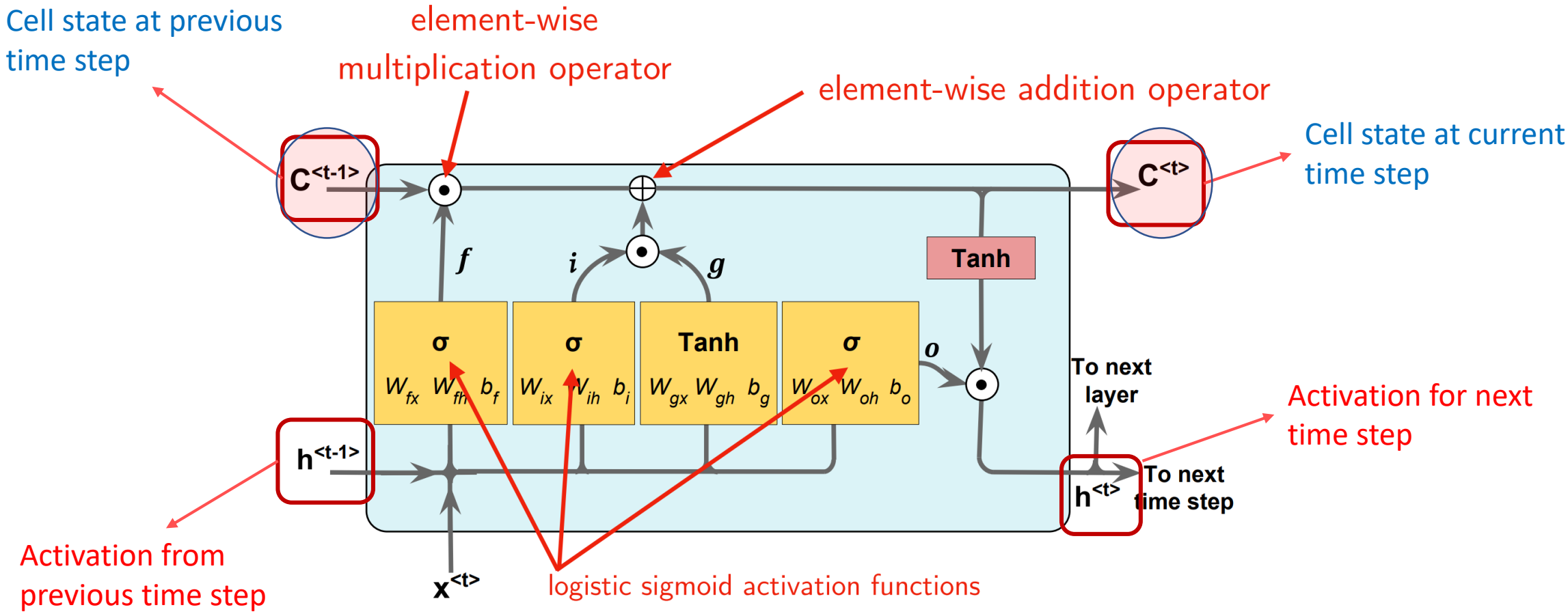
Trick #3: Gated Cells



Long Short Term Memory : Key Concepts

1. Maintain a **cell state**
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with partially **uninterrupted gradient flow**

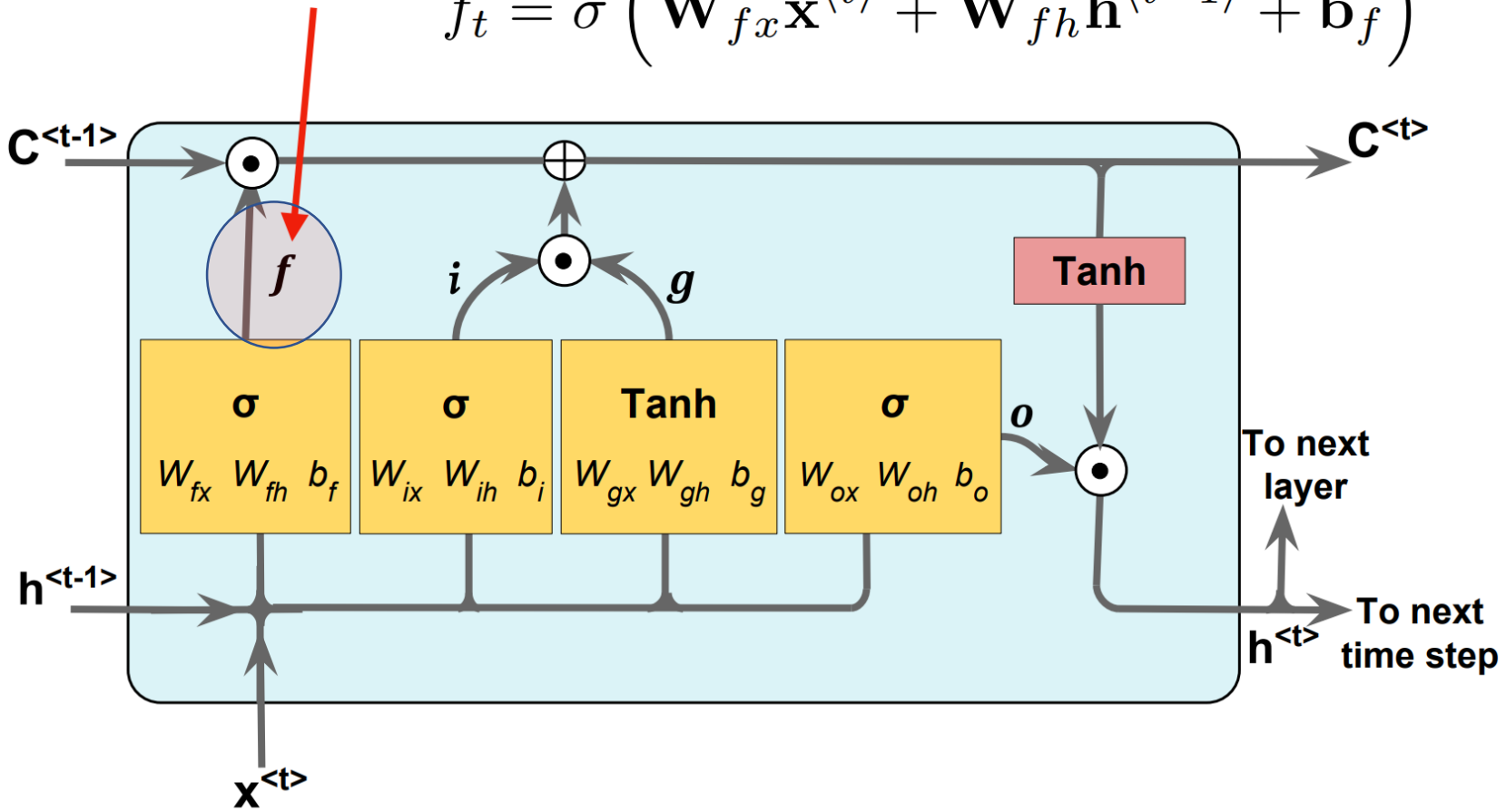
Long-short term memory (LSTM) : Cell state



Long-short term memory (LSTM) : Cell

"Forget Gate": controls which information is remembered, and which is forgotten; can reset the cell state

$$f_t = \sigma\left(\mathbf{W}_{fx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{fh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_f\right)$$

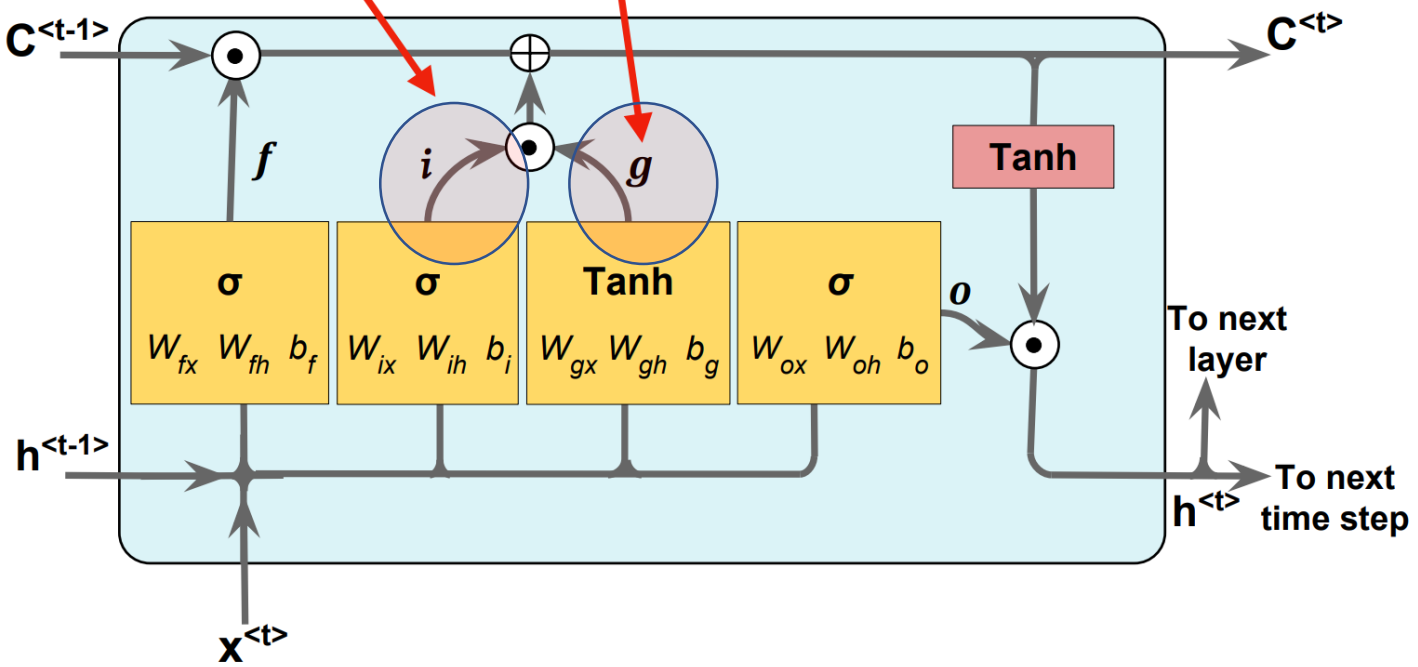


Long-short term memory (LSTM) : Cell

"Input Gate": $\mathbf{i}_t = \sigma\left(\mathbf{W}_{ix}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{ih}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_i\right)$

"Input Node":

$$\mathbf{g}_t = \tanh\left(\mathbf{W}_{gx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{gh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_g\right)$$



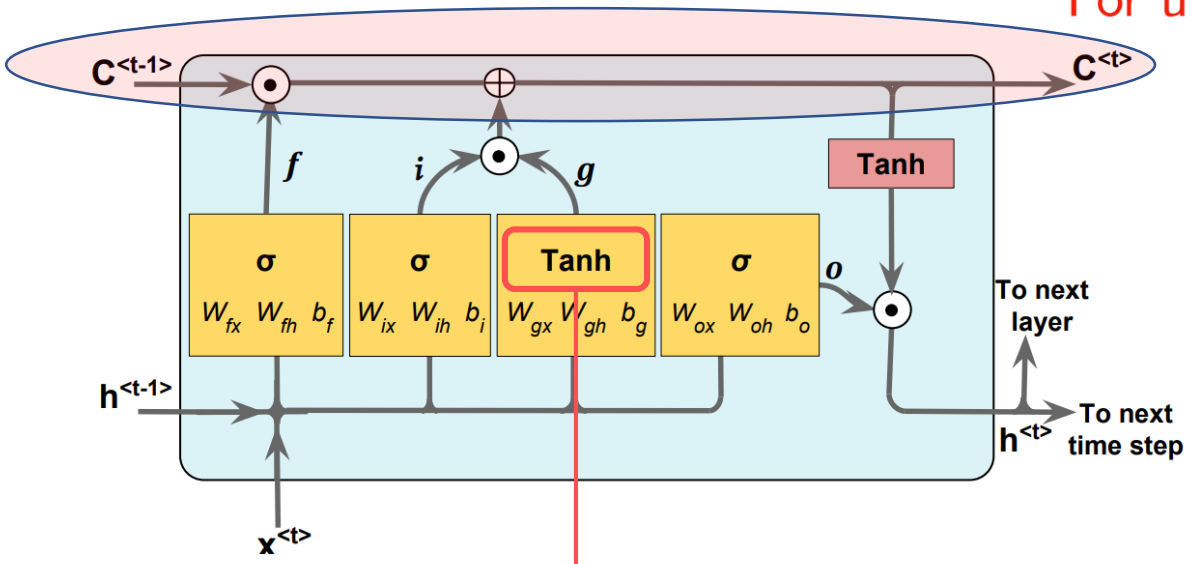
LSTM : Cell state

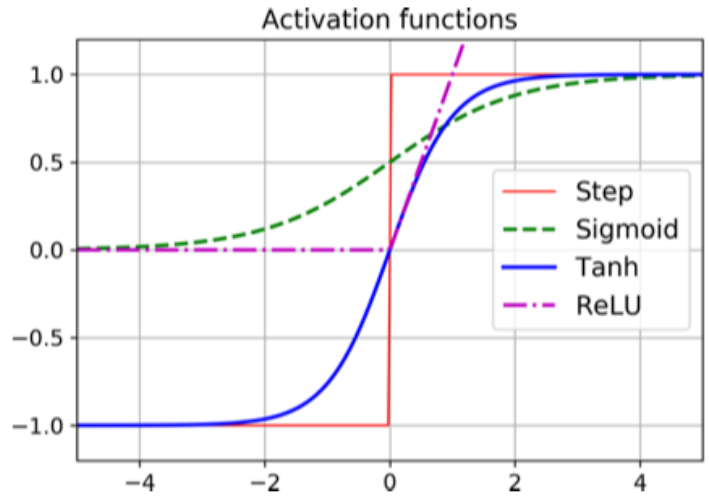
Brief summary of the gates so far ...

Forget Gate Input Node Input Gate

$$C^{\langle t \rangle} = \left(C^{\langle t-1 \rangle} \odot f_t \right) \oplus \left(i_t \odot g_t \right)$$

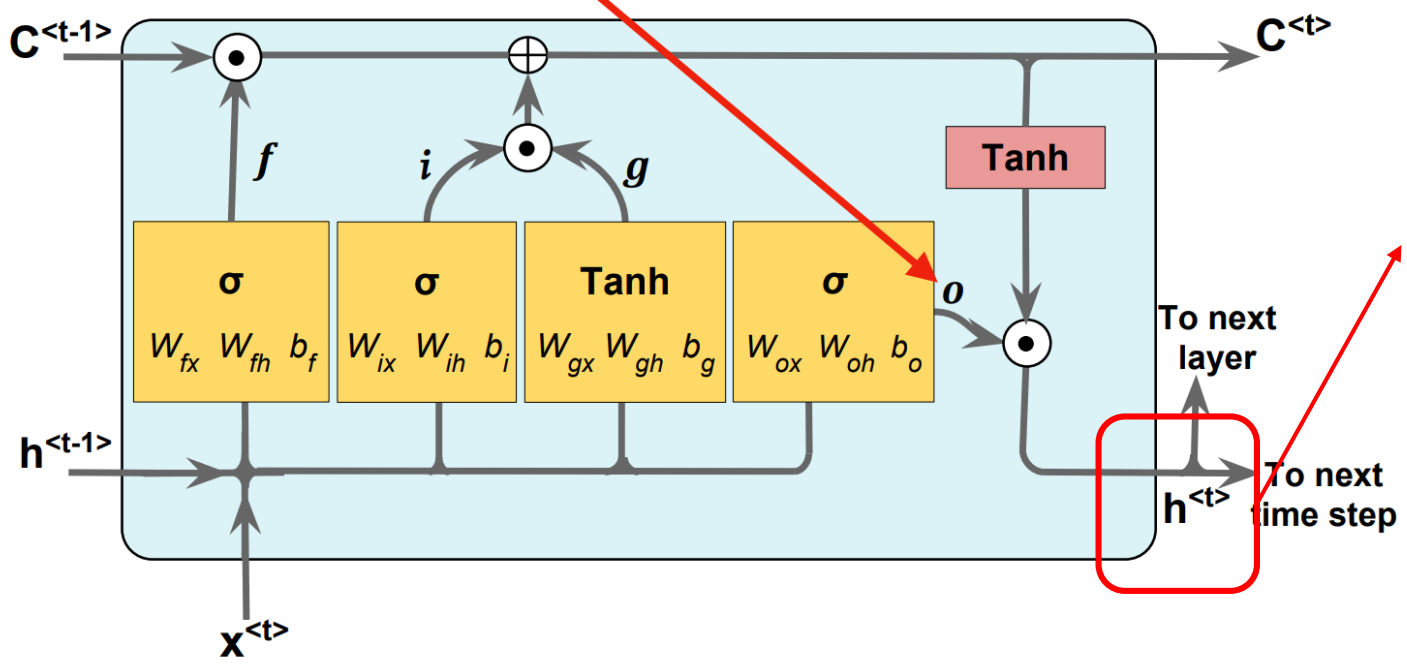
For updating the cell state





Output gate for updating the values of hidden units:

