GATE RECURRENT UNIT, LONG SHORT-TERM MEMORY

Asst.Prof.Dr.Supakit Nootyaskool

IT-KMITL

Topics

- Concept of Gate Recurrent Unit
- Concept of Long Short-Term Memory Unit
- Modify trainr() for GRU and LSTM
- Development trainr()
- Example LSTM in C language

เปลี่ยน นิดหน่อย
RNN → GRU/LSTM ได้

REVIEW
RECURRENT NEURAL NETWORK

Define vectors

0 0 1

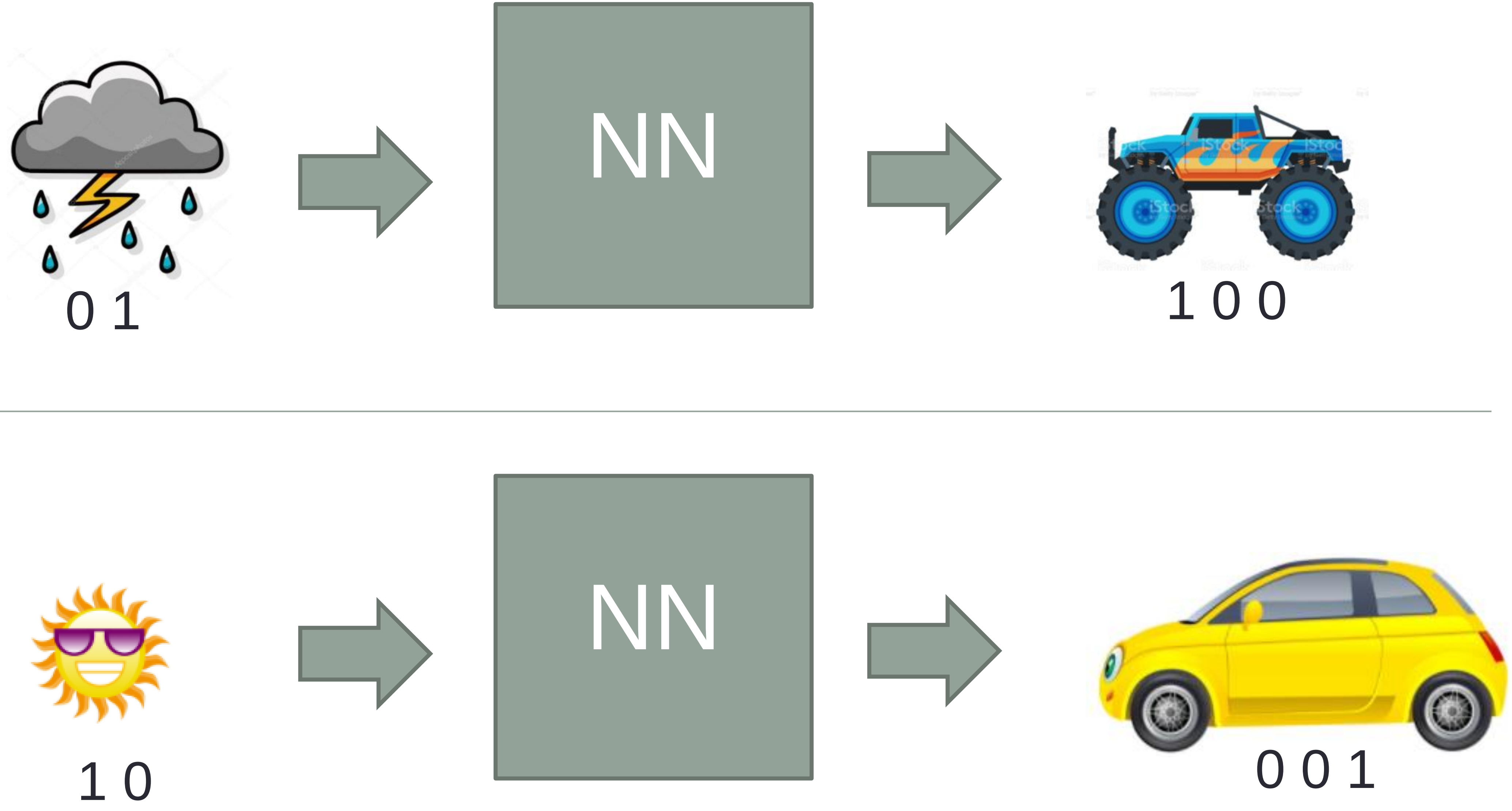
0 1 0

1 0 0

0 1

1 0

NN



0 1

NN

1 0 0

