AI VIETNAM
All-in-One Course

Time-series Forecasting Project

(MLP, RNN, LSTM, Bi-LTSM, XGBoost)

Vinh Dinh Nguyen
PhD in Computer Science

Year 2023

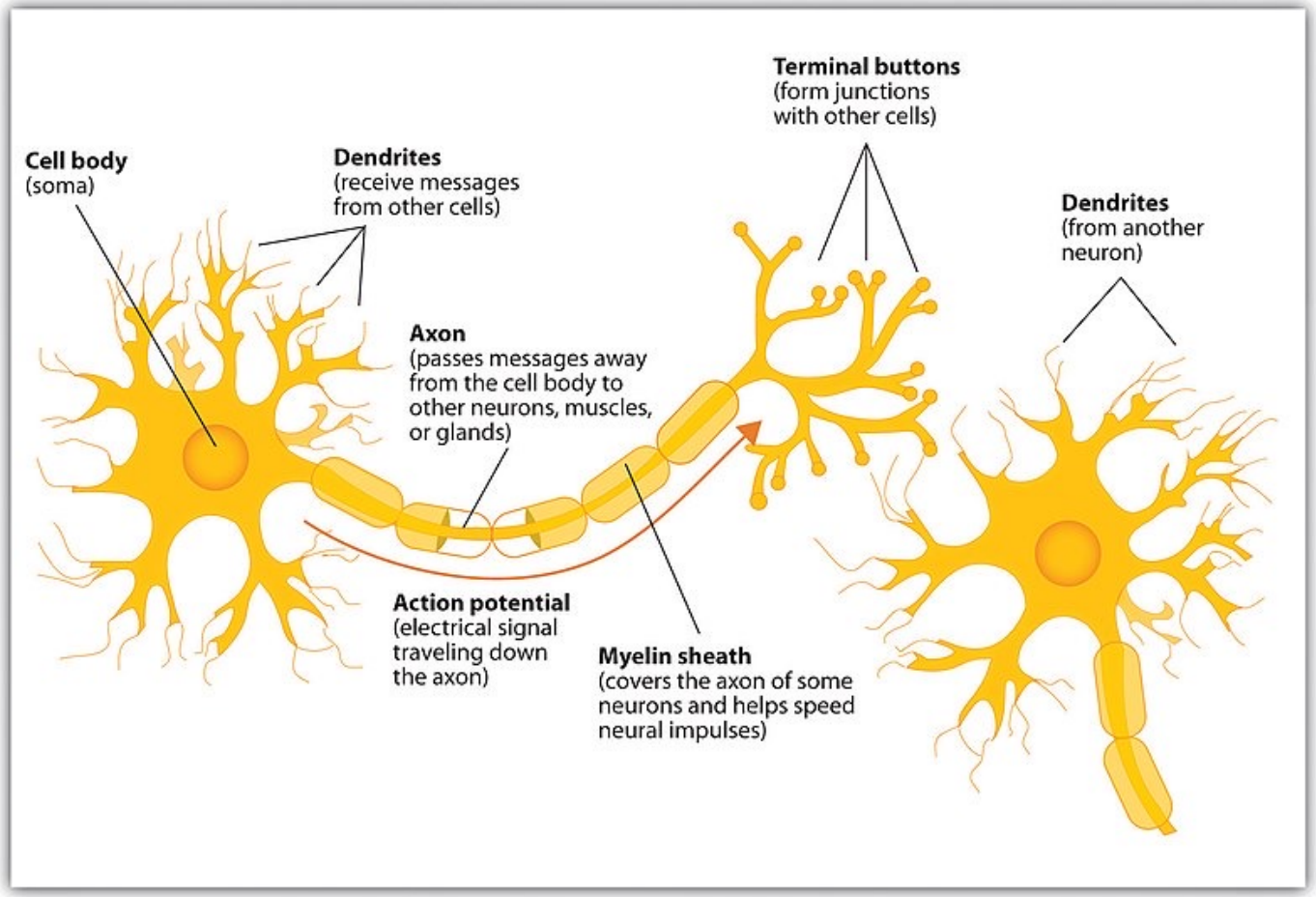
Outline

- Multilayer Perceptron and Time-series Data Forecasting
- RNN and Time-series Data Forecasting
- LSTM and Time-series Data Forecasting
- Bi-LSTM and Time-series Data Forecasting
- XGBoost and Time-series Data Forecasting

Outline

- Multilayer Perceptron and Time-series Data Forecasting
- RNN and Time-series Data Forecasting
- LSTM and Time-series Data Forecasting
- Bi-LSTM and Time-series Data Forecasting
- XGBoost and Time-series Data Forecasting

Neuron: Simple Explanation



Neuron and it's different components

In the early 1940's Warren McCulloch, a neurophysiologist, teamed up with logician Walter Pitts to create a model of how brains work. It was a simple linear model that produced a positive or negative output, given a set of inputs and weights.

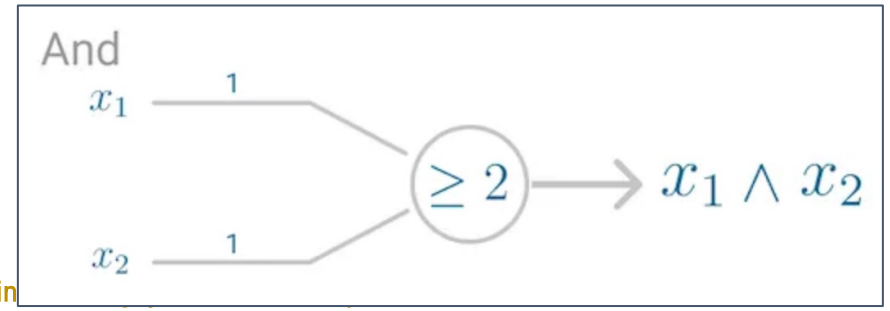
$$\underbrace{f(x,w)}_{\text{output}} = \underbrace{x_1}_{\text{inputs}} w_1 + \cdots + x_n \underbrace{w_n}_{\text{weights}}$$

Neuron: Simple Explanation

The first application of the neuron replicated a logic gate, where you have one or two binary inputs, and a boolean function that only gets activated given the right inputs and weights

AND FUNCTION

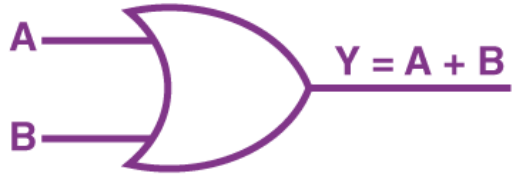
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



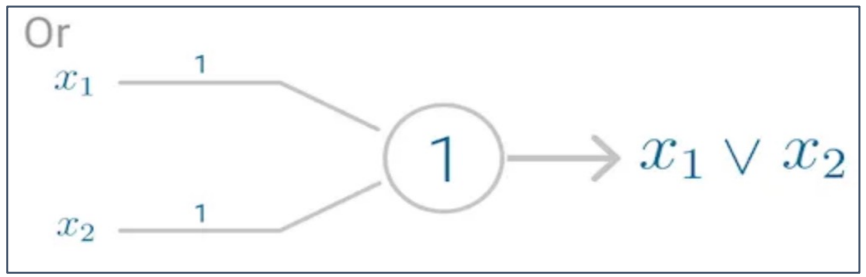
It couldn't *learn* like the brain.

Frank Rosenblatt extended t his model

OR FUNCTION

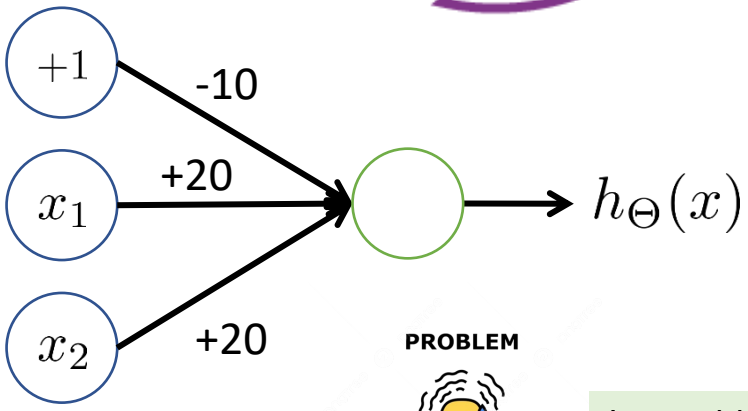


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



Neuron: Simple Explanation

OR FUNCTION

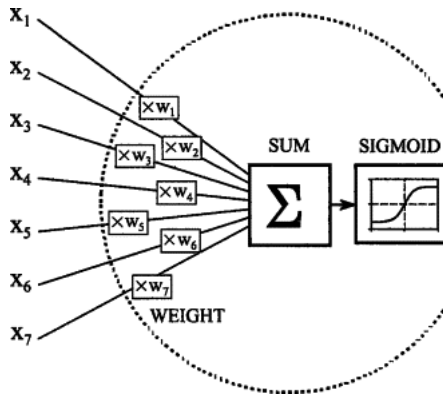


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

It couldn't *learn* like the brain.

x_1	x_2	$h_\Theta(x)$	Result
0	0	s(-10)	0
0	1	s(10)	1
1	0	s(10)	1
1	1	s(30)	1

$$S(x) = \frac{1}{1+e}$$

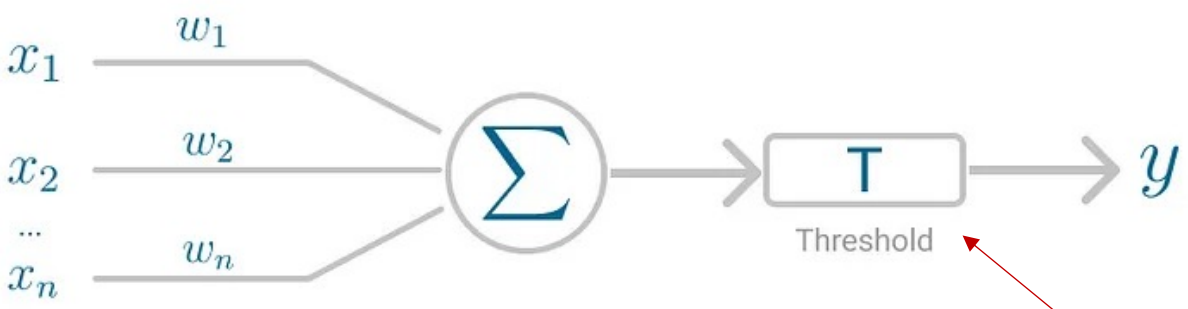


It was only a decade later that Frank Rosenblatt extended this model, and created an algorithm that could *learn* the weights in order to generate an output.

Perceptron: Simple Explanation

Frank Rosenblatt extended neuron model, and created an algorithm that could learn the weights in order to generate an output.
It gets its name from performing the human-like function of perception, seeing and recognizing images.

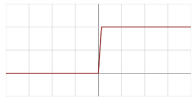
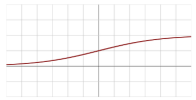
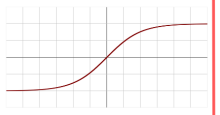
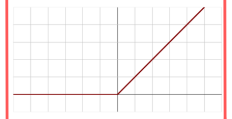
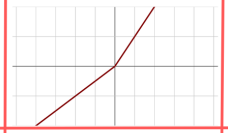
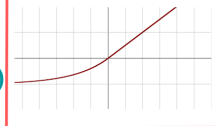
Perceptrons neuron model (left) and threshold logic (right).



$$y = \begin{cases} 1, \text{if } \overbrace{\sum_i w_i x_i}^{\text{weighted sum}} - \underbrace{T}_{\text{threshold}} > 0 \\ 0, \text{otherwise} \end{cases}$$

Activation function

Perception: Activation Fuctions

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		f(x) = x	f'(x) = 1	(-∞, ∞)
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	{0, 1}
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	(0, 1)
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	(-1, 1)
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	[0, ∞)
Softplus		$f(x) = \ln(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	(0, 1)
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	(-1, 1)
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	[0, ∞)

But you might be wondering, Doesn't Perceptron actually learn the weights?

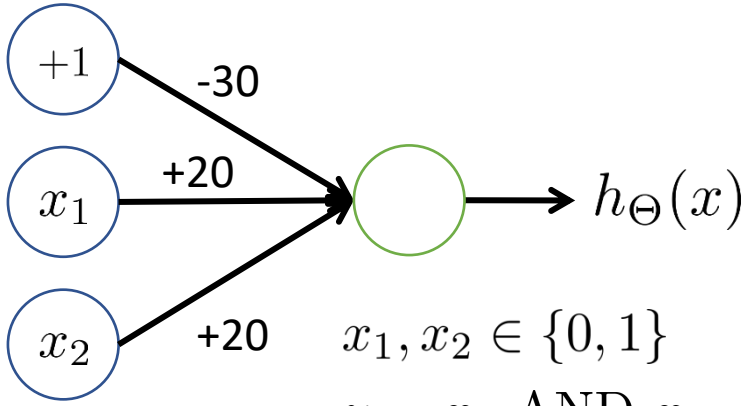
It does! Perceptron uses Stochastic Gradient Descent to find!

Perceptron: Simple Explanation

The first application of the neuron replicated a logic gate, where you have one or two binary inputs, and a boolean function that only gets activated given the right inputs and weights

AND FUNCTION

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

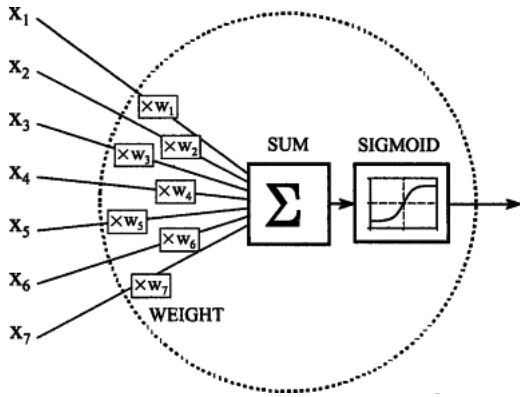


$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$

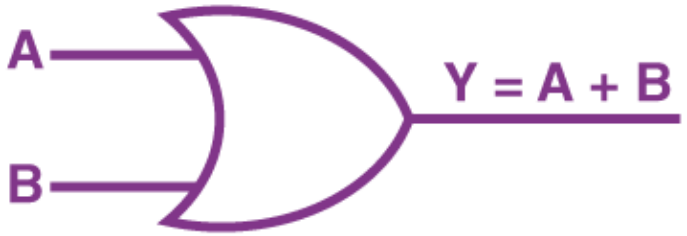
x_1	x_2	$h_\Theta(x)$	Result
0	0	s(-30)	0
0	1	s(-10)	0
1	0	s(-10)	0
1	1	s(10)	1

$$S(x) = \frac{1}{1 + e^{-x}}$$

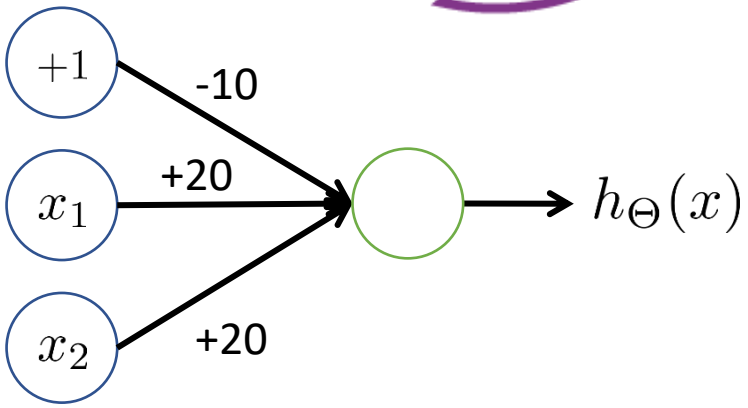


Perceptron: Simple Explanation

OR FUNCTION

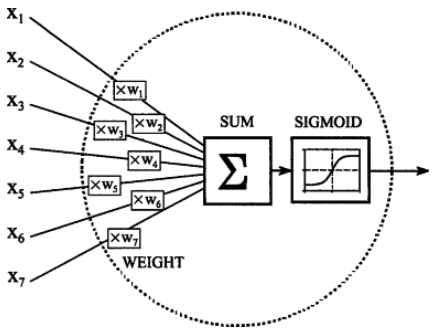


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



x_1	x_2	$h_\Theta(x)$	Result
0	0	s(-10)	0
0	1	s(10)	1
1	0	s(10)	1
1	1	s(30)	1

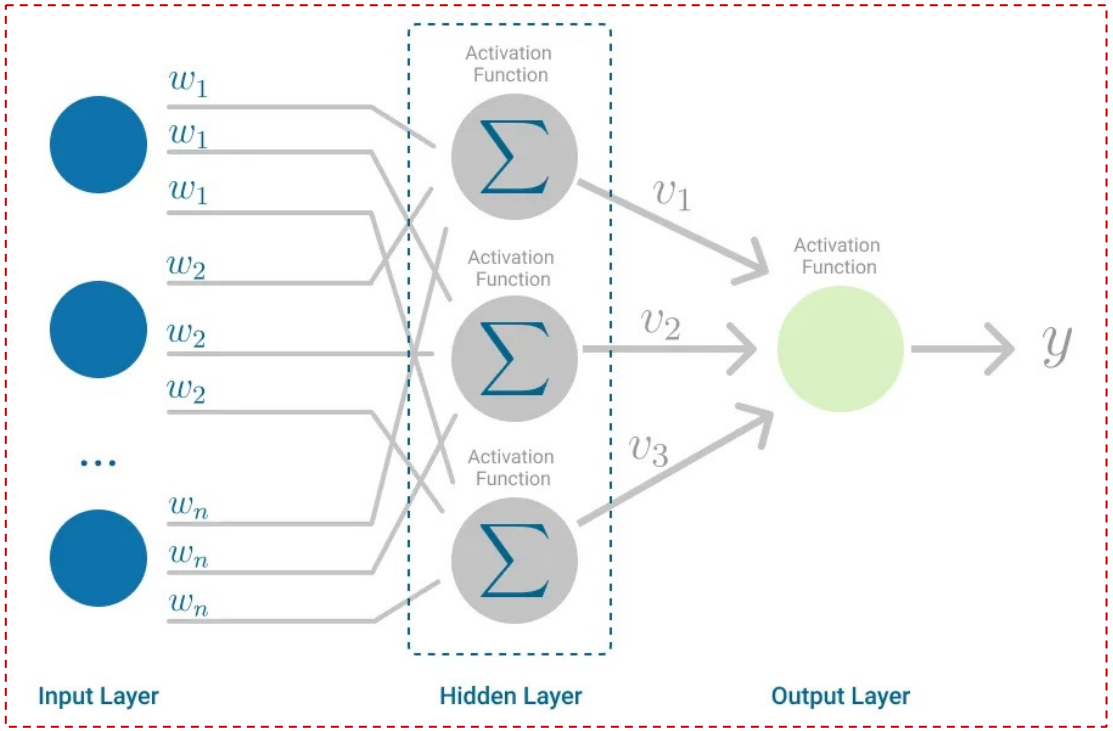
$$S(x) = \frac{1}{1+e^{-x}}$$



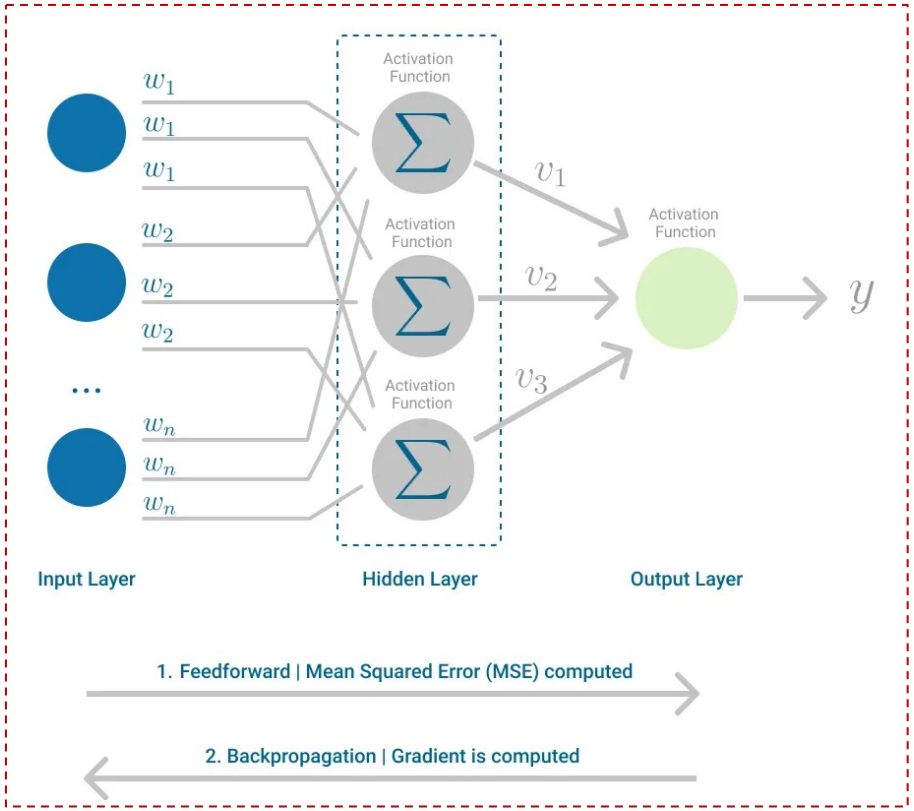
it couldn't represent the XOR gate, *exclusive OR*, where the gate only returns *1* if the inputs are different

Multilayer Perceptron: Simple Explanation

The **Multilayer Perceptron** was developed to tackle this limitation. It is a neural network where the mapping between inputs and output is non-linear.



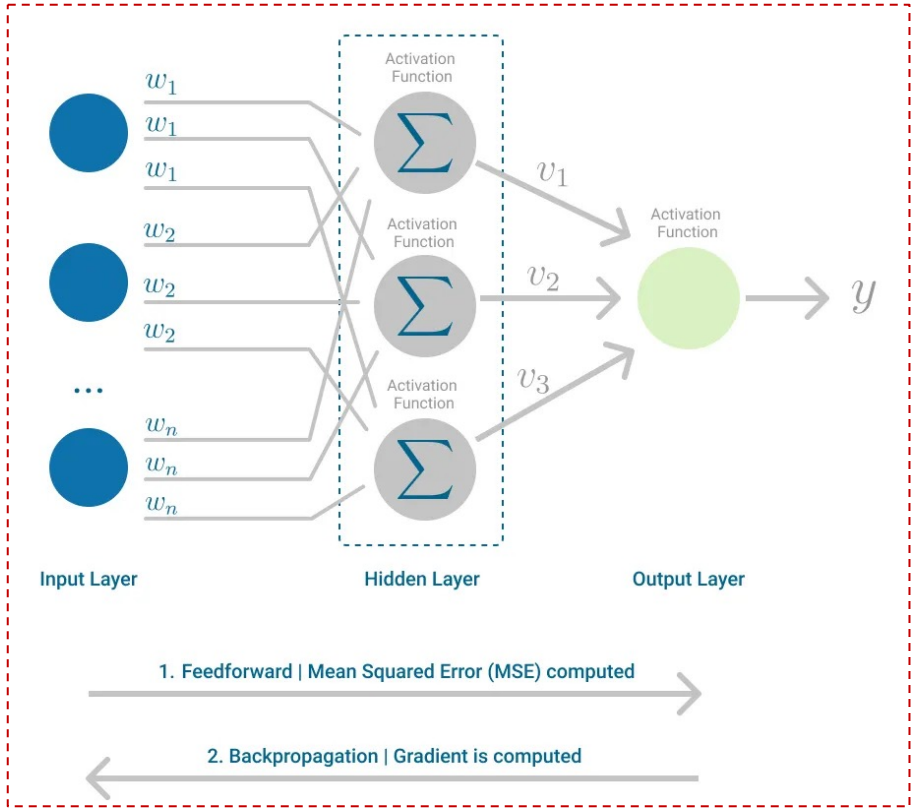
Multilayer Perceptron



the Feedfoward and Backpropagation steps

Multilayer Perceptron: Simple Explanation

The **Multilayer Perceptron** was developed to tackle this limitation. It is a neural network where the mapping between inputs and output is non-linear.

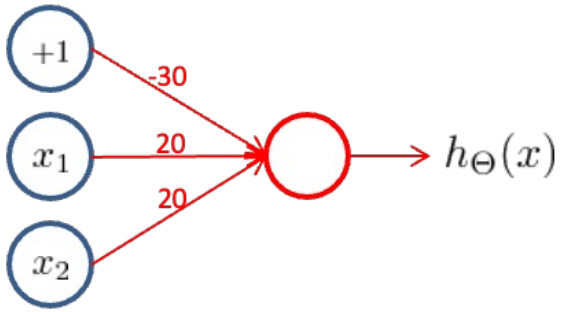


the Feedfoward and Backpropagation steps

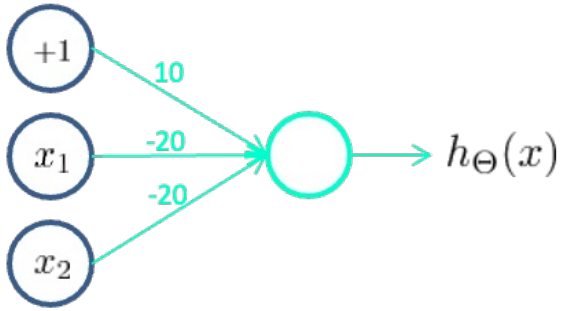
$$\underbrace{\Delta_w(t)}_{\text{Gradient Current Iteration}} = \overbrace{-}^{\text{Bias}}\varepsilon \frac{\overbrace{dE}^{\text{Error}}}{\underbrace{dw_{(t)}}_{\text{Weight vector}}} + \overbrace{\alpha}^{\text{Learning Rate}}\underbrace{\Delta_{w(t-1)}}_{\text{Gradient Previous Iteration}}$$

One iteration of Gradient Descent

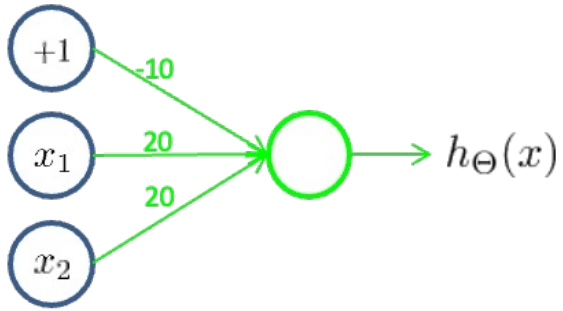
Multilayer Perceptron: Simple Explanation



x_1 AND x_2

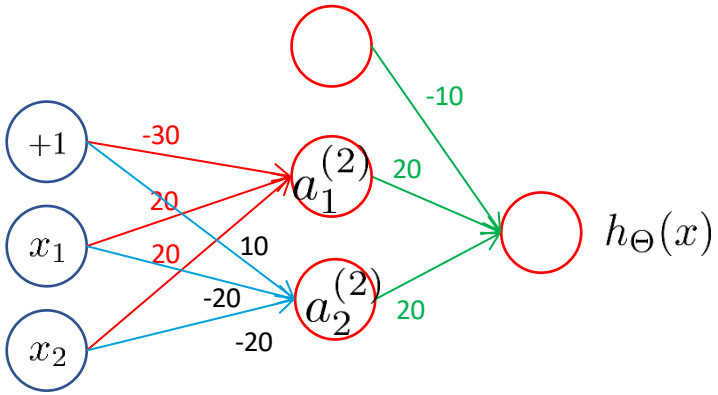


(NOT x_1) AND (NOT x_2)



x_1 OR x_2

x_1 XNOR x_2



x_1 AND x_2

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Multilayer Perceptron: Time-series Data Project

Electricity Transformer Dataset (ETDataset)

	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
0	2016-07-01 00:00:00	5.827	2.009	1.599	0.462	4.203	1.340	30.531000
1	2016-07-01 00:15:00	5.760	2.076	1.492	0.426	4.264	1.401	30.459999
2	2016-07-01 00:30:00	5.760	1.942	1.492	0.391	4.234	1.310	30.038000
3	2016-07-01 00:45:00	5.760	1.942	1.492	0.426	4.234	1.310	27.013000
4	2016-07-01 01:00:00	5.693	2.076	1.492	0.426	4.142	1.371	27.787001

Field	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
Description	The recorded **date**	High UseFul Load	High UseLess Load	Middle UseFul Load	Middle UseLess Load	Low UseFul Load	Low UseLess Load	Oil Temperature (target)

Multilayer Perceptron: Time-series Data Project

Electricity Transformer Dataset (ETDataset)

Data Windowing Techniques

Case 1 : Multivariate to Multivariate Prediction

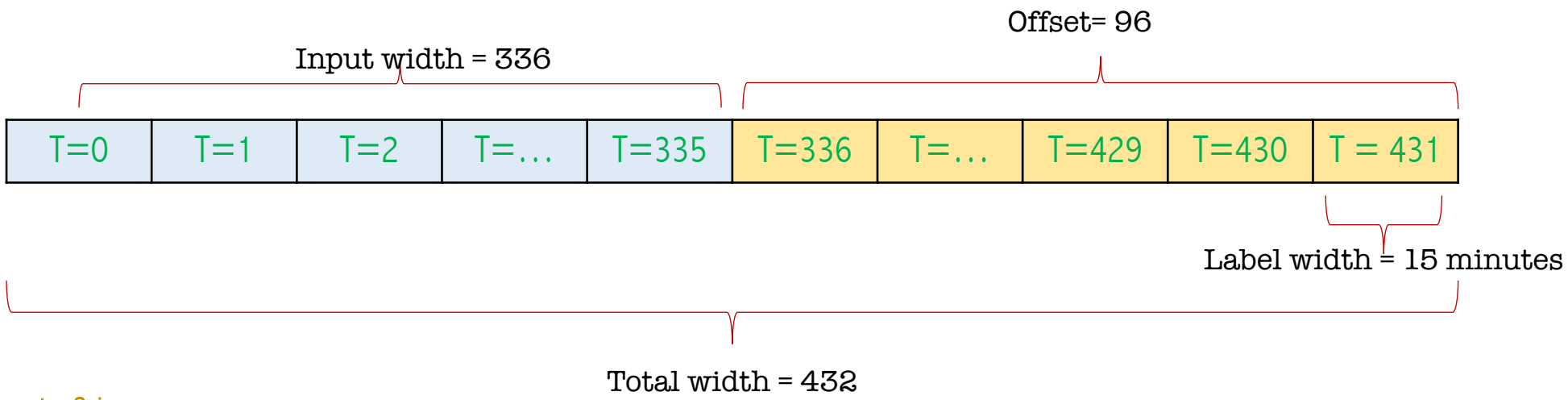
(Input, Output) = (336, 96)

(Input shape)= (15932, 336, 7)

(Output shape)= (15932, 96, 7)

15932

	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
0	2016-07-01 00:00:00	5.827	2.009	1.599	0.462	4.203	1.340	30.531000
1	2016-07-01 00:15:00	5.760	2.076	1.492	0.426	4.264	1.401	30.459999
2	2016-07-01 00:30:00	5.760	1.942	1.492	0.391	4.234	1.310	30.038000
3	2016-07-01 00:45:00	5.760	1.942	1.492	0.426	4.234	1.310	27.013000
4	2016-07-01 01:00:00	5.693	2.076	1.492	0.426	4.142	1.371	27.787001