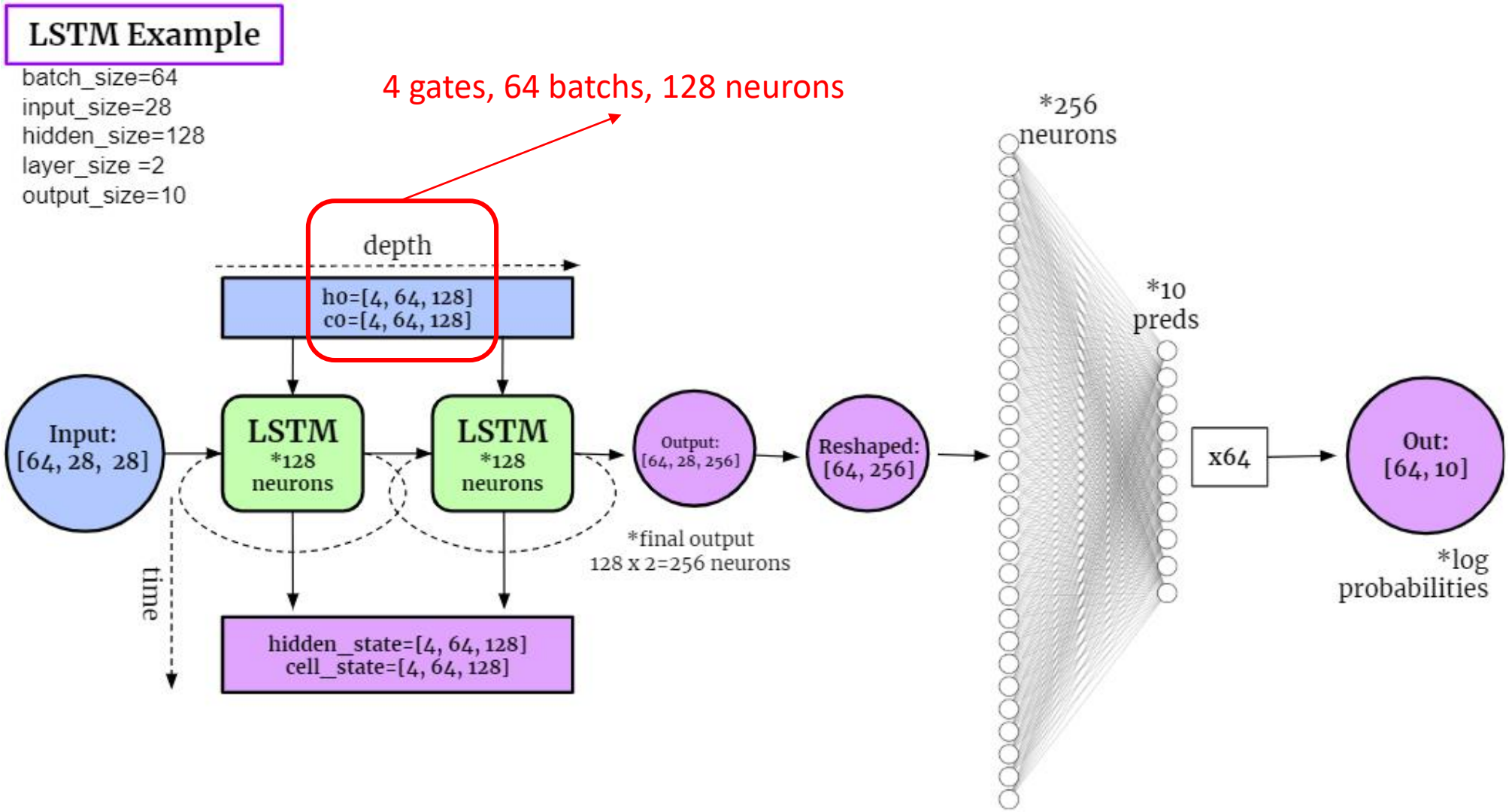
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{ox}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{oh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_o\right)$$



$$\mathbf{h}^{\langle t\rangle} = \mathbf{o}_t \odot \tanh\left(\mathbf{C}^{\langle t\rangle}\right)$$

A Visual Guide to Recurrent Layers in Keras

LSTM Example : Parameters



Lab LSTM Example : many-to-one

```
from keras.models import Model
from keras.layers import Input, Dense, LSTM
import numpy as np

x = np.array([[[1.], [2.], [3.], [4.], [5.]]])
y = np.array([[6.]])

xInput = Input(batch_shape=(None, 5, 1))
xLstm = LSTM(3)(xInput)
xOutput = Dense(1)(xLstm)

model = Model(xInput, xOutput)
model.compile(loss='mean_squared_error', optimizer='adam')
print(model.summary())

model.fit(x, y, epochs=50, batch_size=1, verbose=0)
model.predict(x, batch_size=1)
```

time series input

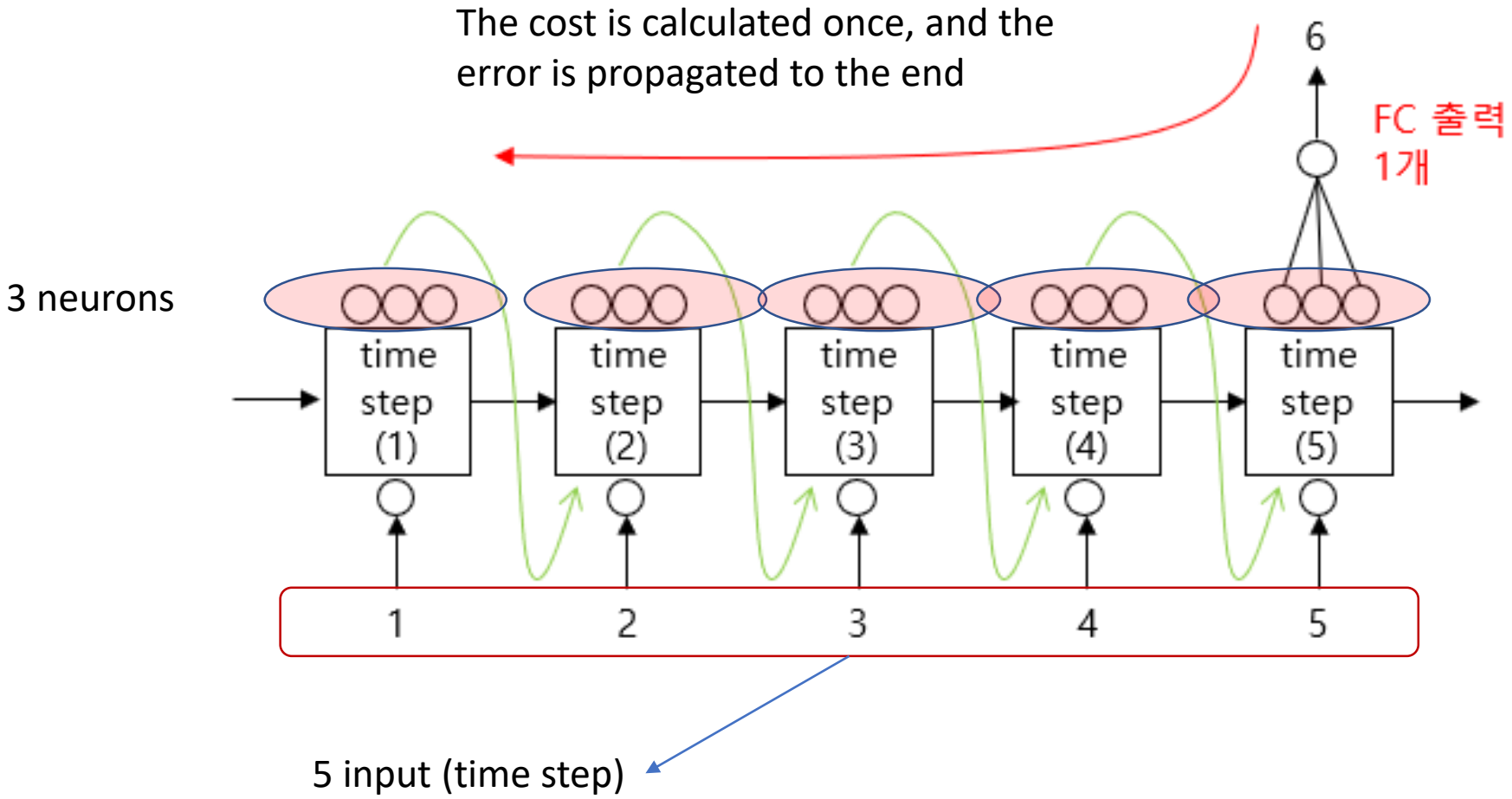
Inputs =[batch, time_step, features]

neurons in LSTM layers

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 5, 1)]            0

lstm_6 (LSTM)                (None, 3)                 60

dense_6 (Dense)              (None, 1)                 4

=================================================================
Total params: 64
Trainable params: 64
Non-trainable params: 0
```

Unfolded LSTM : Many to One

return_sequences=False



LSTM many-to-many with TimeDistributed Layer

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
x = np.array([[[1.], [2.], [3.], [4.], [5.]]])
y = np.array([[[2.], [3.], [4.], [5.], [6.]]])
```

```
model2 = keras.models.Sequential([
    keras.layers.LSTM(3, return_sequences=True, input_shape=[5, 1]),
    keras.layers.TimeDistributed(keras.layers.Dense(1) )                ])
model2.compile(loss='mean_squared_error', optimizer='adam')
model2.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_18 (LSTM)               (None, 5, 3)              60

time_distributed_8 (TimeDis  (None, 5, 1)              4
tributed)

=================================================================
Total params: 64
Trainable params: 64
Non-trainable params: 0
_________________________________________________________________
```

LSTM Output is 5x3(neurons)

Final Output is 5x1

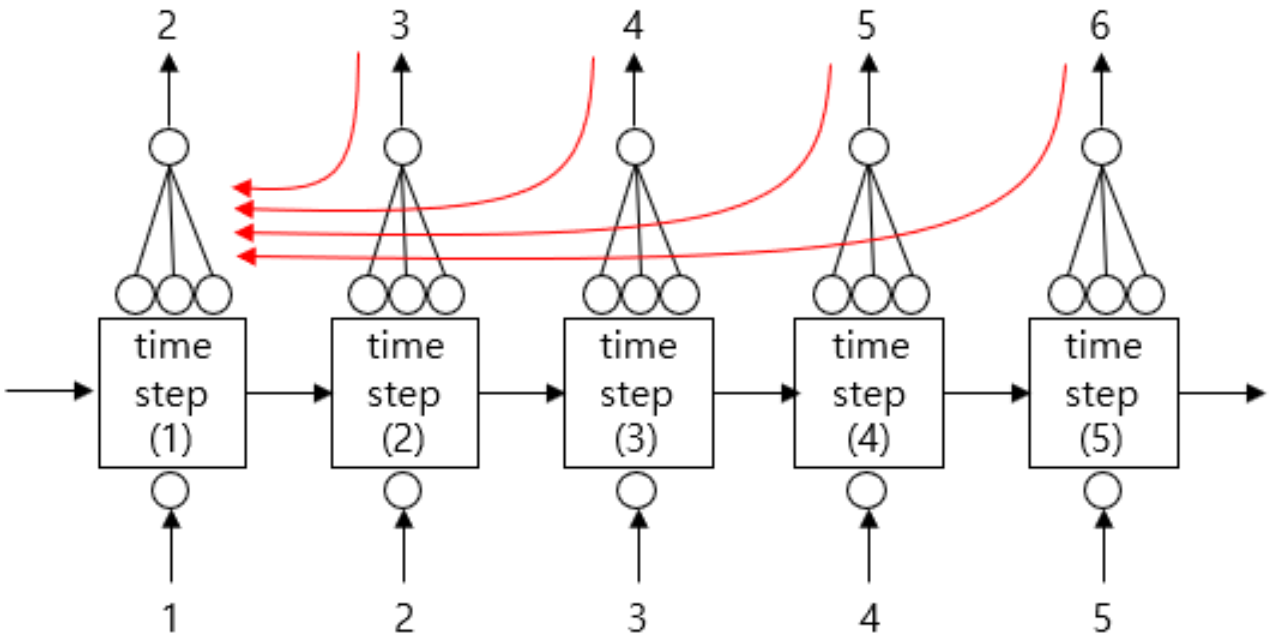
with keras.layers.TimeDistributed()

return_sequences=True

:means to use all the output of the intermediate step of the LSTM

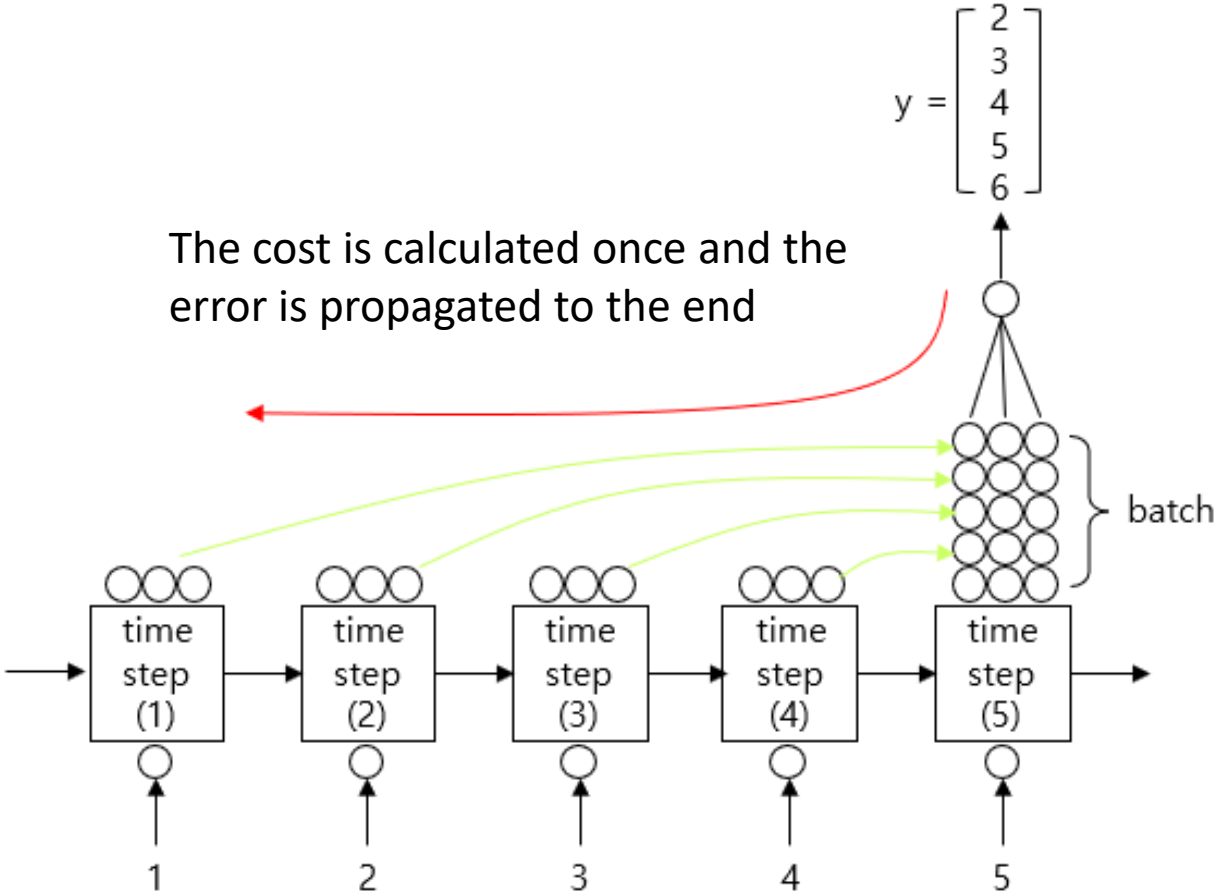
In each step, the cost is calculated and the error is propagated at each point.

(TimeDistributed의 의미)

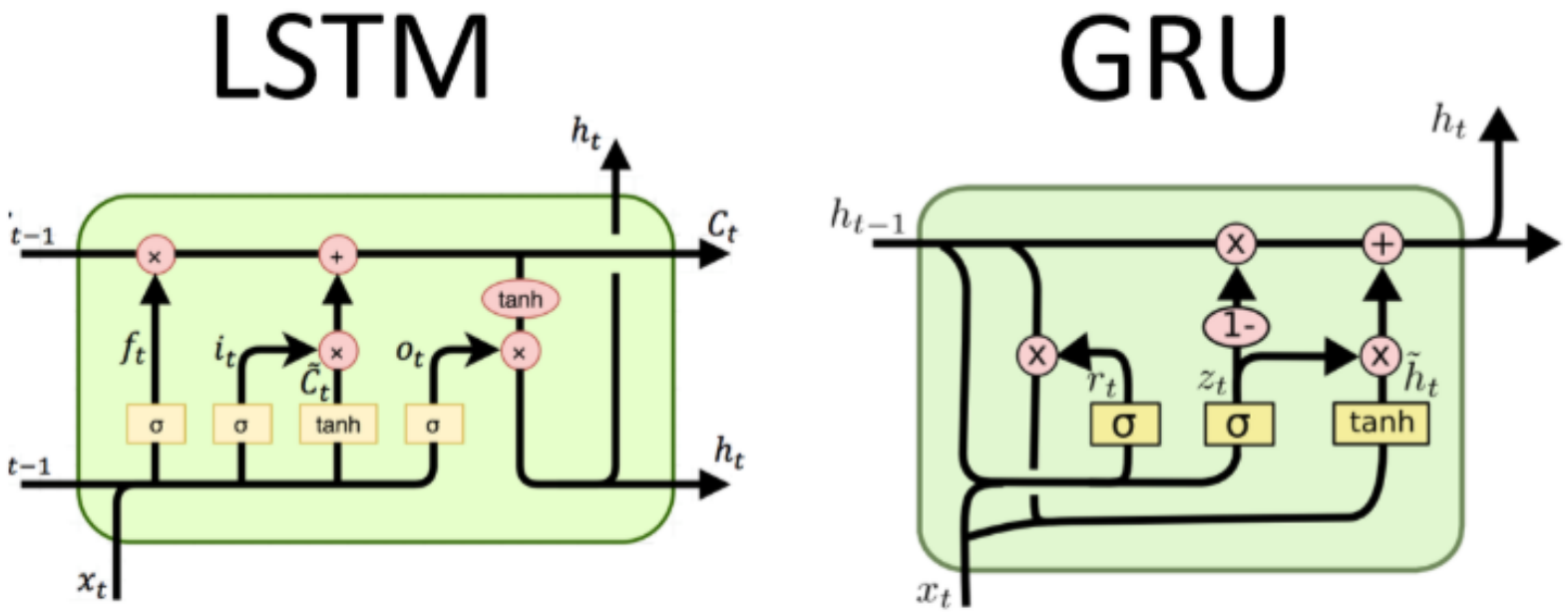


TimeDistributed() :
to calculate the cost for each step and propagate the error to sub-steps to update each weight