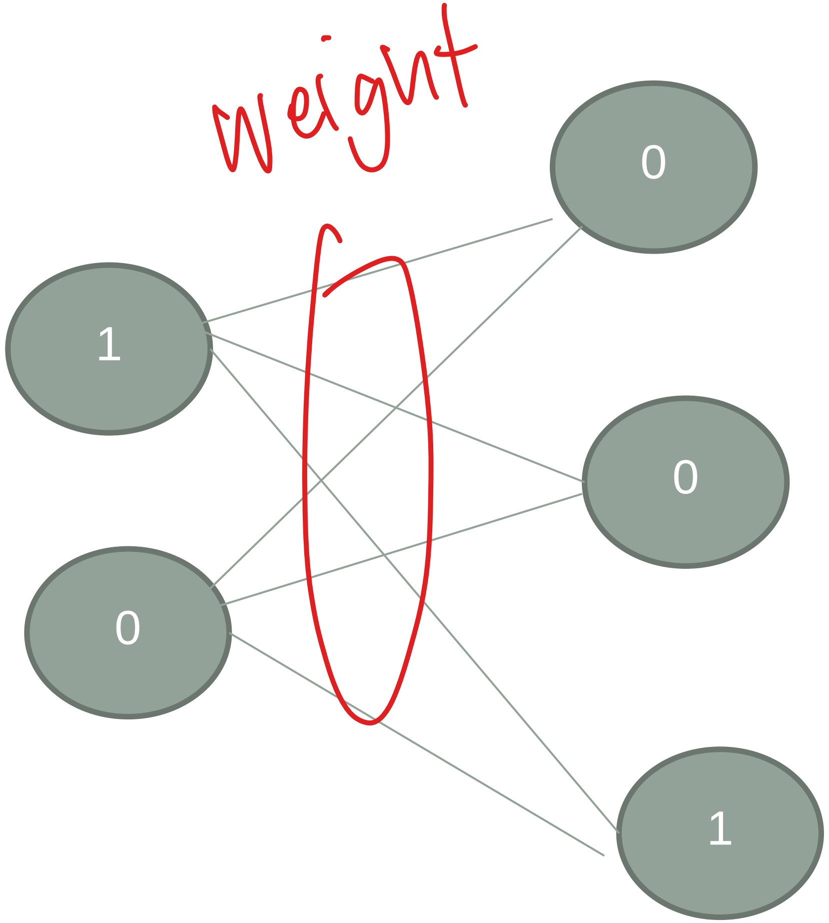
1 0

NN

0 0 1

NN

1 0



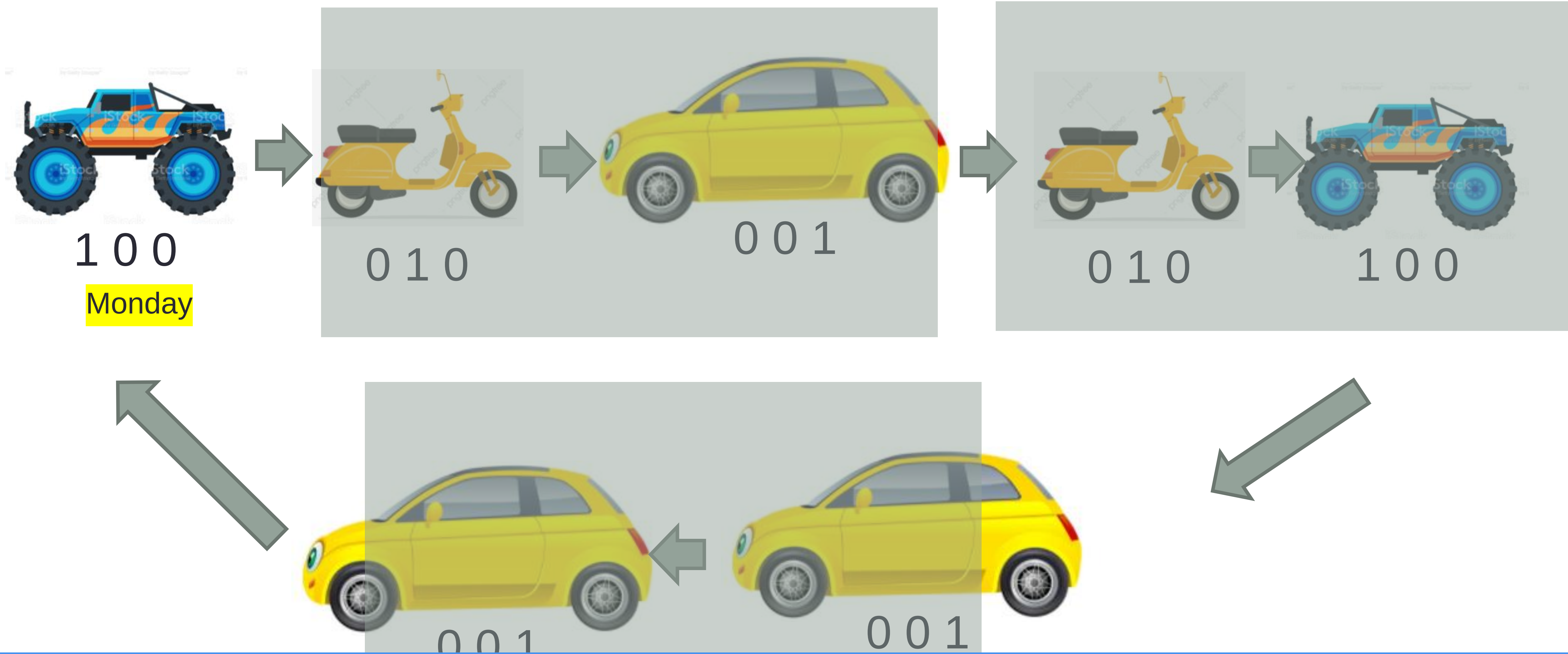
1 0 0

0 1 0

0 0 1

Driving Schedule

Sequence → Time



1 0 0

Monday

0 1 0

0 0 1

0 1 0

1 0 0

0 0 1