Multilayer Perceptron: Time-series Data Project

Electricity Transformer Dataset (ETDataset)

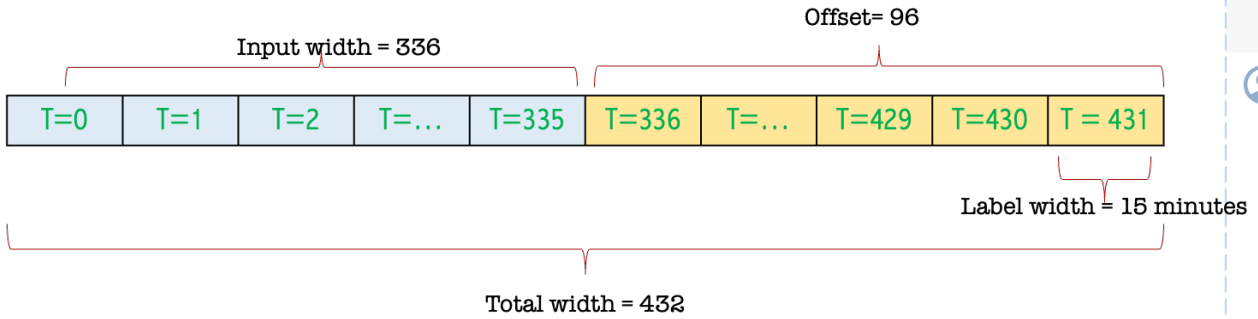
Data Windowing Techniques

Case 1 : Multivariate to Multivariate Prediction

(Input, Output) = (336, 96)

(Input shape)= (15932, 336, 7)

(Output shape)= (15932, 96, 7)



Multivariate-to-Multivartiate

```
features_type='M'
sub_dir = 'multi2multi'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2multi_loader = TimeSeriesDataLoader(file_path,
                                          input_size=input_size,
                                          label_size=label_size,
                                          offset=offset,
                                          train_size=train_size,
                                          val_size=val_size,
                                          target_name=target_name,
                                          features_type=features_type,
                                          date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 7)
self.X_val.shape = (1309, 336, 7)
self.y_val.shape = (1309, 96, 7)
self.X_test.shape = (2861, 336, 7)
self.y_test.shape = (2861, 96, 7)
```

Multilayer Perceptron: Time-series Data Project

Electricity Transformer Dataset (ETDataset)

Data Windowing Techniques

Case 2: Multivariate to Univariate Prediction

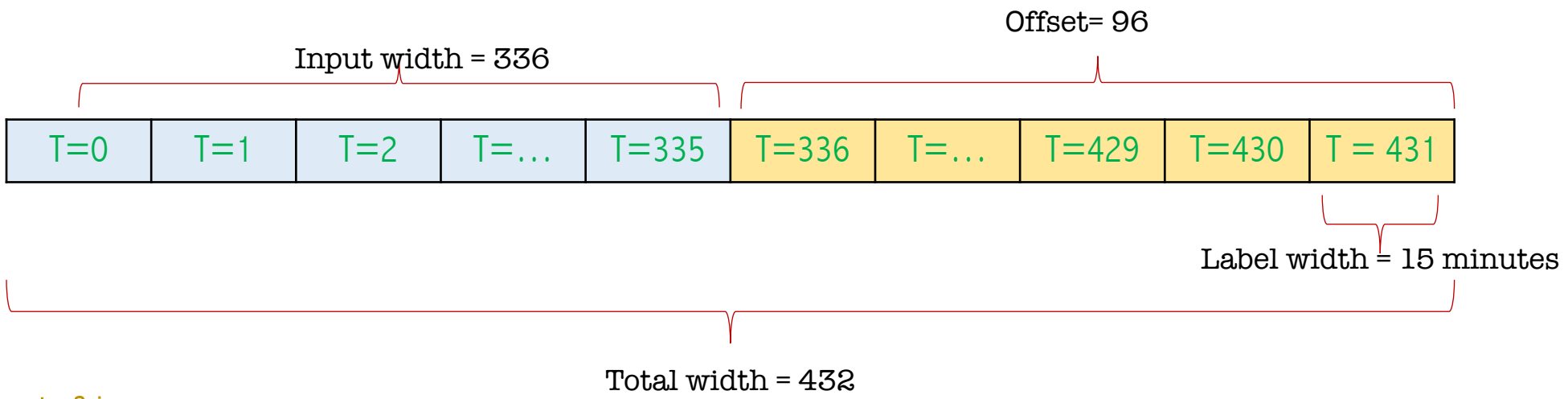
(Input, Output) = (336, 96)

(Input shape)= (15932, 336, 7)

(Output shape)= (15932, 96, 1)

15932

	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
0	2016-07-01 00:00:00	5.827	2.009	1.599	0.462	4.203	1.340	30.531000
1	2016-07-01 00:15:00	5.760	2.076	1.492	0.426	4.264	1.401	30.459999
2	2016-07-01 00:30:00	5.760	1.942	1.492	0.391	4.234	1.310	30.038000
3	2016-07-01 00:45:00	5.760	1.942	1.492	0.426	4.234	1.310	27.013000
4	2016-07-01 01:00:00	5.693	2.076	1.492	0.426	4.142	1.371	27.787001



Multilayer Perceptron: Time-series Data Project

Electricity Transformer Dataset (ETDataset)

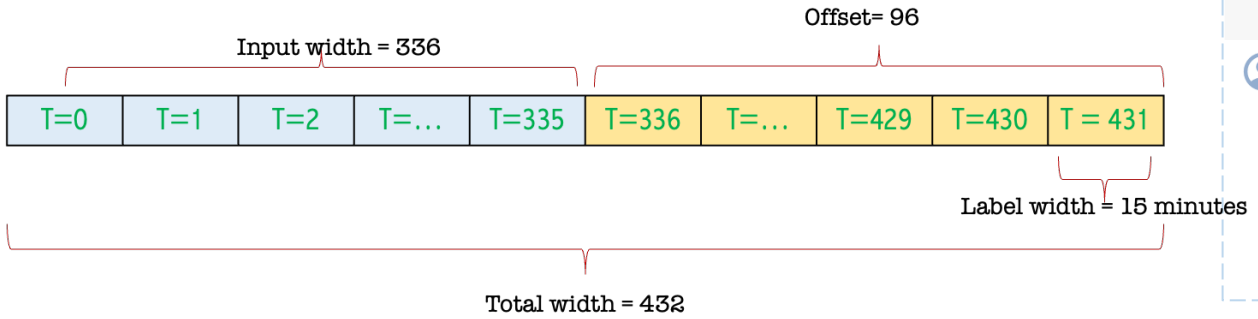
Data Windowing Techniques

Case 2: Multivariate to Univariate Prediction

(Input, Output) = (336, 96)

(Input shape)= (15932, 336, 7)

(Output shape)= (15932, 96, 1)



Multivariate-to-Univartiate

```
features_type='MS'
sub_dir = 'multi2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2uni_loader = TimeSeriesDataLoader(file_path,
                                        input_size=input_size,
                                        label_size=label_size,
                                        offset=offset,
                                        train_size=train_size,
                                        val_size=val_size,
                                        target_name=target_name,
                                        features_type=features_type,
                                        date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 7)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 7)
self.y_test.shape = (2861, 96, 1)
```

Multilayer Perceptron: Time-series Data Project

Electricity Transformer Dataset (ETDataset)

Data Windowing Techniques

Case 3: Univariate to Univariate Prediction

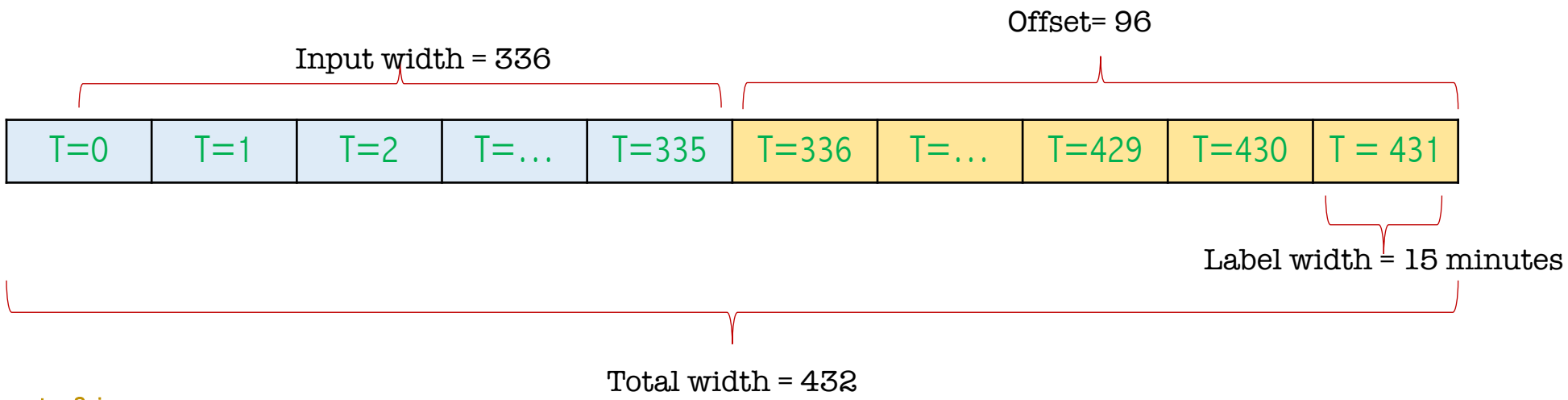
(Input, Output) = (336, 96)

(Input shape)= (15932, 336, 1)

(Output shape)= (15932, 96, 1)

15932

	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
0	2016-07-01 00:00:00	5.827	2.009	1.599	0.462	4.203	1.340	30.531000
1	2016-07-01 00:15:00	5.760	2.076	1.492	0.426	4.264	1.401	30.459999
2	2016-07-01 00:30:00	5.760	1.942	1.492	0.391	4.234	1.310	30.038000
3	2016-07-01 00:45:00	5.760	1.942	1.492	0.426	4.234	1.310	27.013000
4	2016-07-01 01:00:00	5.693	2.076	1.492	0.426	4.142	1.371	27.787001



Multilayer Perceptron: Time-series Data Project

Electricity Transformer Dataset (ETDataset)

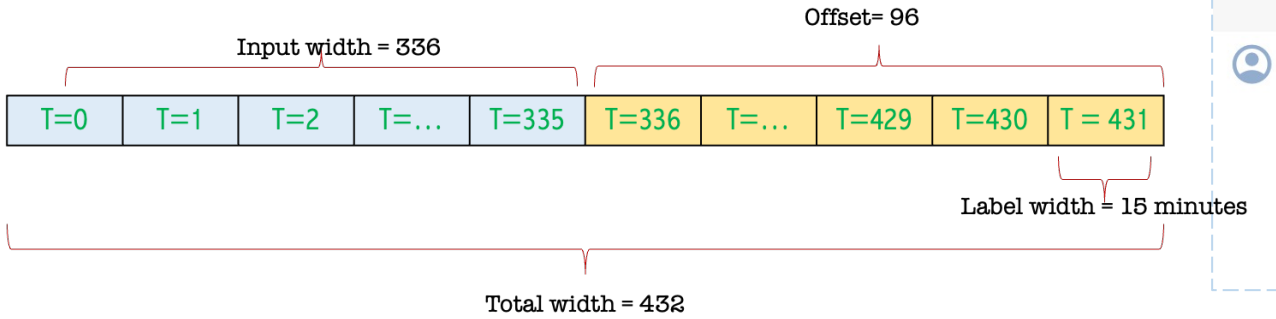
Data Windowing Techniques

Case 3: Univariate to Univariate Prediction

(Input, Output) = (336, 96)

(Input shape)= (15932, 336, 1)

(Output shape)= (15932, 96, 1)



Univariate-to-Univartiate

```
features_type='S'
sub_dir = 'uni2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
uni2uni_loader = TimeSeriesDataLoader(file_path,
                                      input_size=input_size,
                                      label_size=label_size,
                                      offset=offset,
                                      train_size=train_size,
                                      val_size=val_size,
                                      target_name=target_name,
                                      features_type=features_type,
                                      date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 1)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 1)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 1)
self.y_test.shape = (2861, 96, 1)
```

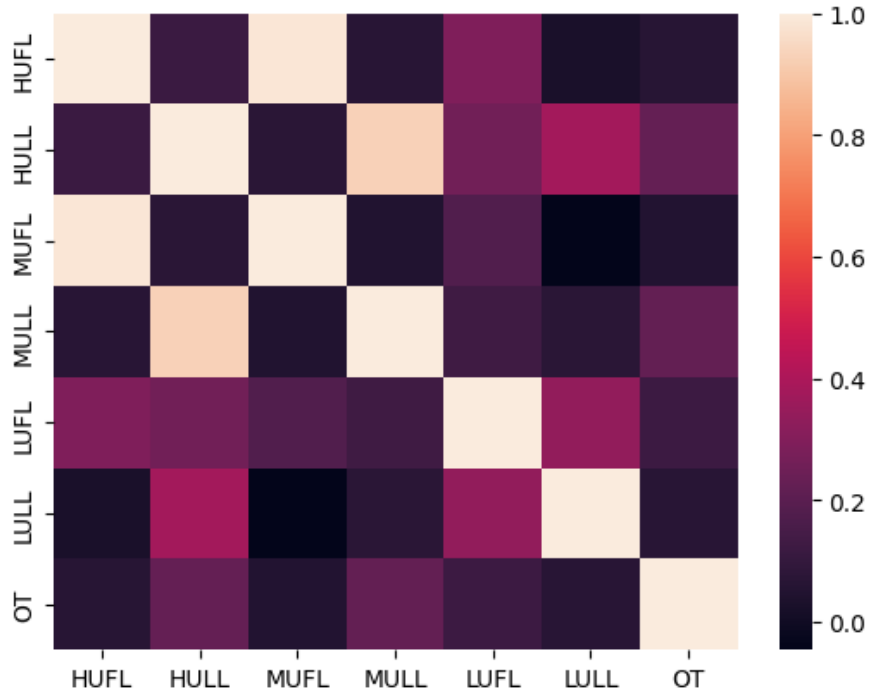
Time-series Data Project: Data Exploration

```
[ ] train_df = pl.read_csv(file_path)
```

```
train_df.describe()
```

shape: (7, 9)

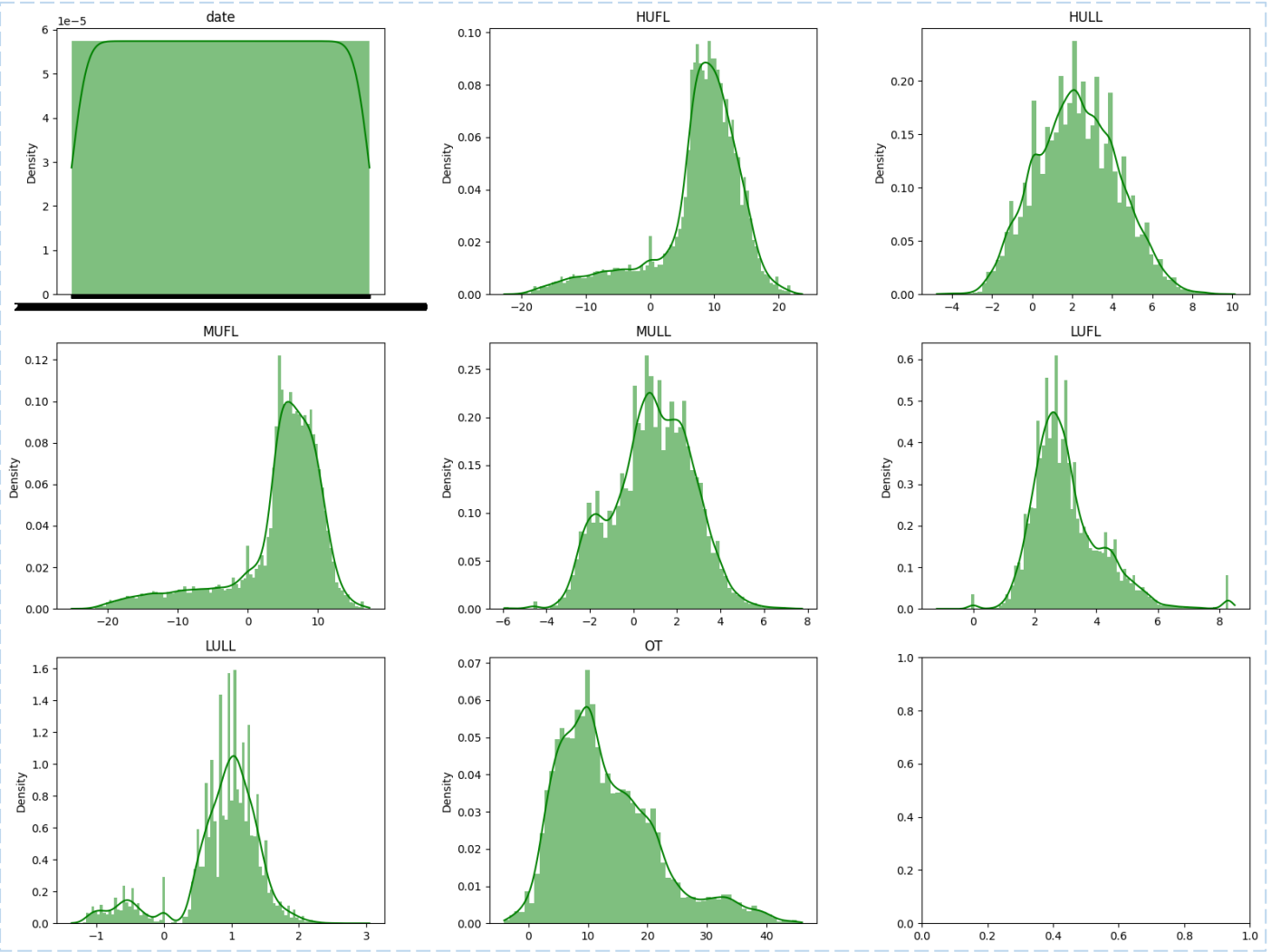
describe	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
str	str	f64	f64	f64	f64	f64	f64	f64
"count"	"17420"	17420.0	17420.0	17420.0	17420.0	17420.0	17420.0	17420.0
"null_count"	"0"	0.0	0.0	0.0	0.0	0.0	0.0	0.0
"mean"	null	7.375141	2.242242	4.300239	0.881568	3.066062	0.856932	13.324672
"std"	null	7.067744	2.042342	6.826978	1.809293	1.164506	0.599552	8.566946
"min"	"2016-07-01 00:…	-22.705999	-4.756	-25.087999	-5.934	-1.188	-1.371	-4.08
"max"	"2018-06-26 19:…	23.643999	10.114	17.341	7.747	8.498	3.046	46.007
"median"	null	8.774	2.21	5.97	0.959	2.833	0.975	11.396




```
train = train_df.to_pandas()
pearson_corr = train.corr(method="pearson")
sns.heatmap(pearson_corr)
```


Experiments were perfomed by TA Khoa Nguyen

Time-series Data Project: Data Exploration




```
columns_to_plot = train_df.columns

num_columns = 3
num_rows = int(np.ceil(len(columns_to_plot) / num_columns))
grid_layout = (num_rows, num_columns)

fig, axes = plt.subplots(*grid_layout, figsize=(16, 12))

axes = axes.flatten()

for i, column in enumerate(columns_to_plot):
    sns.histplot(train_df[column], kde=True, ax=axes[i],
                 color='green', stat="density", linewidth=0)
    axes[i].set_title(column)
plt.tight_layout()
plt.show()
```


Experiments were perfomed by TA Khoa Nguyen

Multilayer Perceptron: Time-series Data Project

Define MLP Structure in Pytorch

```python
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, ahead):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size * ahead)  # Adjust for output sequence length
        self.ahead = ahead
        self.output_size = output_size

    def forward(self, x):
        # Flatten the input
        x = x.view(x.size(0), -1)  # Reshape from [batch, lag, features] to [batch, lag * features]
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x.view(-1, self.ahead, self.output_size)  # Reshape to [batch, ahead, features]
```

```python
# Determine in_variable and out_variable based on features_type
if features_type == 'S':
    self.in_variable = 1
    self.out_variable = 1
elif features_type == 'M':
    self.in_variable = len(self.df.columns)
    self.out_variable = len(self.df.columns)
elif features_type == 'MS':
    self.in_variable = len(self.df.columns)
    self.out_variable = 1
else:
    raise ValueError("Invalid features_type. Choose from 'S' for Univariate-to-Univariate,
```

Configuration

```python
input_size = 336
label_size = 96
offset = 1
train_size = 0.7
val_size = 0.1
num_epochs = 1_000_000
patience = 20
learning_rate = 0.001
hidden_size = 64
num_layers = 2
ele = 5
target_name = 'OT'
date_column = 'date'
file_path = 'ETTh1.csv'
plot_dir = 'plots'
weight_dir = 'weights'
results = []
```

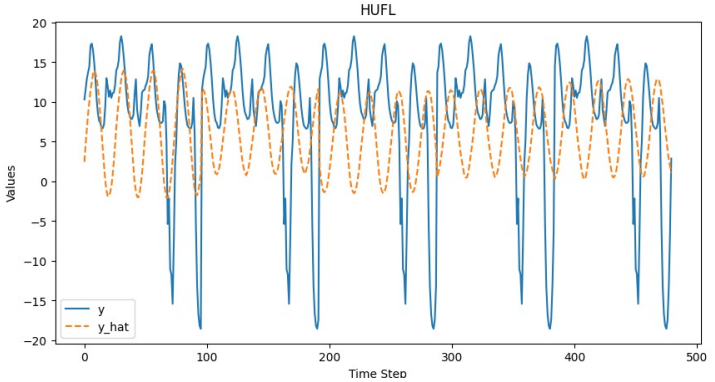
Multilayer Perceptron: Time-series Data Project

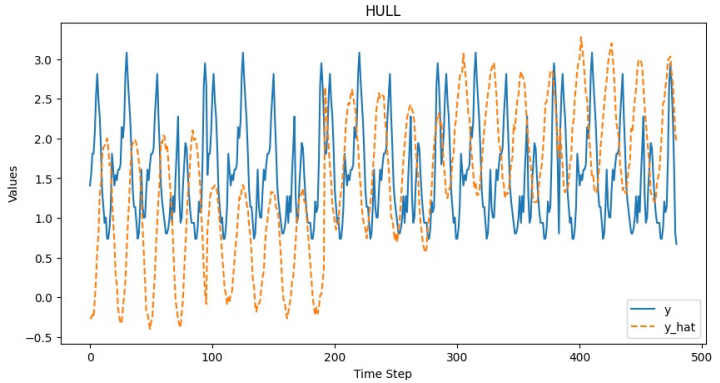
Define MLP Structure in Pytorch

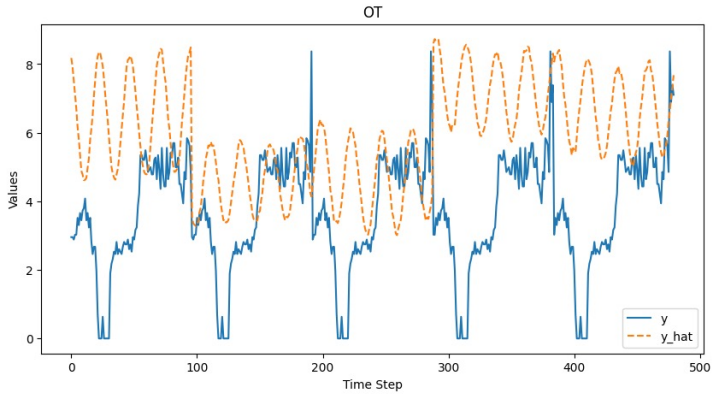
```
[ ]  MLP_multi2multi = MLP(input_size=multi2multi_loader.in_variable*input_size,
                           hidden_size=hidden_size,
                           output_size=multi2multi_loader.out_variable,
                           ahead=label_size)
     MLP_multi2multi_manager = ModelManager(model=MLP_multi2multi,
                                            train_loader=multi2multi_loader.train_loader,
                                            val_loader=multi2multi_loader.val_loader,
                                            lr=learning_rate,
                                            patience=patience)
     MLP_multi2multi_manager.train(num_epochs=num_epochs,
                                   save_dir=os.path.join(weight_dir, sub_dir))
     results.append({
         "Name": MLP_multi2multi_manager.model.__class__.__name__,
         "Type": sub_dir,
         "MAE": MLP_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
     })
     results[-1]

     Model saved to weights\multi2multi\best-MLP.pth
     Epoch [1/1000000], time: 0s, loss: 3.0666, val_loss: 3.6309
     Model saved to weights\multi2multi\best-MLP.pth
     Epoch [2/1000000], time: 0s, loss: 2.3158, val_loss: 3.4847
```

Case 1: Multivariate to Multivariate Prediction







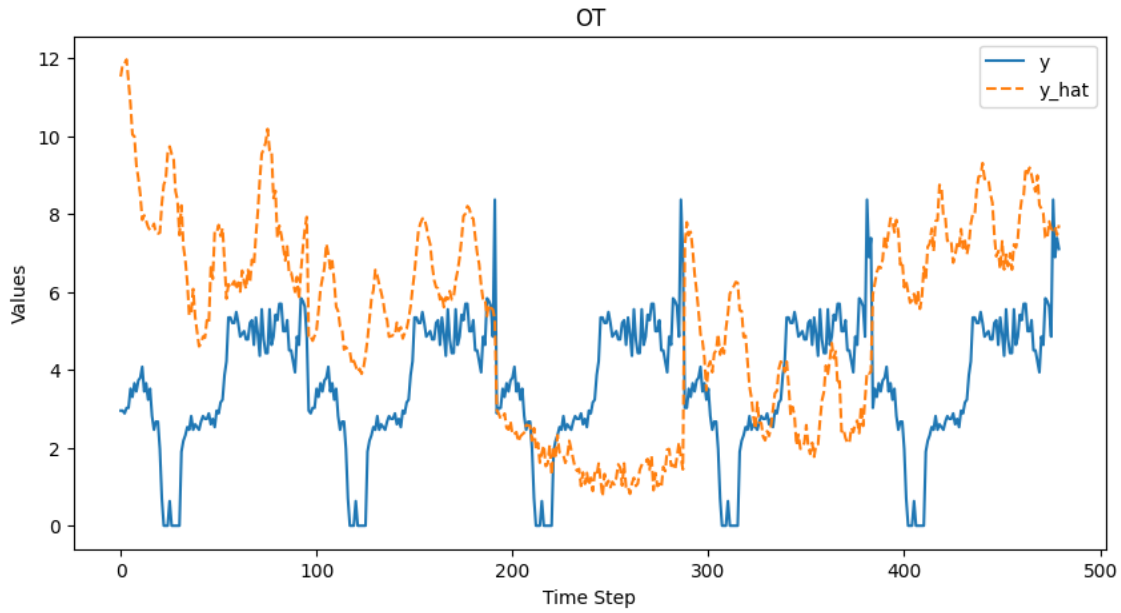
Experiments were perfomed by TA Khoa Nguyen

Multilayer Perceptron: Time-series Data Project

Define MLP Structure in Pytorch

```
features_type='MS'
sub_dir = 'multi2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2uni_loader = TimeSeriesDataLoader(file_path,
                                        input_size=input_size,
                                        label_size=label_size,
                                        offset=offset,
                                        train_size=train_size,
                                        val_size=val_size,
                                        target_name=target_name,
                                        features_type=features_type,
                                        date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 7)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 7)
self.y_test.shape = (2861, 96, 1)
```



Case 2: Multivariate to Univariate Prediction

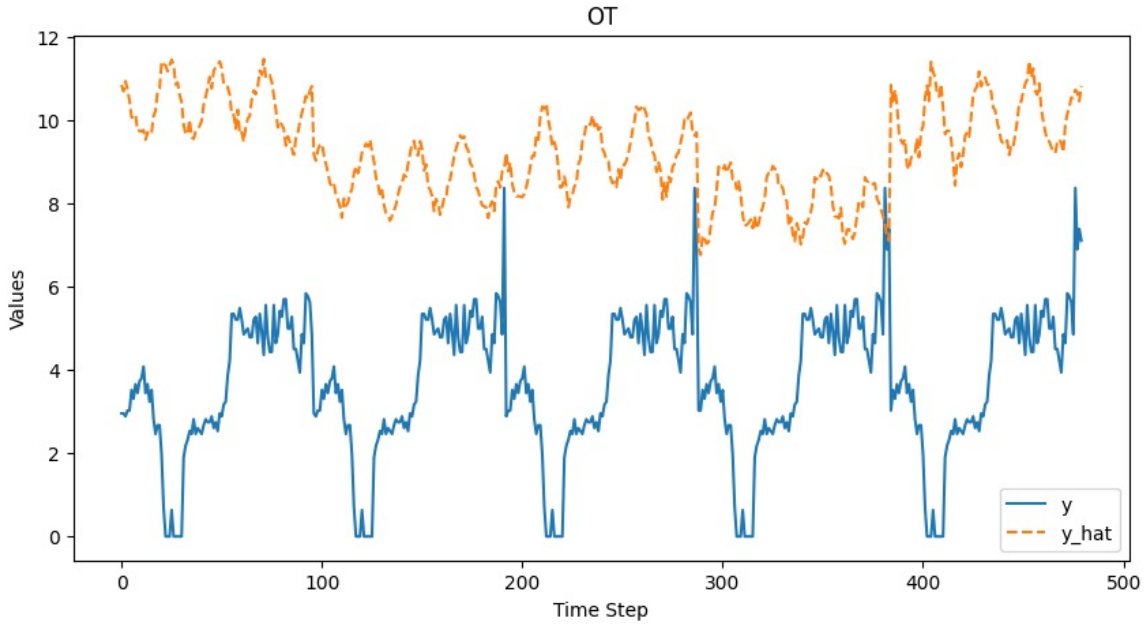
Experiments were perfomed by TA Khoa Nguyen

Multilayer Perceptron: Time-series Data Project

Define MLP Structure in Pytorch

```
features_type='S'
sub_dir = 'uni2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
uni2uni_loader = TimeSeriesDataLoader(file_path,
                                      input_size=input_size,
                                      label_size=label_size,
                                      offset=offset,
                                      train_size=train_size,
                                      val_size=val_size,
                                      target_name=target_name,
                                      features_type=features_type,
                                      date_column=date_column)
```