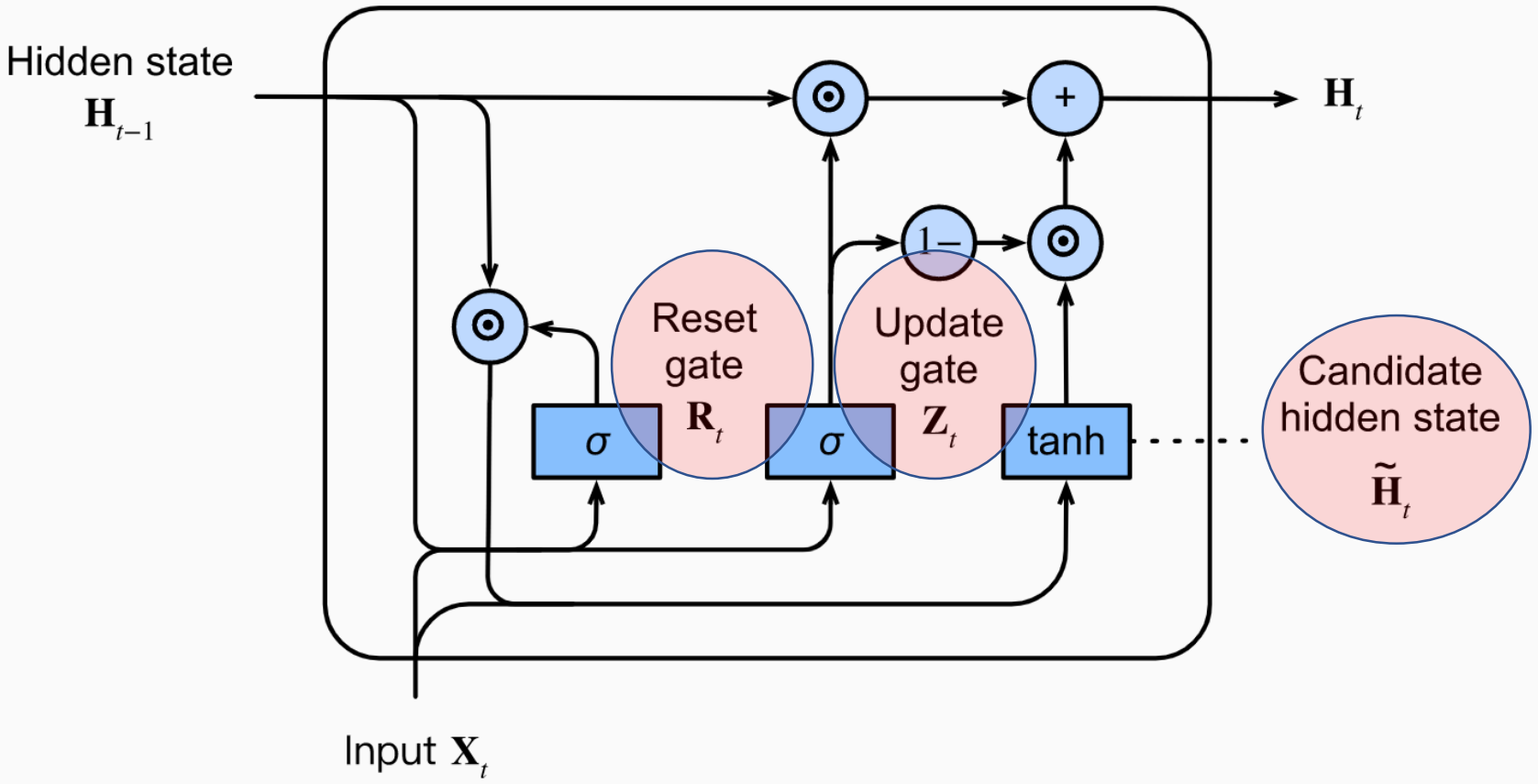
Without keras.layers.TimeDistributed()



- GRUs are very similar to Long Short Term Memory(LSTM).
  - ✓ Just like LSTM, GRU uses gates to control the flow of information.
  - ✓ They are relatively new as compared to LSTM.
  - ✓ This is the reason they offer some improvement over LSTM and have simpler architecture.

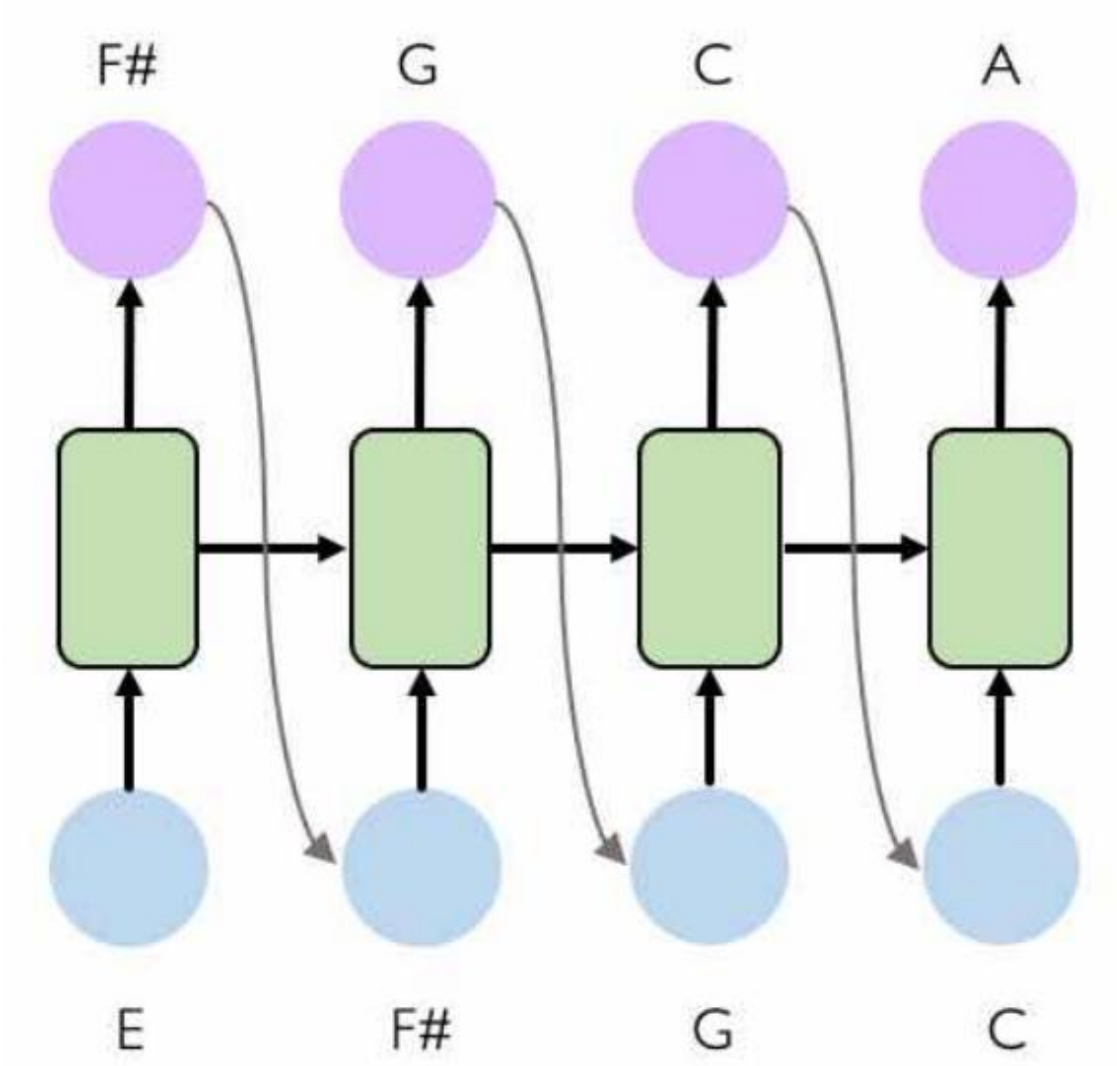


Gated Recurrent Unit

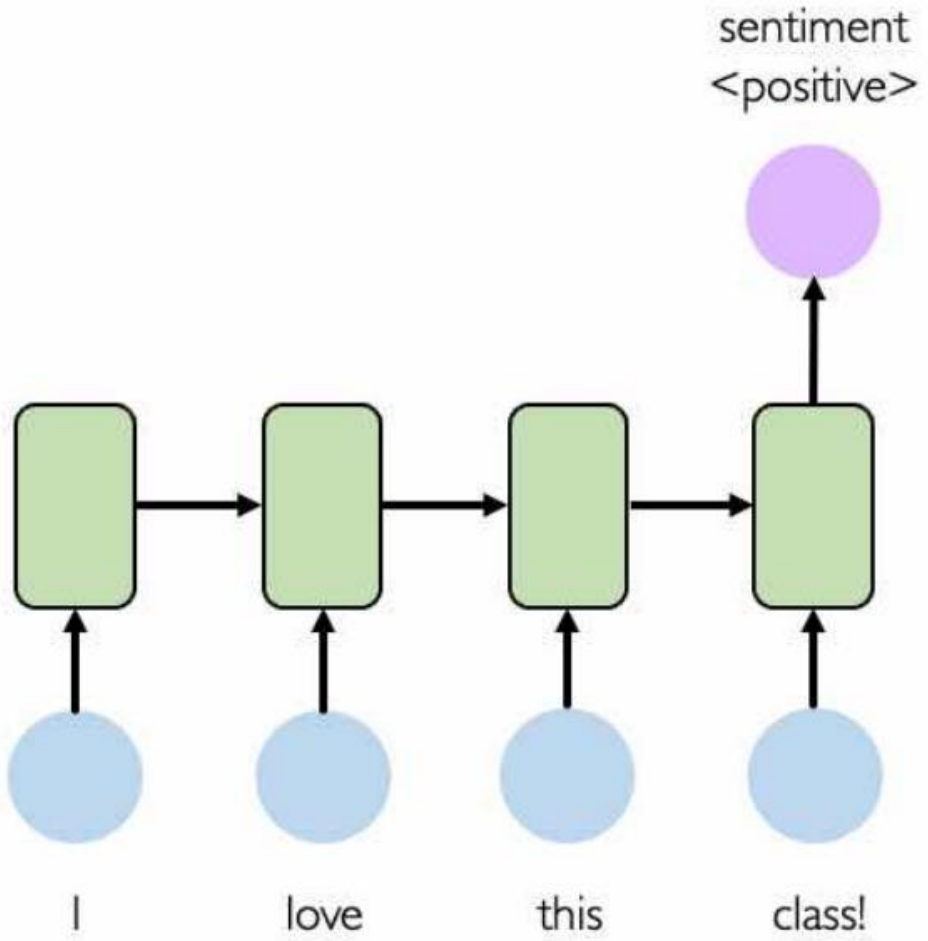


RNN 응용과 한계점

Example Task : Music Generation



Example Task : Sentiment Classification

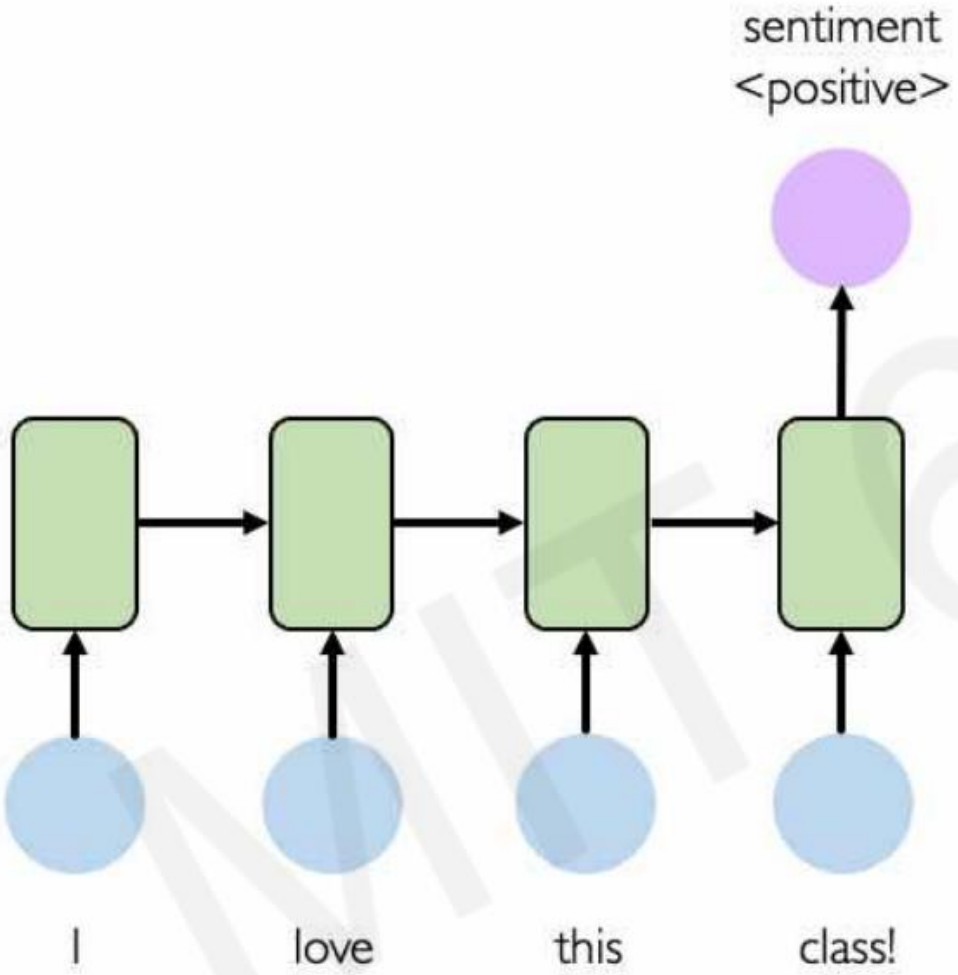


Input: sequence of words

Output: probability of having positive sentiment

```
loss = tf.nn.softmax_cross_entropy_with_logits(y, predicted)
```

Example Task : Sentiment Classification



Tweet sentiment classification

Ivar Hagendoorn
@IvarHagendoorn
Follow

The @MIT Introduction to #DeepLearning is definitely one of the best courses of its kind currently available online introtodeeplearning.com

12:45 PM - 12 Feb 2018

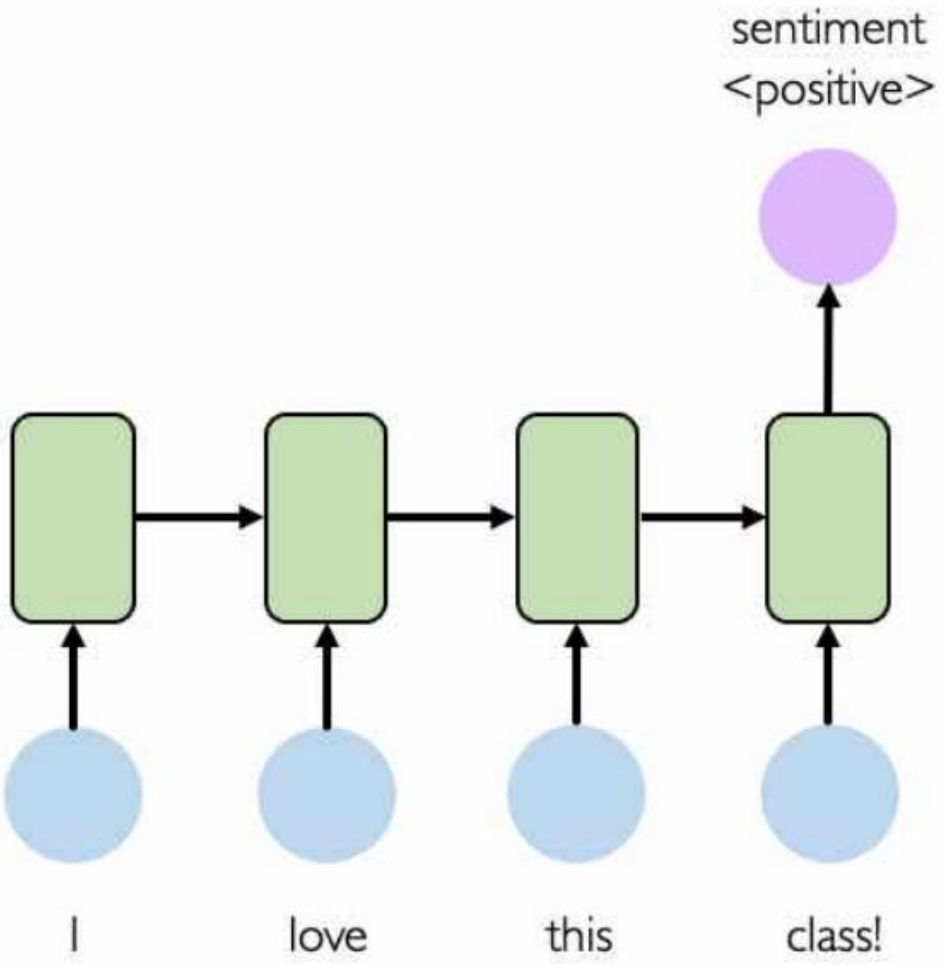
Angels-Cave
@AngelsCave
Follow

Replying to @Kazuki2048

I wouldn't mind a bit of snow right now. We haven't had any in my bit of the Midlands this winter! :(