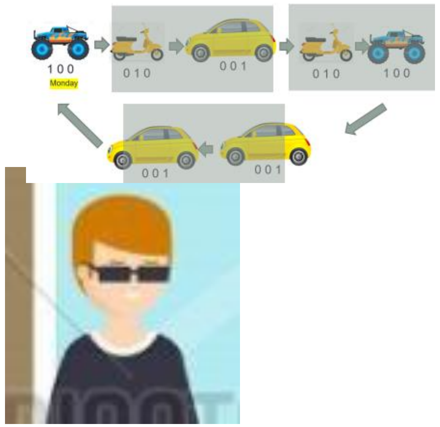
0 0 1

อดีต
พยากรณ์อากาศ

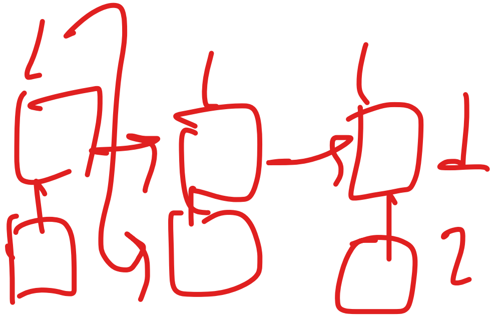
1 0

Yesterday

State
ทรงจำรถ เมื่อวาน



1 0 0



อดีต 1, 2
ผลลัพธ์ ต่าง

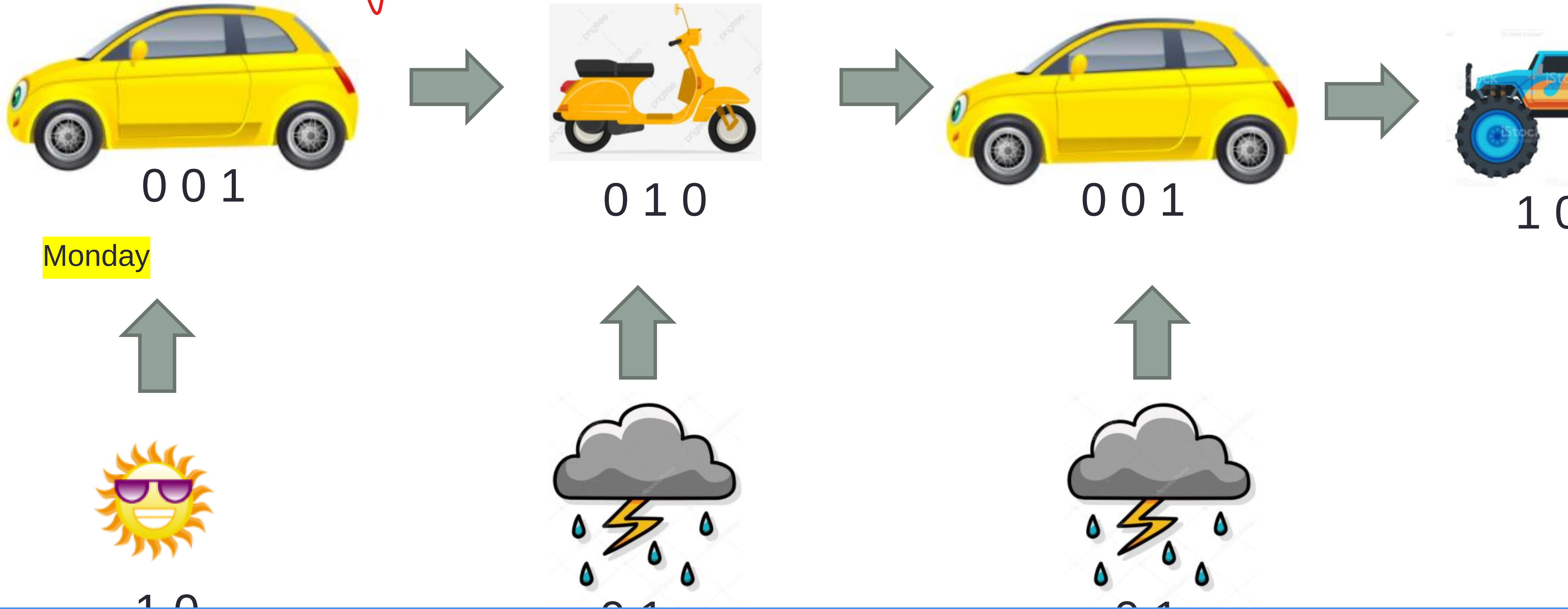
0 1 0

0 1