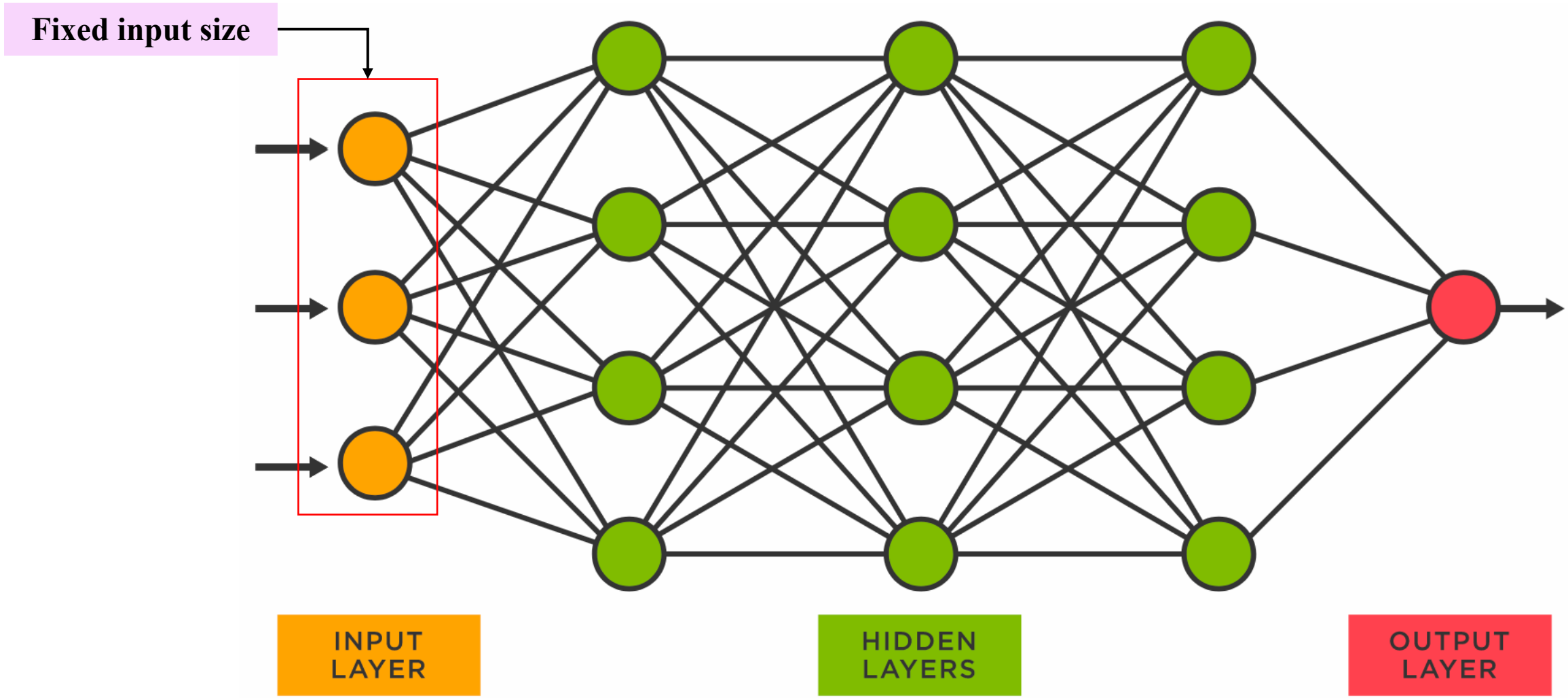
```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 1)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 1)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 1)
self.y_test.shape = (2861, 96, 1)
```



Case 3: Univariate to Univariate Prediction

Experiments were perfomed by TA Khoa Nguyen

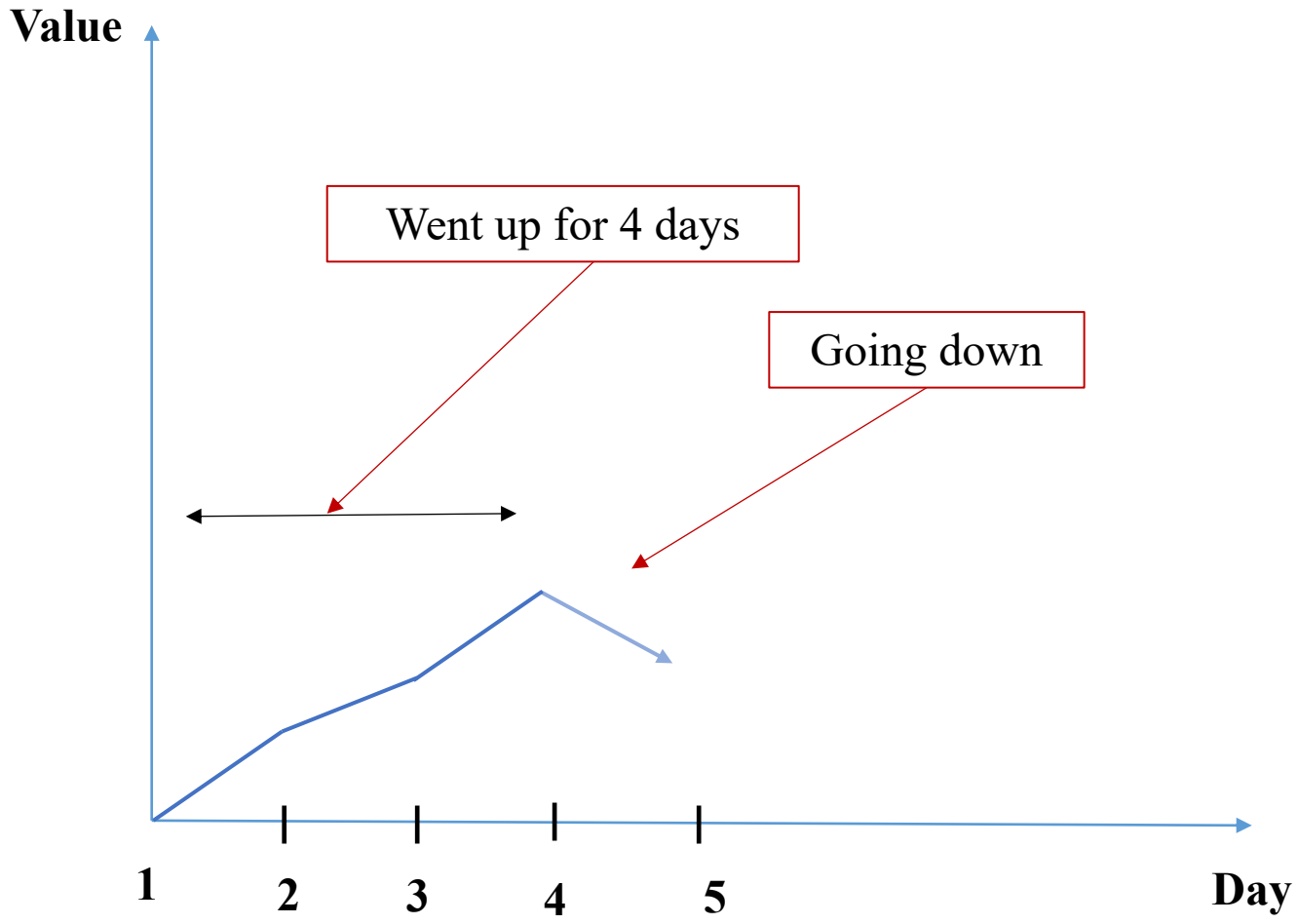
Neural Network : Limitations



Outline

- Multilayer Perceptron and Time-series Data Forecasting
- RNN and Time-series Data Forecasting
- LSTM and Time-series Data Forecasting
- Bi-LSTM and Time-series Data Forecasting
- XGBoost and Time-series Data Forecasting

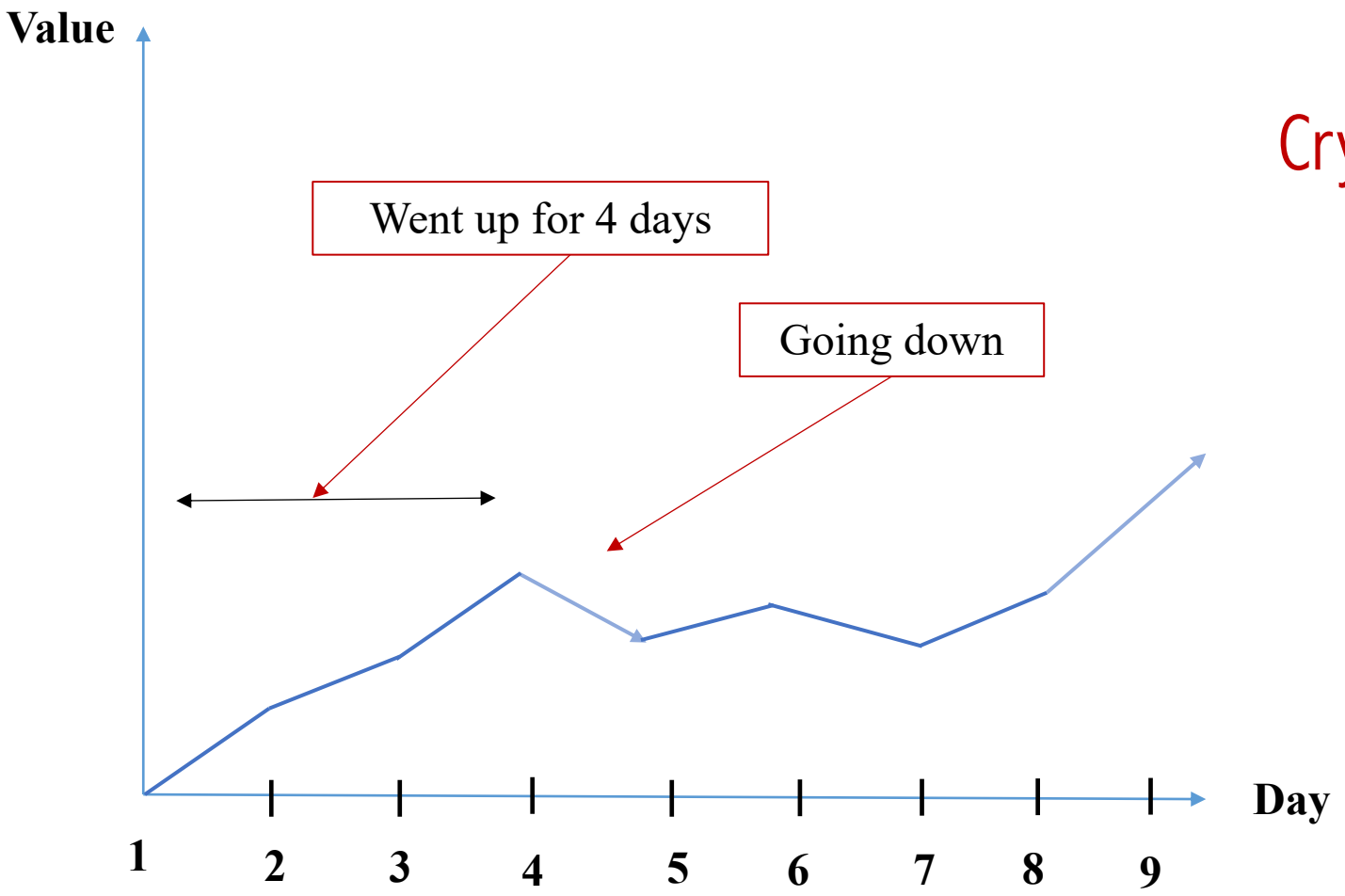
Recurrent Neural Network : Simple Explanation



Cryptocurrency Prediction

BTC-USD

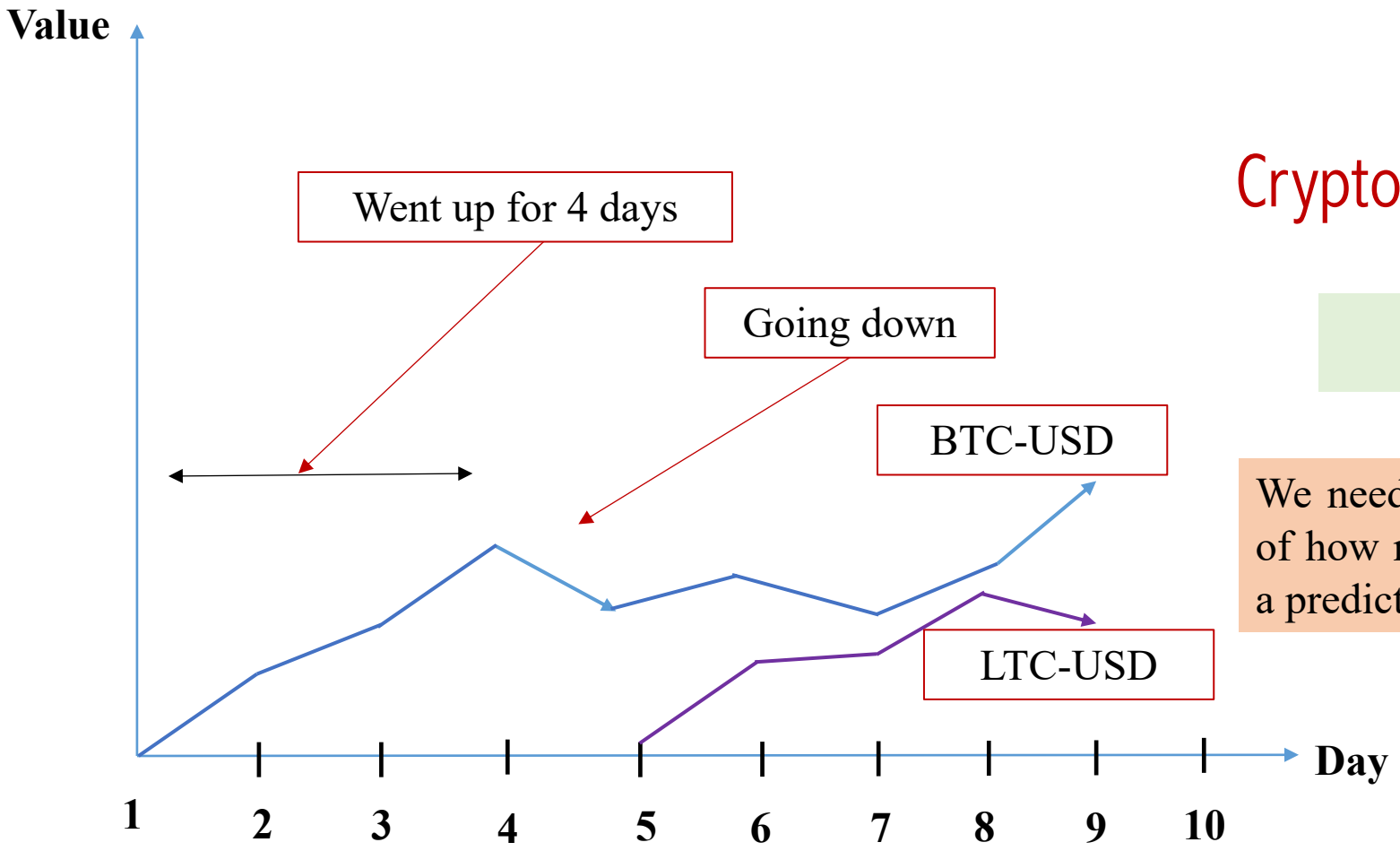
Recurrent Neural Network : Simple Explanation



Cryptocurrency Prediction

BTC-USD

Recurrent Neural Network : Simple Explanation



Cryptocurrency Prediction

BTC-USD

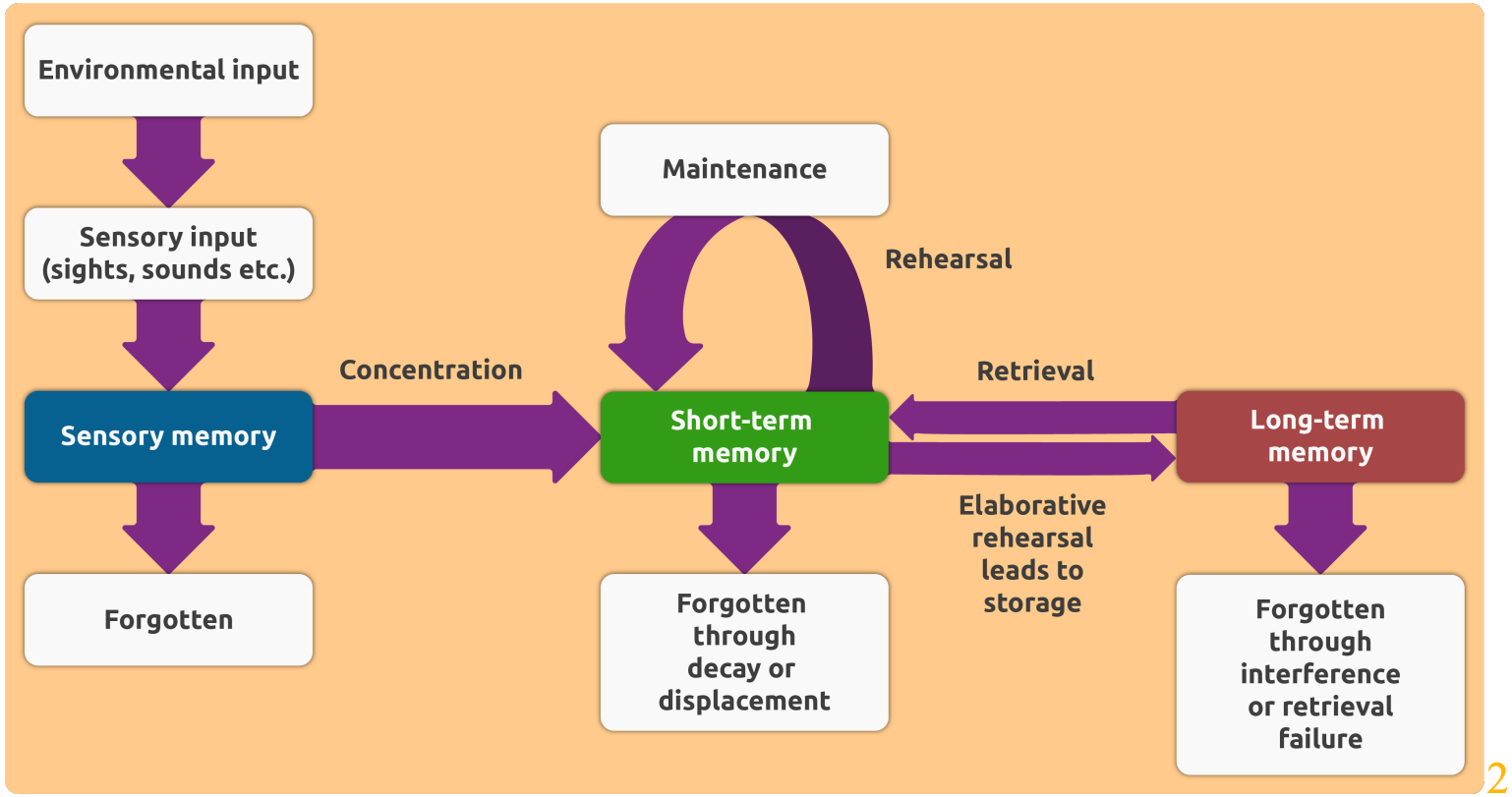
We need the NN to be more flexible in term of how much sequential data we use to make a prediction

Predict LTC's price, we need data for 5 days (5,6,7,8,9)

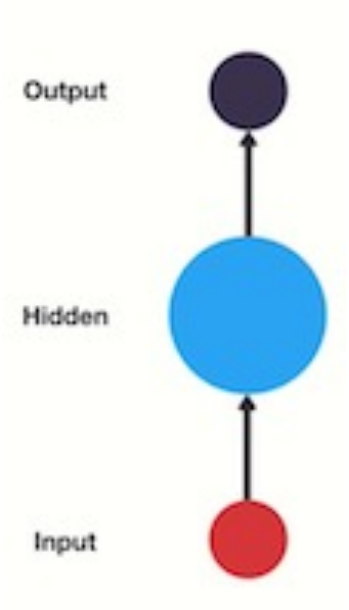
Predict BTC's price, we need data for 9 days (1,2,3,4,5,6,7,8,9)

Recurrent Neural Network : Simple Explanation

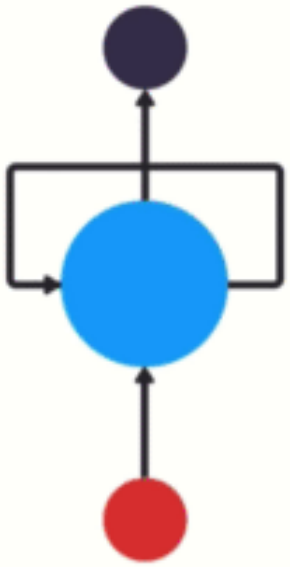
Humans tend to retrieve information from memory, short or long, use current information with it and derive logic to take next action (or impulse/habit, again based on previous experiences).



Recurrent Neural Network : Simple Explanation



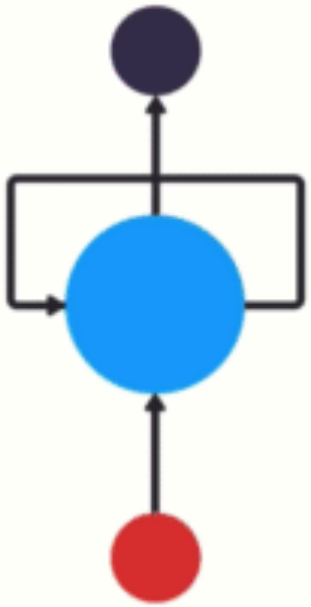
NN



RNN

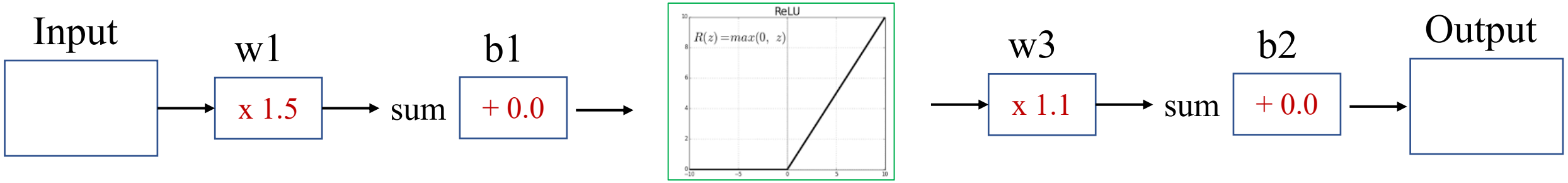
RNN

Recurrent Neural Network : Simple Explanation

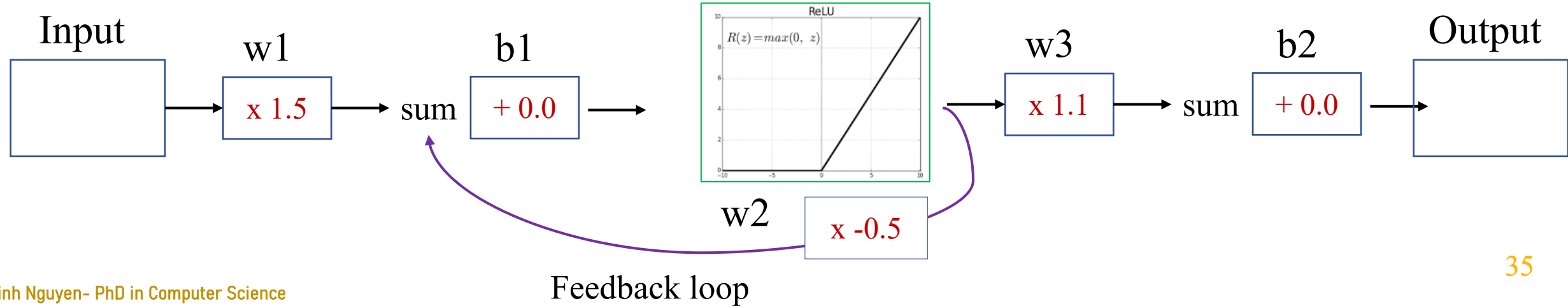


Recurrent Neural Network : Simple Explanation

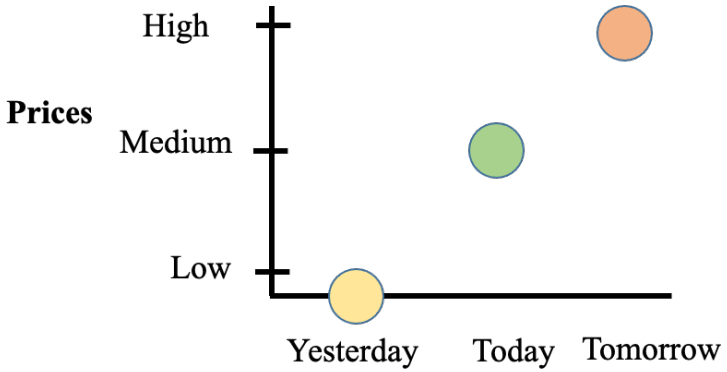
Traditional Neural Network

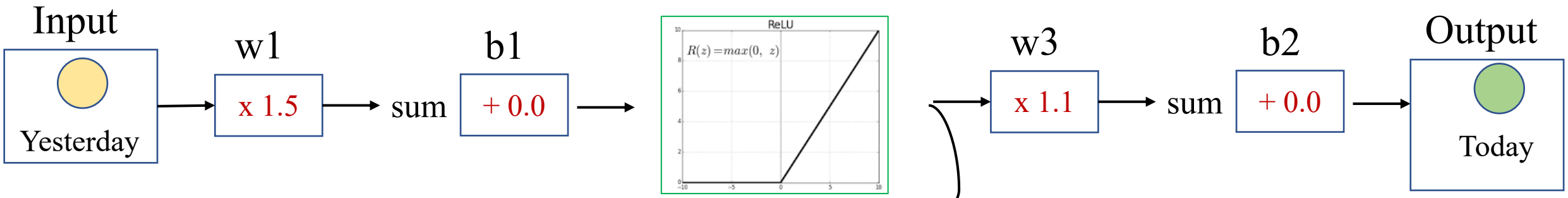


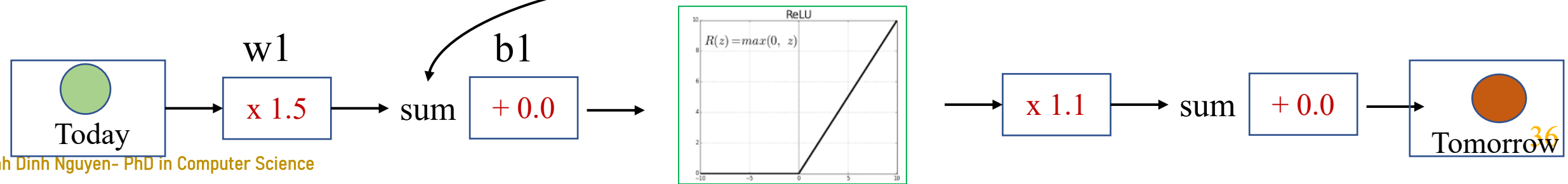
Recurrent Neural Network for sequential input values



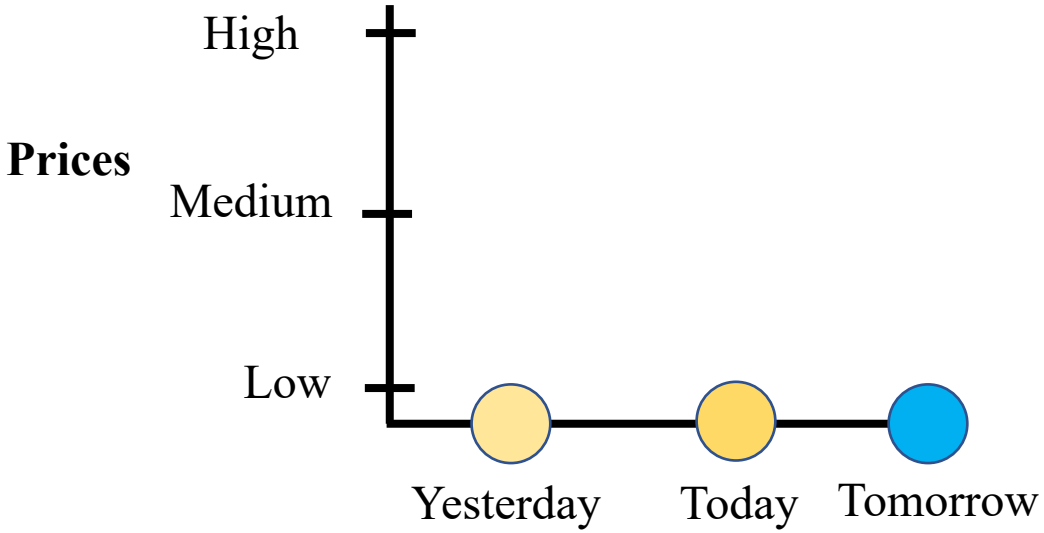
Recurrent Neural Network : Simple Explanation