2:19 AM - 25 Jan 2019

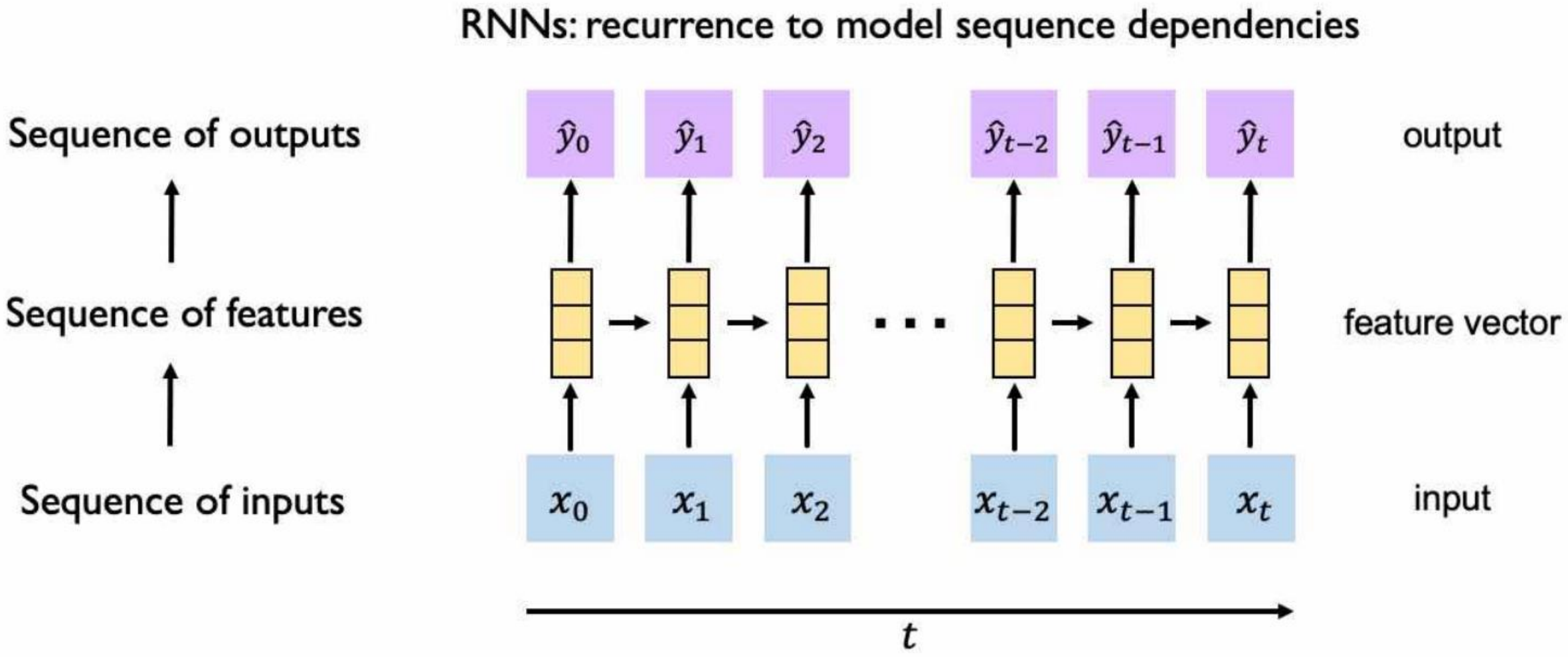
Limitations fo Recurrnet Models



Limitations of RNNs

- Encoding bottleneck
- Slow, no parallelization
- Not long memory

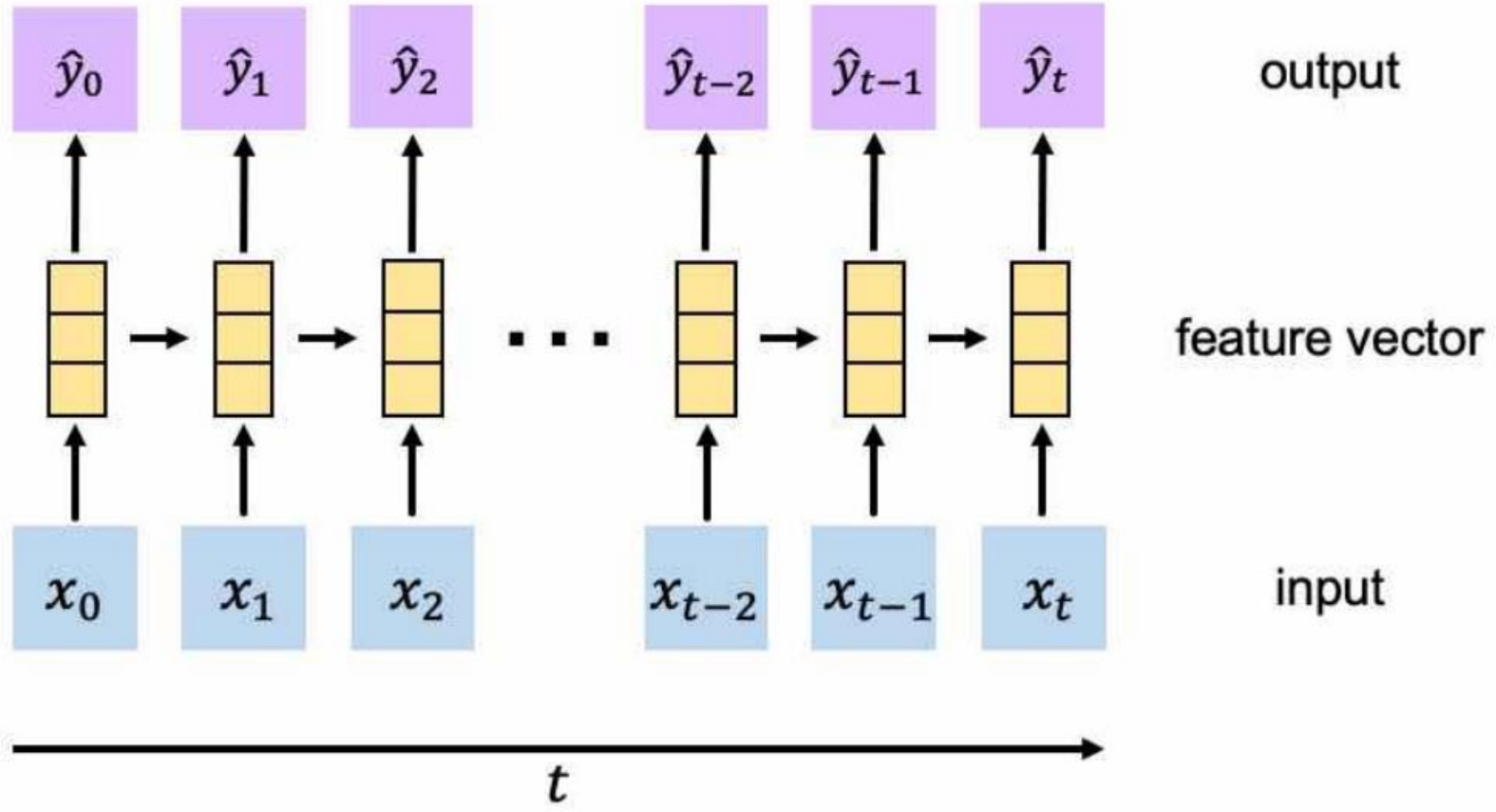
Goal of Sequence Modeling



Goal of Sequence Modeling

Limitations of RNNs

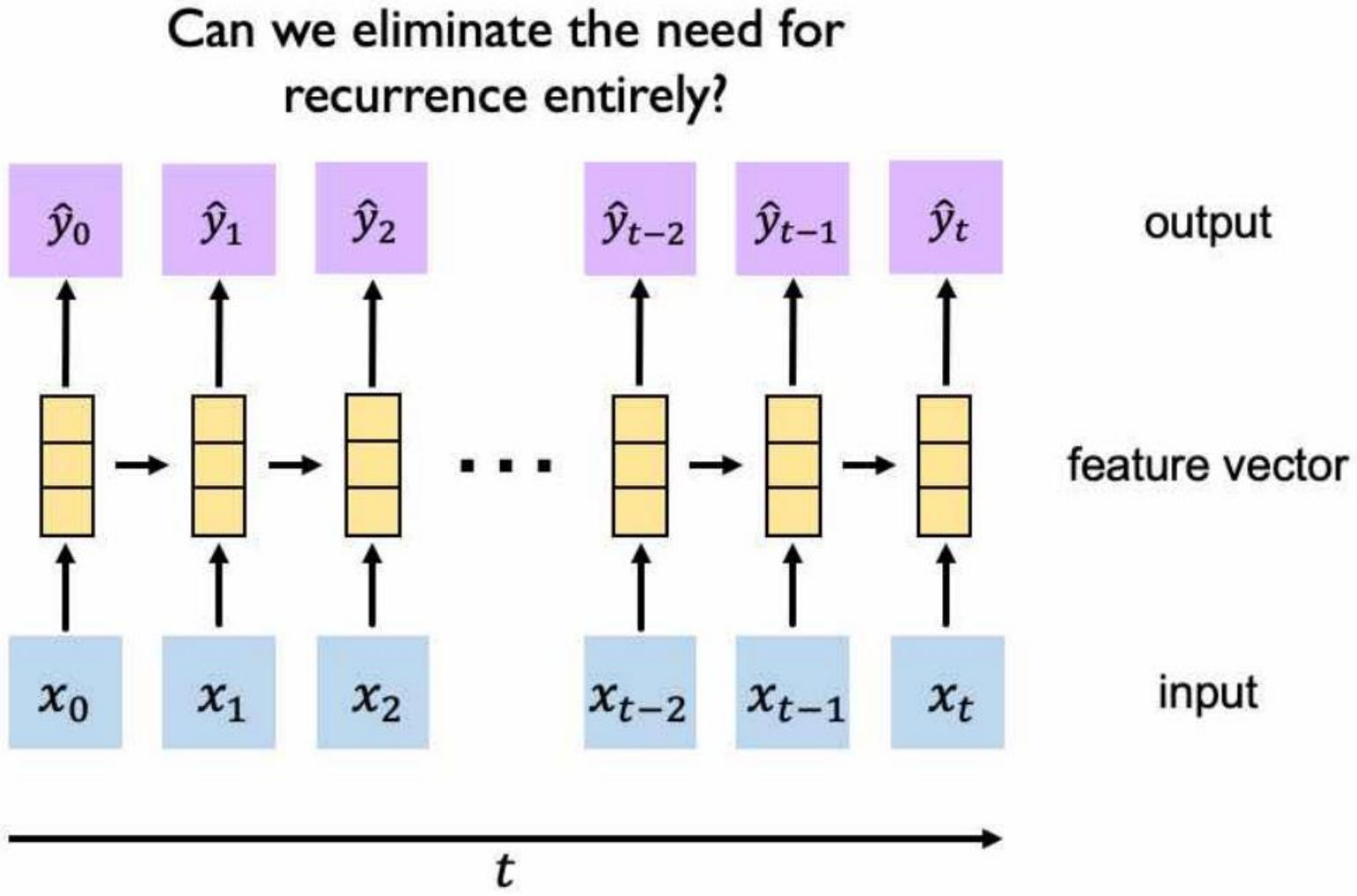
- Encoding bottleneck
- Slow, no parallelization
- Not long memory



Goal of Sequence Modeling

Desired Capabilities

- Continuous stream
- Parallelization
- Long memory

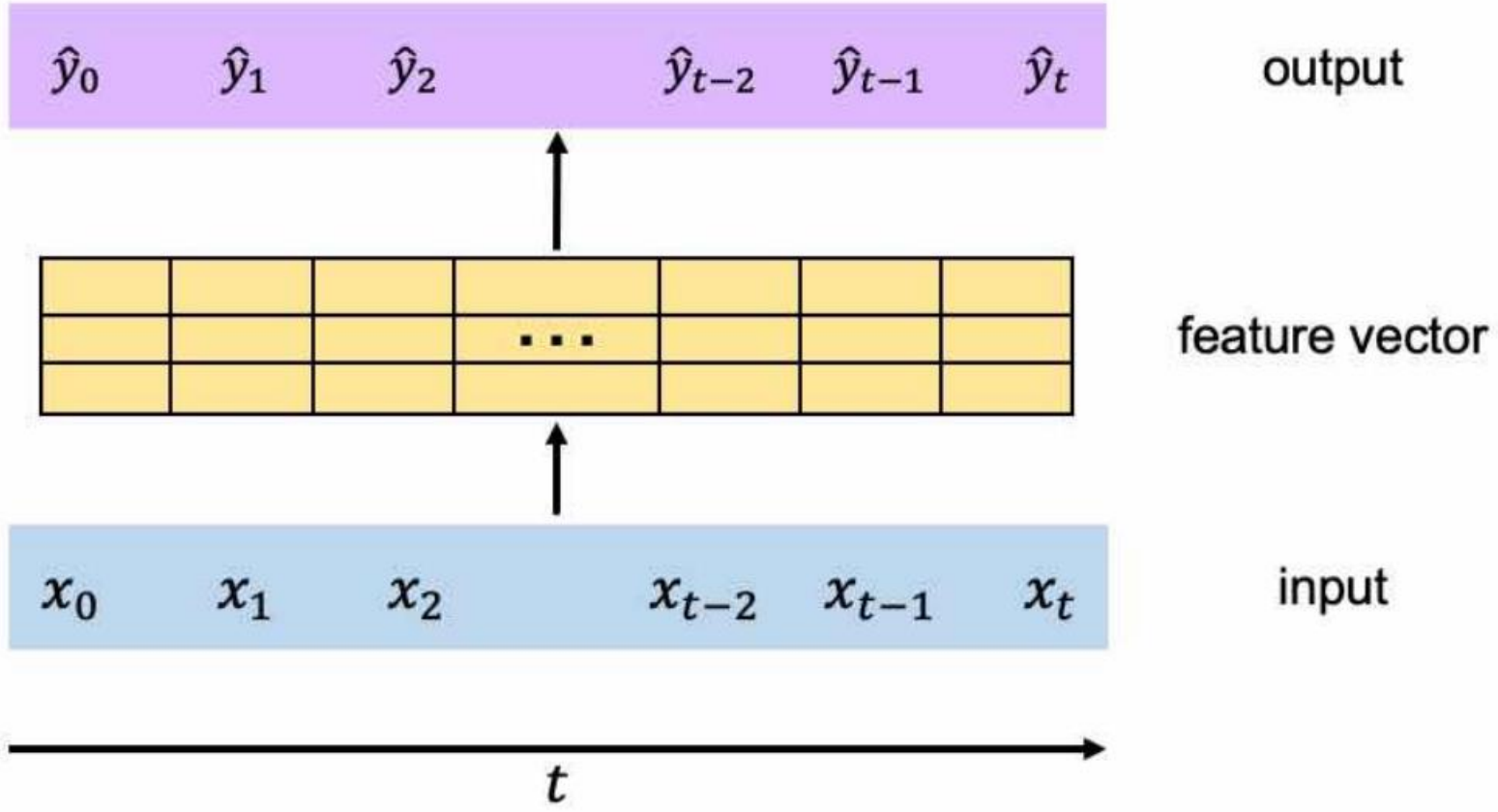


Goal of Sequence Modeling

Desired Capabilities

- Continuous stream
- Parallelization
- Long memory

Can we eliminate the need for recurrence entirely?



Goal of Sequence Modeling

Idea 1: Feed everything into dense network

✓ No recurrence

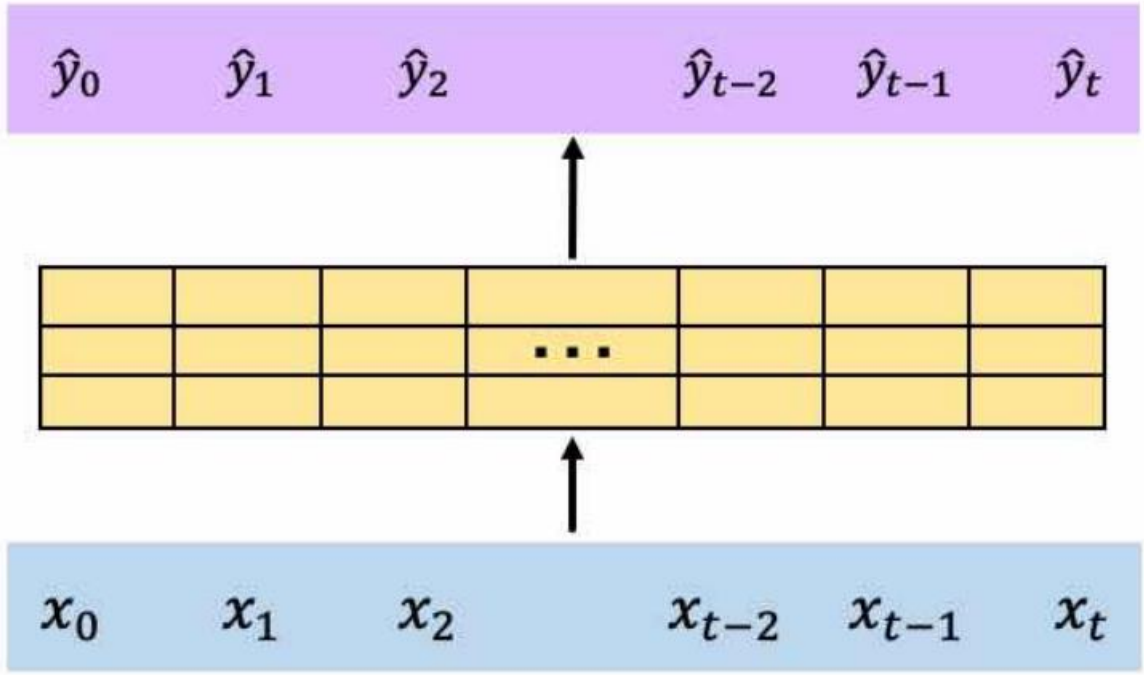
✗ Not scalable

✗ No order

✗ No long memory

Idea: Identify and attend to what's important

Can we eliminate the need for recurrence entirely?