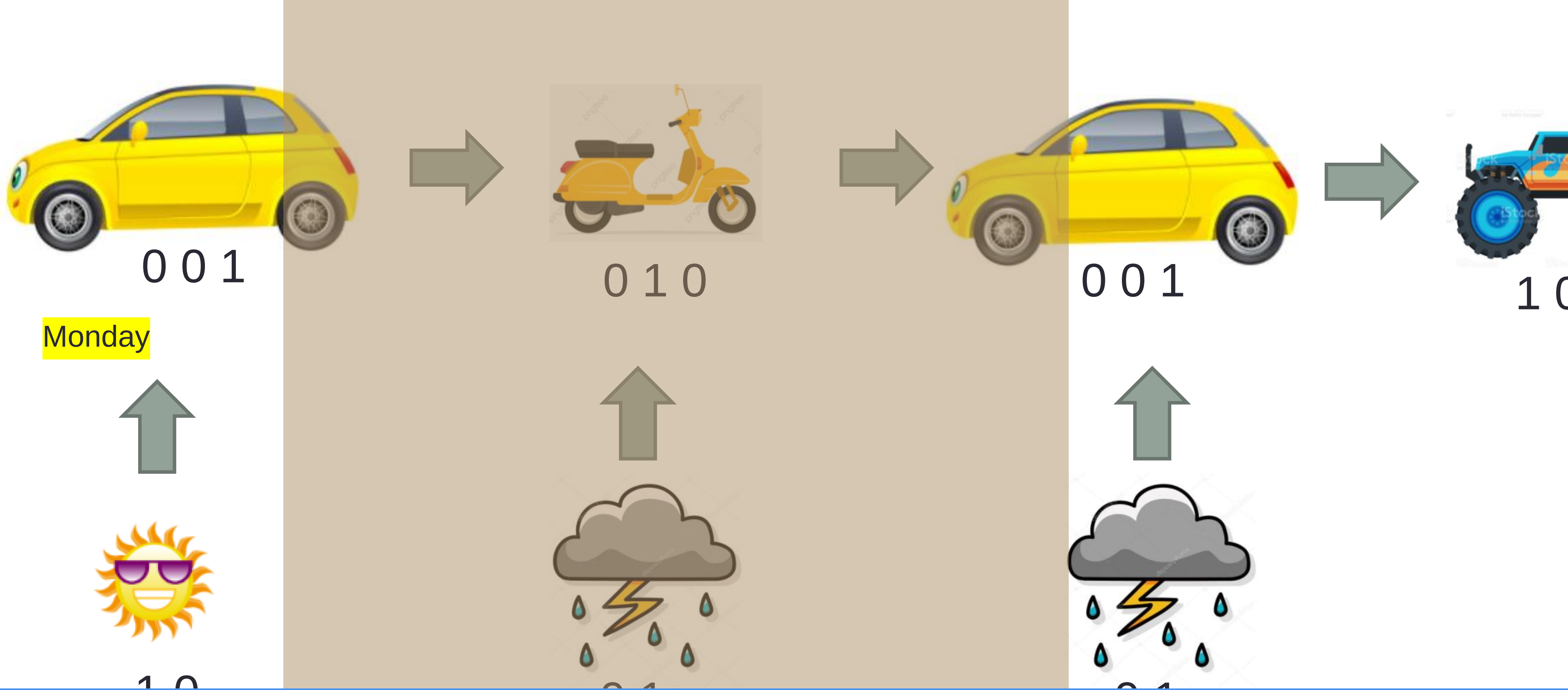
Today

Driving on Thursday

{ state เดิม



Driving on Thursday



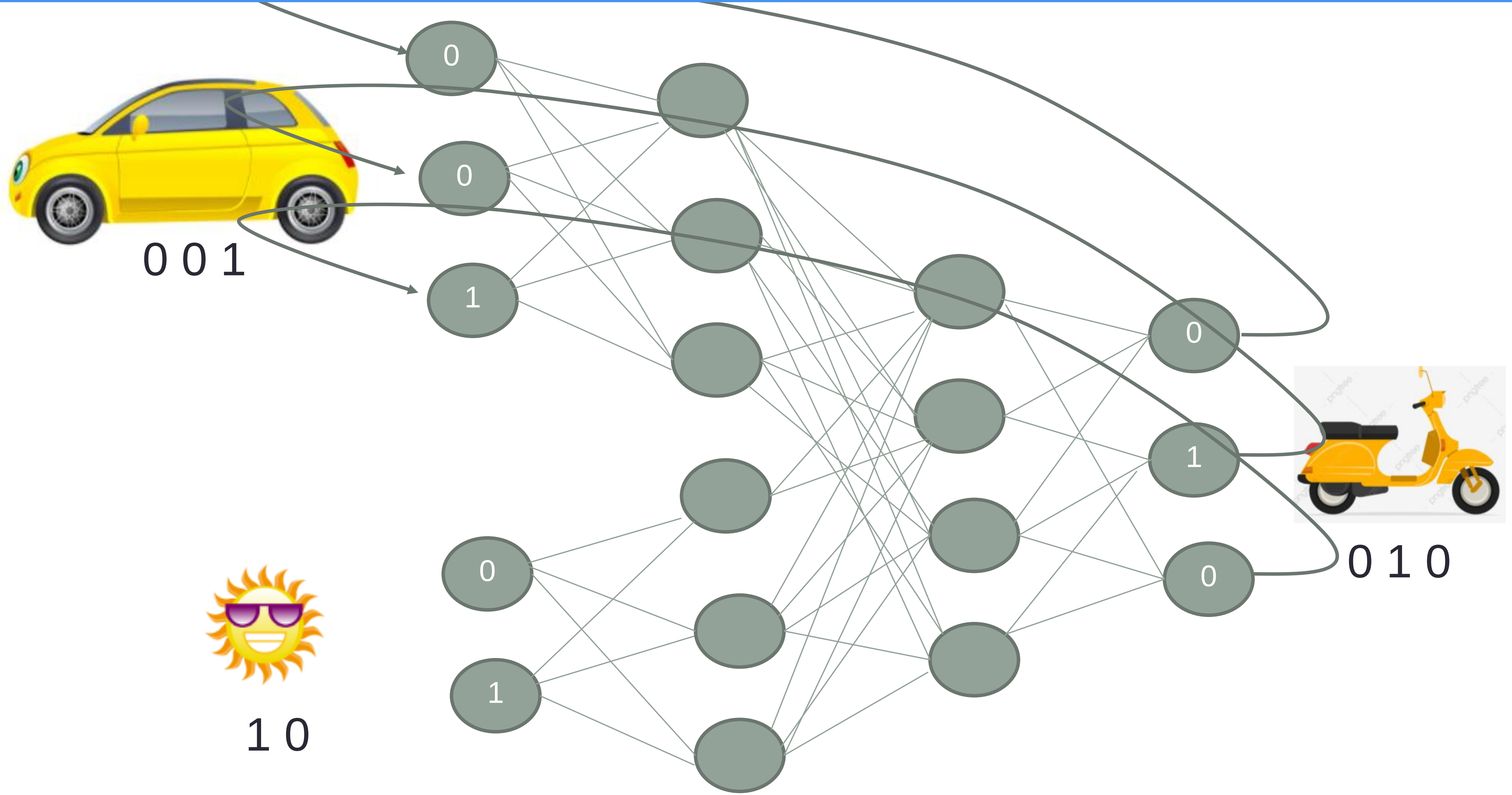
0 0 1

Monday

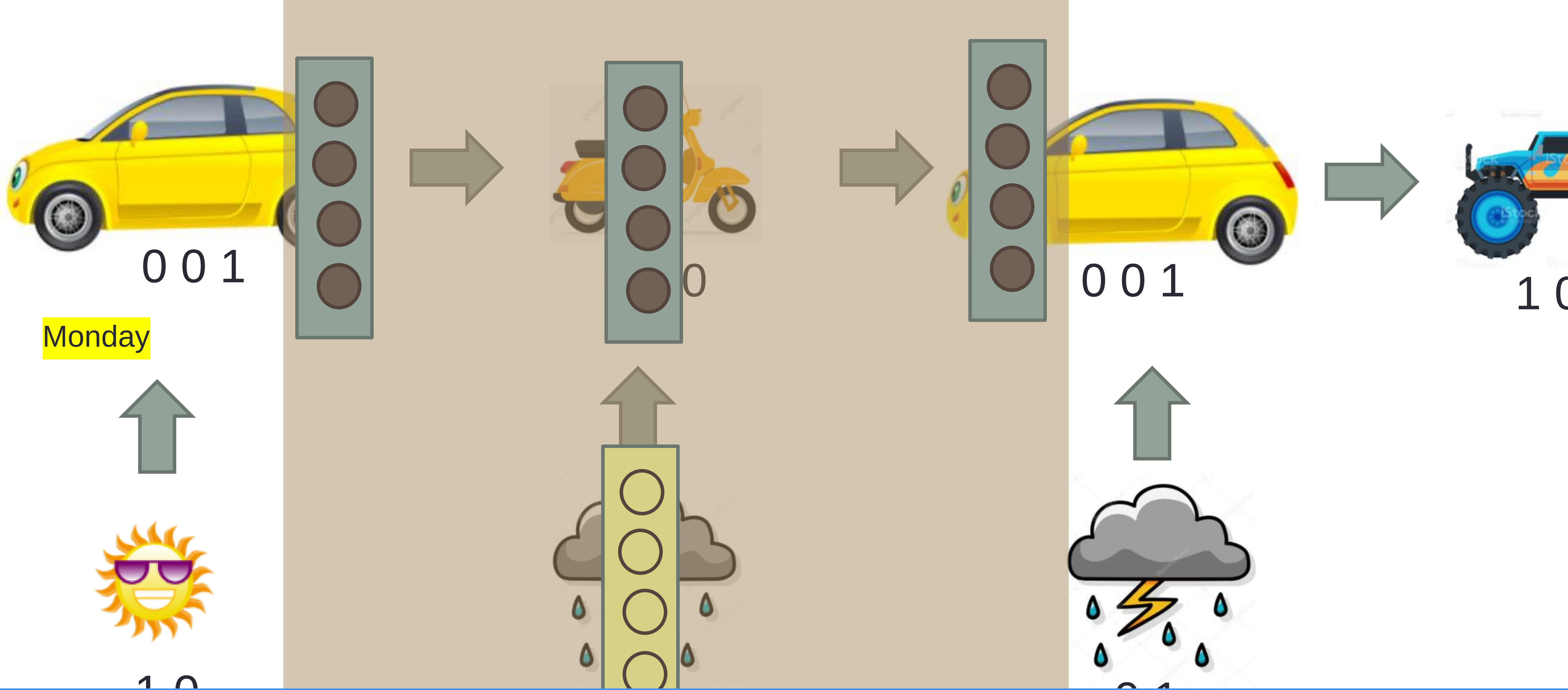
0 1 0

0 0 1

1 0


0
0
1