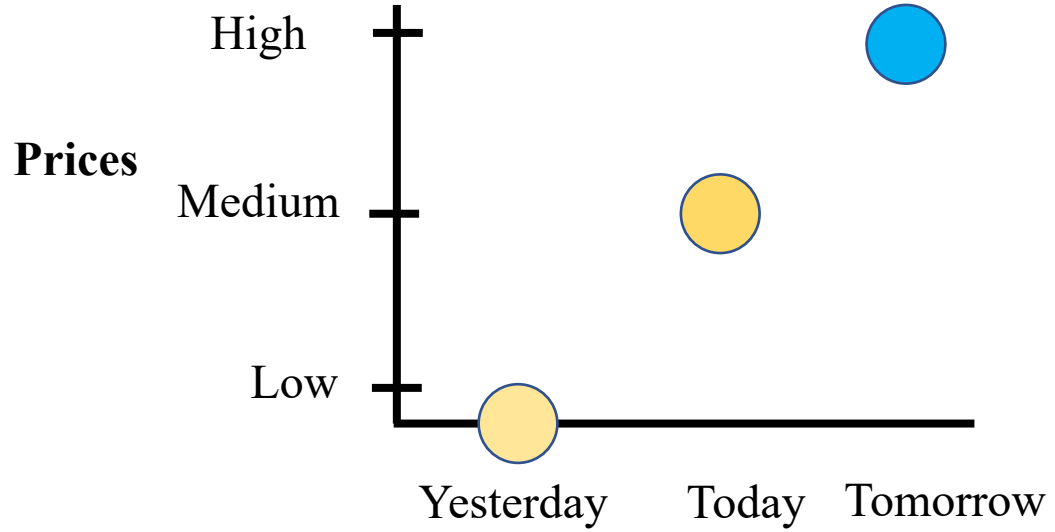


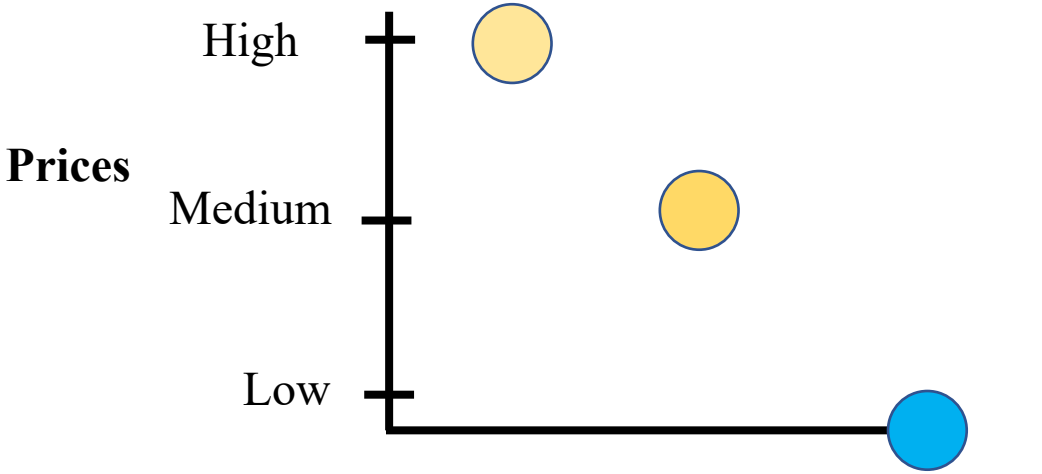


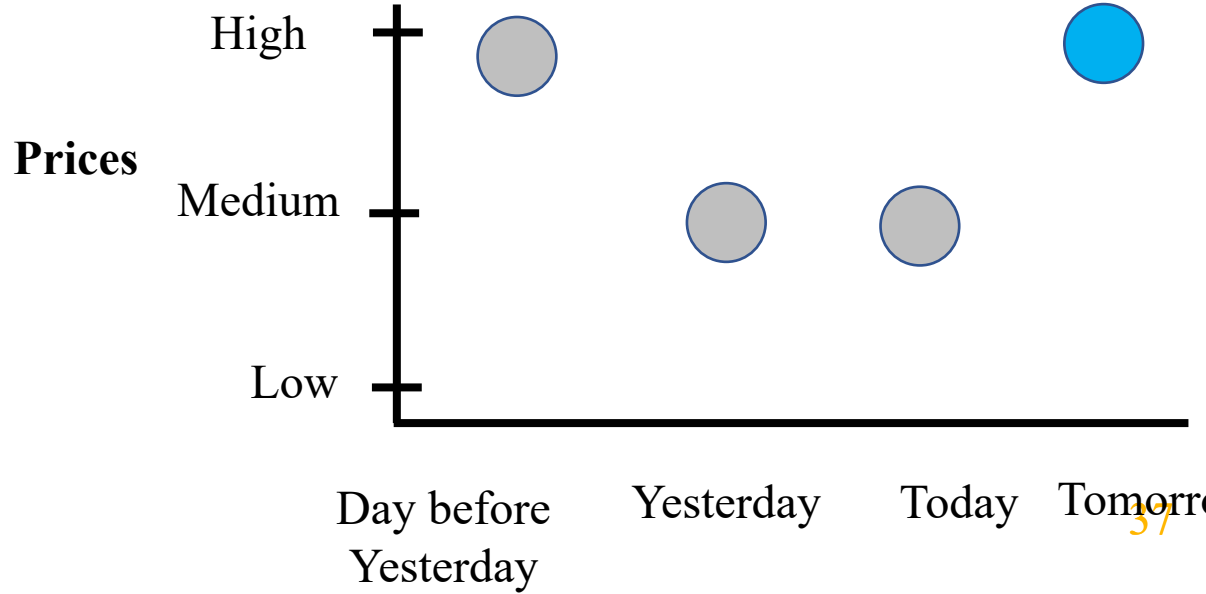


Recurrent Neural Network : Simple Explanation

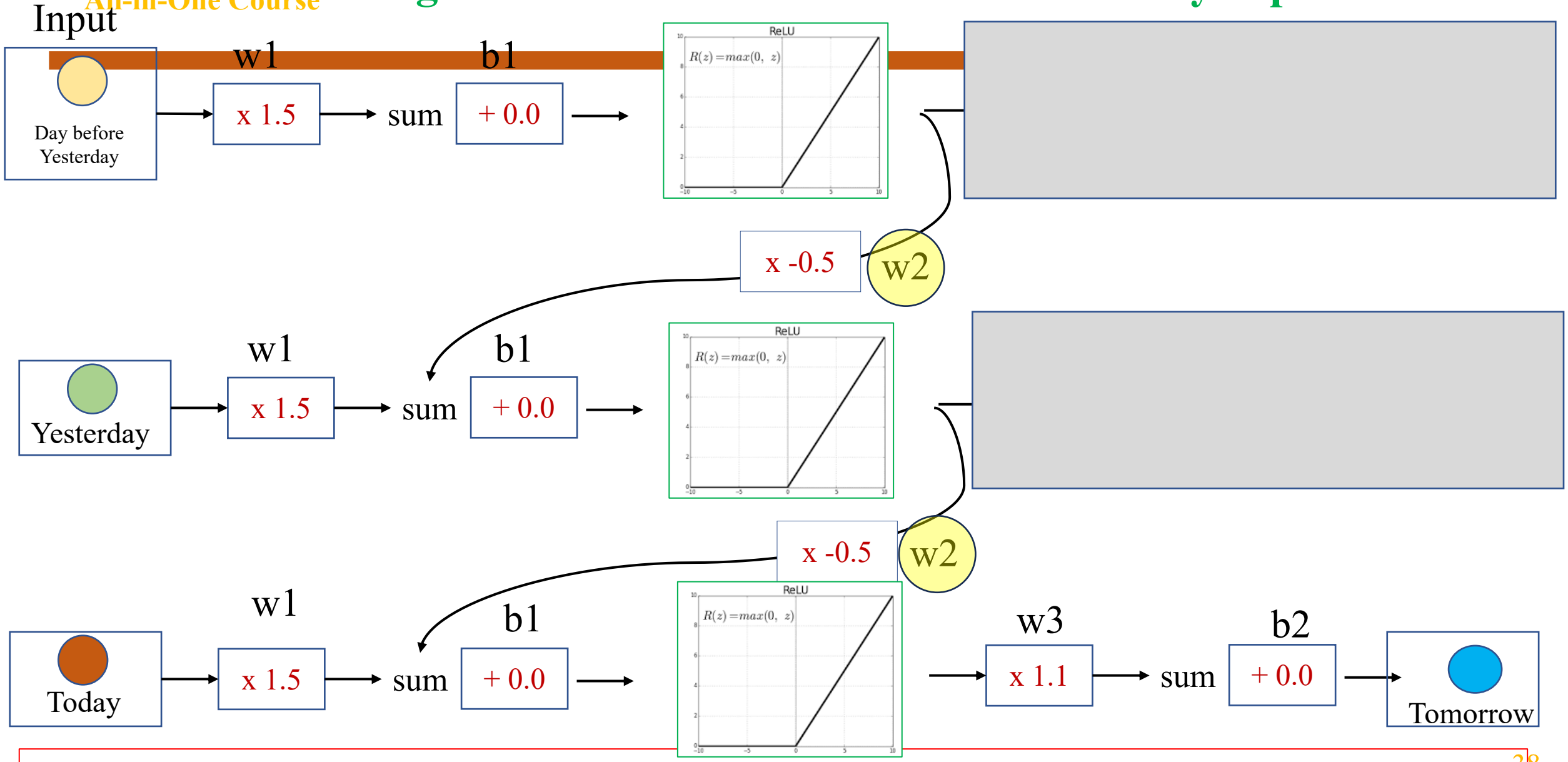








Weight and Biases Are Shared Across Every Input



No matter how many times we unroll a recurrent neural network, we never increase number of weights and bias to train

RNN: Time-series Data Project

Define RNN Structure in Pytorch

```python
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers, ahead):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.ahead = ahead
        self.output_size = output_size

        # RNN Layer - can be replaced with nn.LSTM or nn.GRU
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)

        # Output layer
        self.fc = nn.Linear(hidden_size, output_size * ahead)

    def forward(self, x):
        # Initialize hidden state
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        # Forward propagate RNN
        out, _ = self.rnn(x, h0)

        # Get the last time step's output for each sequence
        out = out[:, -1, :]

        # Pass through the linear layer and reshape
        out = self.fc(out).view(-1, self.ahead, self.output_size)
        return out
```

Configuration

```python
input_size = 336
label_size = 96
offset = 1
train_size = 0.7
val_size = 0.1
num_epochs = 1_000_000
patience = 20
learning_rate = 0.001
hidden_size = 64
num_layers = 2
ele = 5
target_name = 'OT'
date_column = 'date'
file_path = 'ETTh1.csv'
plot_dir = 'plots'
weight_dir = 'weights'
results = []
```

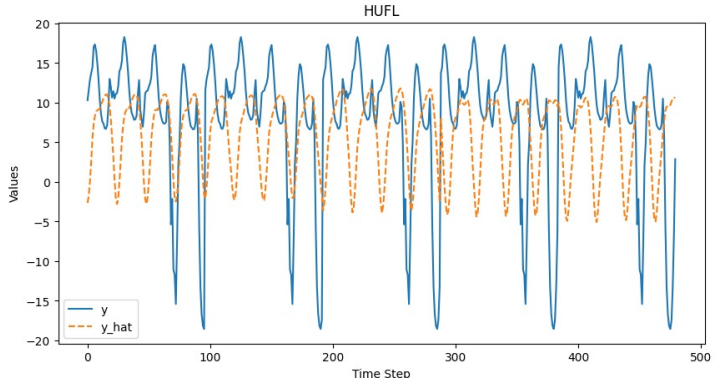
RNN: Time-series Data Project

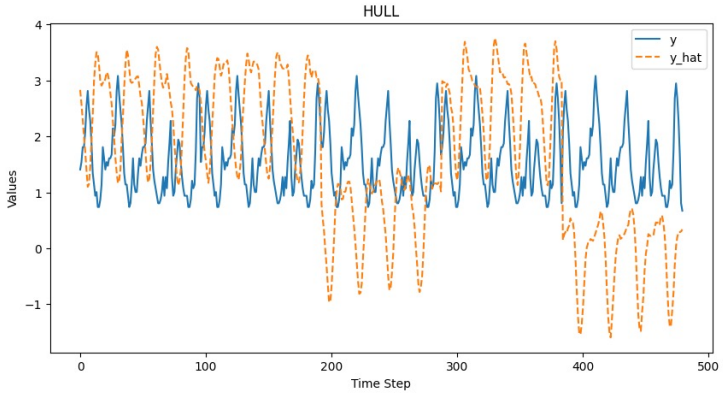
Define RNN Structure in Pytorch

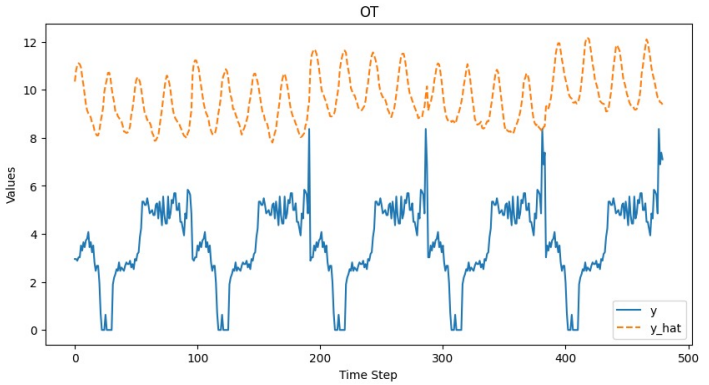
```
RNN_multi2multi = RNN(input_size=multi2multi_loader.in_variable,
                      hidden_size=hidden_size,
                      output_size=multi2multi_loader.out_variable,
                      ahead=label_size,
                      num_layers=num_layers)
RNN_multi2multi_manager = ModelManager(model=RNN_multi2multi,
                                       train_loader=multi2multi_loader.train_loader,
                                       val_loader=multi2multi_loader.val_loader,
                                       lr=learning_rate,
                                       patience=patience)
RNN_multi2multi_manager.train(num_epochs=num_epochs,
                              save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": RNN_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": RNN_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
})
results[-1]
```

Loading...

```
Epoch [29/1000000], time: 13s, loss: 1.7758, val_loss: 2.7365
Model saved to weights\multi2multi\best-RNN.pth
Epoch [30/1000000], time: 13s, loss: 1.7012, val_loss: 2.5789
Epoch [31/1000000], time: 13s, loss: 1.6243, val_loss: 2.6399
Epoch [32/1000000], time: 13s, loss: 1.6731, val_loss: 2.6522
Model saved to weights\multi2multi\best-RNN.pth
```







Case 1: Multivariate to Multivariate Prediction

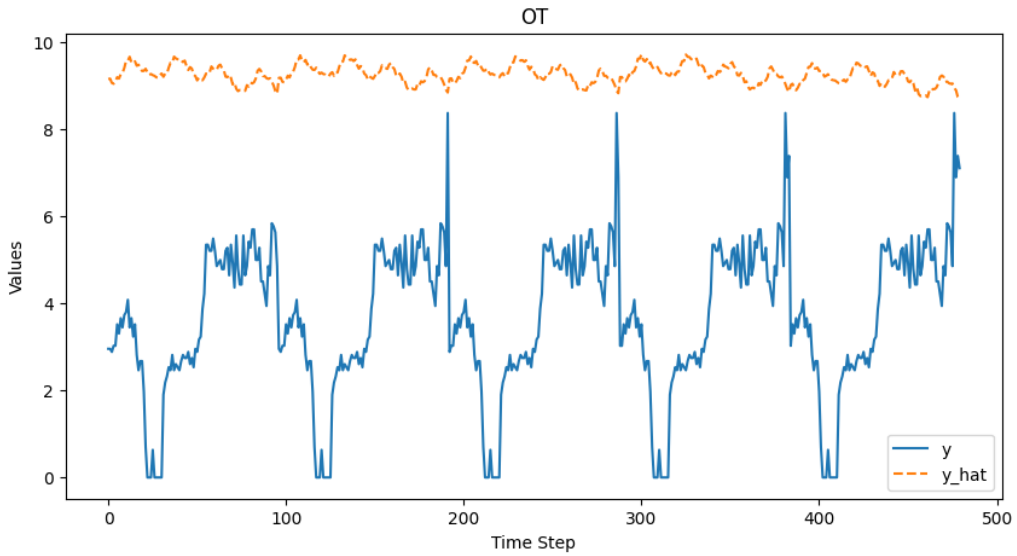
Experiments were perfomed by TA Khoa Nguyen

RNN: Time-series Data Project

Define RNN Structure in Pytorch

```
features_type='MS'
sub_dir = 'multi2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2uni_loader = TimeSeriesDataLoader(file_path,
                                        input_size=input_size,
                                        label_size=label_size,
                                        offset=offset,
                                        train_size=train_size,
                                        val_size=val_size,
                                        target_name=target_name,
                                        features_type=features_type,
                                        date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 7)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 7)
self.y_test.shape = (2861, 96, 1)
```



Case 2: Multivariate to Univariate Prediction

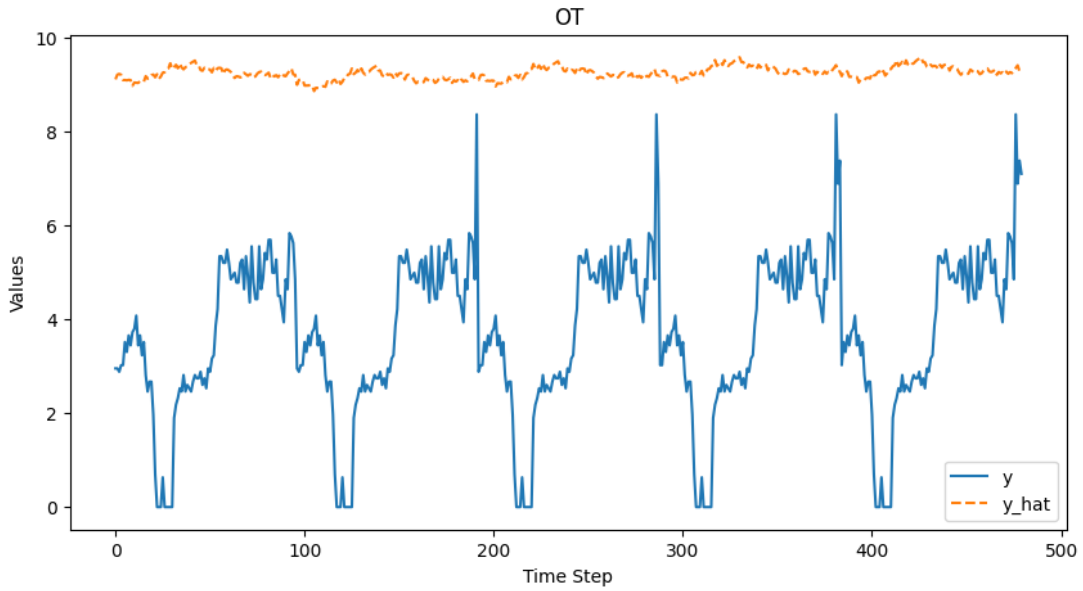
Experiments were perfomed by TA Khoa Nguyen

RNN: Time-series Data Project

Define RNN Structure in Pytorch

```
features_type='S'
sub_dir = 'uni2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
uni2uni_loader = TimeSeriesDataLoader(file_path,
                                      input_size=input_size,
                                      label_size=label_size,
                                      offset=offset,
                                      train_size=train_size,
                                      val_size=val_size,
                                      target_name=target_name,
                                      features_type=features_type,
                                      date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 1)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 1)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 1)
self.y_test.shape = (2861, 96, 1)
```

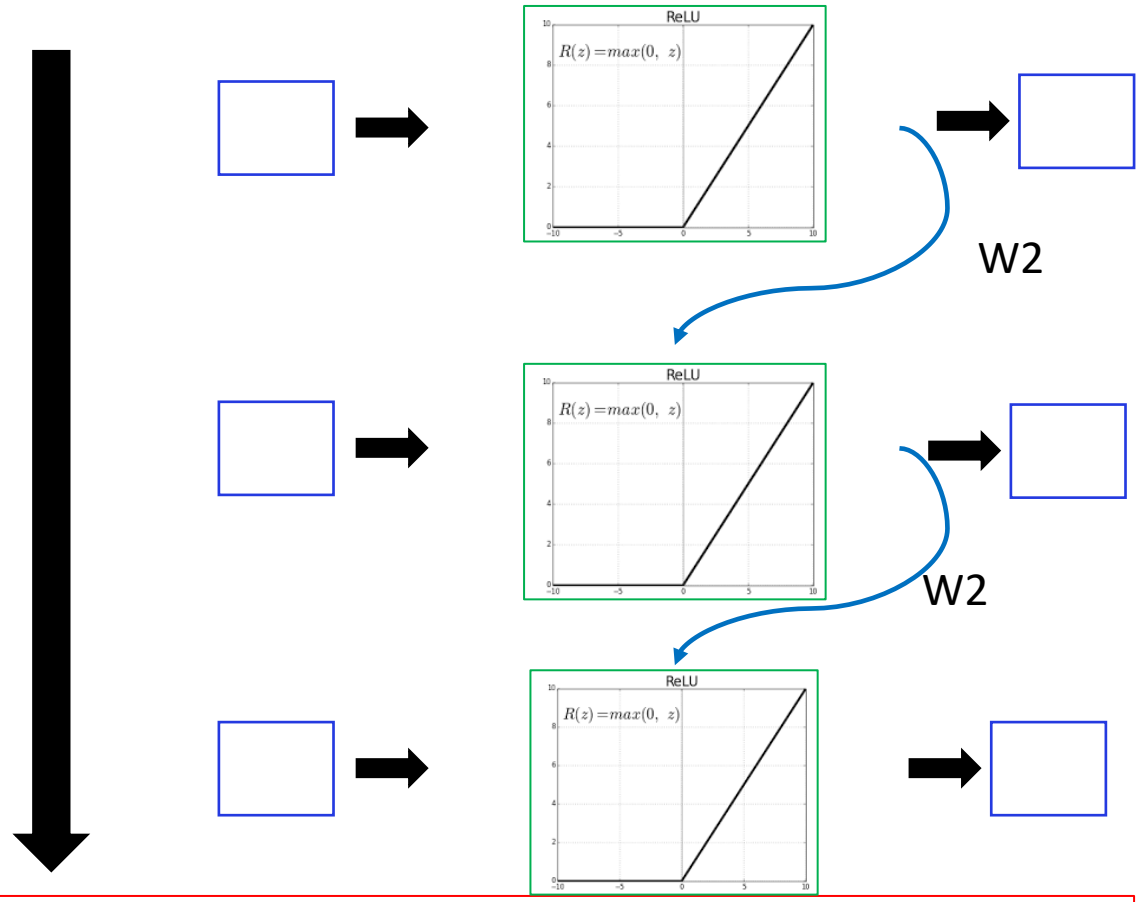


Case 3: Univariate to Univariate Prediction

Experiments were perfomed by TA Khoa Nguyen

Why RRN is not Use Often

The more we unroll the recurrent neural network, the harder it is to train



Vanishing / Exploding Gradient Problem

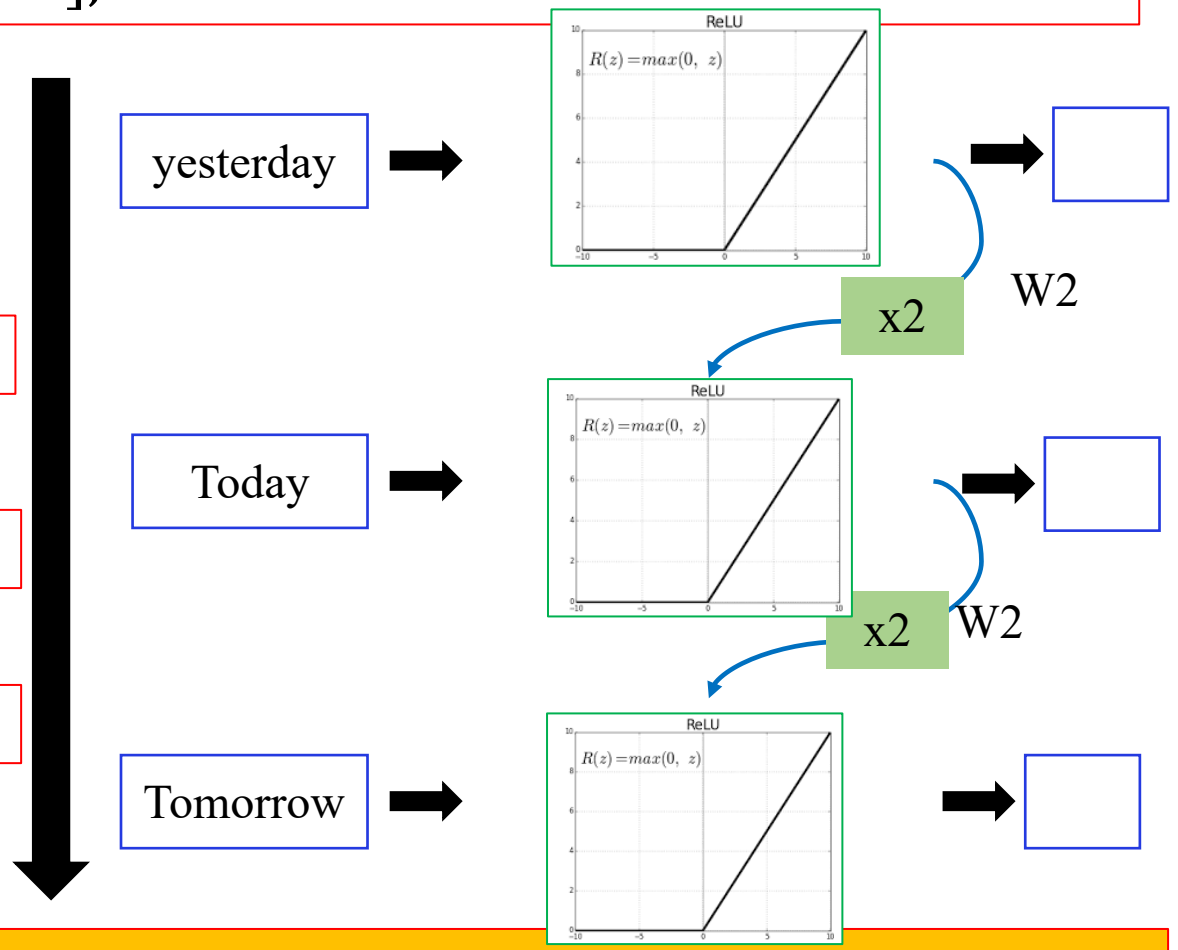
Why RRN is not use Often

If we set W2 ∈ [1, +∞], *Let's start with* $W2 = 2$

$$yesterday \times 2 \times 2 = yesterday \times 2^2 = yesterday \times W_2^2$$

$$Yesterday \times 2^{100} = yesterday \times 2^{100} = yesterday \times W_2^{100}$$

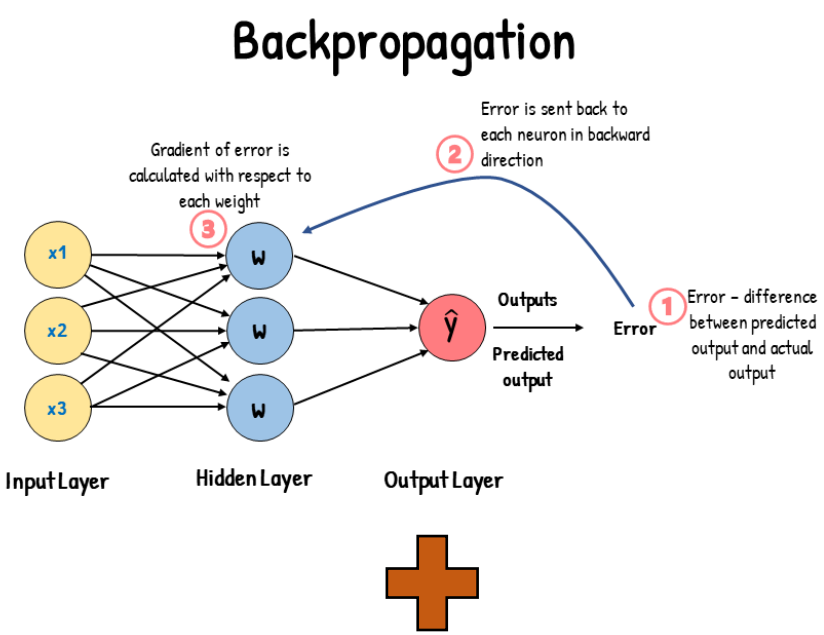
$$Yesterday \times Huge\ Number$$



Exploding Gradient Problem

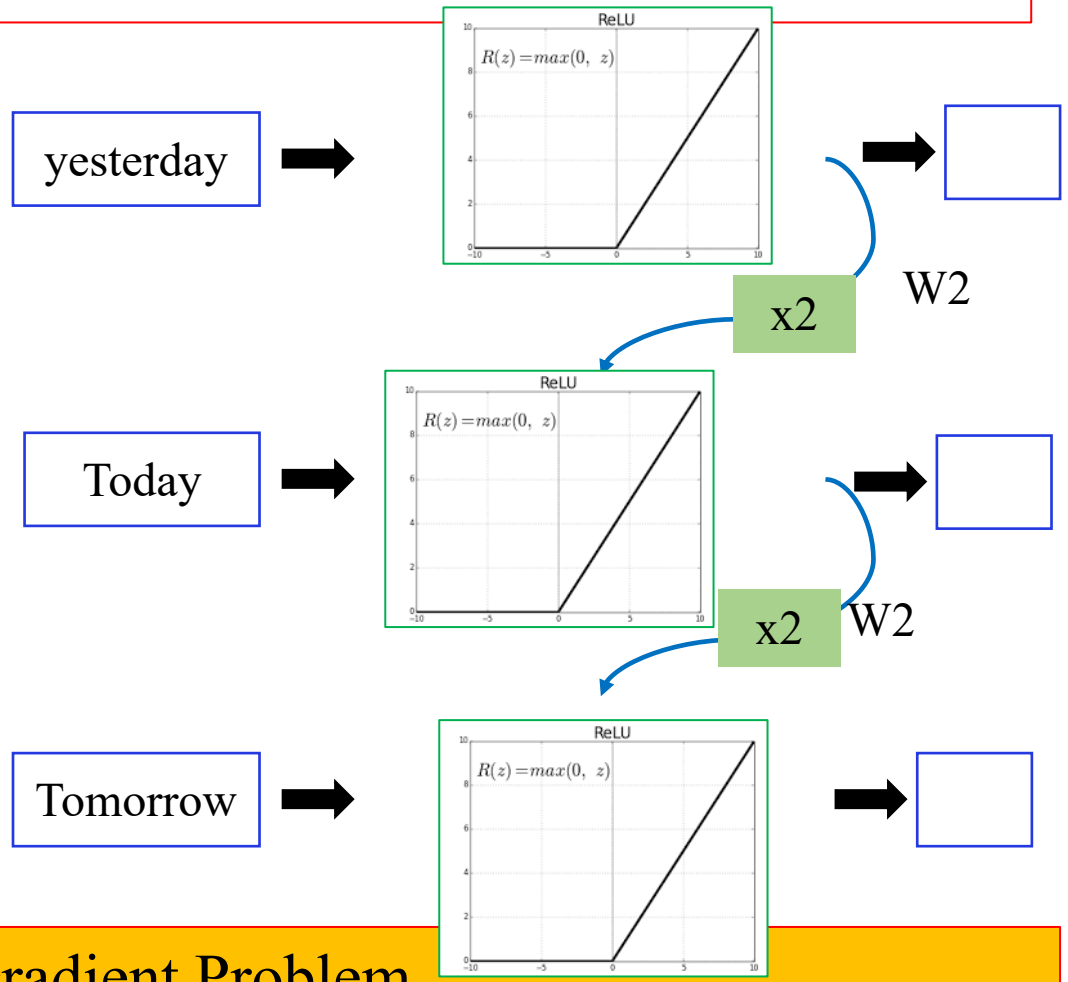
Why RRN is not use Often

If we set W2 ∈ [1, +∞], *Let's start with W2 = 2*



W is a huge numbers

Overshoot the optimal solution



Exploding Gradient Problem

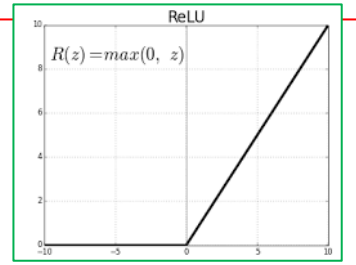
Why RRN is not use Often

If we set W2 ∈ [1, +∞], *Let's start with* $W2 = 2$

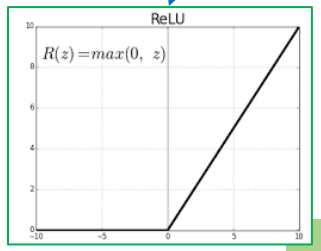
$$yesterday \times 0.5 \times 0.5 = yesterday \times 0.5^2 = yesterday \times W_2^2$$

$$Yesterday \times 0.5^{100} = yesterday \times 0.5^{100} = yesterday \times W_2^{100}$$