어텐션 메커니즘

Attention Is All You Need

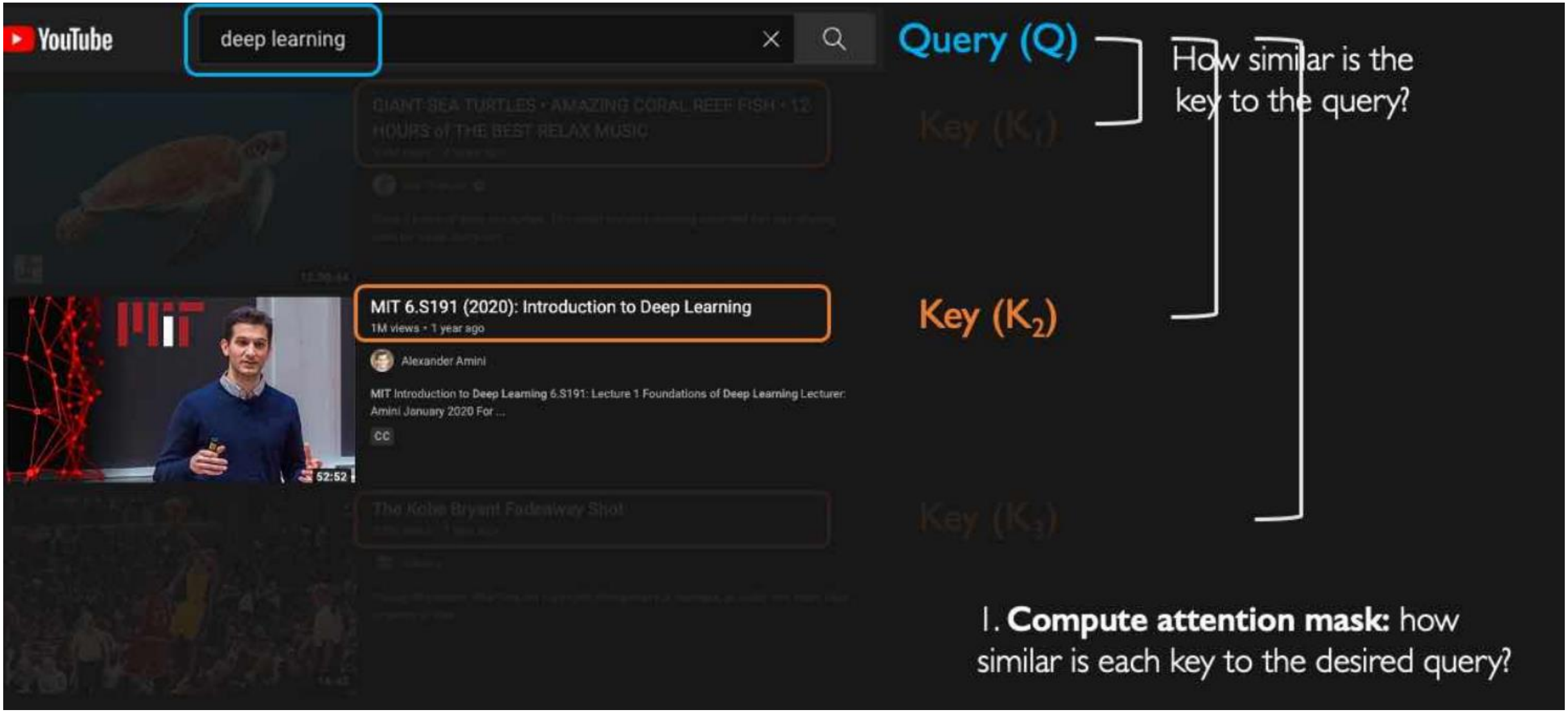
Intution Behind Self-Attention

Attending to the most important parts of an input.

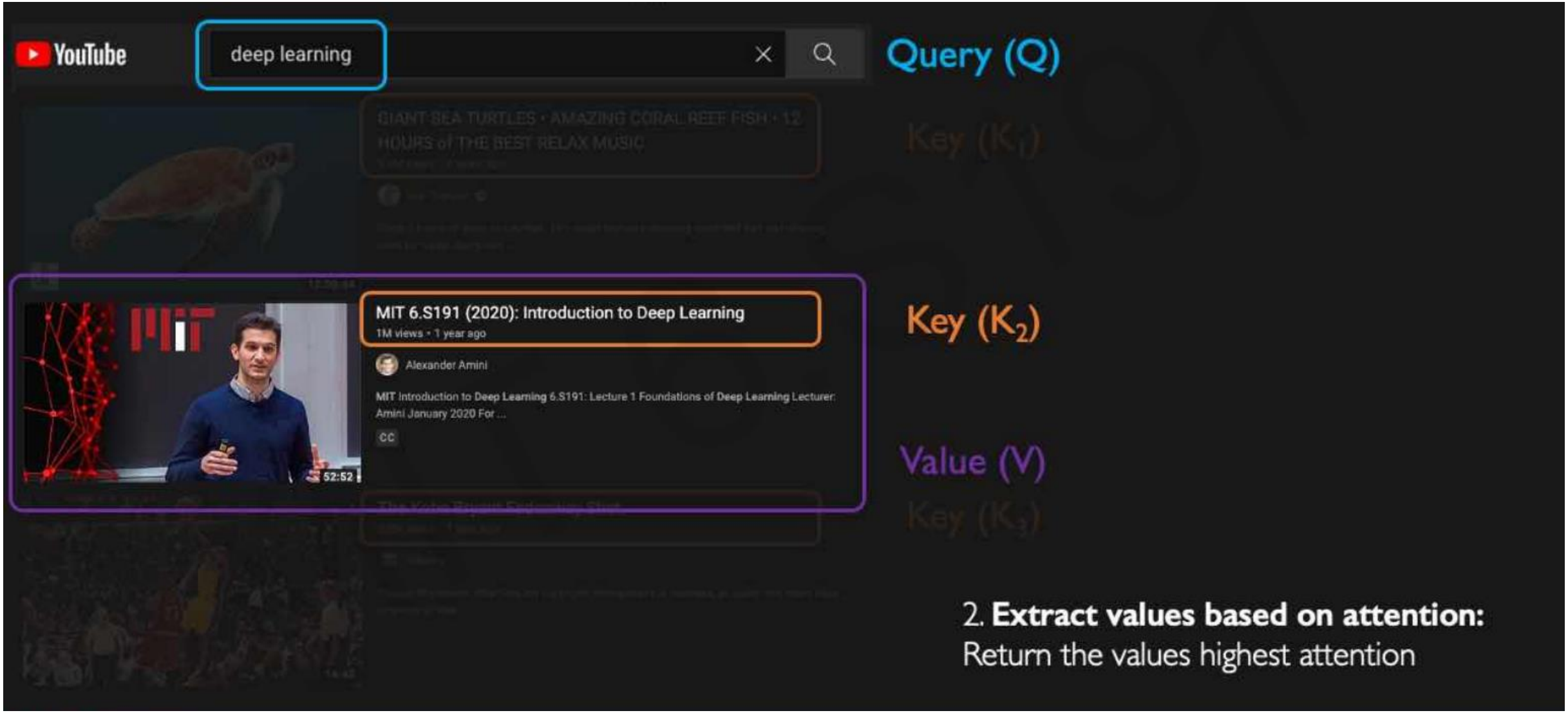
1. Identify which parts to attend to
2. Extract the features with high attention

Similar to a search problem!

Understanding Attention with Search

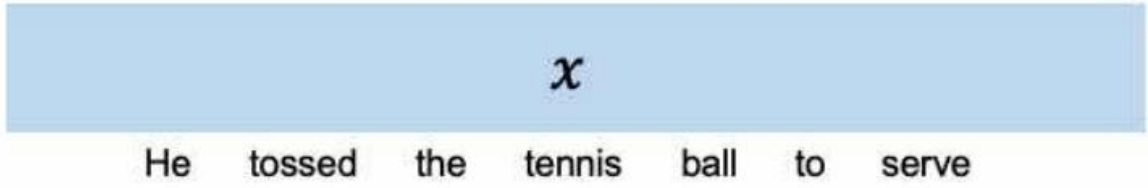


Understanding Attention with Search



Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.



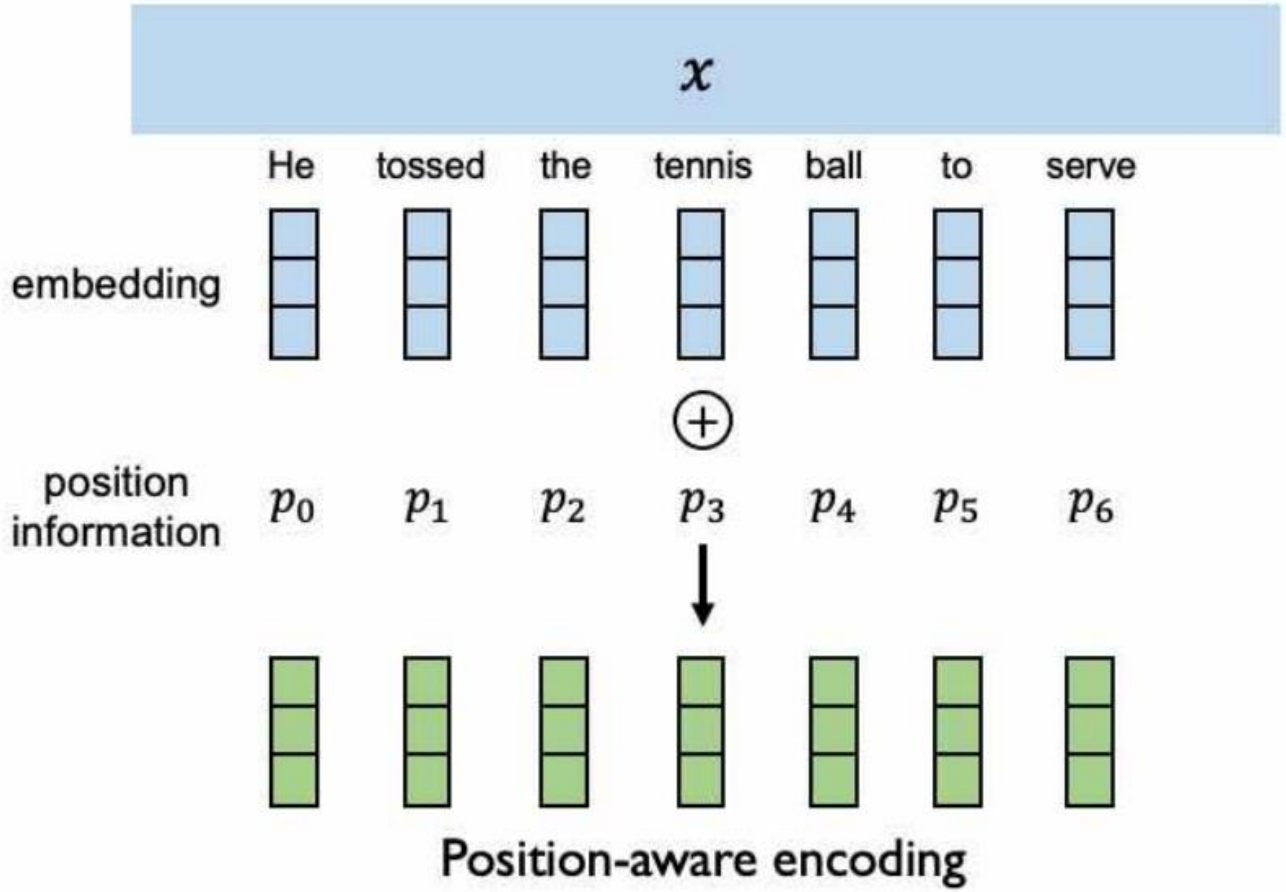
1. Encode **position** information
2. Extract query, key, value for search
3. Compute attention weighting
4. Extract features with high attention

Data is fed in all at once! Need to encode position information to understand order.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract query, key, value for search
3. Compute attention weighting
4. Extract features with high attention

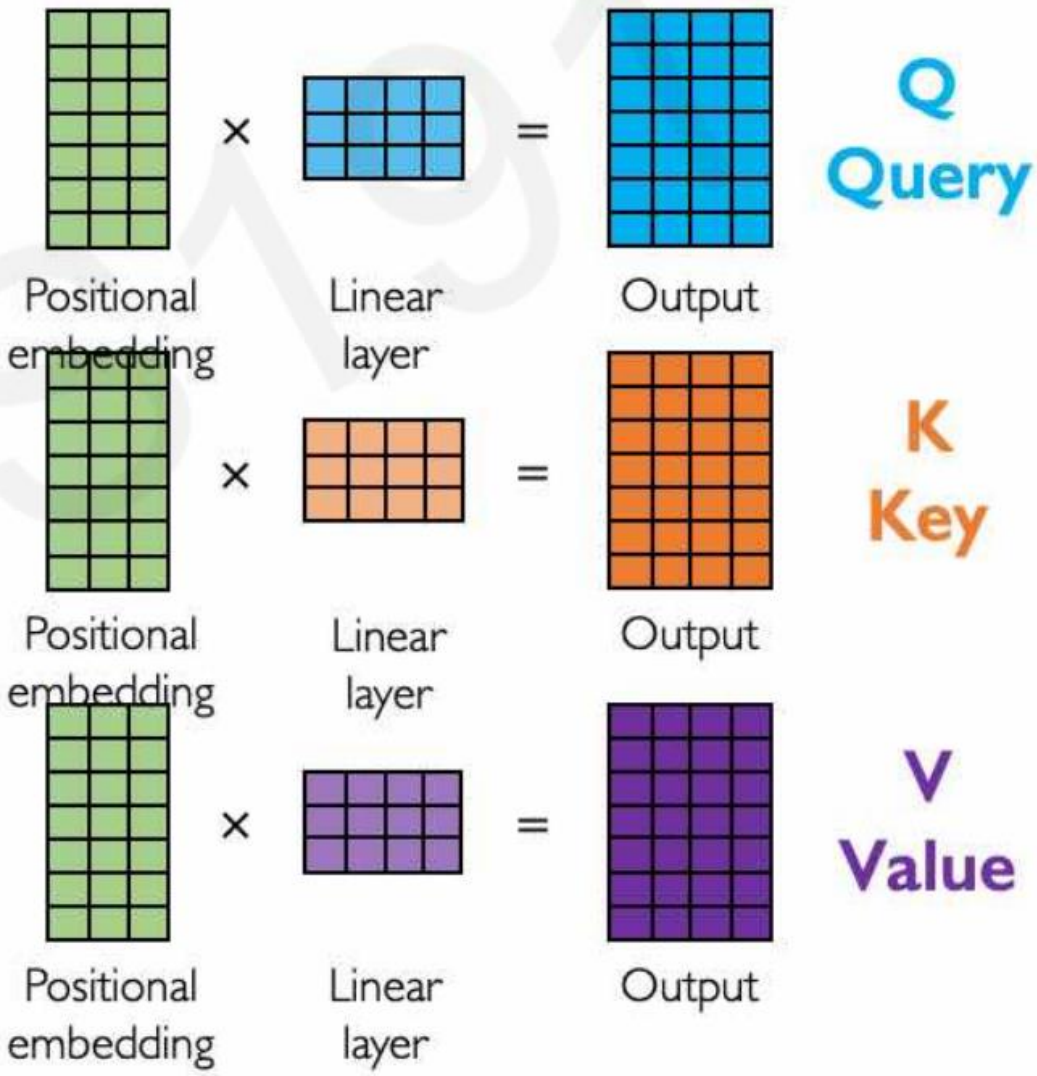


Data is fed in all at once! Need to encode position information to understand order.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute attention weighting
4. Extract features with high attention



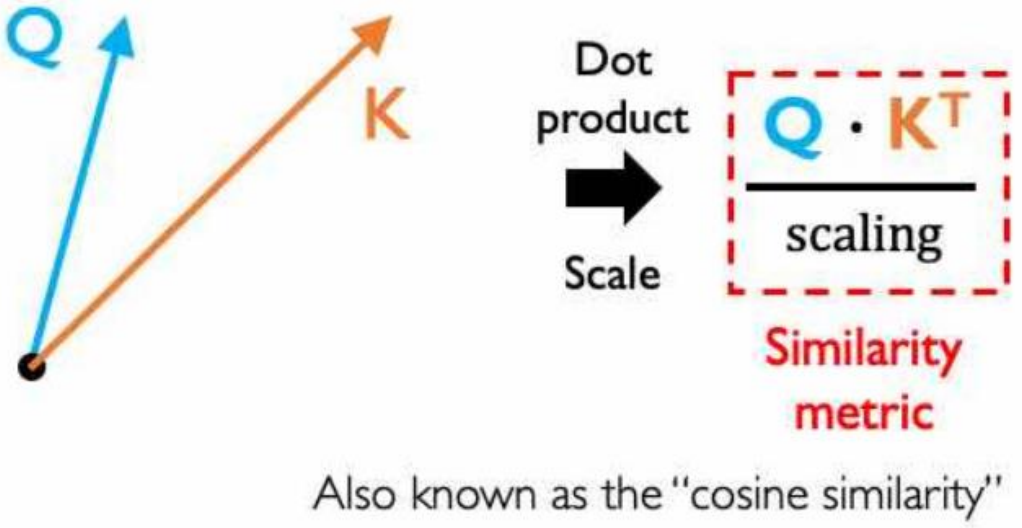
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention score: compute pairwise similarity between each query and key

How to compute similarity between two sets of features?