0 0 1
0
1
1 0
0
1
0
0 1 0

Driving on Thursday



0 0 1

Monday

0

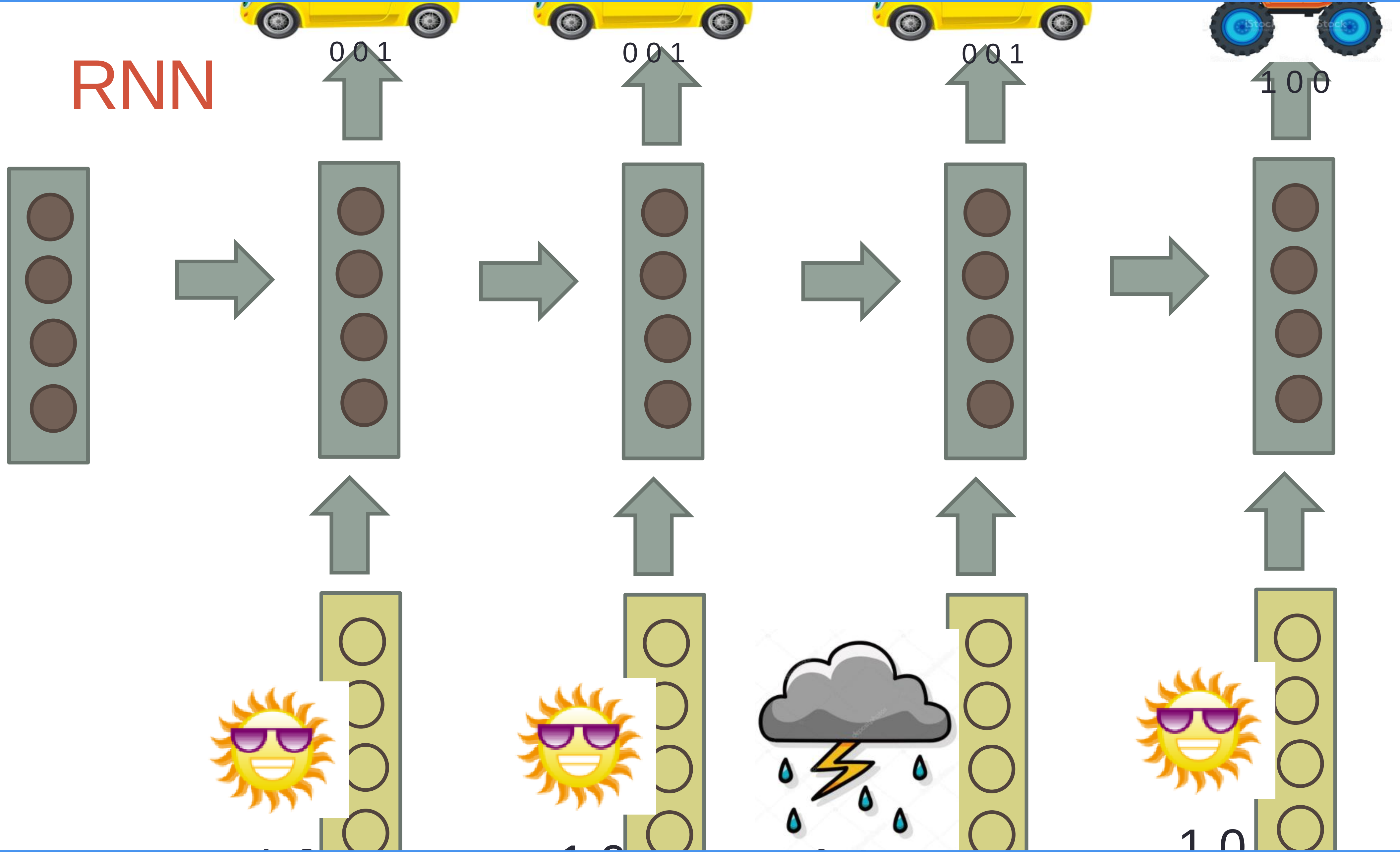
0 0 1

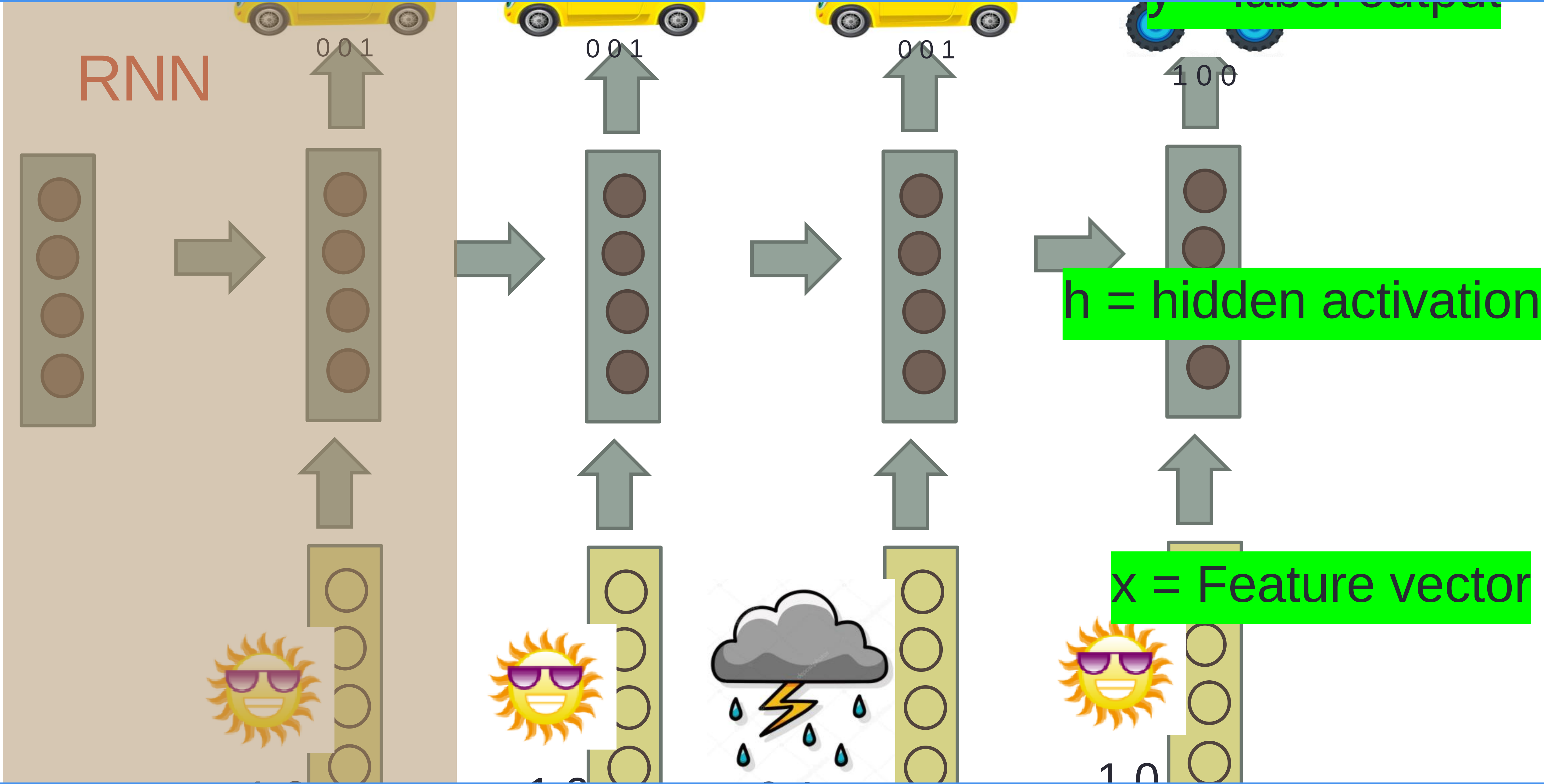
1 0

1 0

0 1

RNN

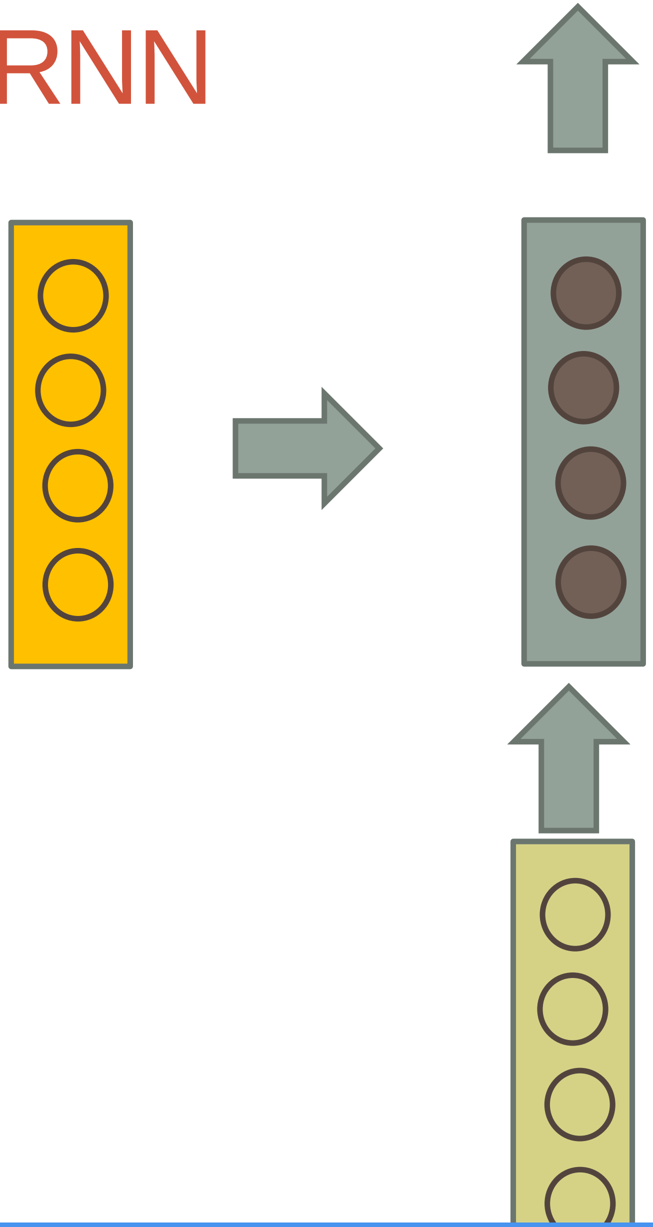