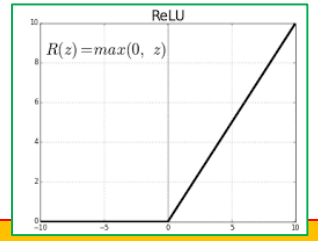
Yesterday × Really Small *Number*

yesterday → 

→ x2 → W2

Today → 

→ x2 → W2

Tomorrow → 

Vanishing Gradient Problem

How to avoid Exploding and Vanishing Gradient Problem in RNN

I grew up in France… I speak fluent *French*

RNN fail to handel

47

Outline

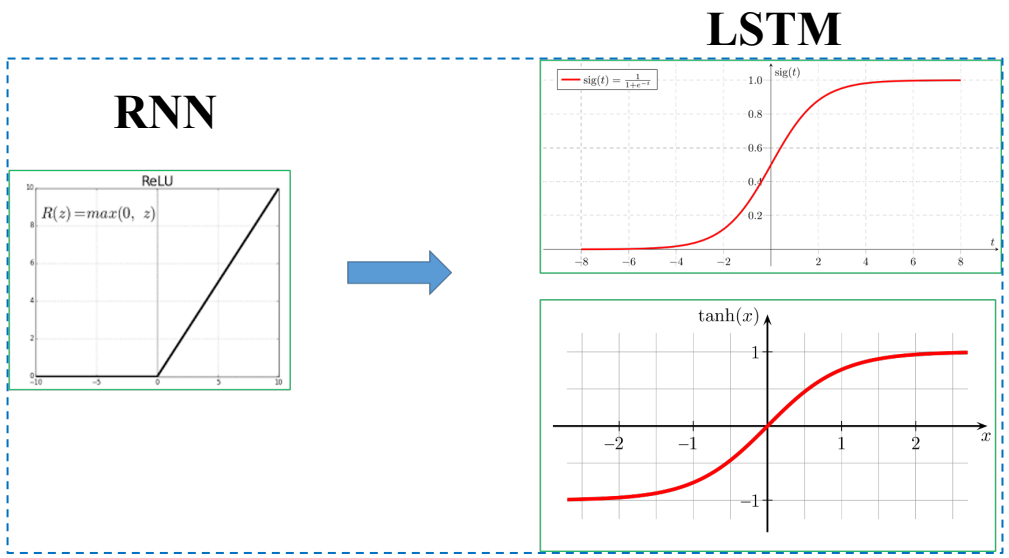
- Multilayer Perceptron and Time-series Data Forecasting
- RNN and Time-series Data Forecasting
- LSTM and Time-series Data Forecasting
- Bi-LSTM and Time-series Data Forecasting
- XGBoost and Time-series Data Forecasting

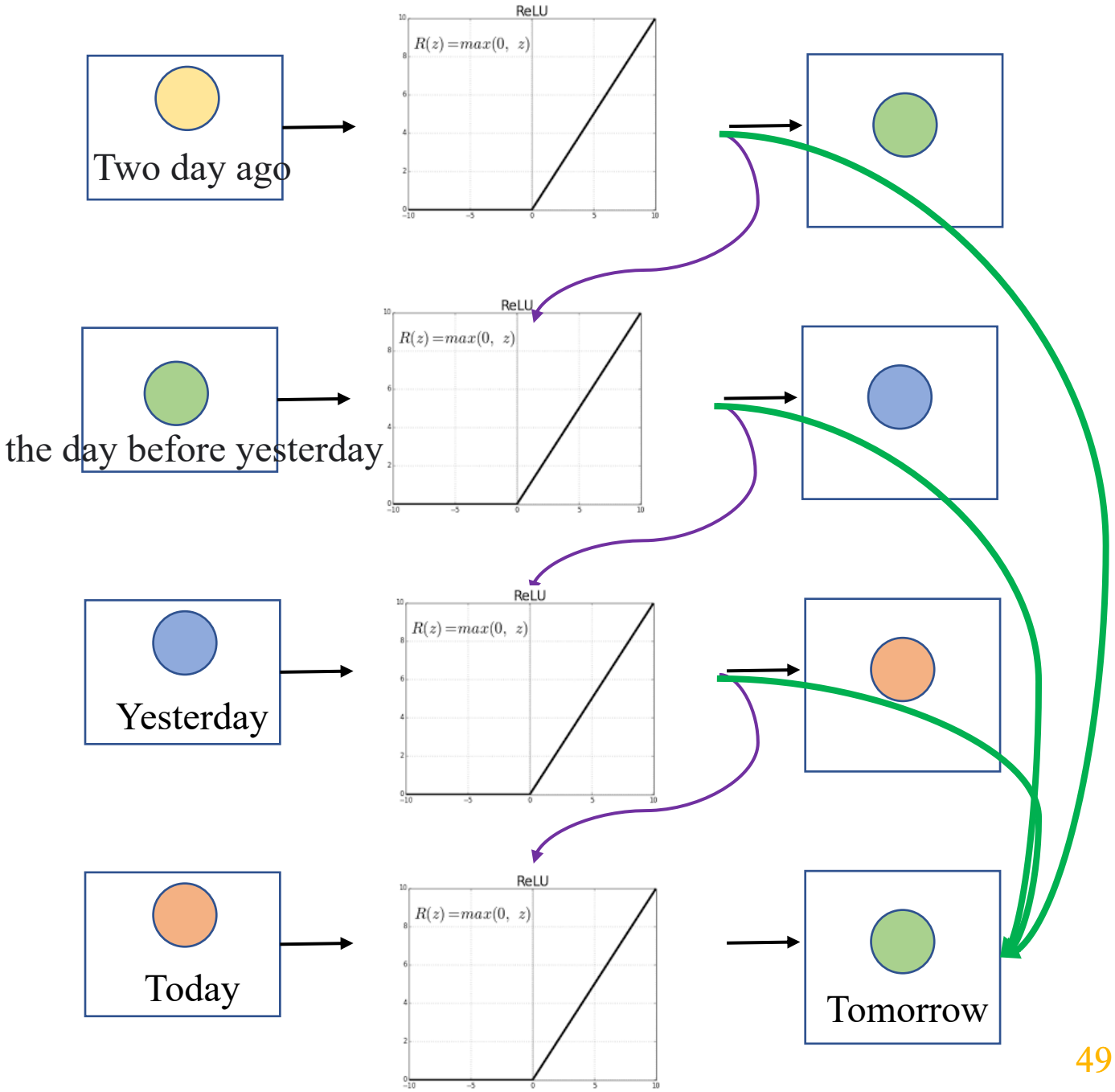
AI VIETNAM
All-in-One Course

Long short-term Memory: Simple Explanation

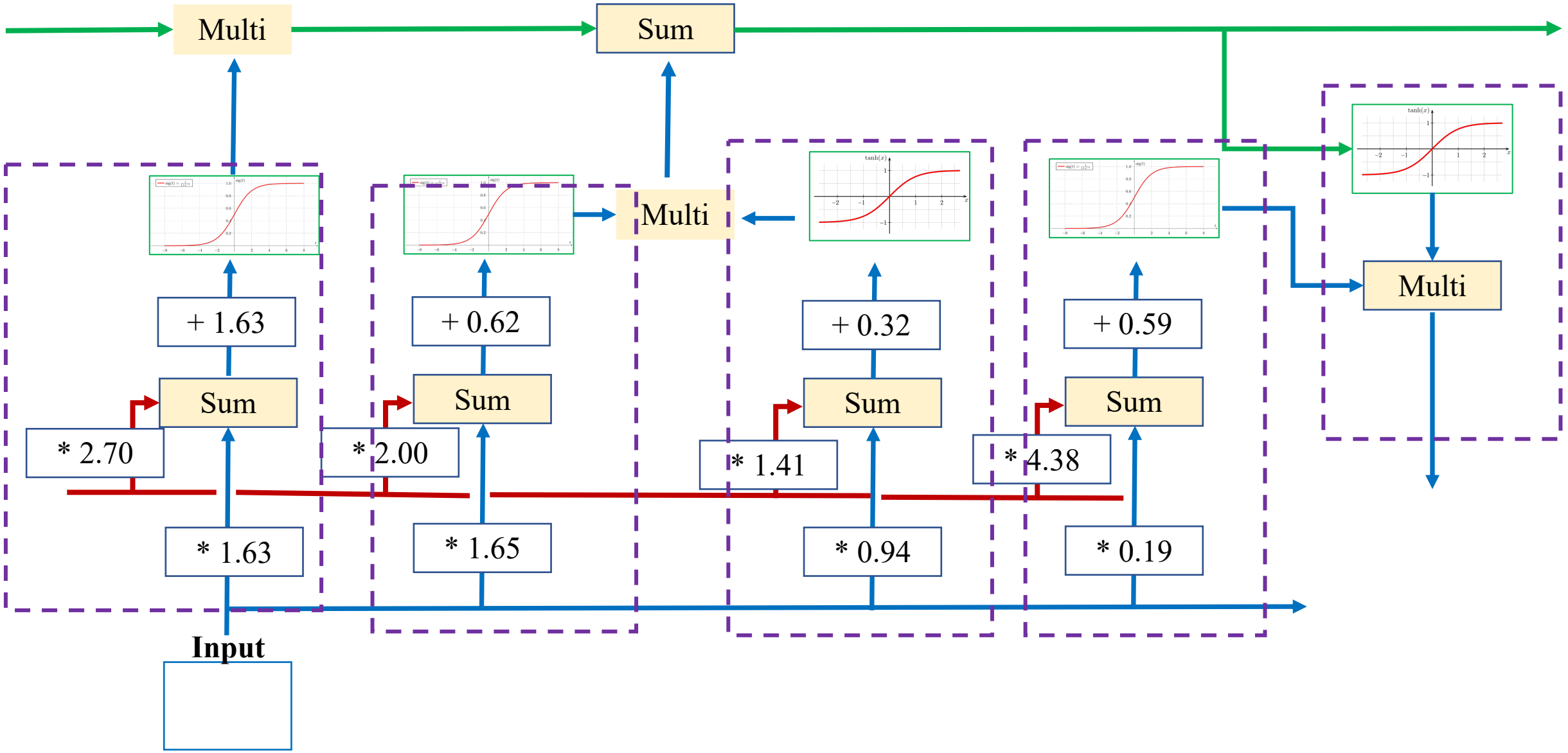
Short memory

Long memory



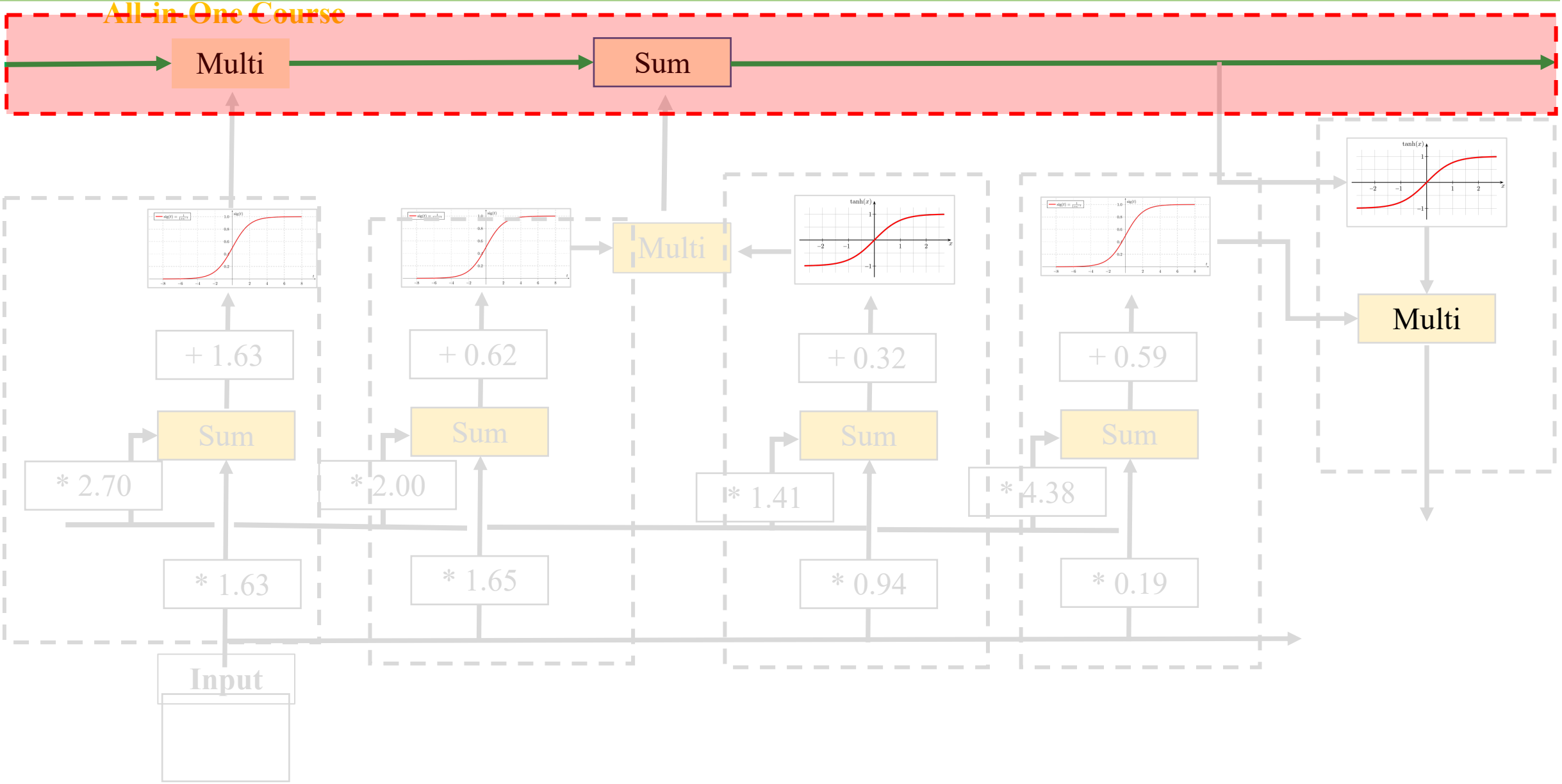


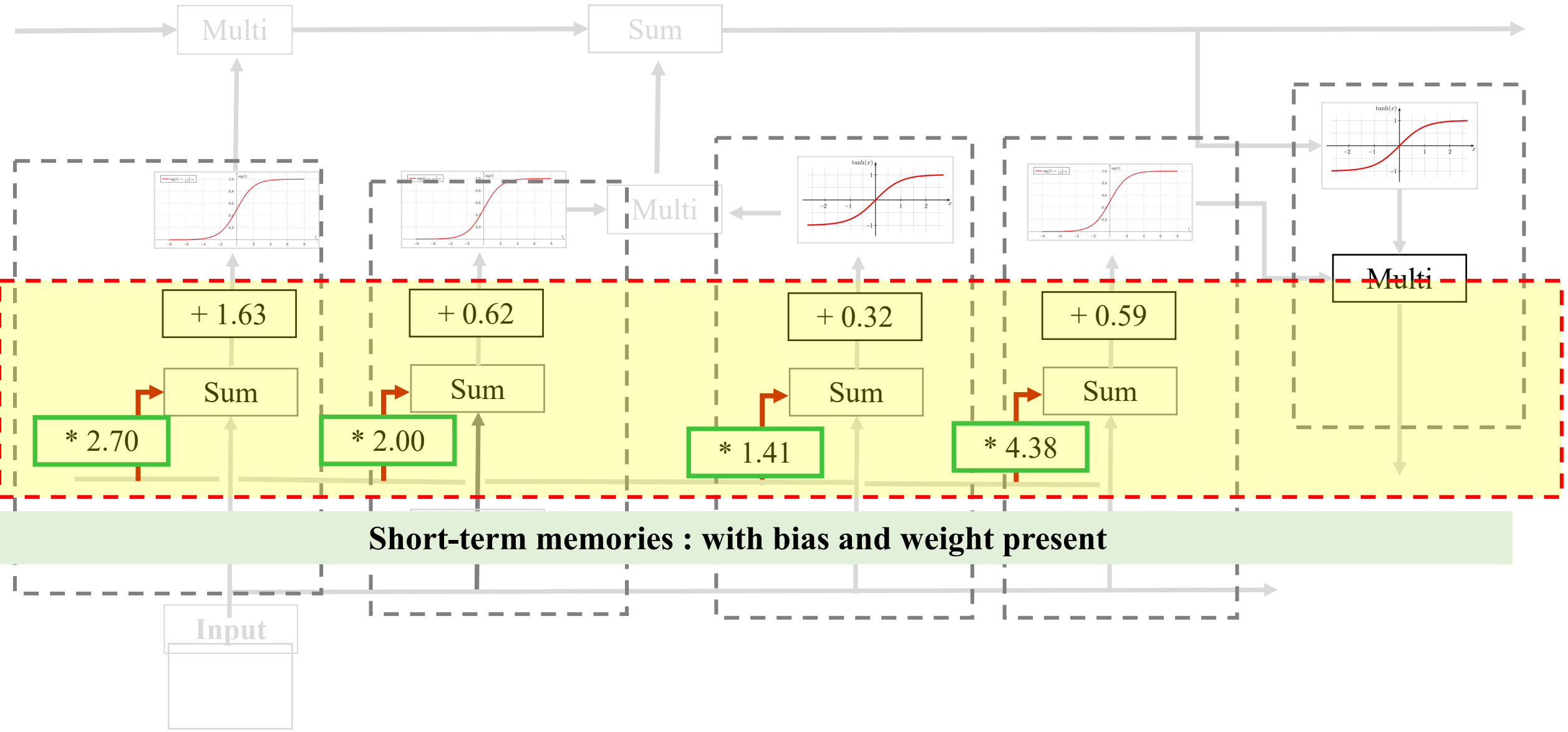
Three Stages in a Single LSTM Unit



Long-term memories : no bias and weight present => avoid causing exploding and vanishing gradient

All-in-One Course

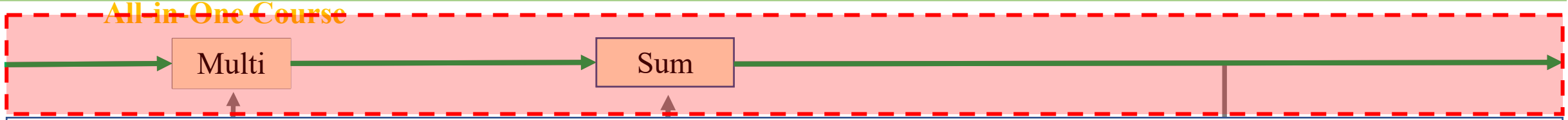




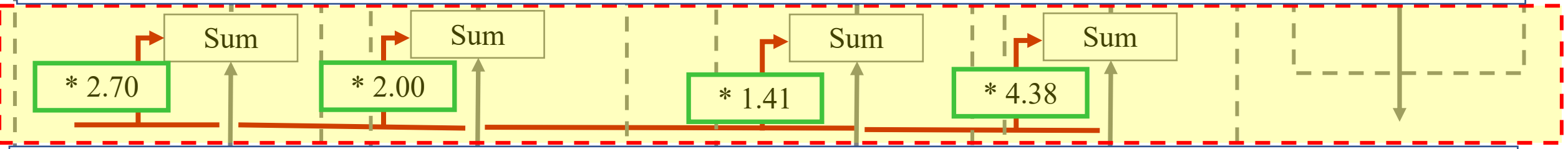
Short-term memories : with bias and weight present

Long-term memories : no bias and weight present => avoid causing exploding and vanishing gradient

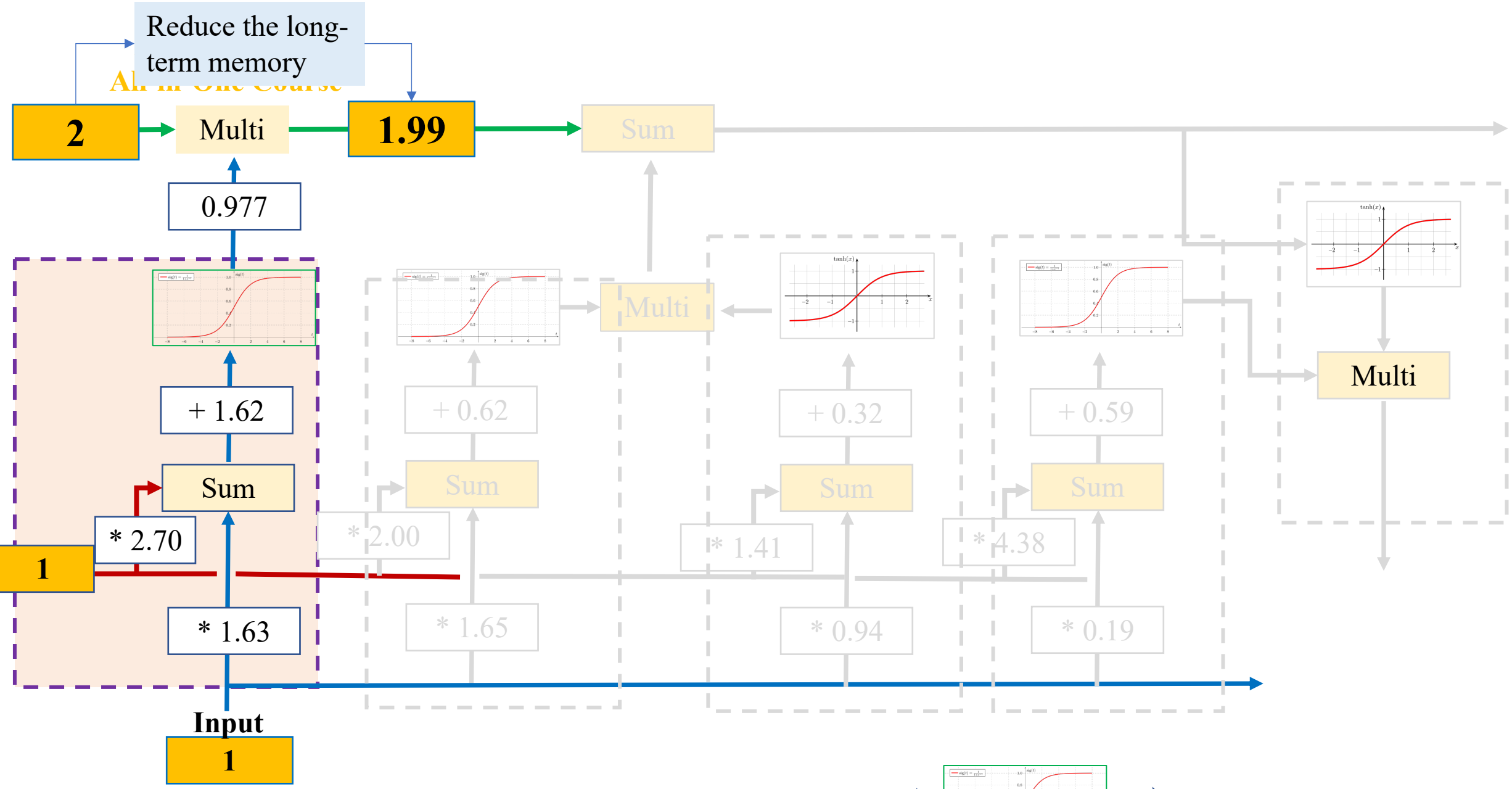
All-in-One Course



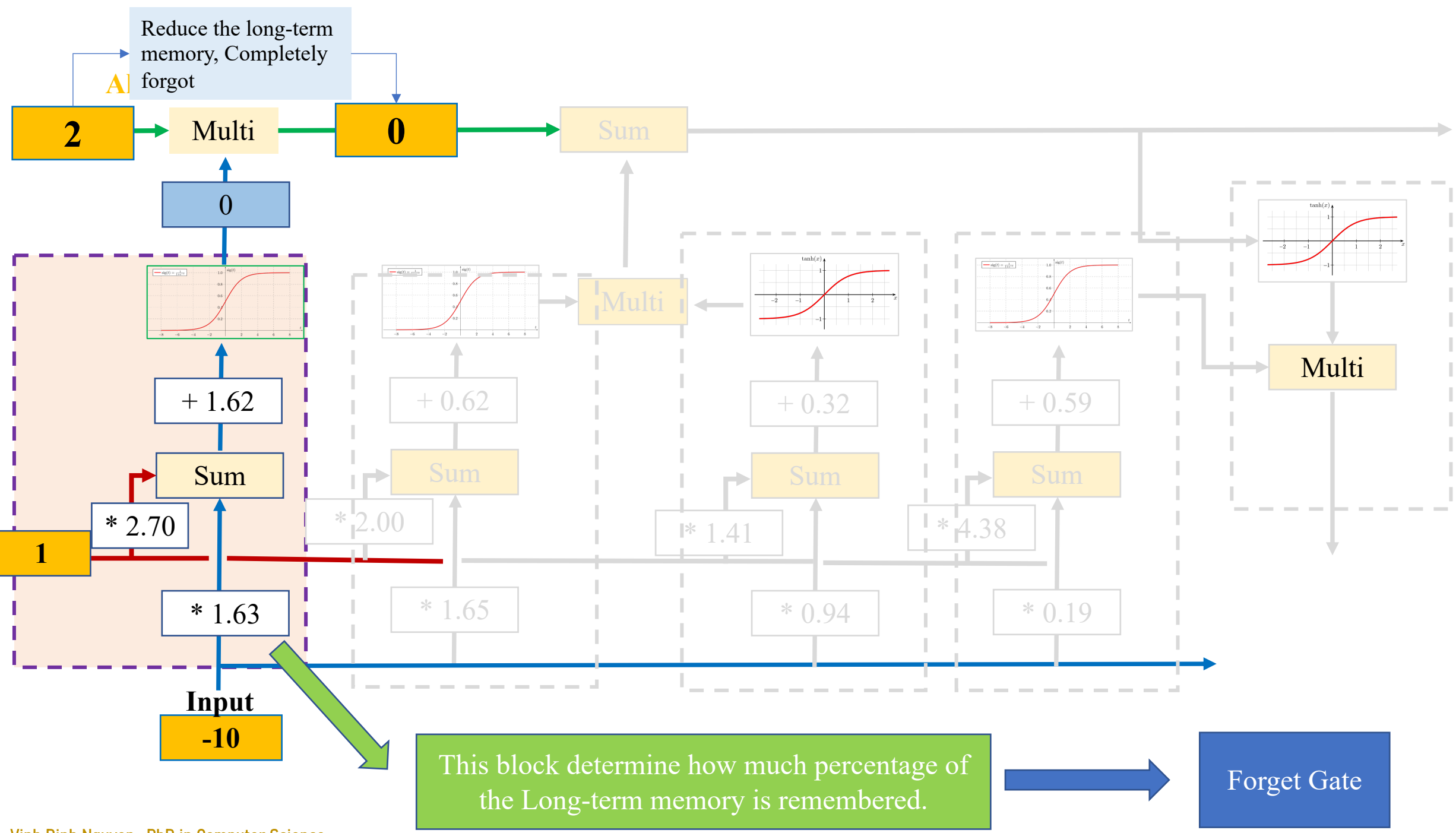
How do long-term memory and short-term memories work?

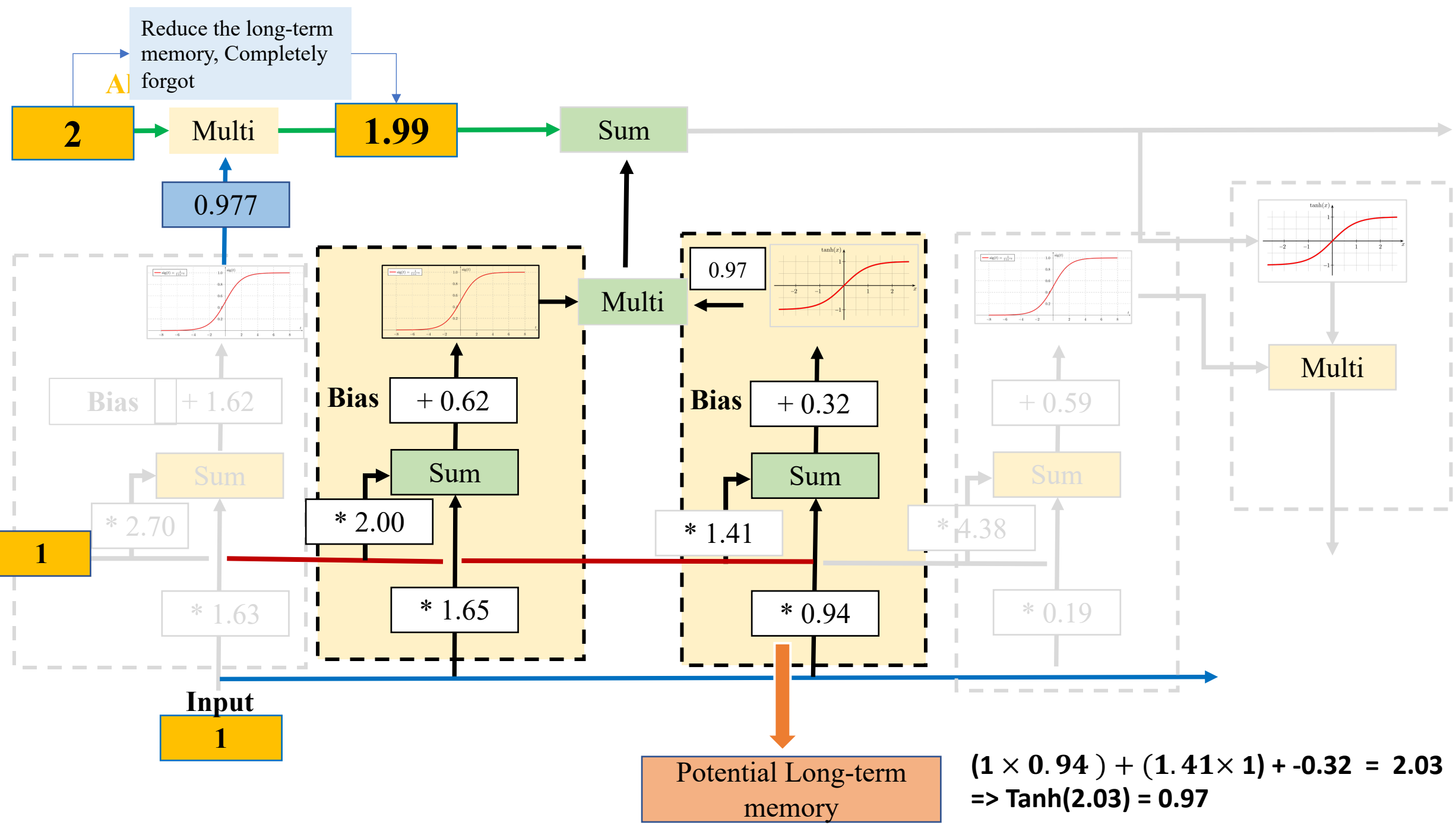


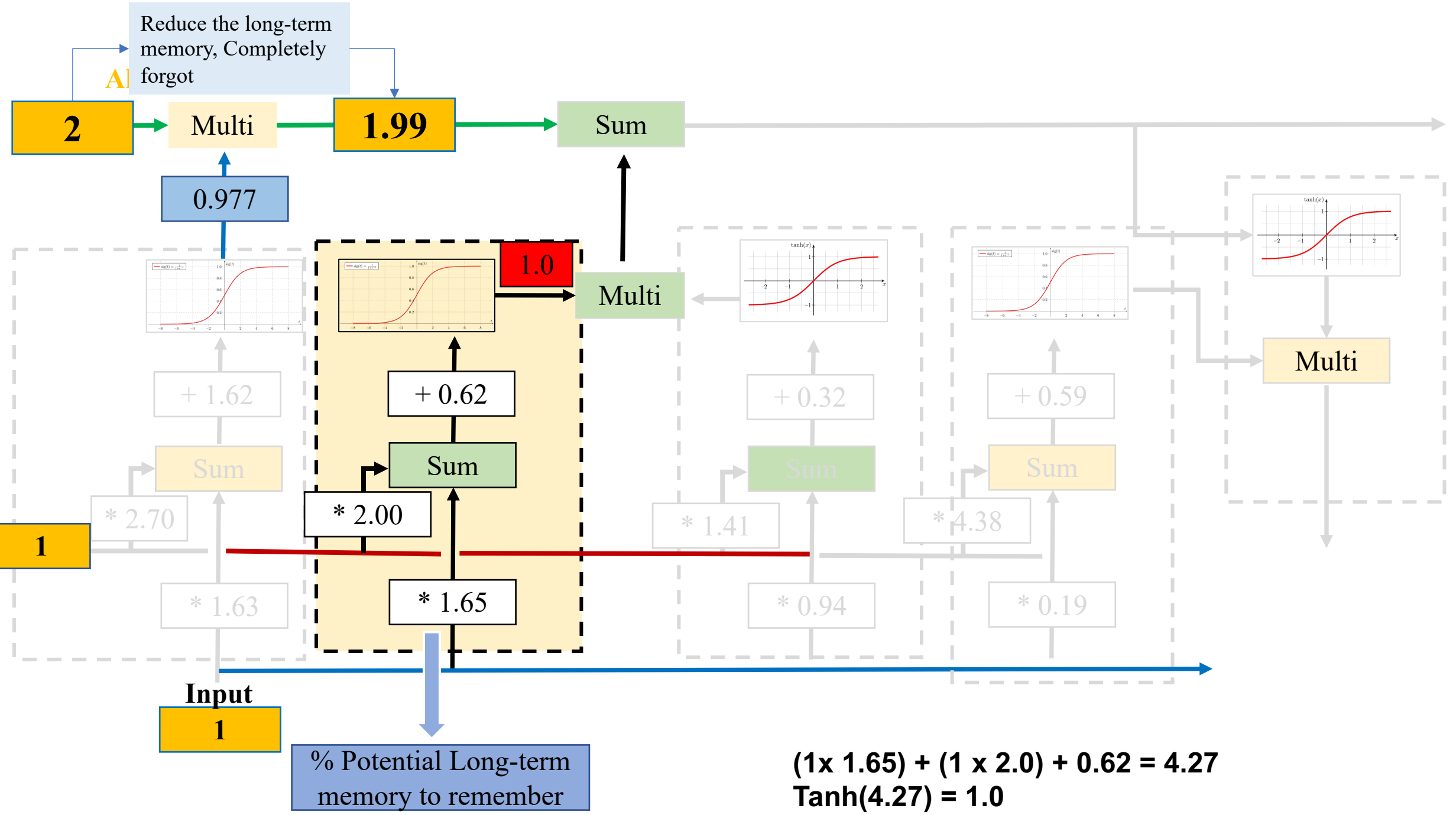
Short-term memories : with bias and weight present