Also known as the "cosine similarity"

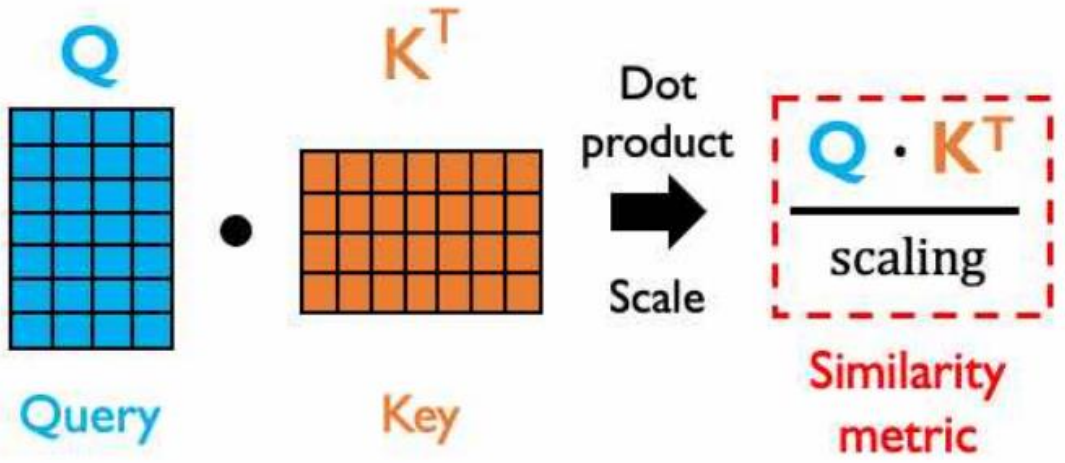
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention score: compute pairwise similarity between each query and key

How to compute similarity between two sets of features?



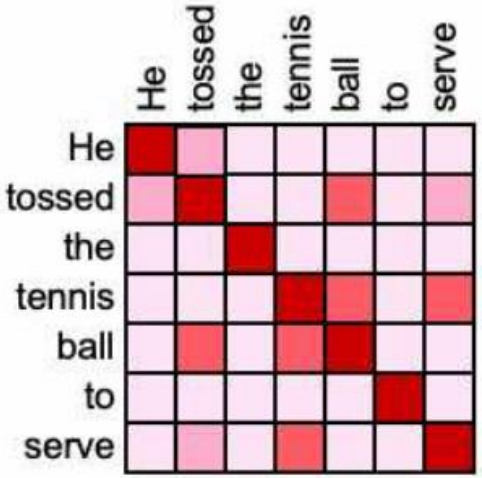
Also known as the "cosine similarity"

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention weighting: where to attend to! How similar is the key to the query?



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right)$$

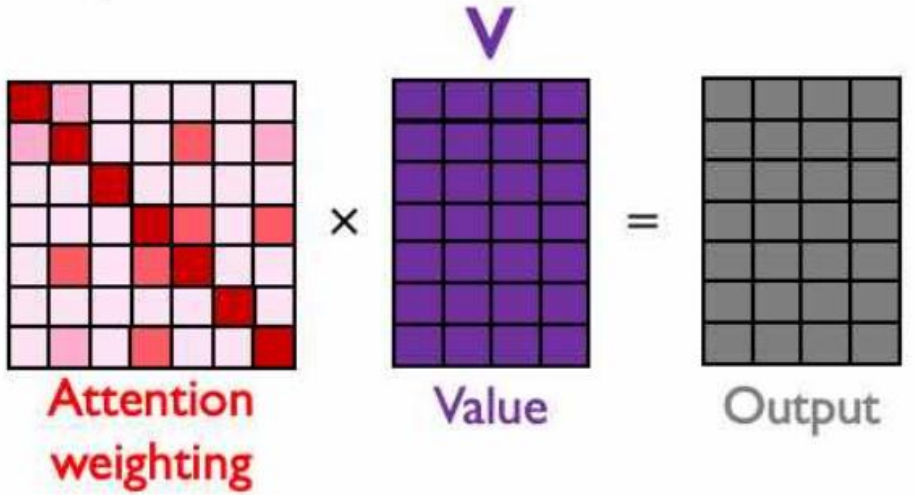
Attention weighting

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Last step: self-attend to extract features



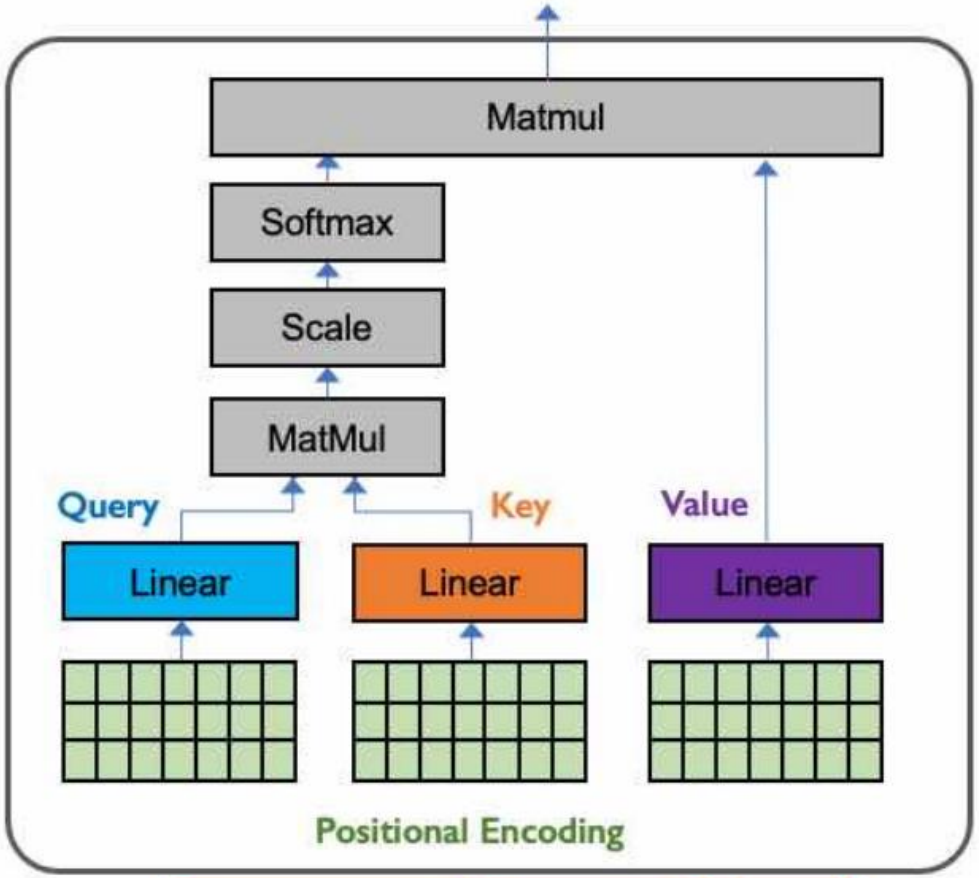
$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V = A(Q, K, V)$$

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

These operations form a self-attention head that can plug into a larger network. Each head attends to a different part of input.



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V$$

Applying Multiple Self-Attention Heads

Attention weighting

×

Value

=

Output

Output of attention head 1

Output of attention head 2

Output of attention head 3

Self-Attention Applied

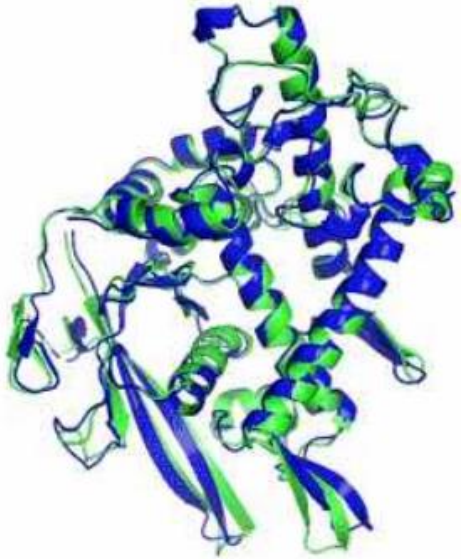
Language Processing

An armchair in the shape of an avocado

BERT, GPT-3

Devlin et al., *NAACL* 2019
Brown et al., *NeurIPS* 2020

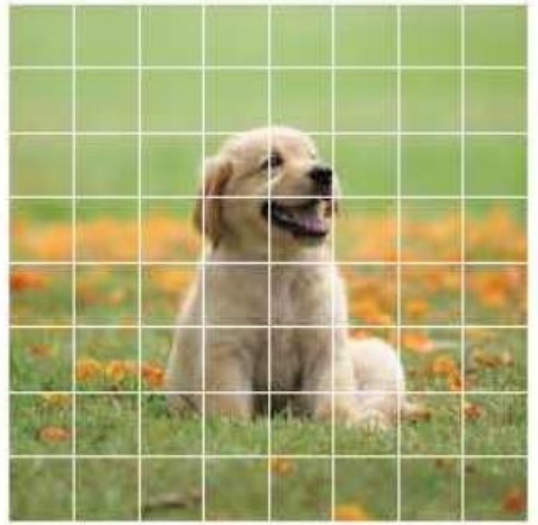
Biological Sequences



AlphaFold2

Jumper et al., *Nature* 2021

Computer Vision



Vision Transformers

Dosovitskiy et al., *ICLR* 2020

교통을 위한 트랜스포머 모델

학습할 내용

- **스마트교통 데이터 실제 및 활용**
  - ✓대전시 유성구 KISTI 정문 앞 교통 흐름 분석
- **학습할 내용**
  - ✓그래프 신경망에 대해서 이해해보자
  - ✓시공간 데이터를 이해하자
  - ✓어텐션과 트랜스포머 알고리즘을 이해하자
- **트랜스포머 모델에 적용할 수 있는 교통 데이터는 무엇있을까요?**

연구 배경 및 필요성

KISTI 본원 정문은 여러 해 동안 교통체증을 발생 시킬 수 있다는 막연한 우려

딥러닝 기반의 교통량 예측 연구가 시급

최근 인공지능 기술의 발전으로 교통량 예측을 매우 정확히 할 수 있음

AI 예측력으로 본원 정문 개통의 의사결정을 지원

한국과학기술정보연구원 본원 정문 개통을 위한 테스트베드 대상지 선정하고,
AI 기반 교통량 예측을 통해 , 교차로 신호 현시 최적화 시뮬레이션을 통한
본원 정문 개통의 의사결정을 지원

선행 연구 : (종료) 딥러닝기반 교통혼잡 예측 및 신호 제어 시스템 개발

❖ (확보한 AI 기술) 시계열 데이터의 패턴을 정확히 예측하는 기술을 적용

- RNN, LSTM(Long Short Term Memory)을 사용하여 교통량(속도, 점유율, 차량수)의 예측정확도 향상.
- 시공간 연관성을 고려하여 Spatial-Temporal 그래프 신경망(Graph Neural Network, ST-GNN) 적용

대전시 유성구 대학로 교통 데이터와 대상도로 현장적용 연구

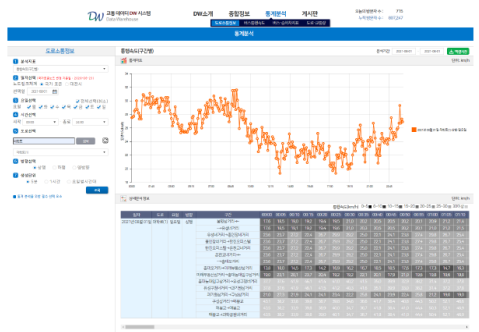
- 연구 대상도로 : 대학로 (유성구청네거리 ~ 구성삼거리)
  - ❖ 약 1.4km 왕복 4차로, RSE 3개, 영상검지기 5개(현재 3개, 향후 2개)

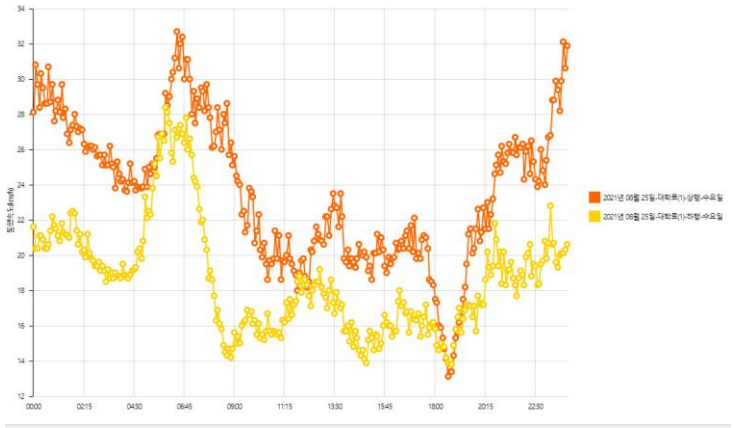
1) 대학로 인근 차량검지기(VDS)와 도로소통정보(RSE) 수집 및 분석

대전시 데이터웨어하우스에서 데이터 수집

❖ KISTI 정문(?) 앞 도로 교통 데이터 수집 : 대학로(1)

- 대전 교통 데이터 DW 시스템에서 대학로(1) RSE 데이터 수집
  - http://tportal.Daejeon.go.kr/
  - 통계분석 → 도로소통정보
    - ① 분석지표: 통행속도(구간별)
    - ② 일자선택
      - 노드링크체계: 국가 표준
      - 선택일: 2020-07-01 ~ 2021-08-31
    - ③ 요일선택: 전체선택
    - ④ 시간선택: 00:00 ~ 24:00
    - ⑤ 도로선택: 대학로(1)
    - ⑥ 방향선택: 상행 및 하행
    - ⑦ 생성단위: 5분

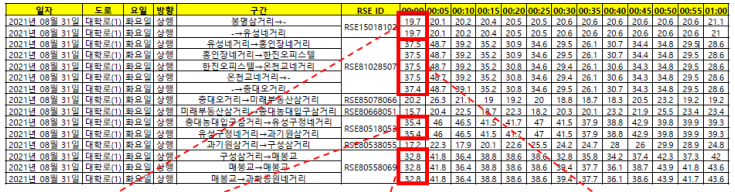




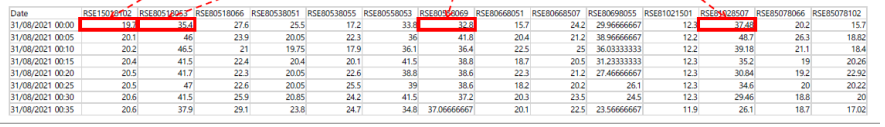
데이터 전처리

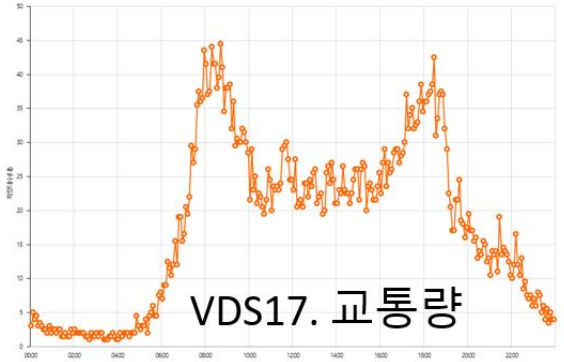
❖ RSE , VDS 데이터 변화

- Date(일자) 및 RSE ID 기준 RSE 측정 값 집계(평균)



Aggregate(Average)

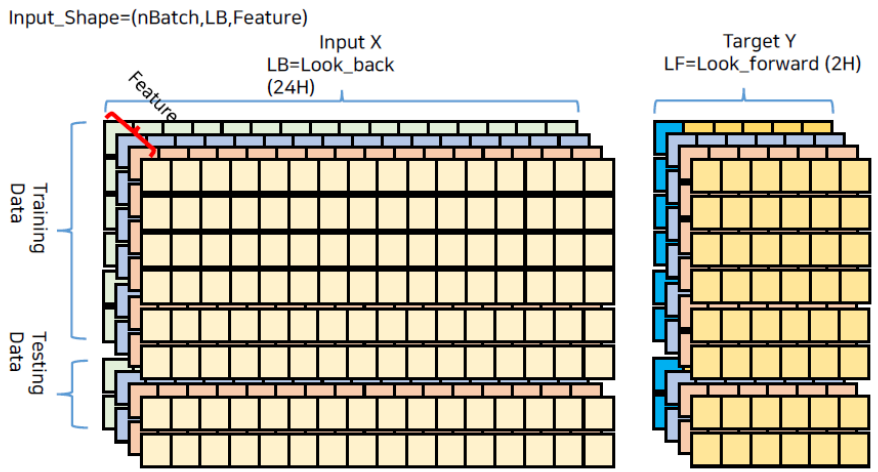




2) Long-Short Term Memory (LSTM)을 이용한 장단기 교통량 예측

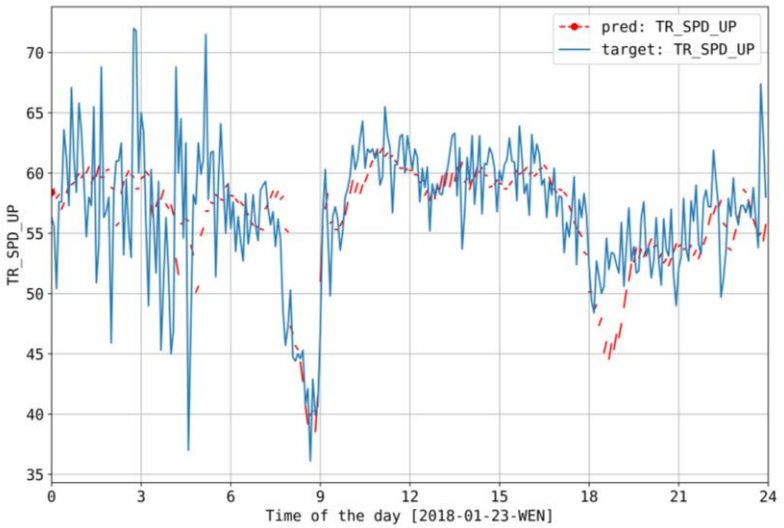
RNN 계열 장기예측을 위한 입력 입력 구조

- Look_back(입력에 사용할 Sequence Length)
- Look_forward(출력에 사용한 Prediction Length)
- Batch_Size 설정
- Train/Test/Validation 데이터 설정



차량 검지기 (VDS17, 상행,하행) 차량의 속도 예측

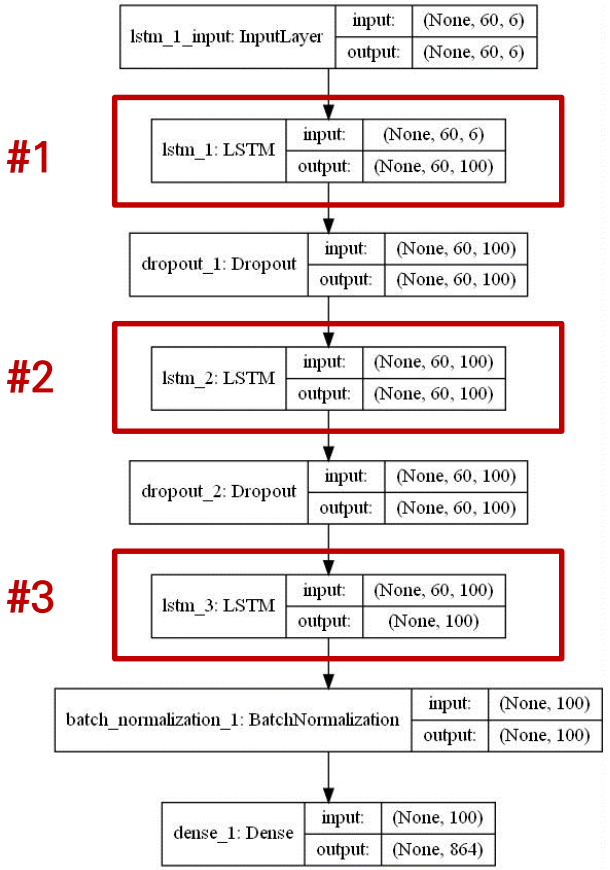
- 단기 예측은 Look_forward=1로 설정하며, 매 5분 단위 예측
- 보통 1시간 (12*5분) 예측은 정확도가 단기에 비하여 떨어진다.