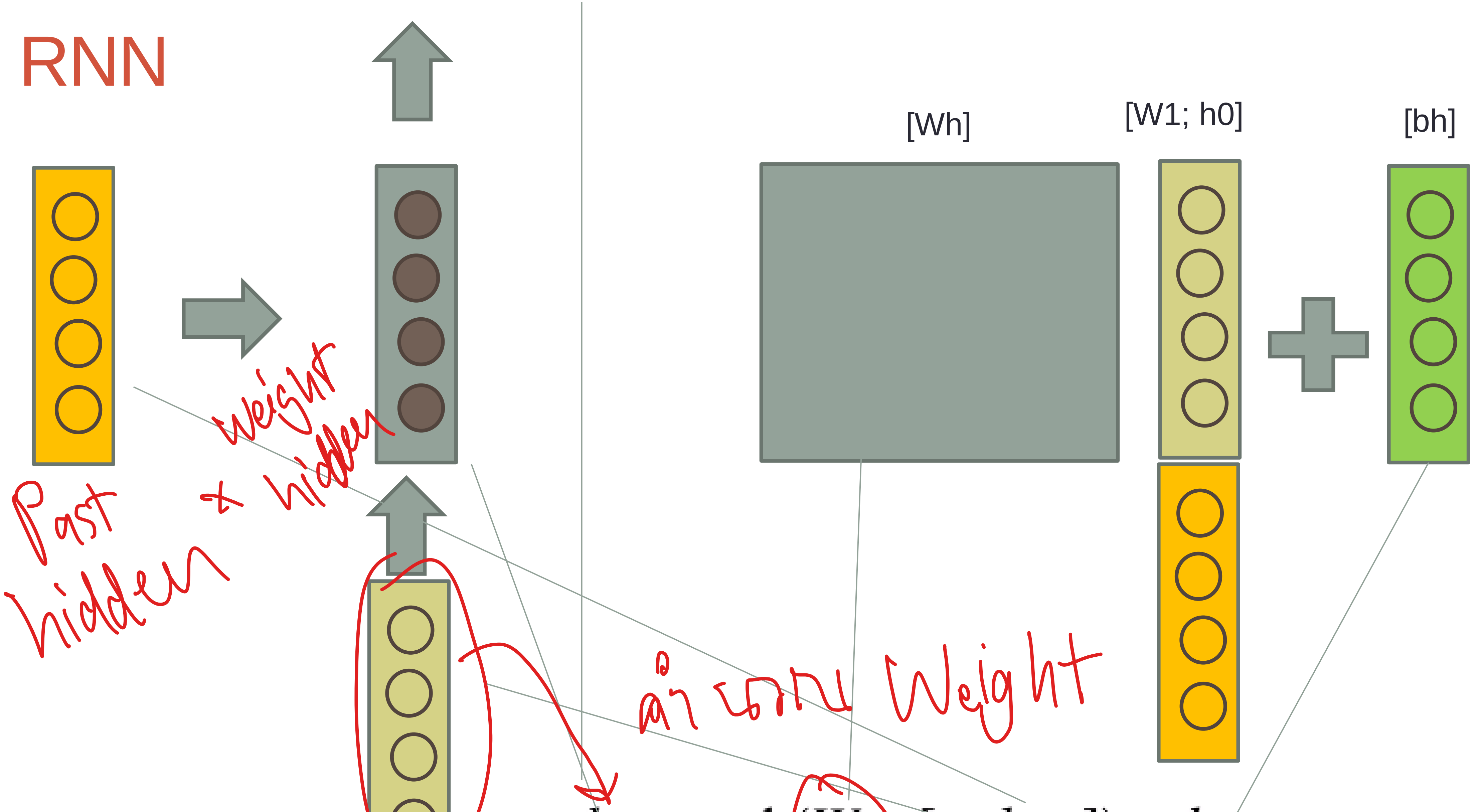
h = hidden activation

x = Feature vector

RNN

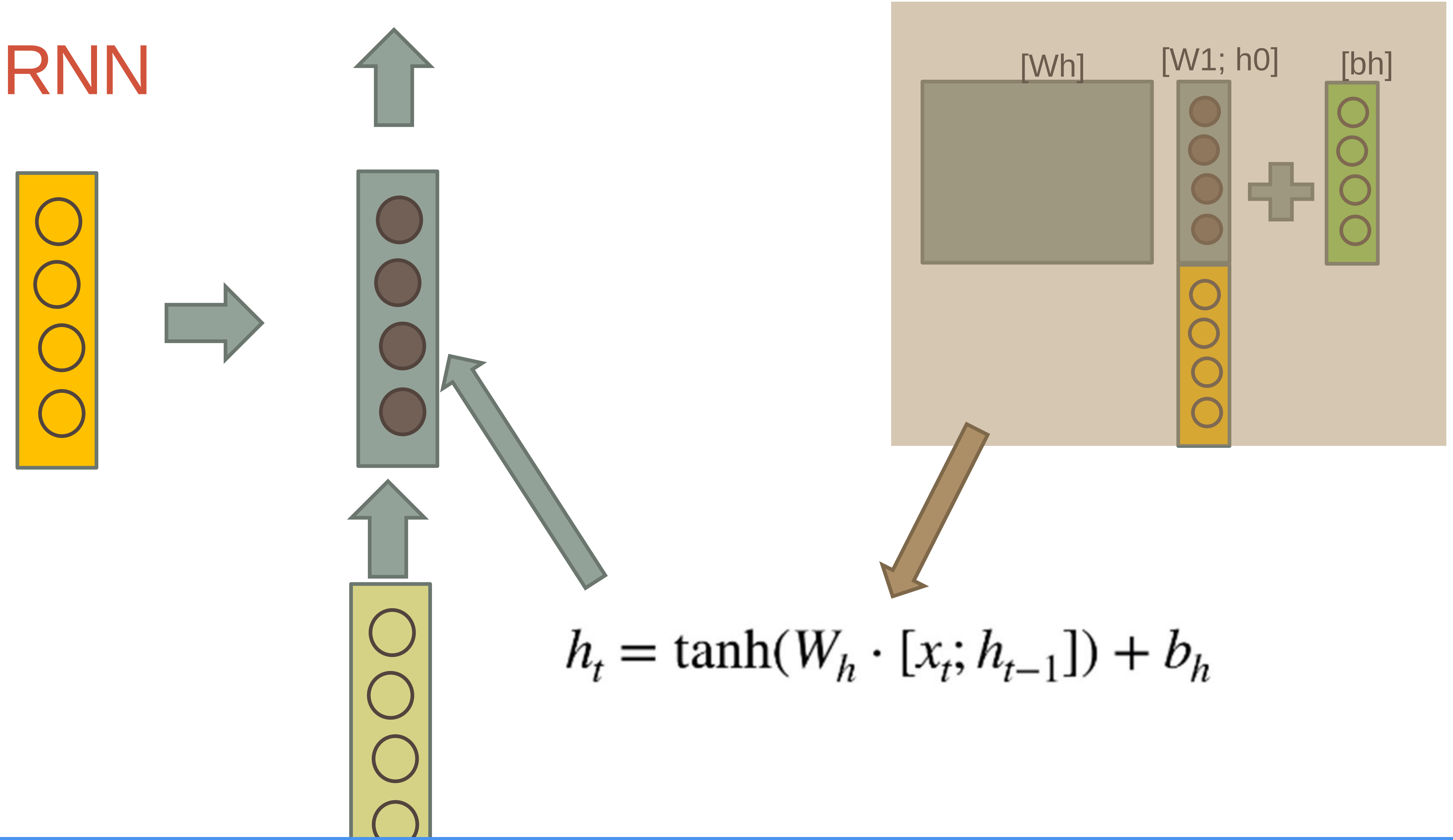


RNN



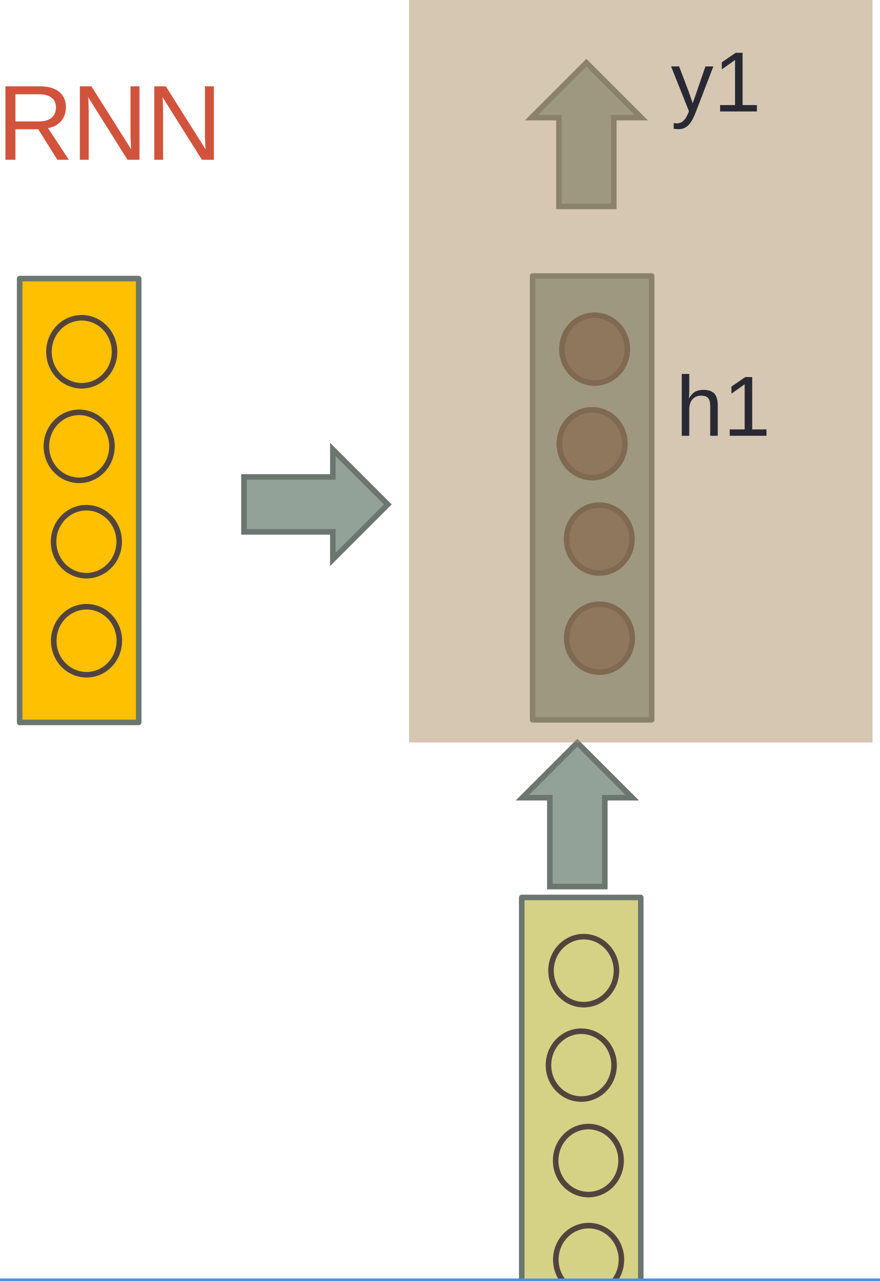
$$h_t = \tanh(W_h \cdot [x_t ; h_{t-1}]) + b_h$$