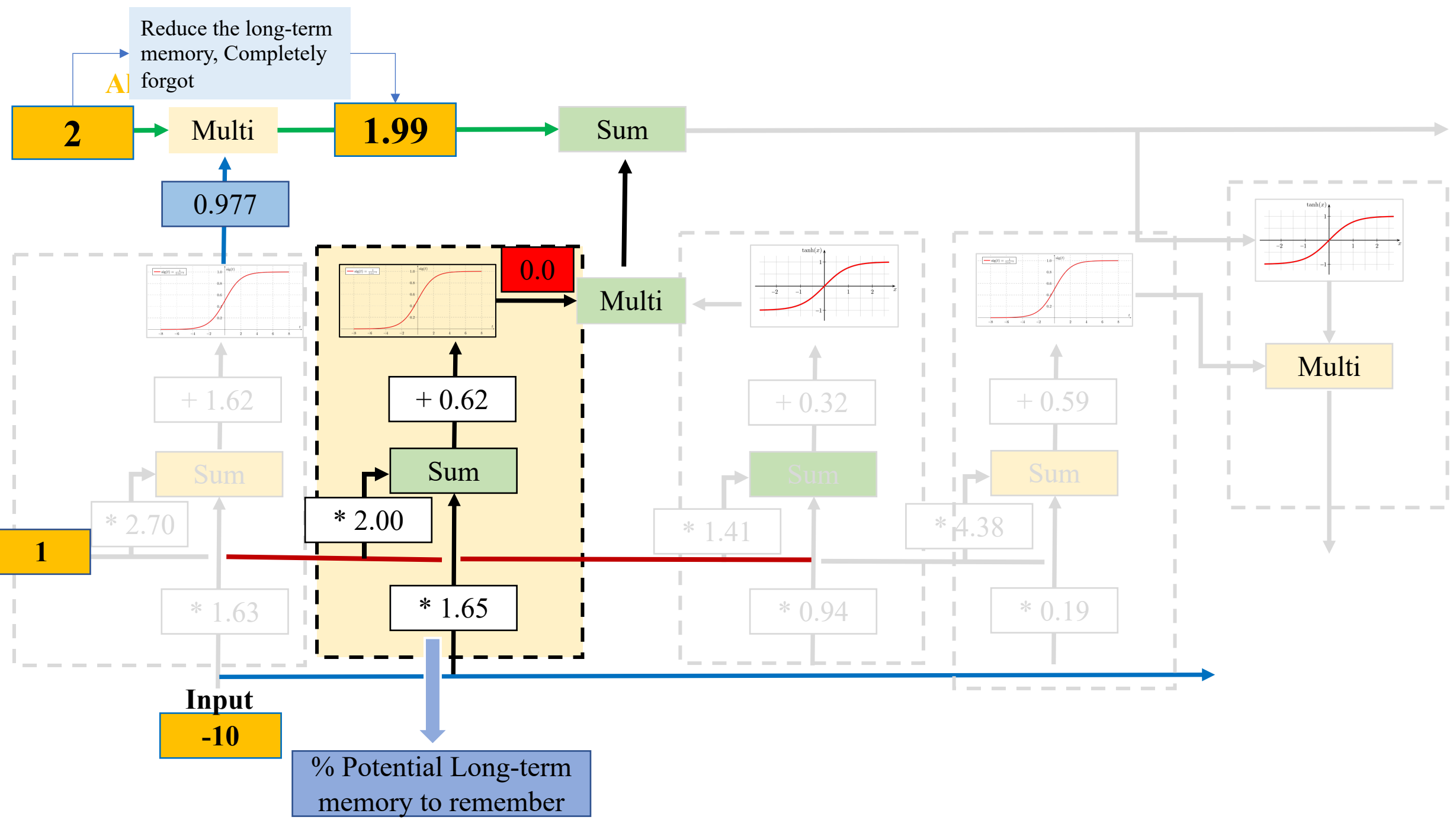
$(1 \times 1.63) + (1 \times 2.70) + 1.62 = 5.95$ → $0.997 \times 2 = 1.99$

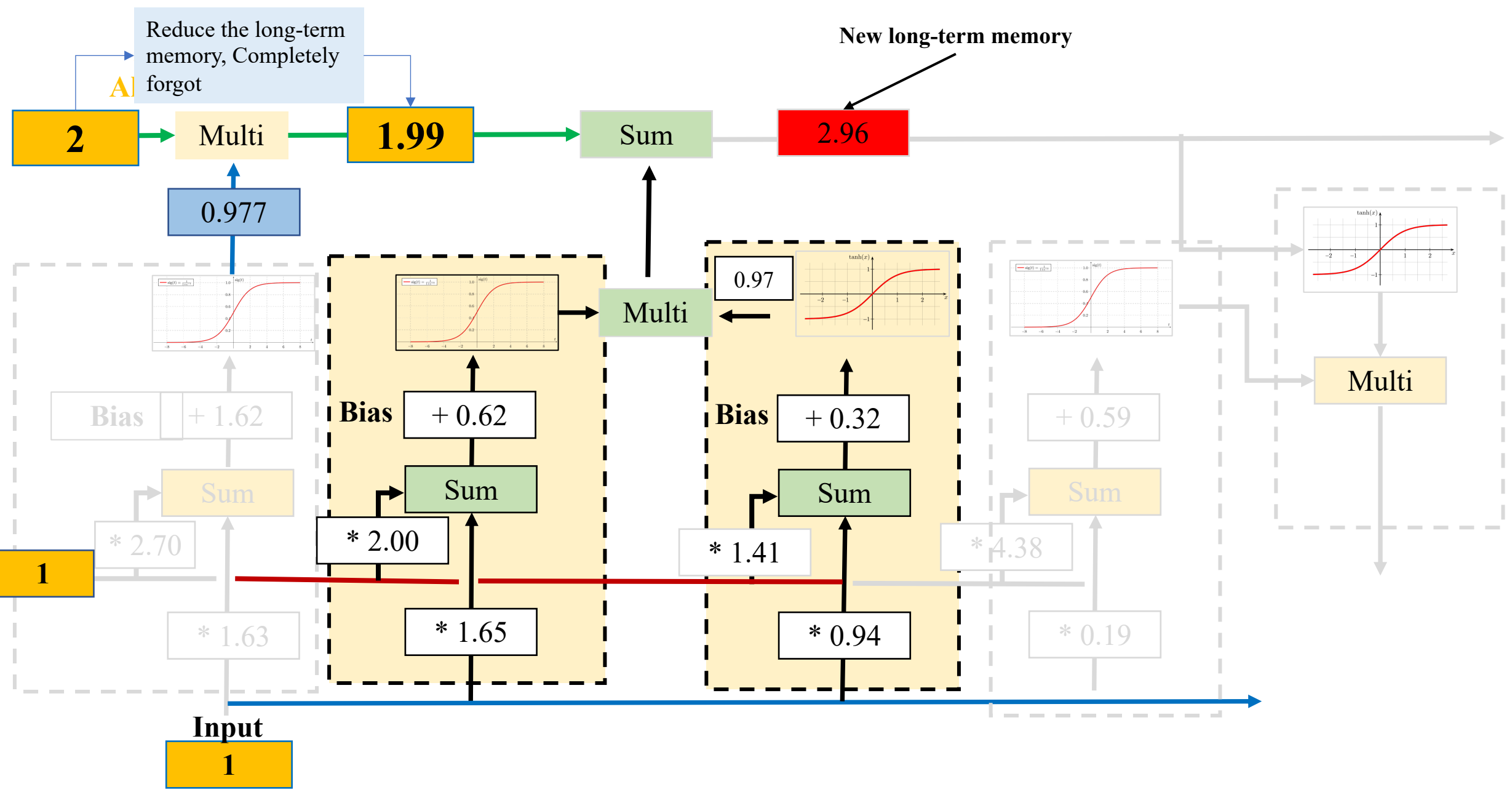

Reduce the long-term memory, Completely forgot
2
Multi
0
Sum
0
+ 1.62
Sum
* 2.70
1
* 1.63
Input
-10
+ 0.62
Sum
* 2.00
* 1.65
Multi
+ 0.32
Sum
* 1.41
* 0.94
+ 0.59
Sum
* 4.38
* 0.19
Multi
This block determine how much percentage of the Long-term memory is remembered.
Forget Gate

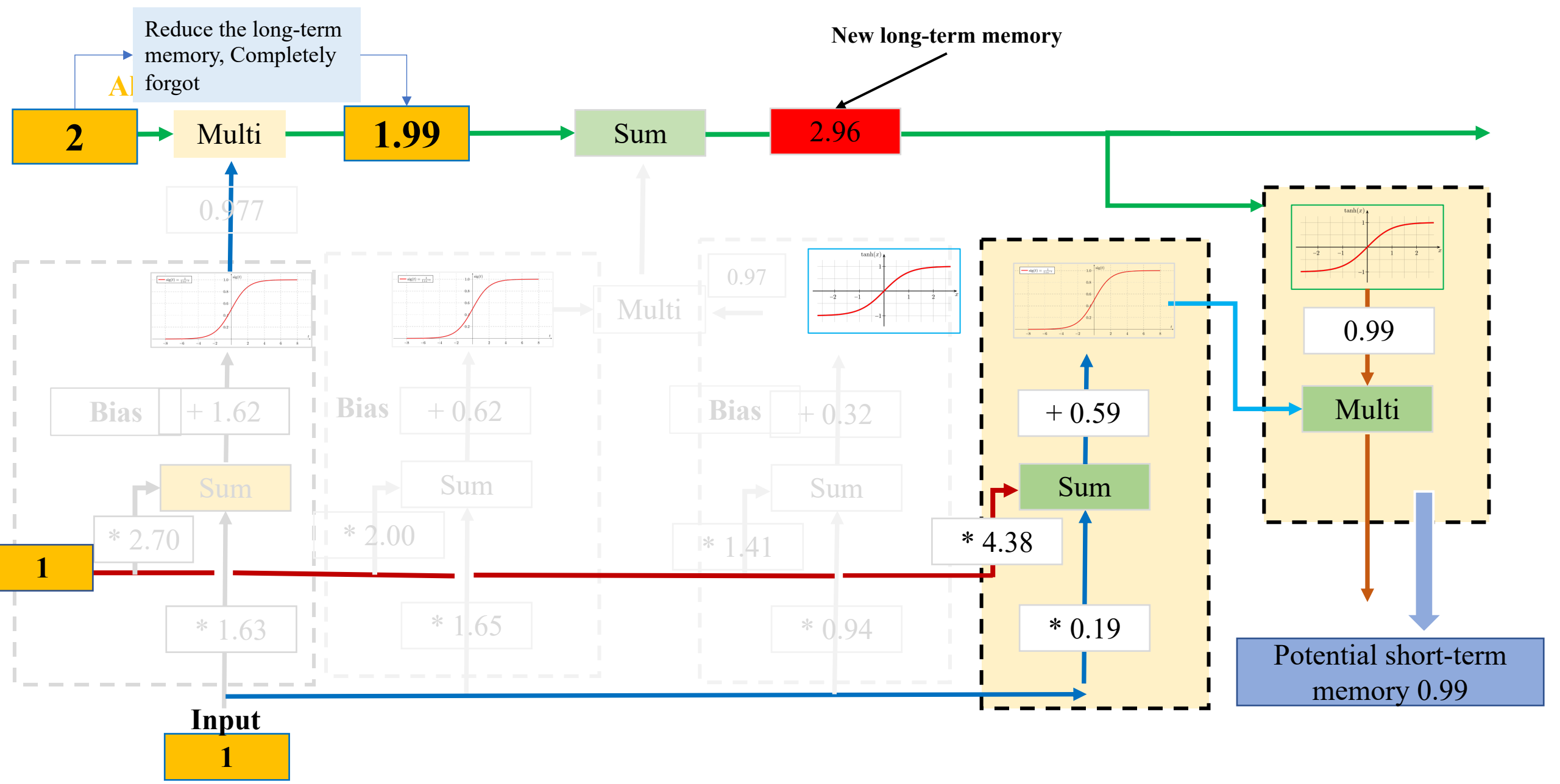

Reduce the long-term memory, Completely forgot
2
Multi
1.99
Sum
0.977
Bias
+ 0.62
Sum
* 2.00
* 1.65
Multi
0.97
Bias
+ 0.32
Sum
* 1.41
* 0.94
Multi
Bias
+ 1.62
Sum
* 2.70
* 1.63
+ 0.59
Sum
* 4.38
* 0.19
1
Input
1
Potential Long-term memory
(1 × 0.94) + (1.41× 1) + -0.32 = 2.03
=> Tanh(2.03) = 0.97

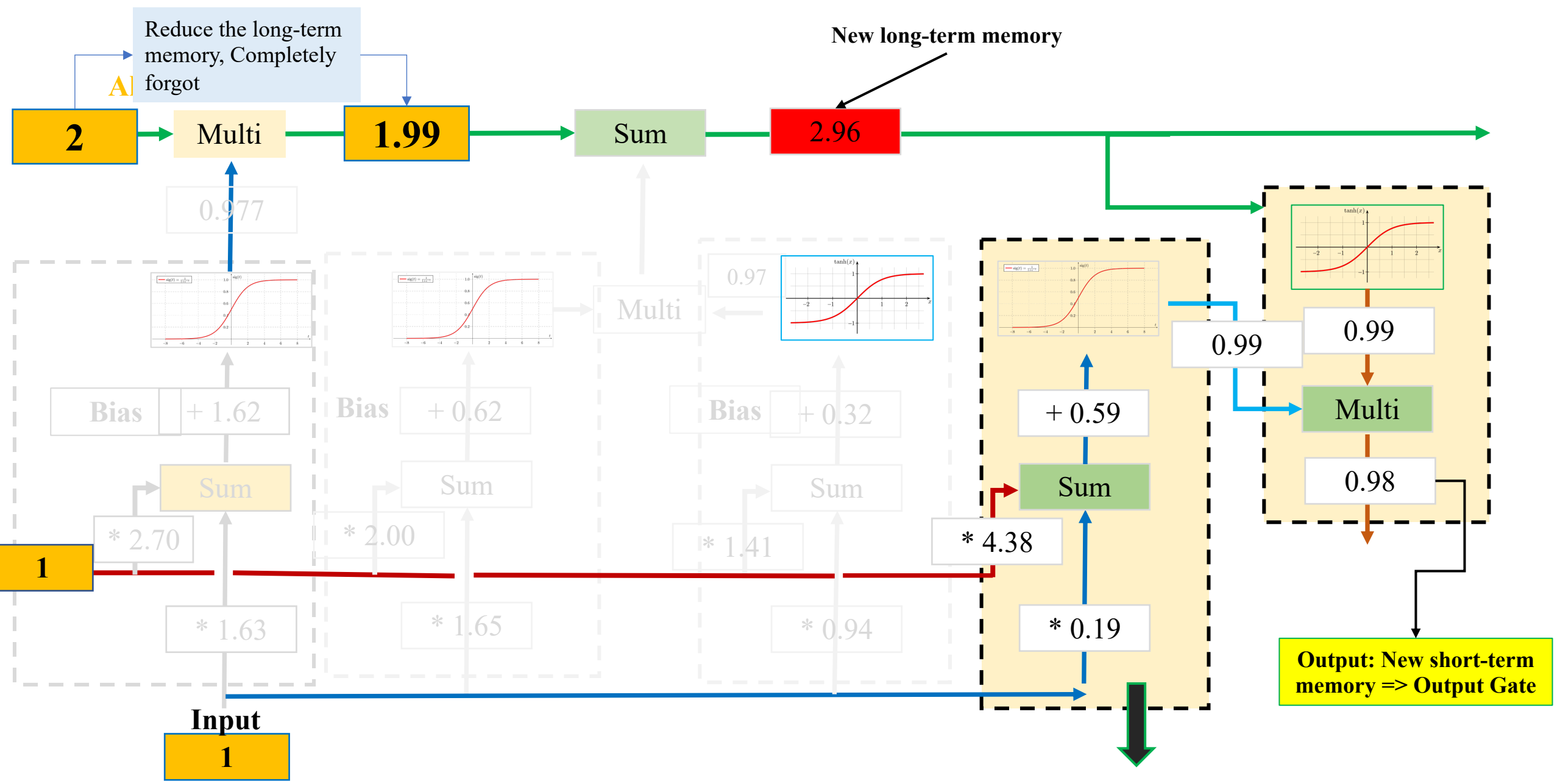


(1x 1.65) + (1 x 2.0) + 0.62 = 4.27
Tanh(4.27) = 1.0

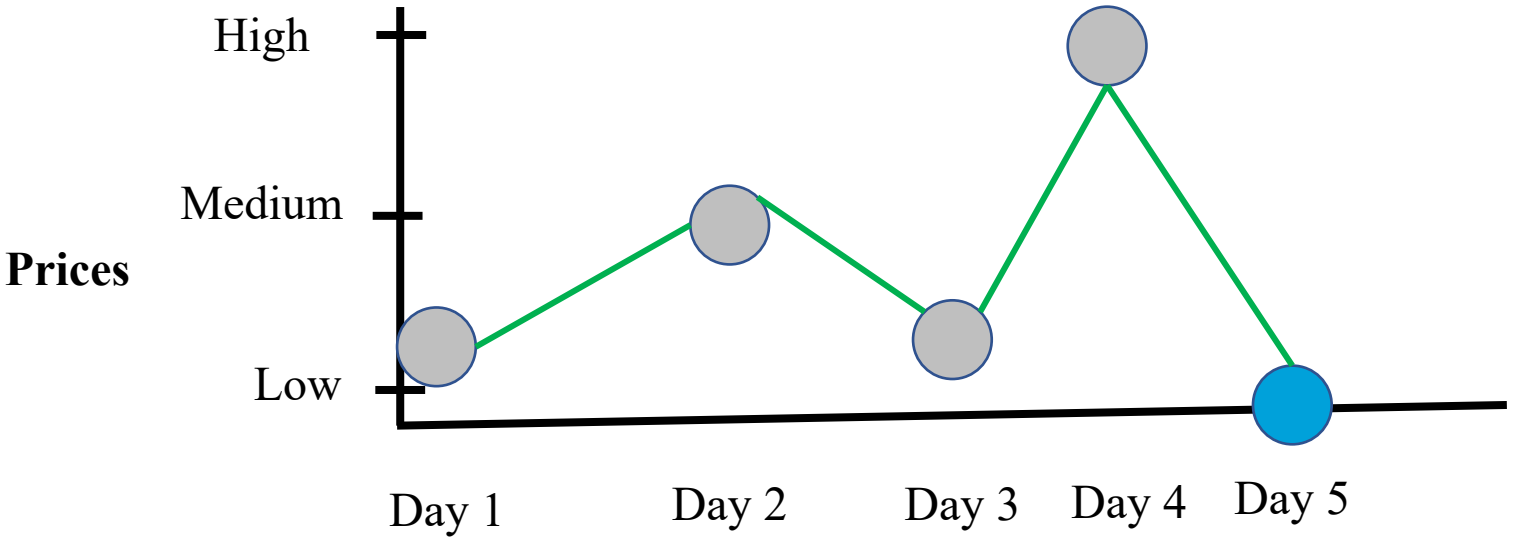

Reduce the long-term memory, Completely forgot
2
Multi
1.99
Sum
0.977
0.0
Multi
+ 1.62
+ 0.62
+ 0.32
+ 0.59
Sum
Sum
Sum
Multi
* 2.70
* 2.00
* 1.41
* 4.38
1
* 1.63
* 1.65
* 0.94
* 0.19
Input
-10
% Potential Long-term memory to remember


Reduce the long-term memory, Completely forgot
New long-term memory
2
Multi
1.99
Sum
2.96
0.977
Multi
0.97
Bias
+ 0.62
Sum
* 2.00
* 1.65
Bias
+ 0.32
Sum
* 1.41
* 0.94
Multi
Bias
+ 1.62
Sum
* 2.70
* 1.63
+ 0.59
Sum
* 4.38
* 0.19
1
Input
1


Reduce the long-term memory, Completely forgot
New long-term memory
2
Multi
1.99
Sum
2.96
0.977
Bias
+ 1.62
Sum
* 2.70
* 1.63
Bias
+ 0.62
Sum
* 2.00
* 1.65
Multi
0.97
Bias
+ 0.32
Sum
* 1.41
* 0.94
1
Input
1
+ 0.59
Sum
* 4.38
* 0.19
0.99
Multi
Potential short-term memory 0.99

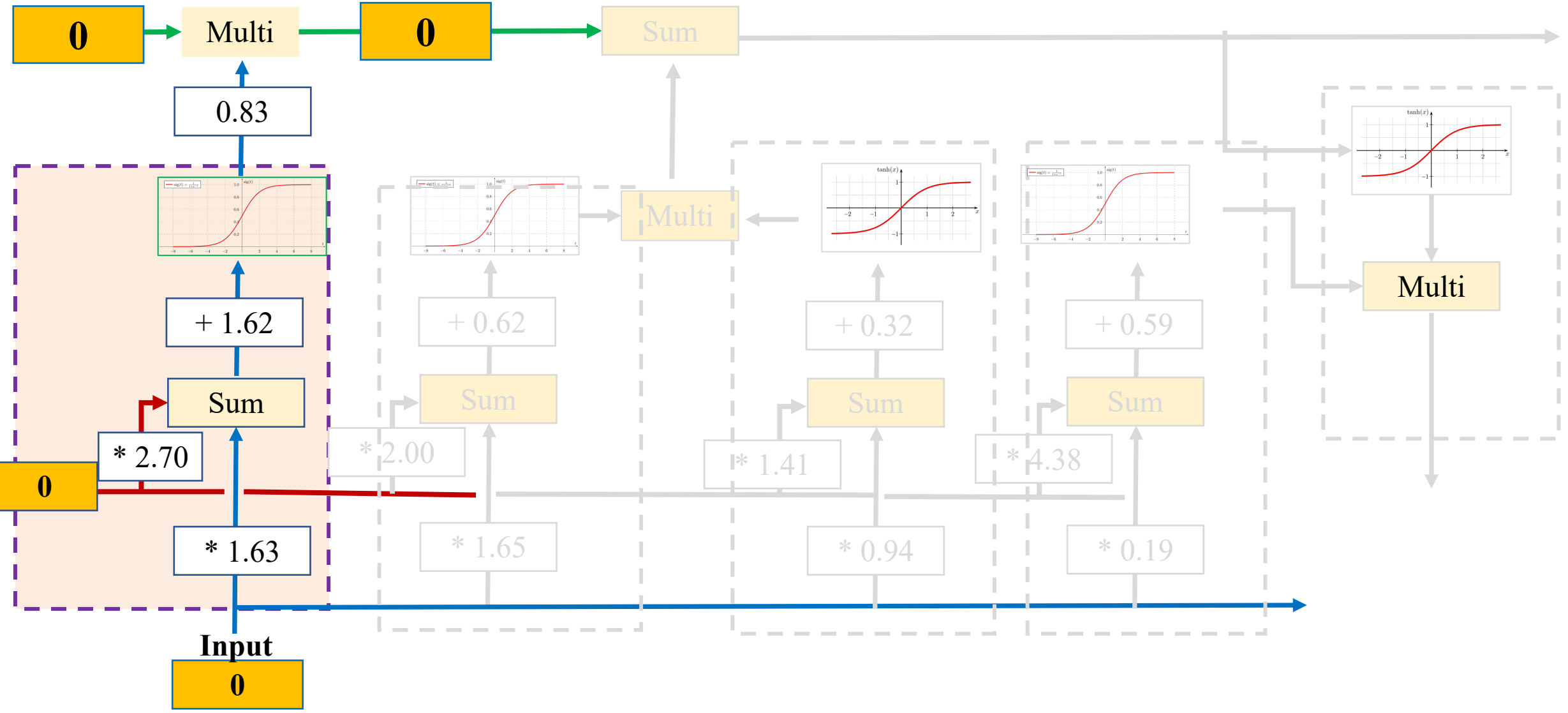

Reduce the long-term memory, Completely forgot
New long-term memory
2
Multi
1.99
Sum
2.96
0.977
Bias
+ 1.62
Sum
* 2.70
1
* 1.63
Input
1
Bias
+ 0.62
Sum
* 2.00
* 1.65
Multi
0.97
Bias
+ 0.32
Sum
* 1.41
* 0.94
+ 0.59
Sum
* 4.38
* 0.19
0.99
0.99
Multi
0.98
Output: New short-term memory => Output Gate
How much % Potential short-term memory to remember

LSTM Unit: Example

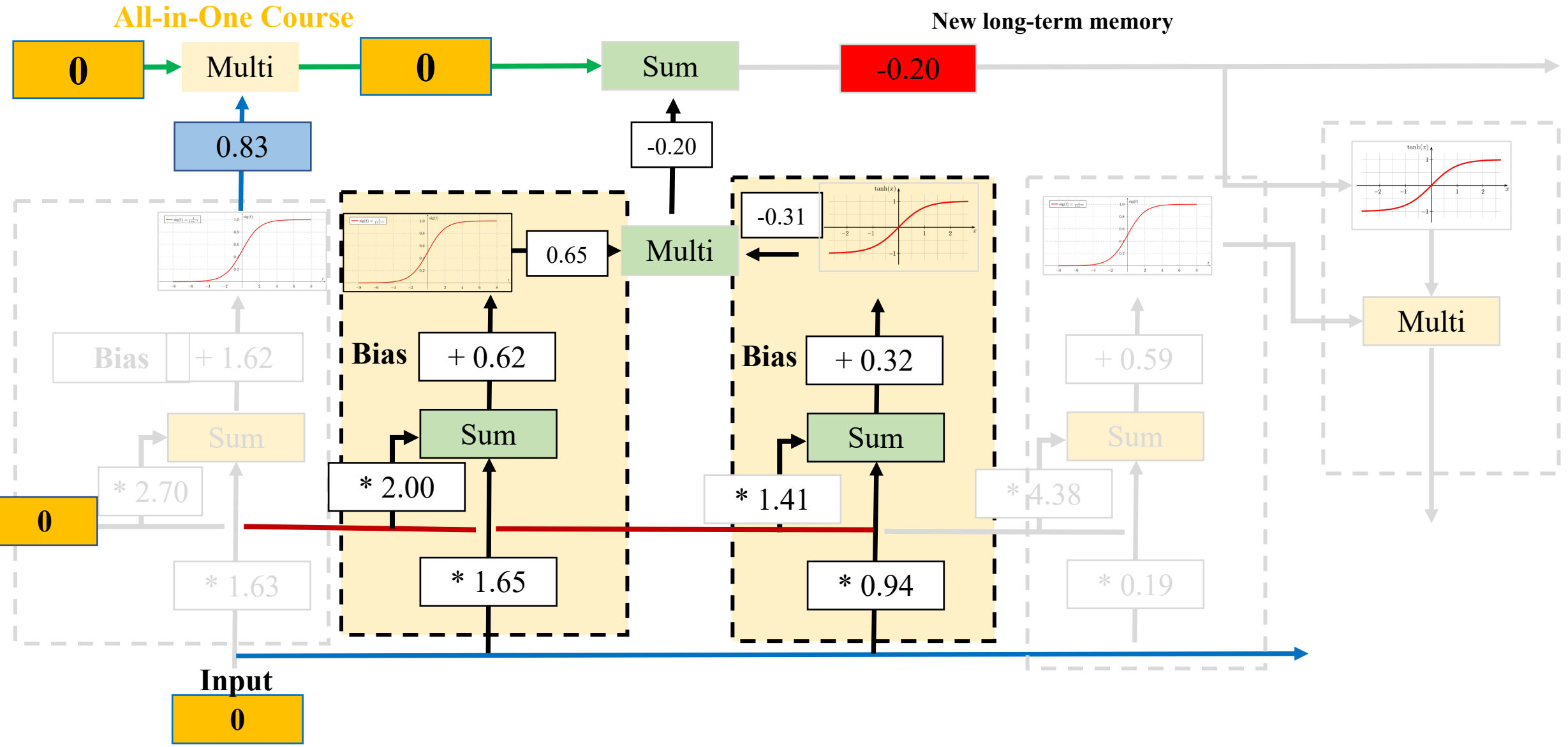


Input: day 1, day 2, day 3, day 4
Output: day 5
Algorithm: Long-short term memory

Day 1 input


0
Multi
0
Sum
0.83
+ 1.62
Sum
* 2.70
0
* 1.63
Input
0
* 2.00
+ 0.62
Sum
* 1.65
Multi
* 1.41
+ 0.32
Sum
* 0.94
* 4.38
+ 0.59
Sum
* 0.19
Multi