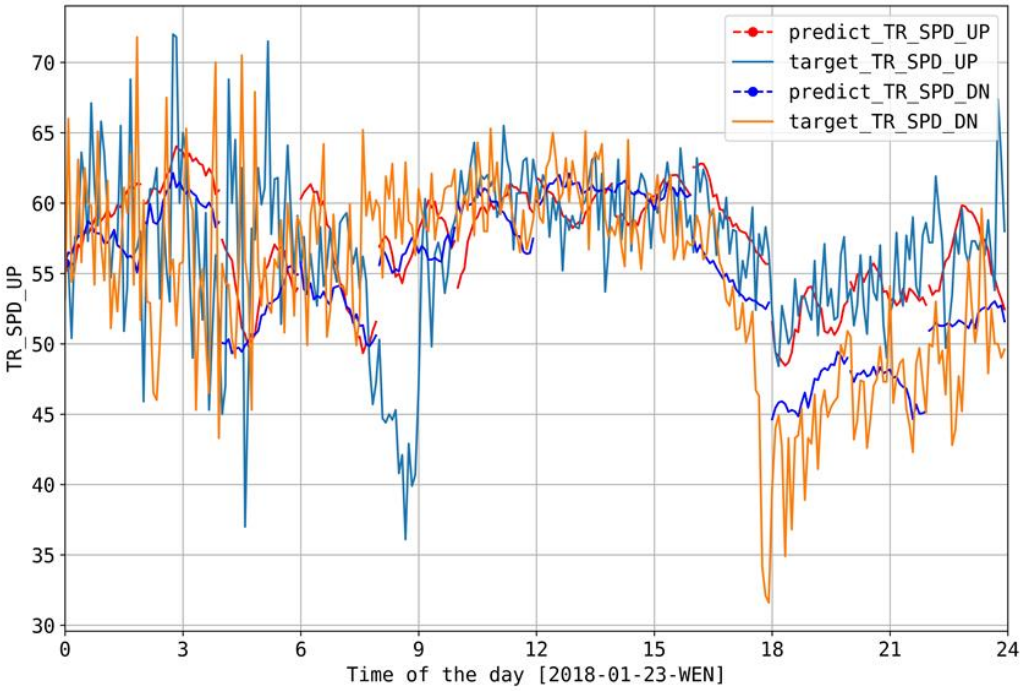
☆ LSTM 단기 예측(5분) 결과는 상대적으로 정확성이 높게 나왔다.

연구 사업 결과

최적 LSTM 아키텍처: 3개의 LSTM 층 사용함



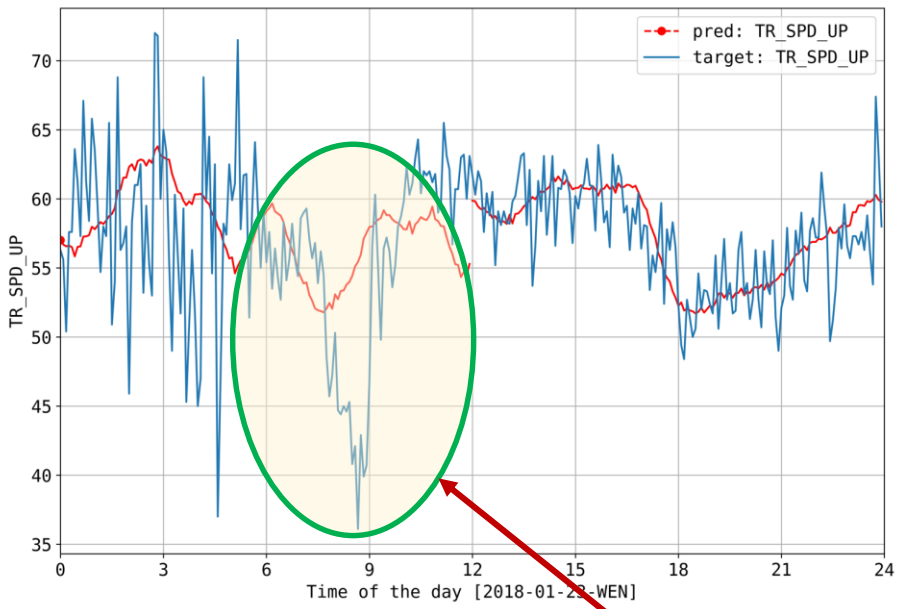
차량 검지기 (VDS17, 상행,하행) 차량의 속도 예측



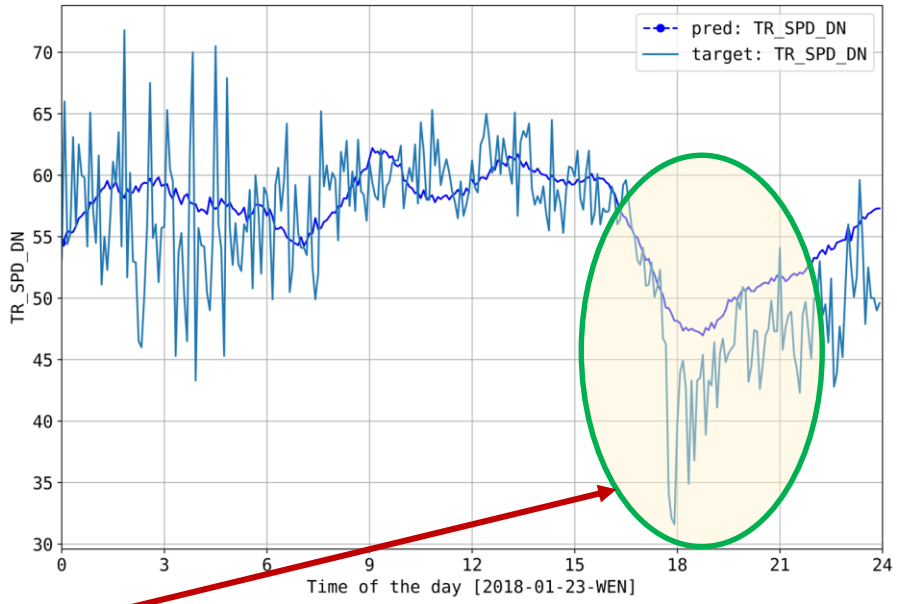
☆ LSTM 중기 예측(2시간) 결과는 상대적으로 정확도가 낮다.

LSTM 예측은 RNN에 비하여 성능은 좋은나, Long-Sequence 장기예측에는 성능저하가 발생한다.

상행 12시간예측 (2018.01.23.,수요일)



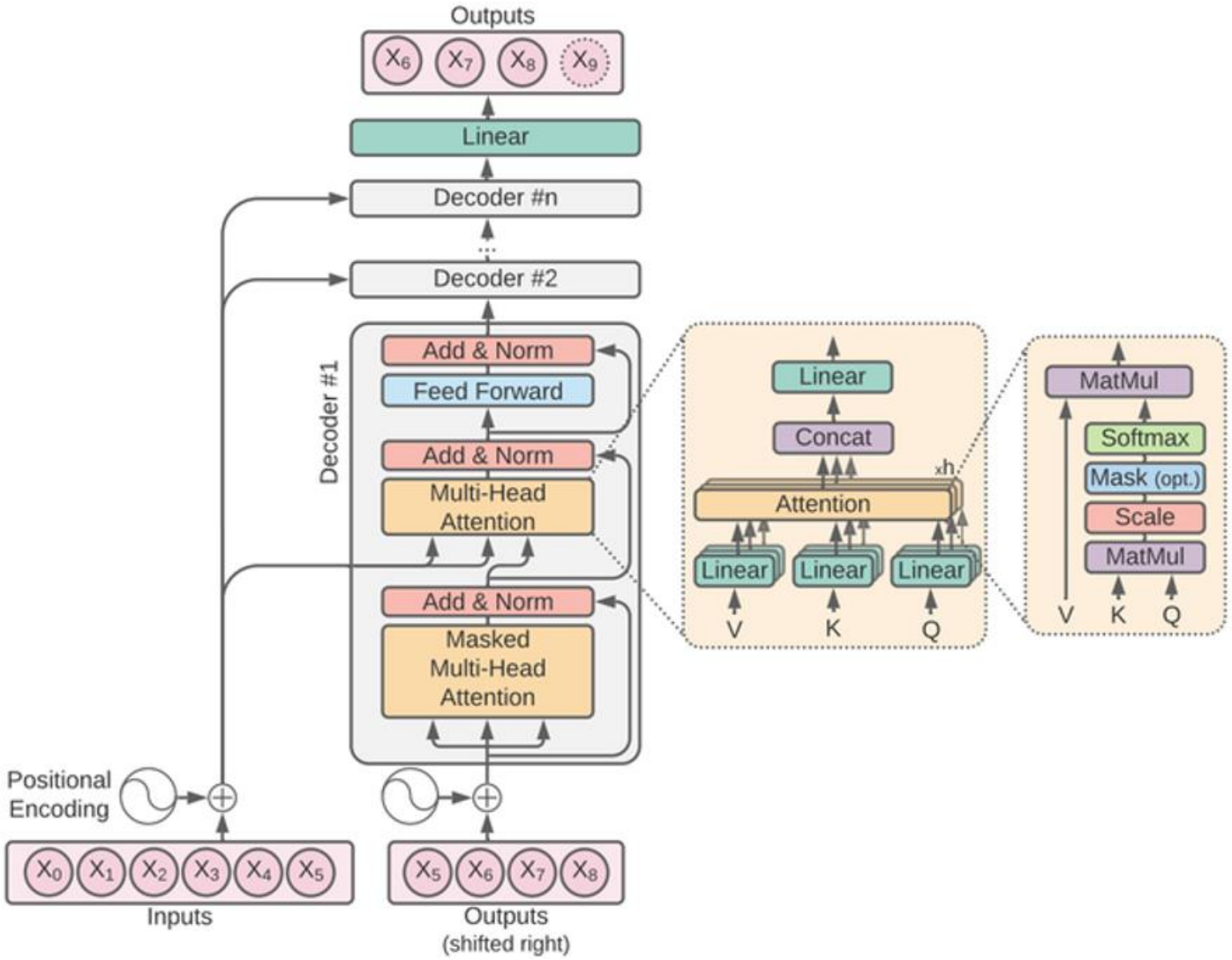
하행 12시간예측 (2018.01.23.,수요일)



LSTM 장기예측(12시간)과 예측 정확은 재고할 필요가 있다. → 새로운 모델이 필요하다.

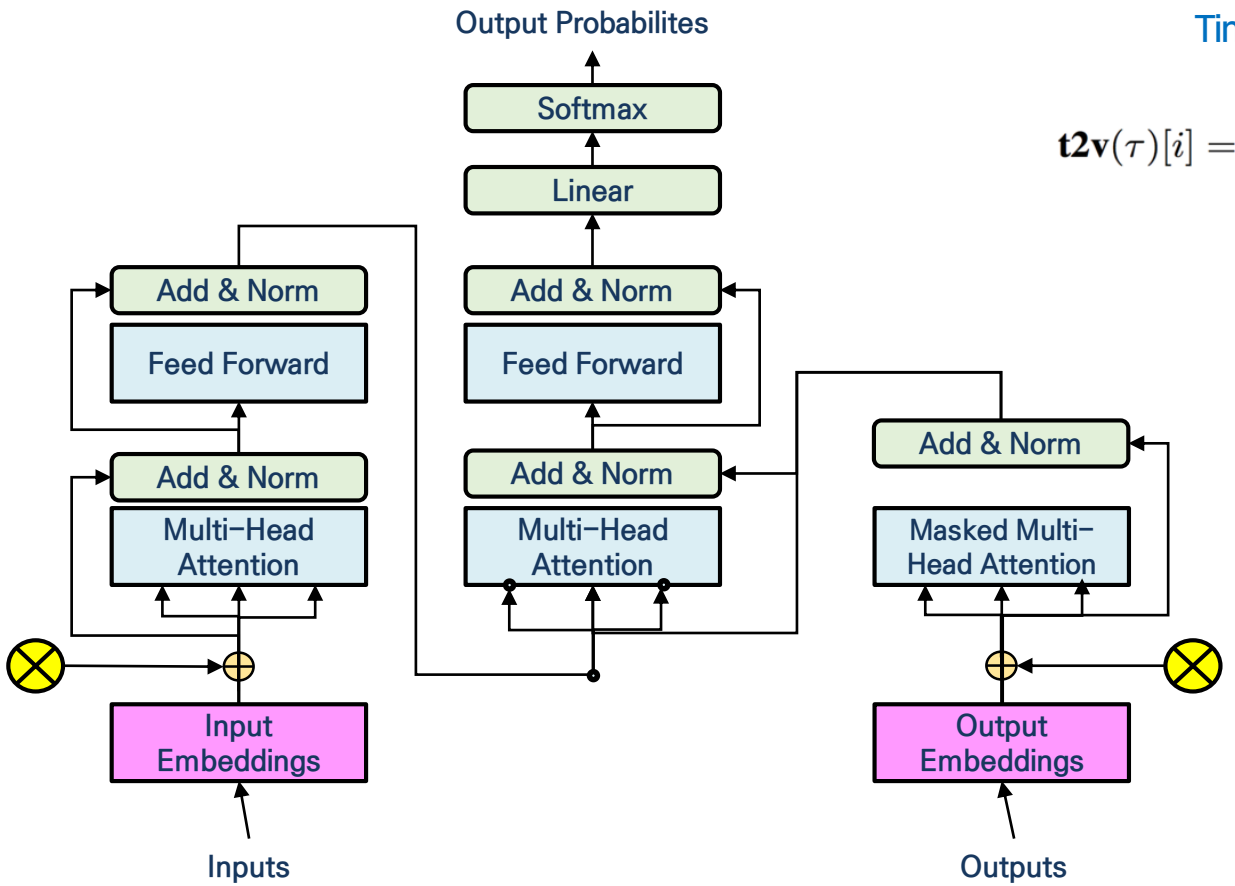
☆ LSTM 장기 예측(12시간) 결과는 상대적으로 정확도가 현저히 떨어진다.

멀티 헤드 어텐션 메커니즘



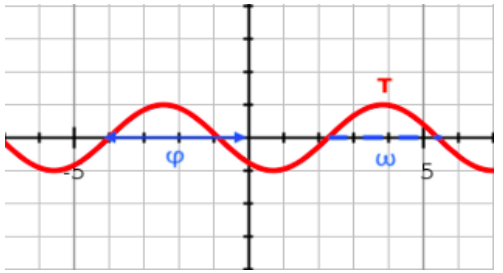
교통에 적합한 Position Encoding : Time2Vec

자연어 처리 트랜스포머 모델의 포지셔널 인코딩을 교통에 적합하게 Time2Vec 개발이 핵심이다.



❖ Attention 도입으로 시계열성을 없어지고, 대신에 Time2Vec 임베딩을 이용하여 시공간적인 정보 유지

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0. \\ \mathcal{F}(\omega_i \tau + \varphi_i), & \text{if } 1 \leq i \leq k. \end{cases}$$



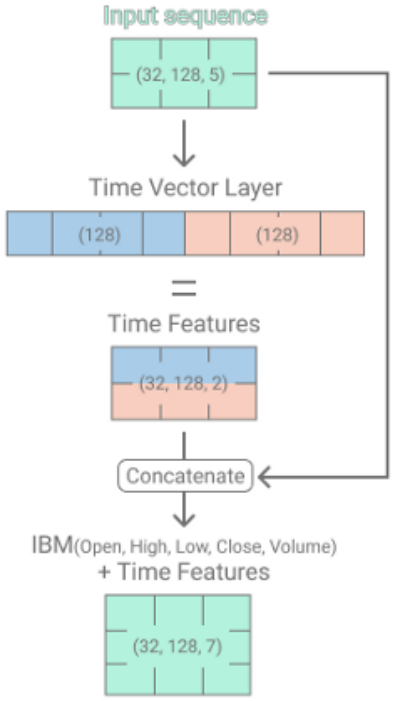
주기성: 삼각함수(sin)를 사용

비주기성: 선형함수

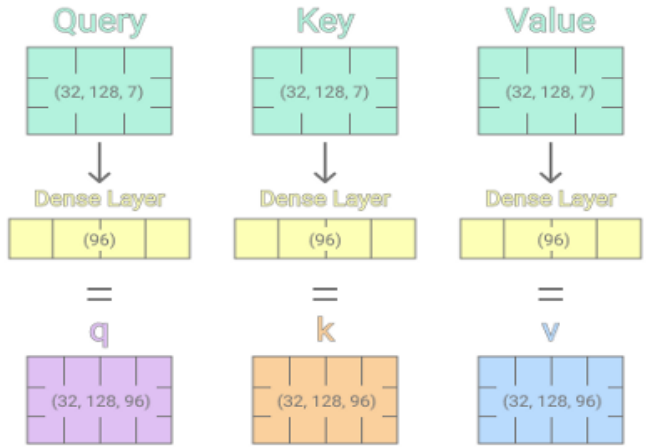
어텐션의 Query, Key, Value 전략

Time2Vec

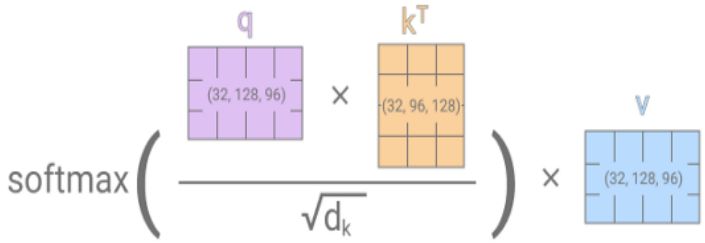
- 5개 VDS 17, VDS 18, VDS 19, VDS 20, VDS 21
- 트랜스포머 모델 입력크기(input-size)
  - batch_size: 32, sequence_length: 128, feature : 5



Single Head Attention : Query, Key, Value



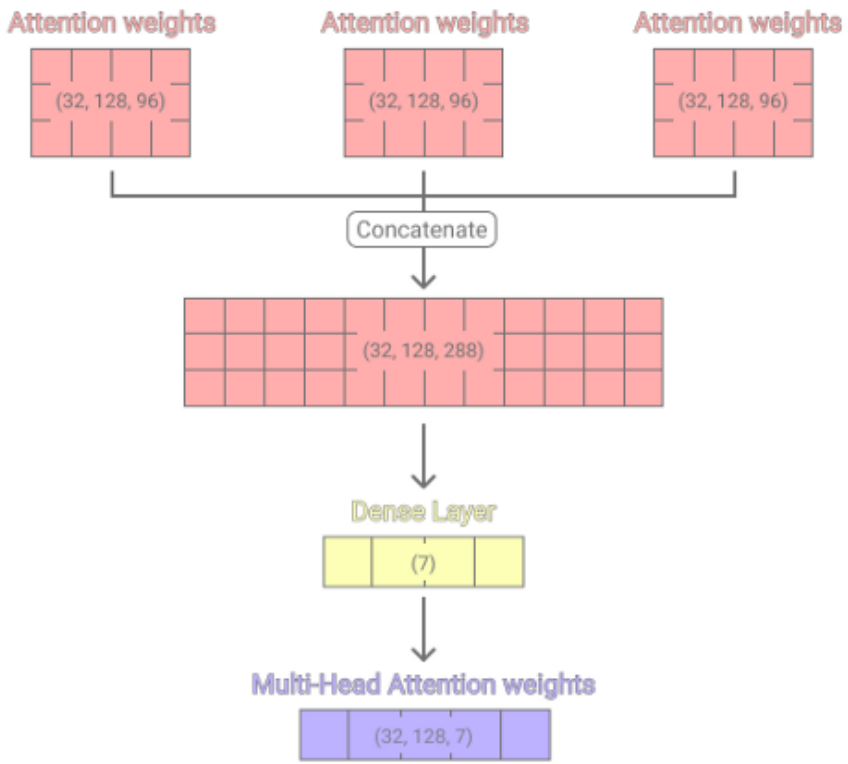
- Attention Weight는 Softmax를 사용함. Q*K*V



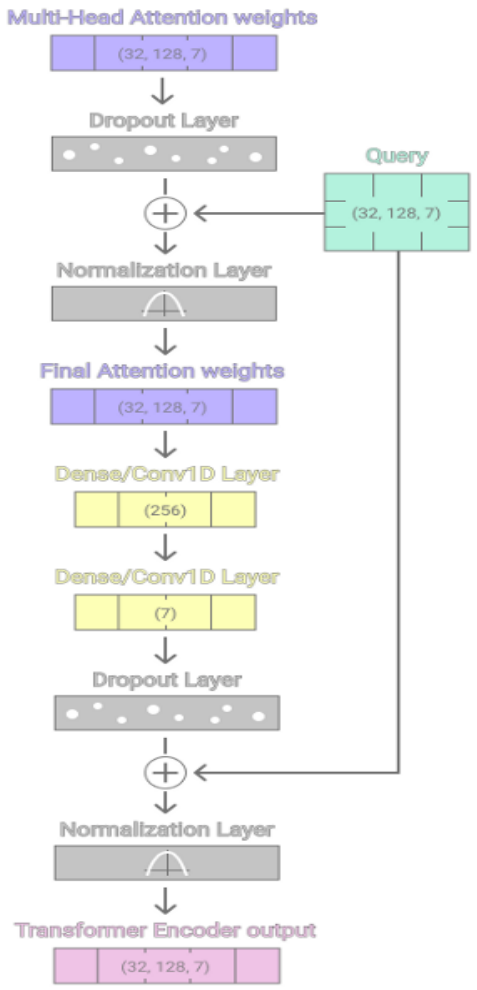
인코더-디코더 아키텍처 기반 Transformer 교통 모델

Mulit-Head Attention으로 병렬처리가 가능함

❖ VDS 17, VDS 18 데이터 구조 (train, test, validation)



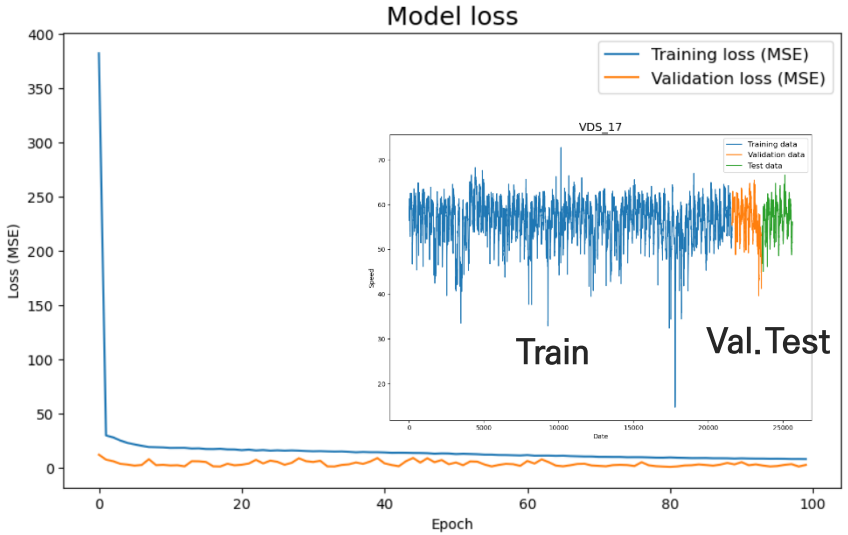
Transformer Encoder 출력이되 Decoder에 사용됨



트랜스포머 VDS 데이터로 장기 예측

KISTI SINCE 1962

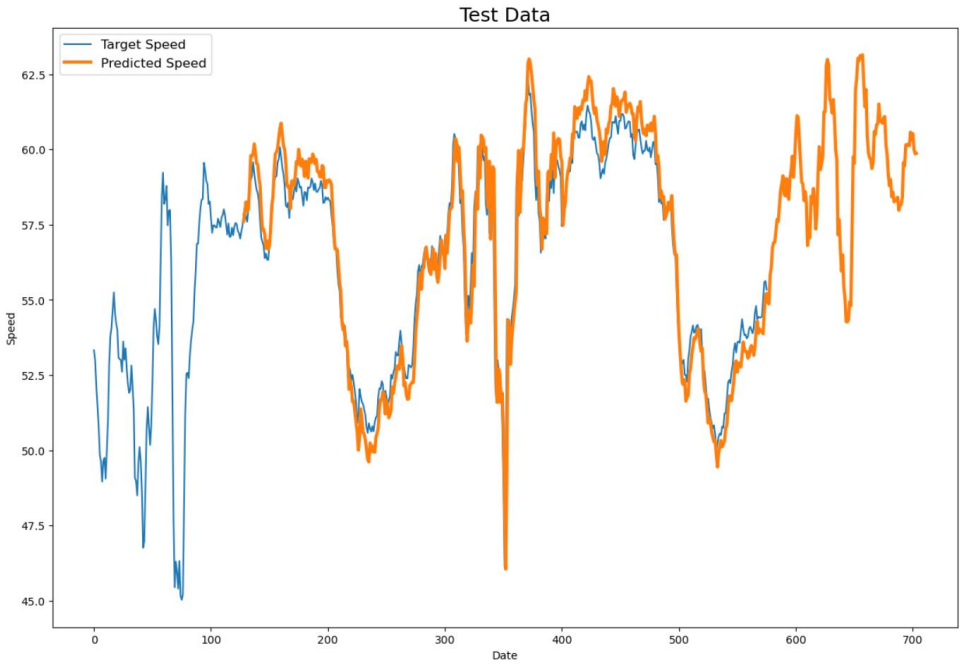
대전시 차량검지기(VDS) 데이터 105개



- 대학로 관련 5개 위치 데이터 90일
  - 데이터 개수: 25,920개
  - 90일*24*12 (5분 단위)

장기예측 성능 : 2016개 예측(1주일, 288*7)

- TransformerVDS: Transformer+TimeEmbedding

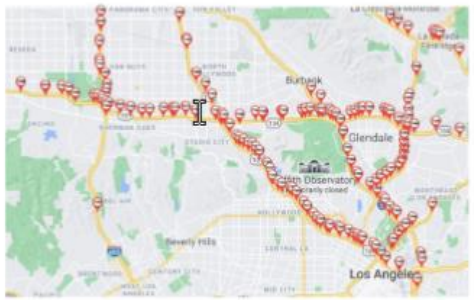


- TransformerVDS 결과와 대전 UVDS 데이터에 적용하고 있으며, SCI 논문에 투고할 예정

Dynamic Spatial Transformer WaveNet Network

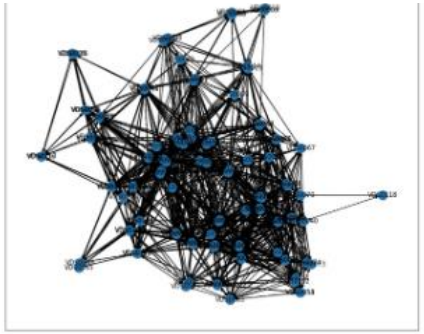
UVDS: 대전 VDS 데이터 기반 시공간 그래프 신경망 구축
Spatial-temporal Graph neural Network

- ❖ Metr-LA : 대표적 교통 공개 데이터 (미국, LA지역 4개월)
- ❖ UVDS 공개 : (시공간) 대전시 차량검지기 데이터 (3개월)

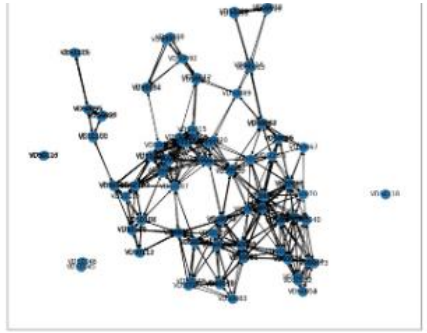


a) METR-LA

b) UVDS



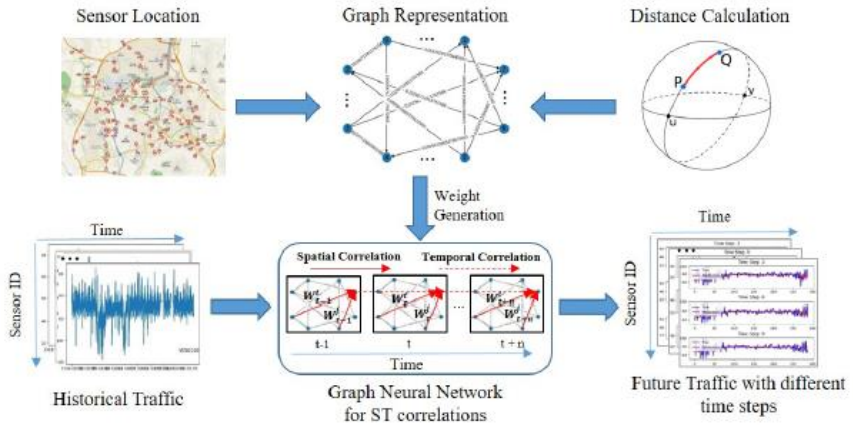
a) β = 0.1



b) β = 0.5

대전시 VDS 데이터 기반 새로운 UVDS 데이터 공개
UVDS: A New Dataset for Trac Forecasting with Spatial-Temporal Correlation

- ❖ General Framework for Traffic Forecasting using Spatial-temporal Correlaitons



Dynamic Spatial Transformer WaveNet Network

UVDS 데이터 기반 동적 시공간-트랜스포머 웨이브넷 신경망 개발 및 UVDS 데이터 성능 테스트
DSTWN 모델 : Dynamic Spatial Transformer WaveNet Network로 Spatial-Temporal Graph 신경망 보다 우수성 검증

❖ 기존 Spatial-Temporal Graph Network과 비교 성능 향상 목표

❖ 개발한 DSTWN 알고리즘 성능 벤치마크 : Metr-LA 데이터

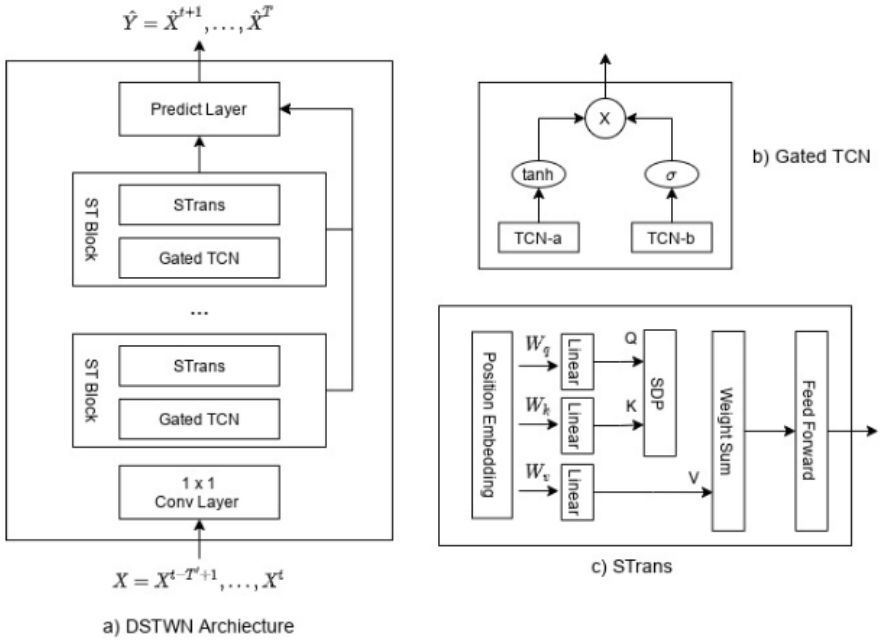


Fig. 3. The DSTWN Architecture

Table 2. Results on METR-LA dataset.

Step	Metrics	GraphWN	MTGNN	STWaNet	STTN	DSTWN
1	MAE	2.2408	2.2441	2.2791	2.4080	2.3220
	RMSE	3.8636	3.9185	3.9793	4.2339	4.0704
	MAPE	0.0540	0.0551	0.0550	0.0588	0.0567
3	MAE	2.7127	2.6762	2.7457	2.9160	2.8758
	RMSE	5.1690	5.1428	5.3162	5.6556	5.6285
	MAPE	0.0695	0.0686	0.0711	0.0778	0.0759
6	MAE	3.0974	3.0605	3.0947	3.3819	3.2909
	RMSE	6.1839	6.2002	6.2781	6.8651	6.7740
	MAPE	0.0847	0.0816	0.0838	0.0958	0.0900
9	MAE	3.3617	3.3100	3.3239	3.6961	3.5461
	RMSE	6.8279	6.8380	6.8639	7.5749	7.3837
	MAPE	0.0950	0.0912	0.0924	0.1085	0.0984
12	MAE	3.5760	3.4937	3.5036	3.9533	3.7446
	RMSE	7.2883	7.2421	7.2761	8.1262	7.8246
	MAPE	0.1035	0.0982	0.0993	0.1181	0.1047
Training		45.6927	62.8770	54.5744	77.5496	133.2374
Inference		1.4630	1.6825	1.5830	6.6945	3.8088

딥러닝을 이용한 교통흐름 예측 연구

KISTI SINCE 1962

❖ 대전시 데이터웨어하우스에서 차량검지기(VDS)와 RSE 데이터 수집 및 분석함

- ✓KISTI 정문 앞 도로 차량검지기(VDS 17데이터)는 오전과 출근과 오후 퇴근에 일부 속도가 떨어지는 경향이 있음.
- ✓대학로 인근 RSE 데이터는 전체적으로 속도가 너무 낮게 측정되었음.

❖ (딥러닝 기반 교통 흐름 예측)

- ✓RNN 기반 LSTM을 양방향 장단기로 교통 흐름 예측하였고 교통 혼잡은 없는 것으로 예측됨
- ✓Transformer_VDS 모델 개발로 장기 교통흐름 예측 정확도가 향상됨
- ✓Spatial-Temporal Graph 데이터 기반 Dynamic Spatial Transformer WaveNet 모델 개발 및 성능 테스트 함

KISTI SINCE 1962

2022 TRUST KISTI

Korea Institute of Science and Technology Information