Day 1 input


New long-term memory
0
Multi
0
Sum
-0.20
0.83
-0.20
Bias
+ 1.62
Sum
* 2.70
* 1.63
0
Bias
+ 0.62
Sum
* 2.00
* 1.65
0.65
Multi
-0.31
Bias
+ 0.32
Sum
* 1.41
* 0.94
+ 0.59
Sum
* 4.38
* 0.19
Multi
Input
0

Day 1 input

Reduce the long-term memory, Completely forgot

0 → Multi → 0.0 → Sum → -0.20

0.977

Bias + 1.62 → Sum → * 2.70

Bias + 0.62 → Sum → * 2.00

0.97 → Multi

Bias + 0.32 → Sum → * 1.41

0 → * 1.63 → * 1.65 → * 0.94

Input 0

* 4.38 → Sum → + 0.59

* 0.19

0.64

-0.20 → Multi → - 0.13

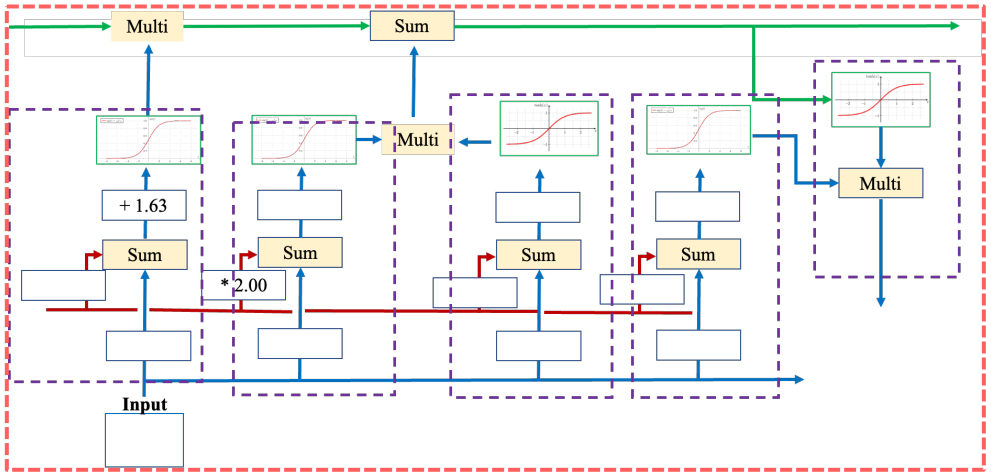
Output: New short-term memory => Output Gate

How much % Potential short-term memory to remember

Day 1 input

Initialize long term memory: 0

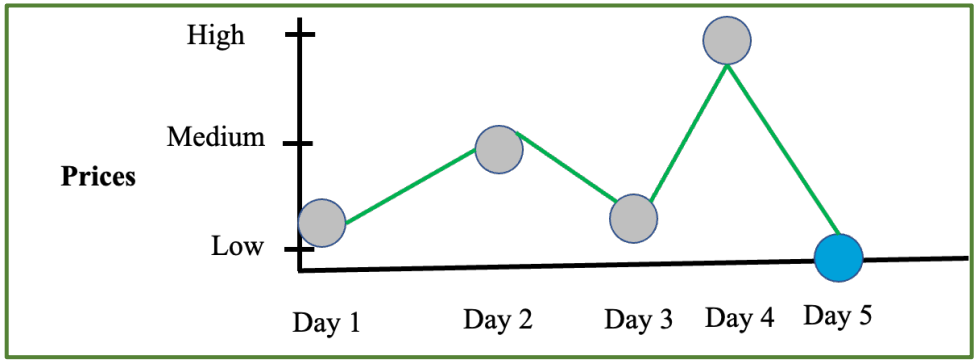
Initialize short term memory: 0



Updated long term memory: -0.2

Updated short term memory: -0.1

Day 1: 0

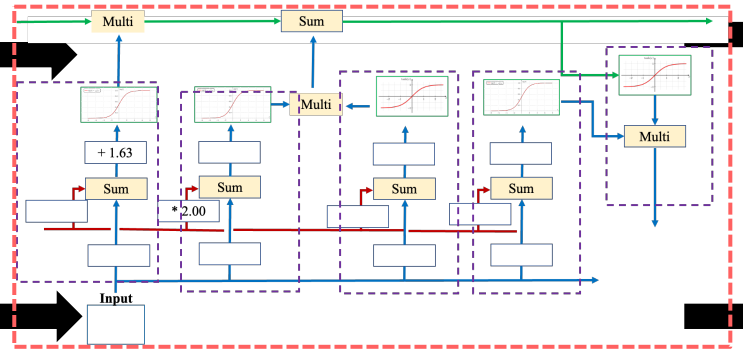


Day 2 input

Day 1 output

Initialize long term memory: -0.2

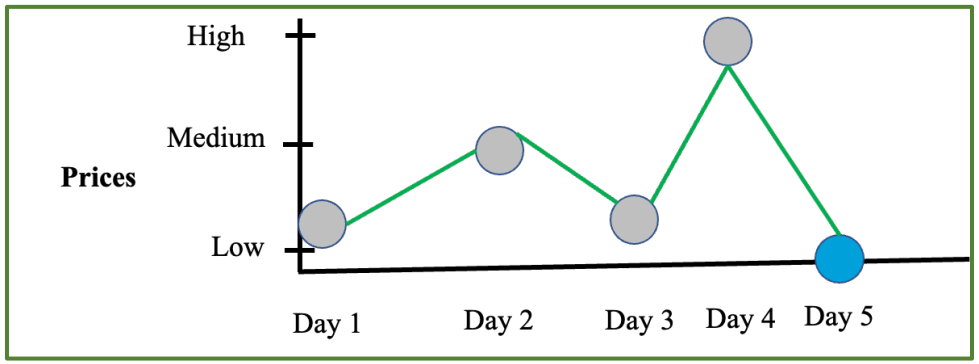
Initialize short term memory: -0.1



Updated long term memory: -0.2

Updated short term memory: -0.1

Day 2: 0.5



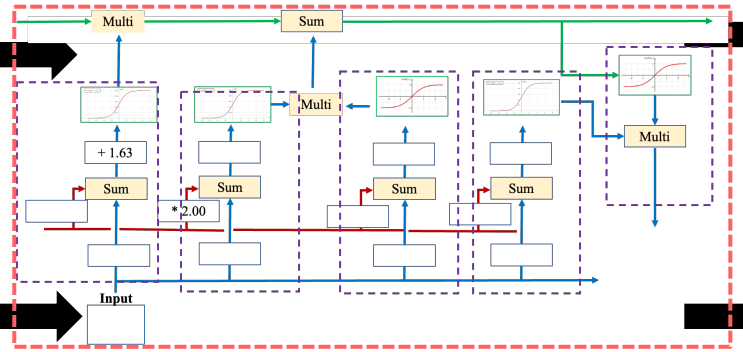
67

Day 3 input

Day 2 output

Initialize long term memory: -0.2

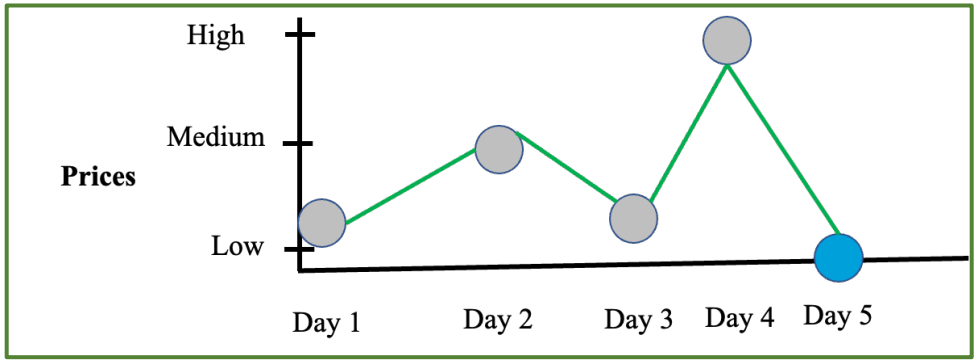
Initialize short term memory: -0.1



Updated long term memory: -0.3

Updated short term memory: -0.2

Day 2: 0.25

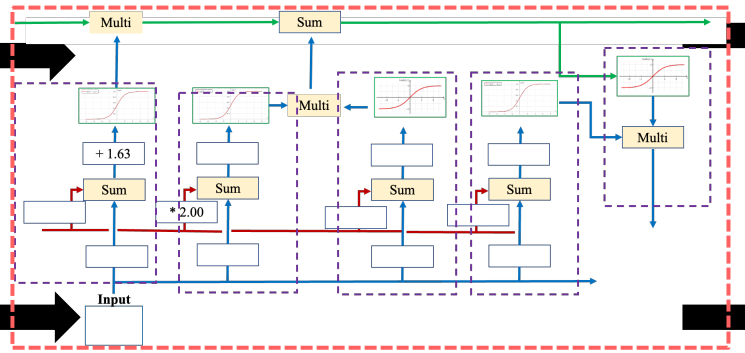


Day 4 input

Day 3 output

Initialize long term memory: -0.3

Initialize short term memory: -0.2

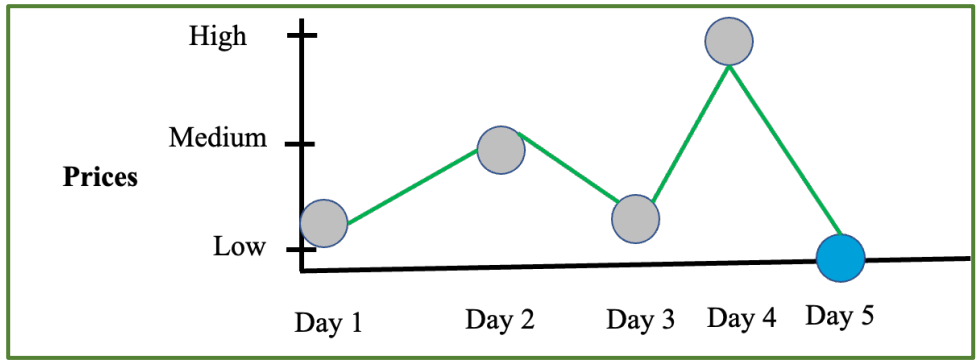


Updated long term memory: 0

Updated short term memory: 0

Predict value of day 5

Day 4: 1



LSTM : Time-series Data Project

Define LSTM Structure in Pytorch

```python
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers, ahead):
        super(LSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.ahead = ahead
        self.output_size = output_size

        # LSTM Layer
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)

        # Output layer
        self.fc = nn.Linear(hidden_size, output_size * ahead)

    def forward(self, x):
        # Initialize hidden and cell states
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        # Forward propagate LSTM
        out, _ = self.lstm(x, (h0, c0))

        # Get the last time step's output for each sequence
        out = out[:, -1, :]

        # Pass through the linear layer and reshape
        out = self.fc(out).view(-1, self.ahead, self.output_size)
        return out
```

Configuration

```python
input_size = 336
label_size = 96
offset = 1
train_size = 0.7
val_size = 0.1
num_epochs = 1_000_000
patience = 20
learning_rate = 0.001
hidden_size = 64
num_layers = 2
ele = 5
target_name = 'OT'
date_column = 'date'
file_path = 'ETTh1.csv'
plot_dir = 'plots'
weight_dir = 'weights'
results = []
```

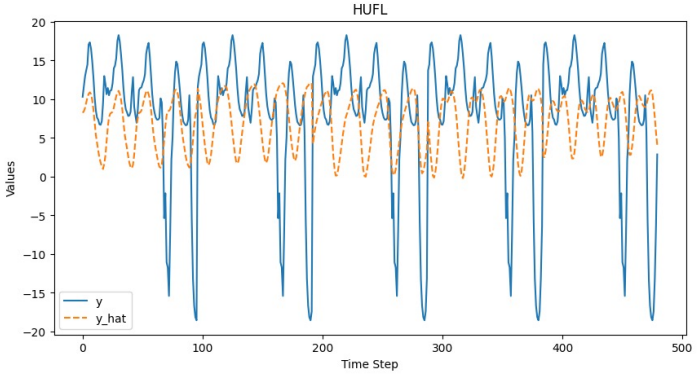
LSTM : Time-series Data Project

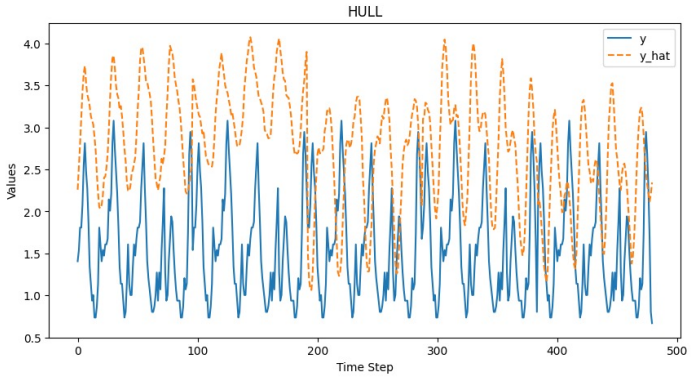
Define LSTM Structure in Pytorch

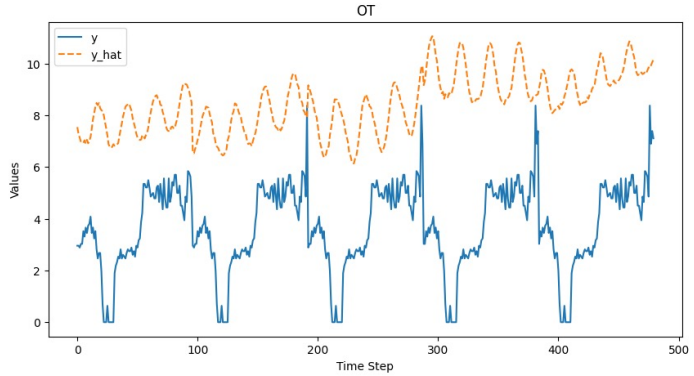
```
LSTM_multi2multi = LSTM(input_size=multi2multi_loader.in_variable,
                        hidden_size=hidden_size,
                        output_size=multi2multi_loader.out_variable,
                        ahead=label_size,
                        num_layers=num_layers)
LSTM_multi2multi_manager = ModelManager(model=LSTM_multi2multi,
                                        train_loader=multi2multi_loader.train_loader,
                                        val_loader=multi2multi_loader.val_loader,
                                        lr=learning_rate,
                                        patience=patience)
LSTM_multi2multi_manager.train(num_epochs=num_epochs,
                               save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": LSTM_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": LSTM_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
})
results[-1]
```

```
Model saved to weights\multi2multi\best-LSTM.pth
Epoch [1/1000000], time: 50s, loss: 3.6222, val_loss: 3.2154
Epoch [2/1000000], time: 49s, loss: 2.7683, val_loss: 3.6213
Epoch [3/1000000], time: 49s, loss: 2.7491, val_loss: 3.6428
Epoch [4/1000000], time: 49s, loss: 2.5998, val_loss: 3.5903
Epoch [5/1000000], time: 49s, loss: 2.3548, val_loss: 3.5180
Epoch [6/1000000], time: 49s, loss: 2.2306, val_loss: 3.2928
Model saved to weights\multi2multi\best-LSTM.pth
Epoch [7/1000000], time: 49s, loss: 2.1372, val_loss: 3.0611
Model saved to weights\multi2multi\best-LSTM.pth
```

Case 1: Multivariate to Multivariate Prediction







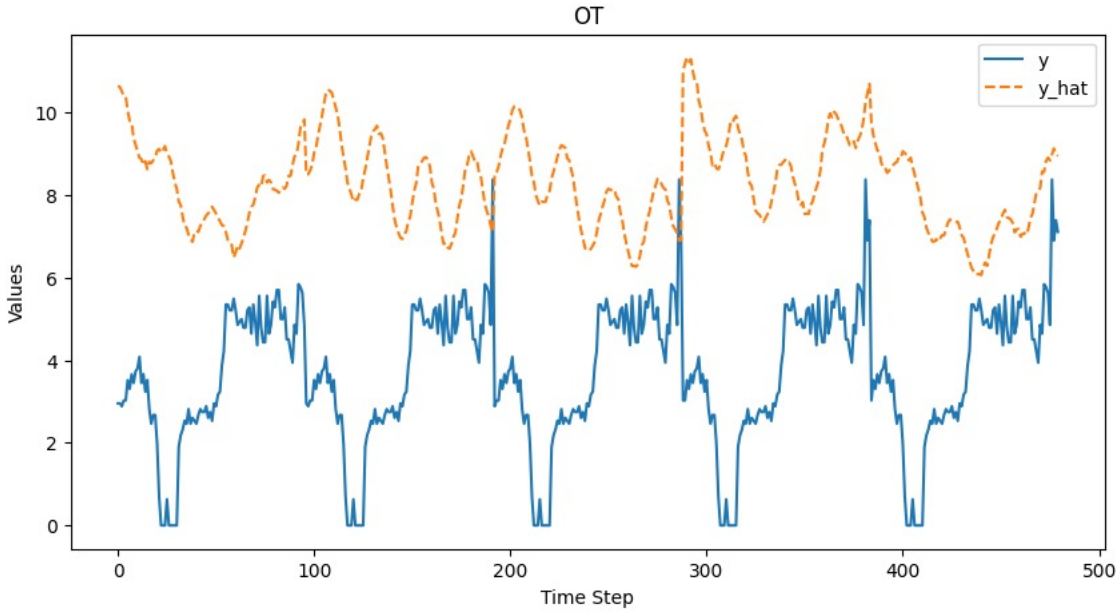
Experiments were perfomed by TA Khoa Nguyen

LSTM : Time-series Data Project

Define LSTM Structure in Pytorch

```
features_type='MS'
sub_dir = 'multi2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2uni_loader = TimeSeriesDataLoader(file_path,
                                        input_size=input_size,
                                        label_size=label_size,
                                        offset=offset,
                                        train_size=train_size,
                                        val_size=val_size,
                                        target_name=target_name,
                                        features_type=features_type,
                                        date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 7)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 7)
self.y_test.shape = (2861, 96, 1)
```



Case 2: Multivariate to Univariate Prediction

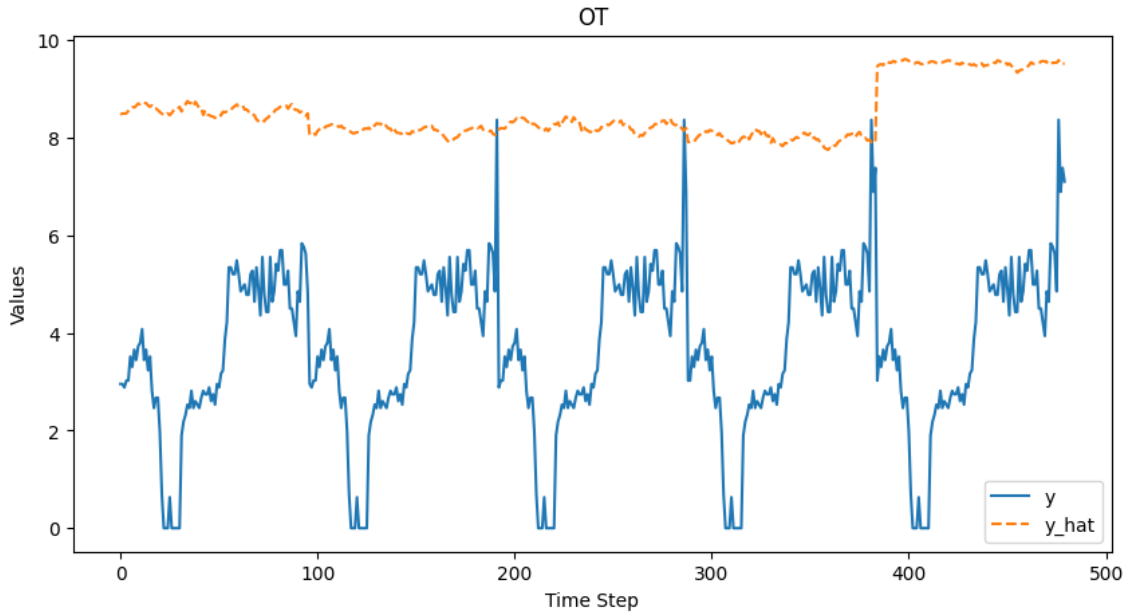
Experiments were perfomed by TA Khoa Nguyen

LSTM: Time-series Data Project

Define LSTM Structure in Pytorch

```
features_type='S'
sub_dir = 'uni2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
uni2uni_loader = TimeSeriesDataLoader(file_path,
                                      input_size=input_size,
                                      label_size=label_size,
                                      offset=offset,
                                      train_size=train_size,
                                      val_size=val_size,
                                      target_name=target_name,
                                      features_type=features_type,
                                      date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 1)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 1)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 1)
self.y_test.shape = (2861, 96, 1)
```



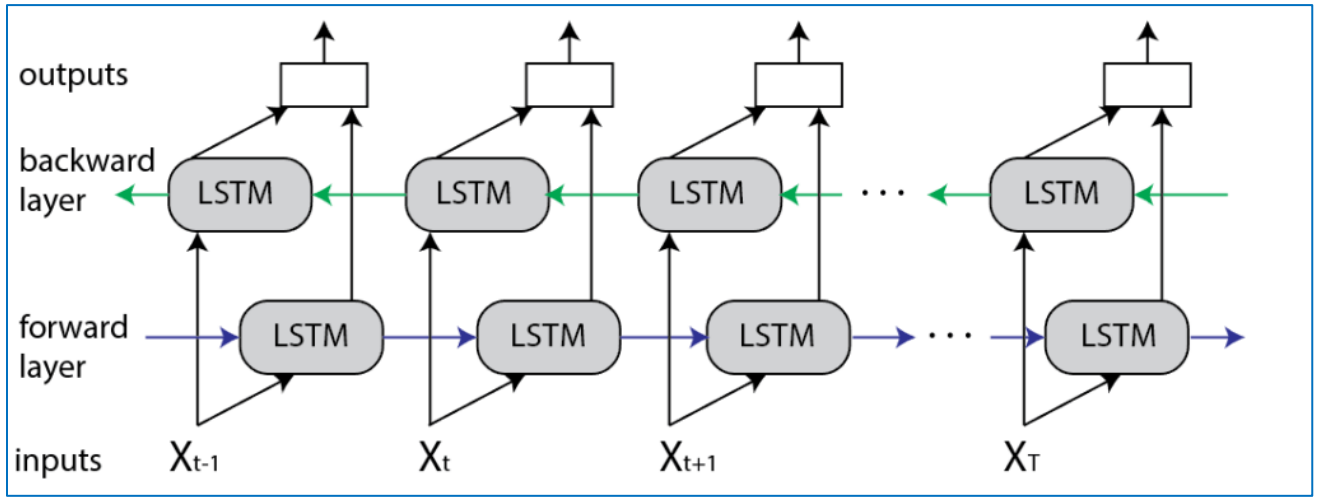
Case 3: Univariate to Univariate Prediction

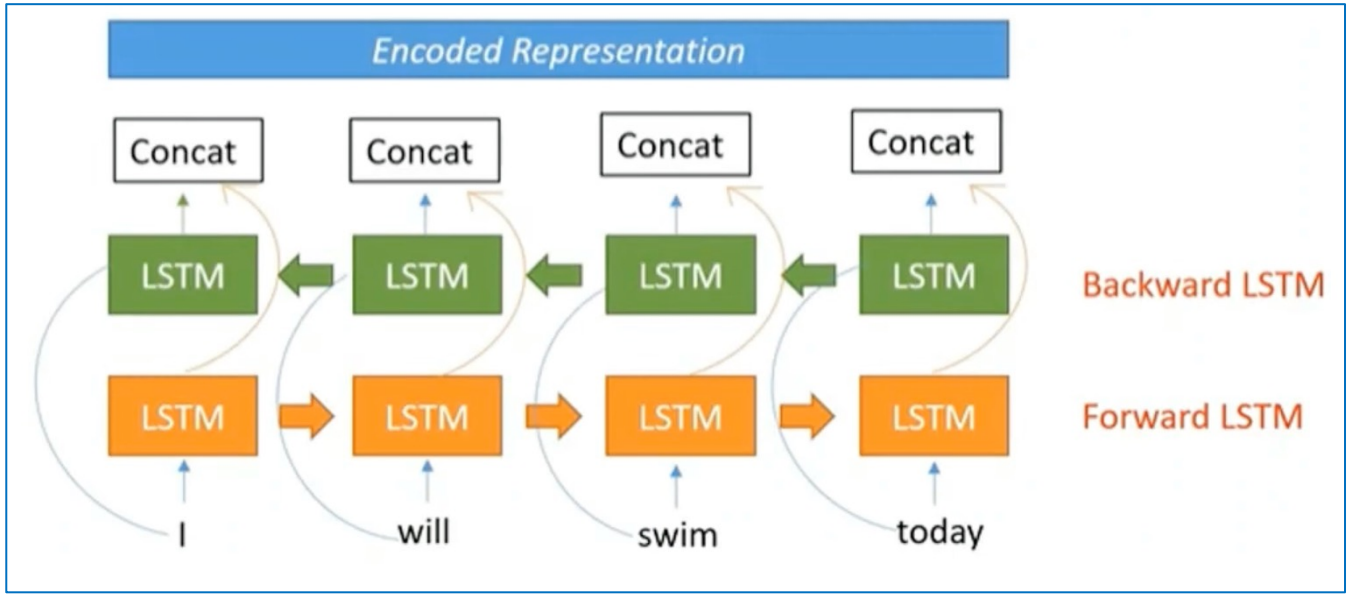
Experiments were perfomed by TA Khoa Nguyen

Outline

- Multilayer Perceptron and Time-series Data Forecasting
- RNN and Time-series Data Forecasting
- LSTM and Time-series Data Forecasting
- Bi-LSTM and Time-series Data Forecasting
- XGBoost and Time-series Data Forecasting

Bidirectional-LSTM: Simple Explanation





Bidirectional-LSTM: Simple Explanation

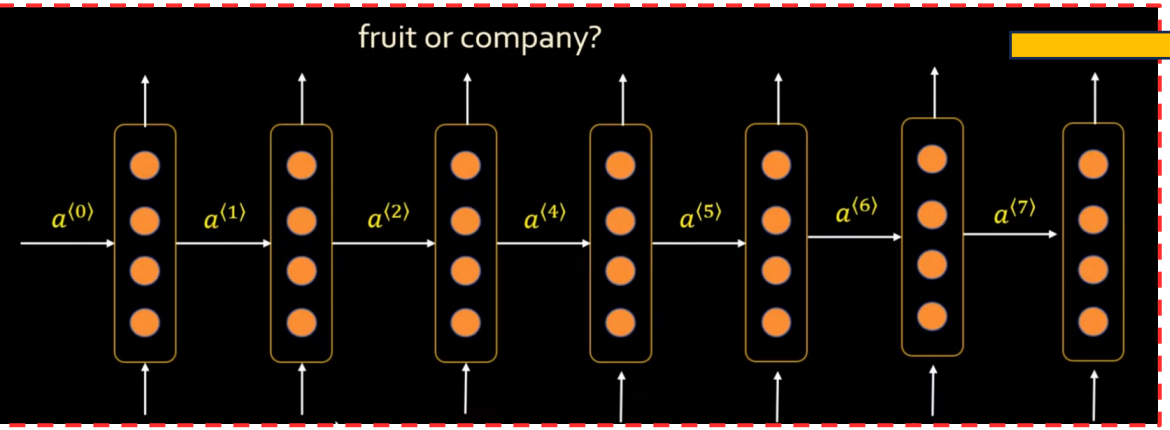
Fruit

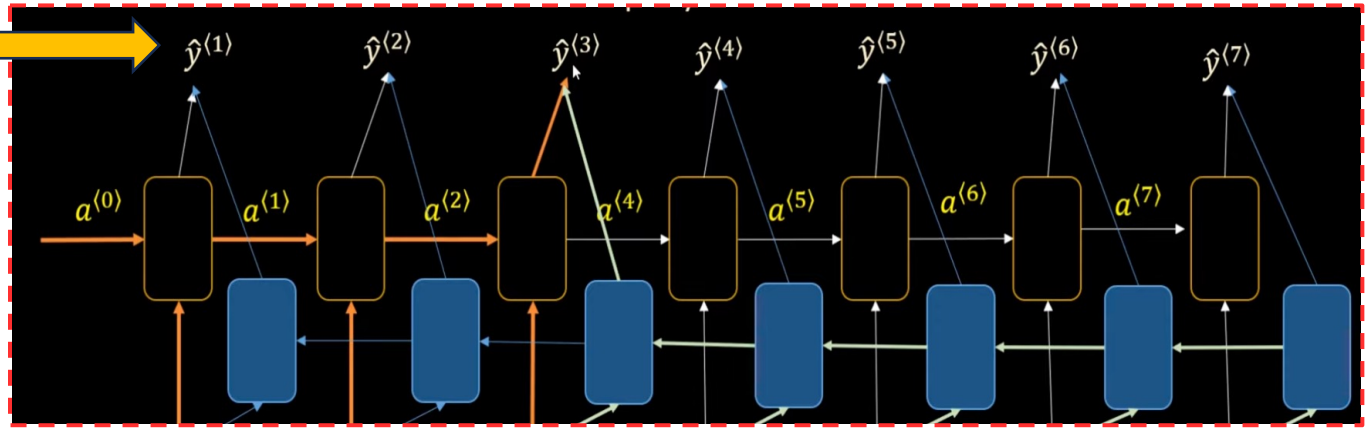
Vinh loves apple, it keeps him healthy

Person

Vinh loves apple, the company produces best electronics

Company

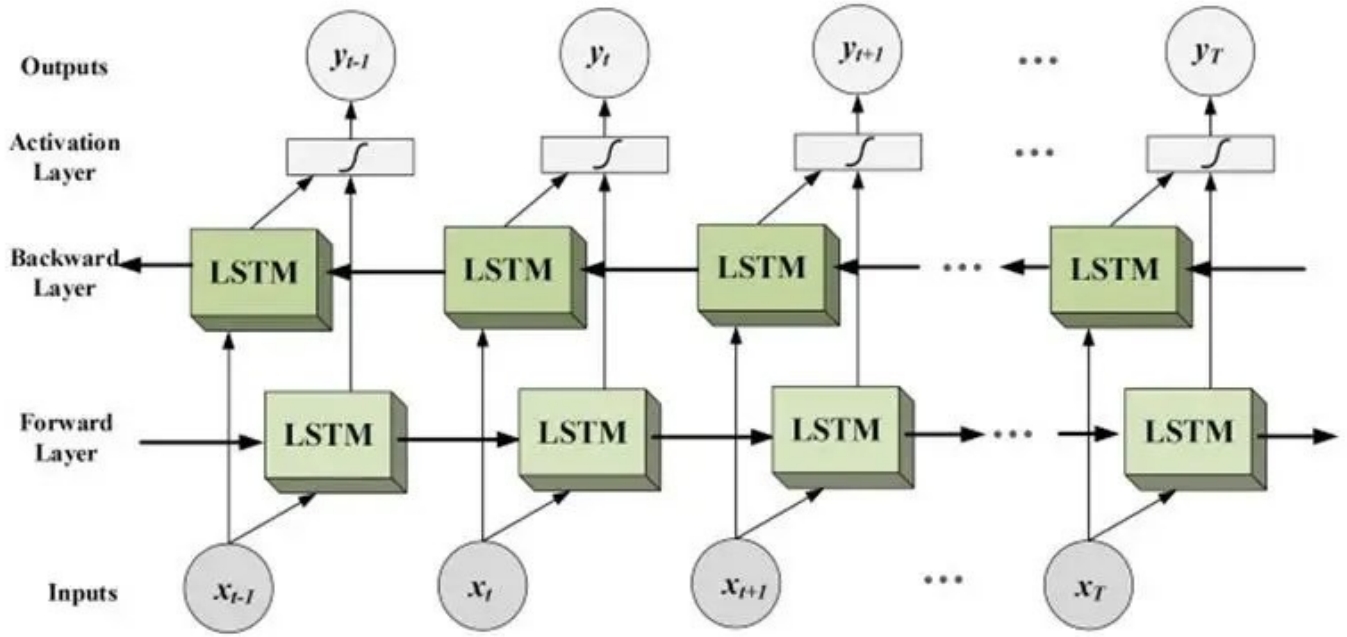




Bidirectional-LSTM: Simple Explanation

In the sentence “boys go to .....” we can not fill the blank space.

We have a future sentence “boys come out of school”



Bi-LSTM : Time-series Data Project

Define Bi-LSTM Structure in Pytorch

```python
class BiLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers, ahead):
        super(BiLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.ahead = ahead
        self.output_size = output_size

        # BiLSTM Layer
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, bidirectional=True)

        # Output layer
        # Output size is doubled because BiLSTM has two hidden states for each layer
        self.fc = nn.Linear(hidden_size * 2, output_size * ahead)

    def forward(self, x):
        # Initialize hidden and cell states
        h0 = torch.zeros(self.num_layers * 2, x.size(0), self.hidden_size).to(x.device)  # 2 for bidirec
        c0 = torch.zeros(self.num_layers * 2, x.size(0), self.hidden_size).to(x.device)

        # Forward propagate BiLSTM
        out, _ = self.lstm(x, (h0, c0))

        # Get the last time step's output for each sequence
        out = out[:, -1, :]

        # Pass through the linear layer and reshape
        out = self.fc(out).view(-1, self.ahead, self.output_size)
        return out
```

Configuration

```python
input_size = 336
label_size = 96
offset = 1
train_size = 0.7
val_size = 0.1
num_epochs = 1_000_000
patience = 20
learning_rate = 0.001
hidden_size = 64
num_layers = 2
ele = 5
target_name = 'OT'
date_column = 'date'
file_path = 'ETTh1.csv'
plot_dir = 'plots'
weight_dir = 'weights'
results = []
```

Bi-LSTM : Time-series Data Project

Define Bi-LSTM Structure in Pytorch

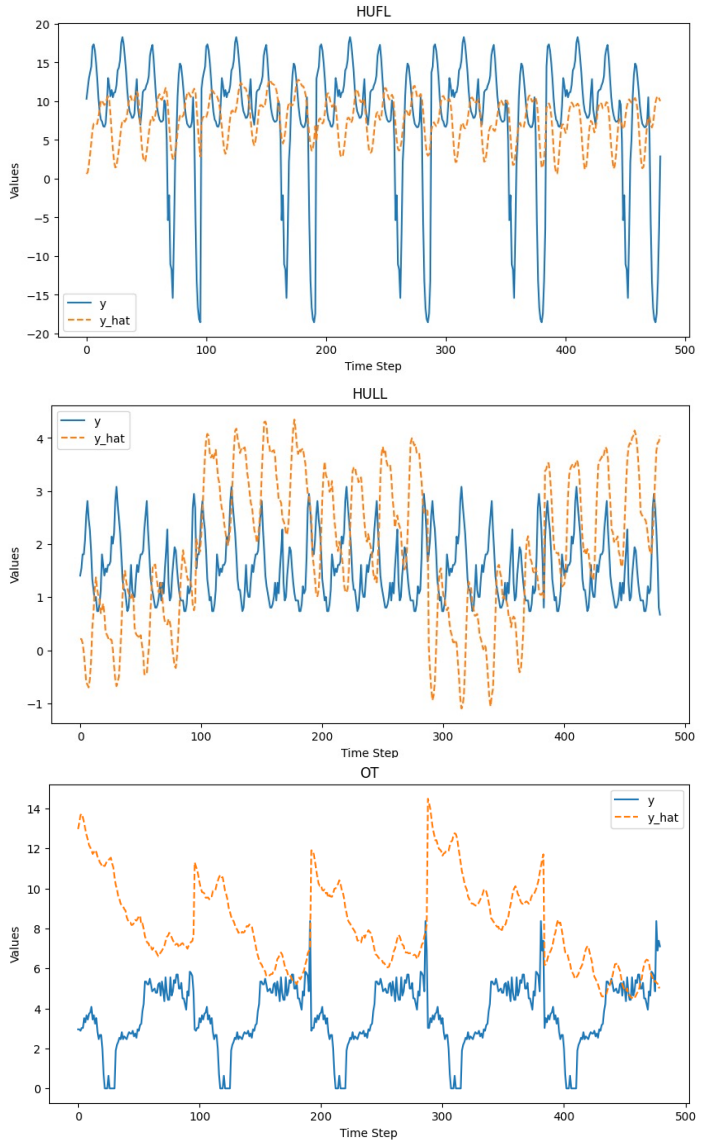
```
BiLSTM_multi2multi = BiLSTM(input_size=multi2multi_loader.in_variable,
                            hidden_size=hidden_size,
                            output_size=multi2multi_loader.out_variable,
                            ahead=label_size,
                            num_layers=num_layers)
BiLSTM_multi2multi_manager = ModelManager(model=BiLSTM_multi2multi,
                                          train_loader=multi2multi_loader.train_loader,
                                          val_loader=multi2multi_loader.val_loader,
                                          lr=learning_rate,
                                          patience=patience)
BiLSTM_multi2multi_manager.train(num_epochs=num_epochs,
                                 save_dir=os.path.join(weight_dir, sub_dir))

results.append({
    "Name": BiLSTM_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": BiLSTM_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
})
results[-1]
```

Loading...

```
Model saved to weights\multi2multi\best-BiLSTM.pth
Epoch [1/1000000], time: 101s, loss: 3.3011, val_loss: 3.5839
Model saved to weights\multi2multi\best-BiLSTM.pth
Epoch [2/1000000], time: 101s, loss: 2.5628, val_loss: 3.5039
Model saved to weights\multi2multi\best-BiLSTM.pth
```

Case 1: Multivariate to Multivariate Prediction



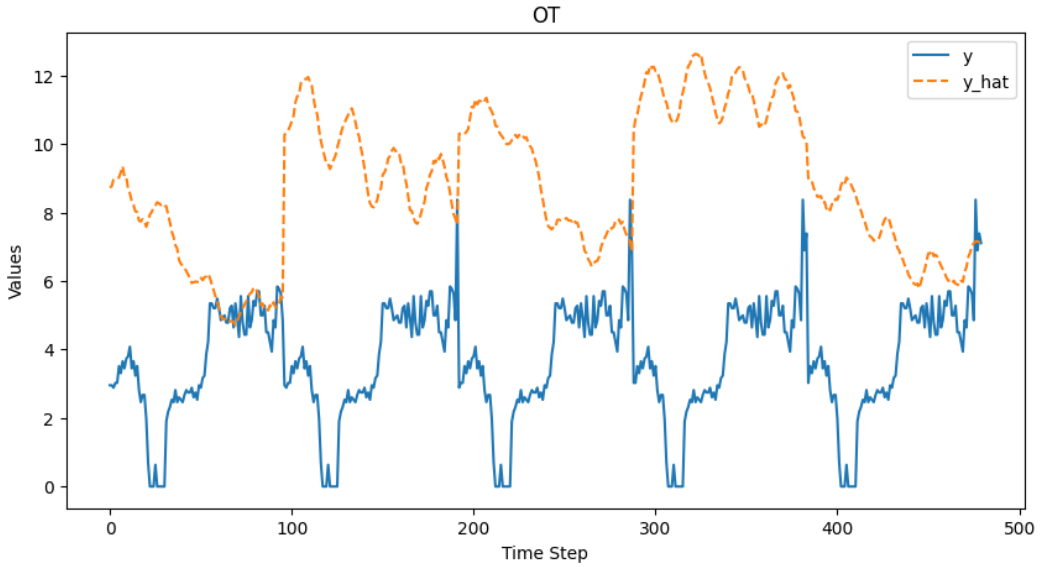
Experiments were perfomed by TA Khoa Nguyen

Bi-LSTM : Time-series Data Project

Define Bi-LSTM Structure in Pytorch

```
features_type='MS'
sub_dir = 'multi2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2uni_loader = TimeSeriesDataLoader(file_path,
                                        input_size=input_size,
                                        label_size=label_size,
                                        offset=offset,
                                        train_size=train_size,
                                        val_size=val_size,
                                        target_name=target_name,
                                        features_type=features_type,
                                        date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 7)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 7)
self.y_test.shape = (2861, 96, 1)
```



Case 2: Multivariate to Univariate Prediction

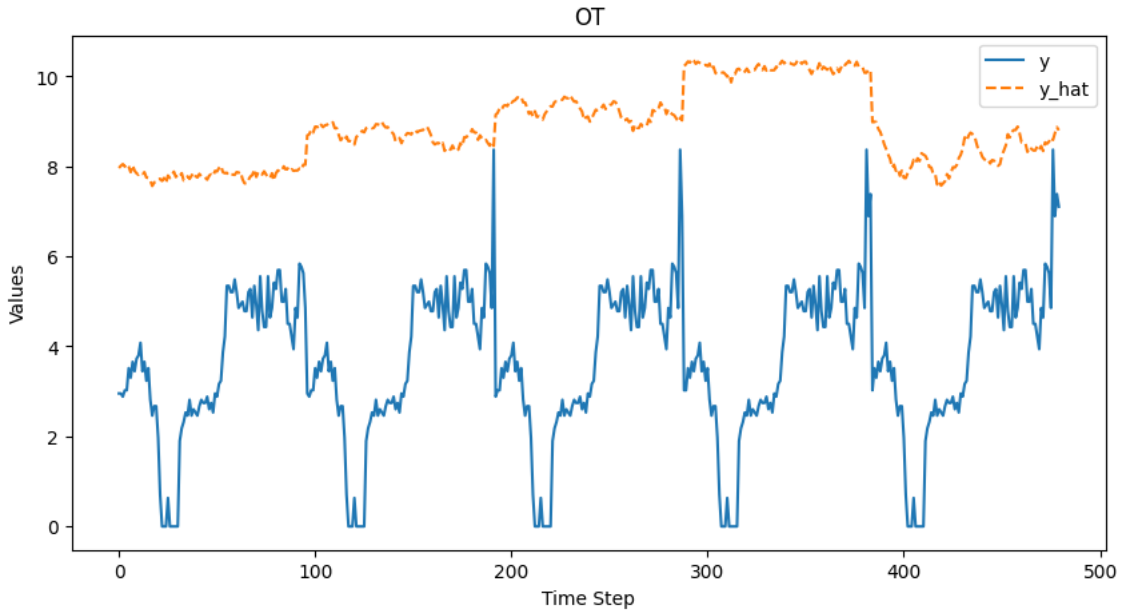
Experiments were perfomed by TA Khoa Nguyen

Bi-LSTM : Time-series Data Project

Define Bi-LSTM Structure in Pytorch

```
features_type='S'
sub_dir = 'uni2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
uni2uni_loader = TimeSeriesDataLoader(file_path,
                                      input_size=input_size,
                                      label_size=label_size,
                                      offset=offset,
                                      train_size=train_size,
                                      val_size=val_size,
                                      target_name=target_name,
                                      features_type=features_type,
                                      date_column=date_column)
```

```
Offset will be change from 1 to 96
self.X_train.shape = (11762, 336, 1)
self.y_train.shape = (11762, 96, 1)
self.X_val.shape = (1309, 336, 1)
self.y_val.shape = (1309, 96, 1)
self.X_test.shape = (2861, 336, 1)
self.y_test.shape = (2861, 96, 1)
```



Case 3: Univariate to Univariate Prediction

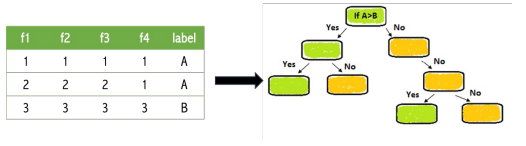
Experiments were perfomed by TA Khoa Nguyen

Outline

- Multilayer Perceptron and Time-series Data Forecasting
- RNN and Time-series Data Forecasting
- LSTM and Time-series Data Forecasting
- Bi-LSTM and Time-series Data Forecasting
- XGBoost and Time-series Data Forecasting

Evolution of Tree and Its Variant

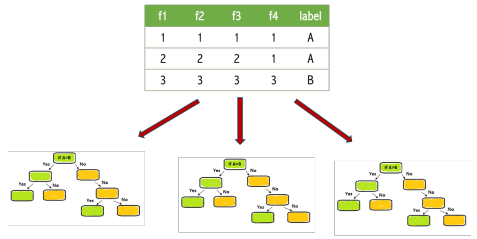
Decision Tree



$$Gini = 1 - \sum_j p_j^2 \qquad H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

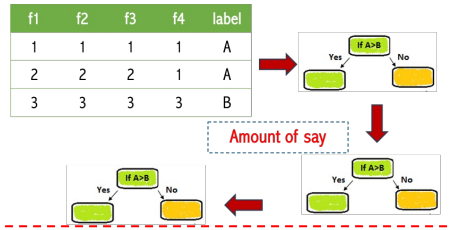
Tree is created by using GNI or Entropy metrics

Random Forest



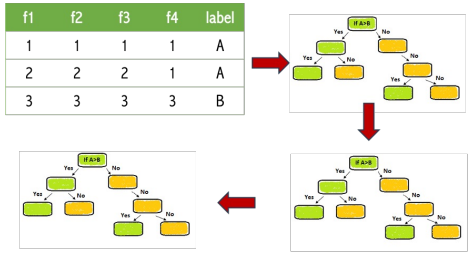
Tree are created independently
Same contribution of each tree to the final prediction

Adaboost



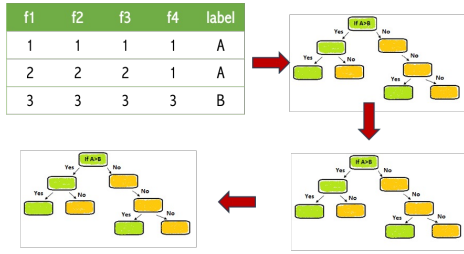
Stumps are created sequentially.
Different contribution of each stump to the final prediction

Gradient Boosting



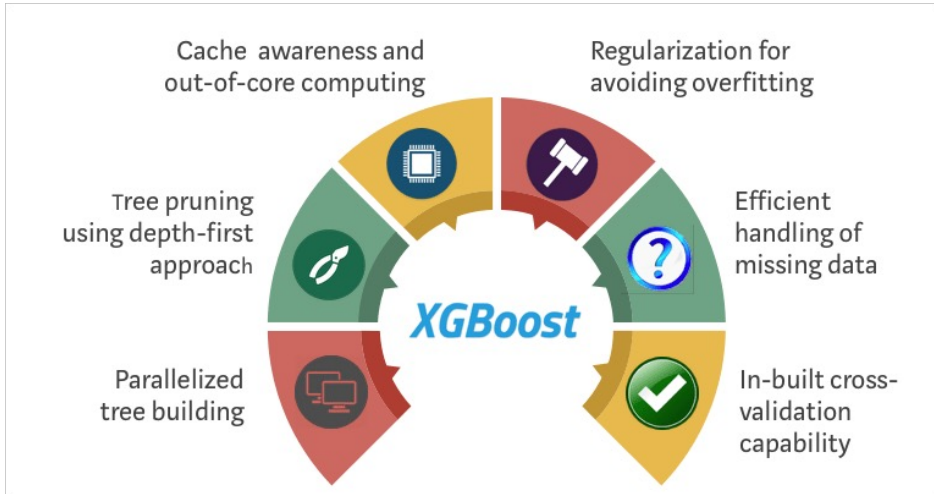
Made up of Gradient descent and Boosting.
Minimize the cost function of the ensemble

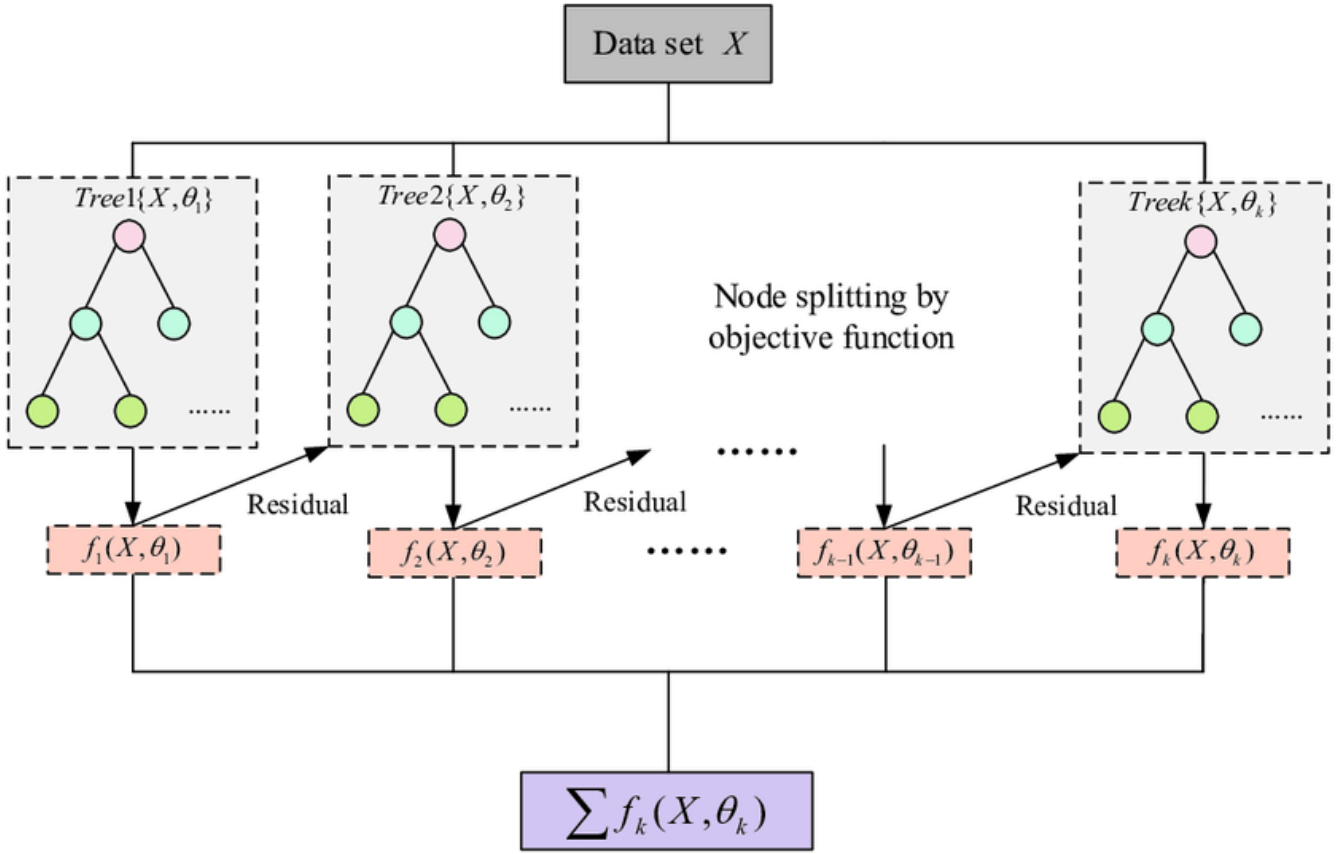
XGBoost



Regularization
Approximate Greed Algorithm
Parallel Learning
Weighted Quantile Sketching
Sparsity-Aware Finding
Cache-Aware Access

XGBoost: Simple Explanation





XGBoost: Simple Explanation

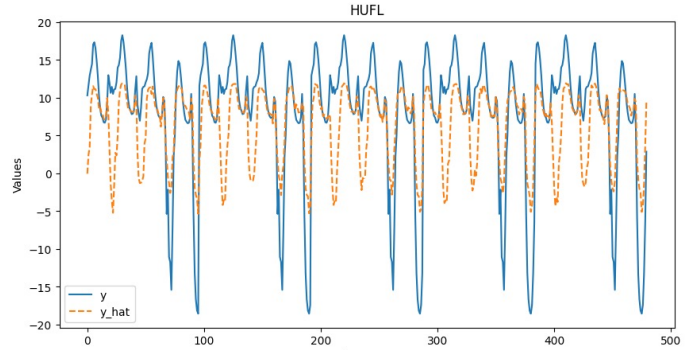
- Gradient Boost
- Regularization
- Approximate Greedy Algorithm
- Parallel Learning
- Weighted Quantile Sketch
- Sparsity-Aware Split Finding
- Cache-Aware Access

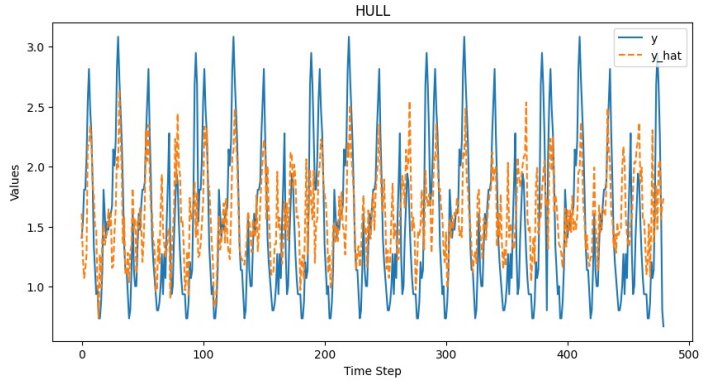
XGBoost : Time-series Data Project

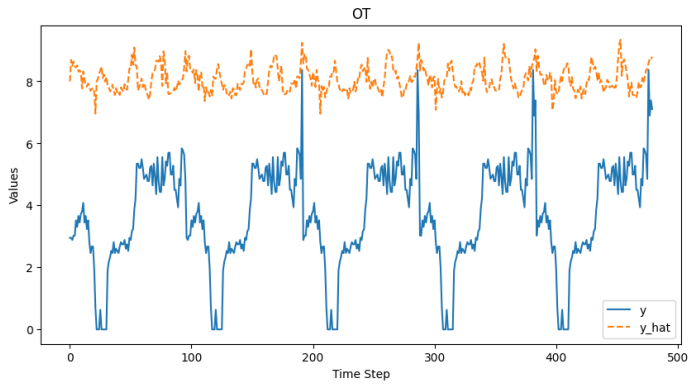
Define XGBoost Structure in Pytorch

```
XGBoost_multi2multi = XGBRegressor(**xgboost_config)
XGBoost_multi2multi_manager = MachineLearningModelManager(model=XGBoost_multi2multi,
                                                          xtrain=multi2multi_loader.X_train,
                                                          ytrain=multi2multi_loader.y_train,
                                                          xval=multi2multi_loader.X_val,
                                                          yval=multi2multi_loader.y_val)
XGBoost_multi2multi_manager.train(save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": XGBoost_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": XGBoost_multi2multi_manager.evaluate(x=multi2multi_loader.X_test, y=multi2multi_loader.y_test
})
results[-1]
```

```
[0]     validation_0-mae:3.90950
[1]     validation_0-mae:3.61946
[2]     validation_0-mae:3.38305
[3]     validation_0-mae:3.19688
[4]     validation_0-mae:3.03972
```



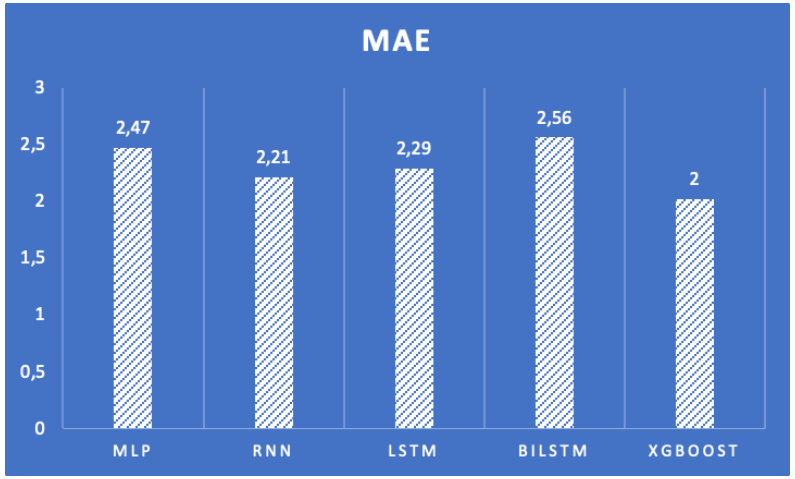




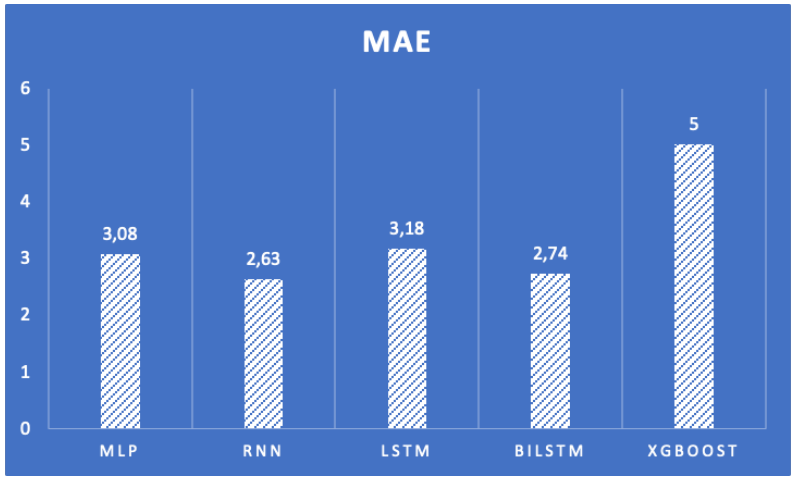
Multivariate to Multivariate Prediction

Experiments were perfomed by TA Khoa Nguyen

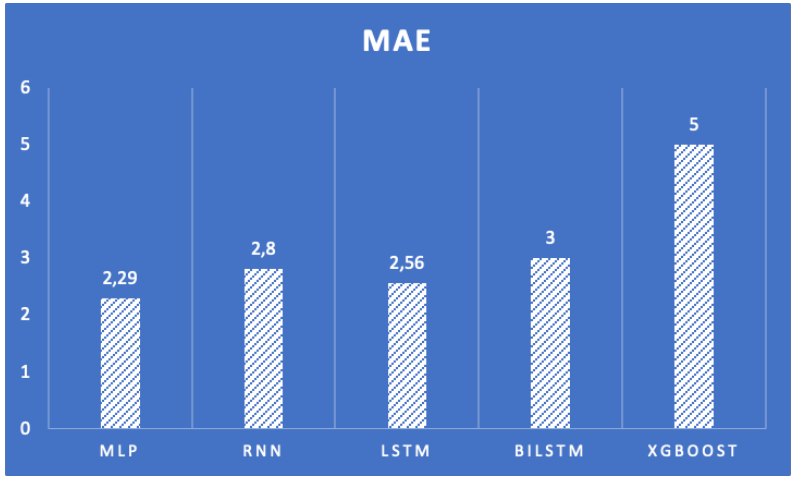
Performance Evaluation



Multivariate to Multivariate



Multivariate to Univariate



Multivariate to Univariate

Further Reading

Publisher	Year	Paper	Model
AAAI	2023	Are Transformers Effective for Time Series Forecasting?	NLinearDLinear
ICLR	2023	A TIME SERIES IS WORTH 64 WORDS: LONG-TERM FORECASTING WITH TRANSFORMERS	PatchTST
	2023	iTransformer: Inverted Transformers Are Effective for Time Series Forecasting	iTransformer
ICLR	2023	TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis	TimesNet
	2023	An adaptive standardisation model for Day-Ahead electricity price forecasting	None
ICLR	2023	MICN: Multi-scale Local and Global Context Modeling for Long-term Series Forecasting	MICN
PeerJ Computer Science	2023	Multi-horizon short-term load forecasting using hybrid of LSTM and modified split convolution	LSTM-SC
	2023	PatchMixer: A Patch-Mixing Architecture for Long-Term Time Series Forecasting	PatchMixer
	2023	SegRNN: Segment Recurrent Neural Network for Long-Term Time Series Forecasting	SegRNN
TMLR	2023	TSMixer: An All-MLP Architecture for Time Series Forecasting	TSMixer
NeurIPS	2023	One Fits All: Power General Time Series Analysis by Pretrained LM	None
	2023	Distilling Universal and Joint Knowledge for Cross-Domain Model Compression on Time Series Data	UNI-KD
	2023	TSMixer: An All-MLP Architecture for Time Series Fore casting	TSMixer
IEEE	2023	Prophet: Prompting Large Language Models with Complementary Answer Heuristics for Knowledge-based Visual Question Answering	Prophet
	2023	Alternative Telescopic Displacement: An Efficient Multimodal Alignment Method	None
	2023	Long-term Forecasting with TiDE: Time-series Dense Encoder	TiDE
NeurIPS	2022	Non-stationary Transformers: Exploring the Stationarity in Time Series Forecasting	Stationary
ICML	2022	FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting	FEDformer
NeurIPS	2022	SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction	SCINet
ICLR	2022	CROSSFORMER: A VERSATILE VISION TRANSFORMER HINGING ON CROSS-SCALE ATTENTION	Crossformer
	2022	Less Is More: Fast Multivariate Time Series Forecasting with Light Sampling-oriented MLP Structures	LightTS
	2022	N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting	N-HiTS
	2022	Self-Supervised Transformer for Sparse and Irregularly Sampled Multivariate Clinical Time-Series	STraTS
NeurIPS	2022	WaveBound: Dynamic Error Bounds for Stable Time Series Forecasting	None
	2022	ETSformer: Exponential Smoothing Transformers for Time-series Forecasting	ETSformer
NeurIPS	2021	Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting	Autoformer
AAAI	2021	Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting	Informer
CIKM	2021	AdaRNN: Adaptive Learning and Forecasting of Time Series	AdaRNN
	2021	Prediction of the Position of External Markers Using a Recurrent Neural Network Trained With Unbiased Online Recurrent Optimization for Safe Lung Cancer Radiotherapy	UORO
	2021	Long-term series forecasting with Query Selector -- efficient model of sparse attention	QuerySelector
	2021	A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting	ES-RNN
	2021	Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting	TimeGrad
	2021	Systematic Generalization in Neural Networks-based Multivariate Time Series Forecasting Models	None
	2021	ES-dRNN: A Hybrid Exponential Smoothing and Dilated Recurrent Neural Network Model for Short-Term Load Forecasting	ES-dRNN
ICLR	2021	Multivariate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows	LSTM-MAF
	2020	Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting	TFT
NeurIPS	2020	Normalizing Kalman Filters for Multivariate Time Series Analysis	NKF

https://docs.google.com/spreadsheets/d/17J6ldQogp186AZDZv8ZVlBF5s2EaeZwQ/edit?usp=sharing&ouid=118245386452054287147&rtpof=true&sd=true


Thank
you