RNN

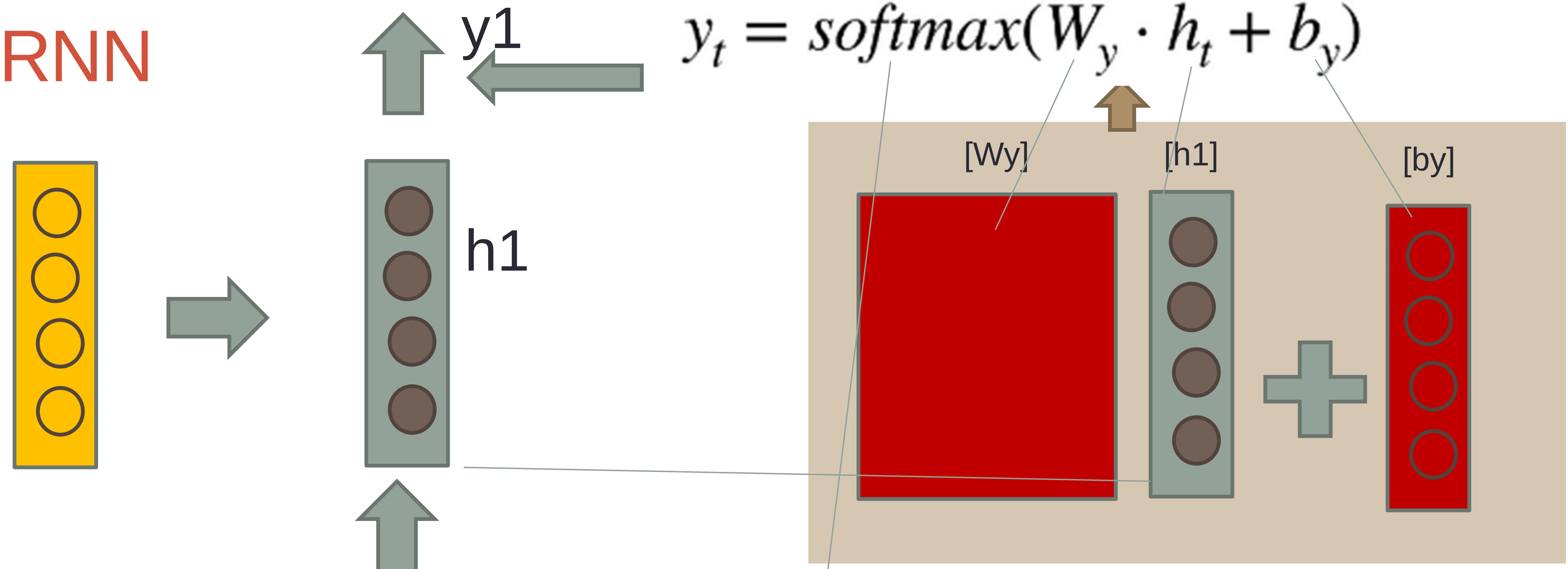


$$h_t = \tanh(W_h \cdot [x_t; h_{t-1}]) + b_h$$

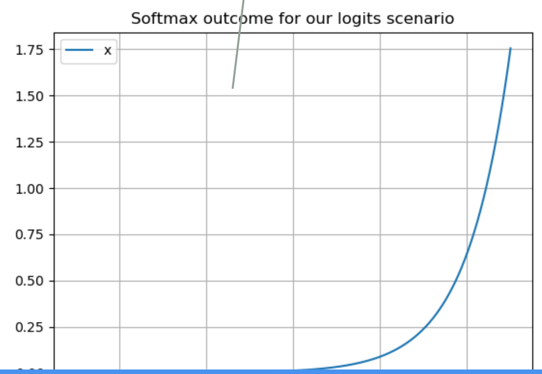
RNN



RNN



$$y_t = softmax(W_y \cdot h_t + b_y)$$



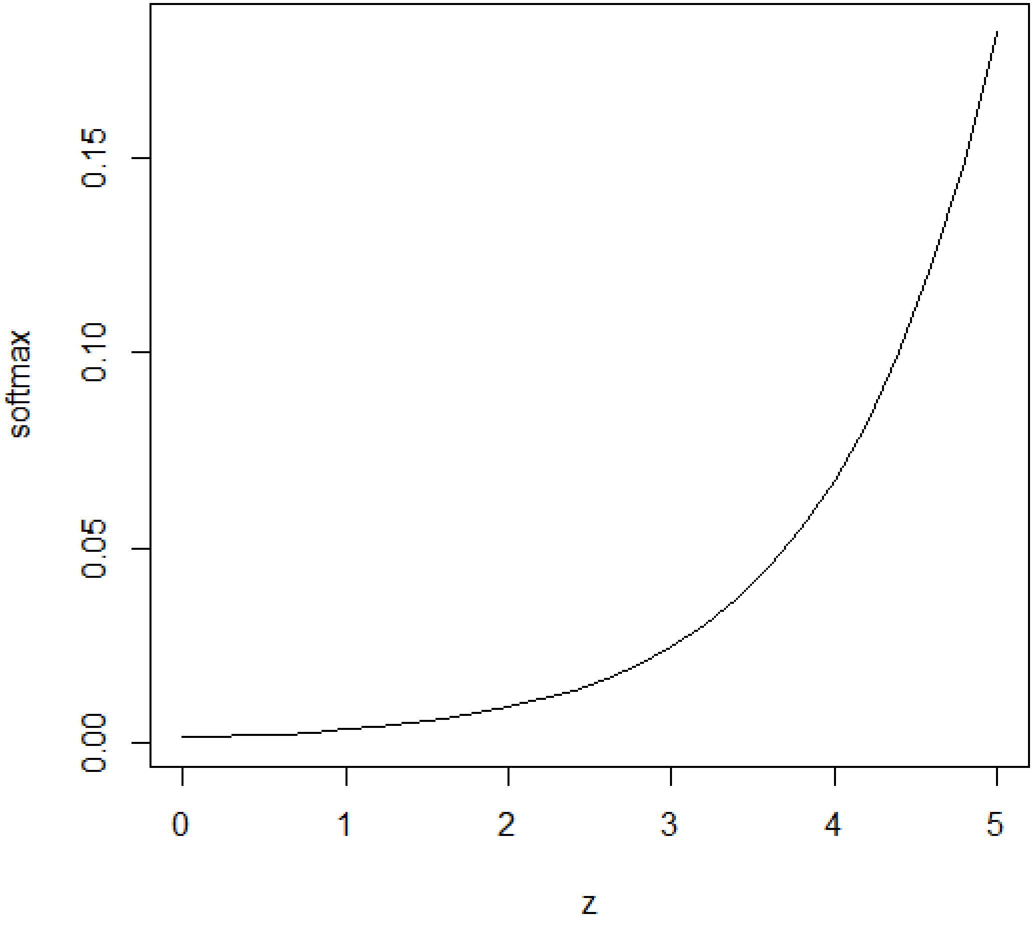
Softmax outcome for our logits scenario

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

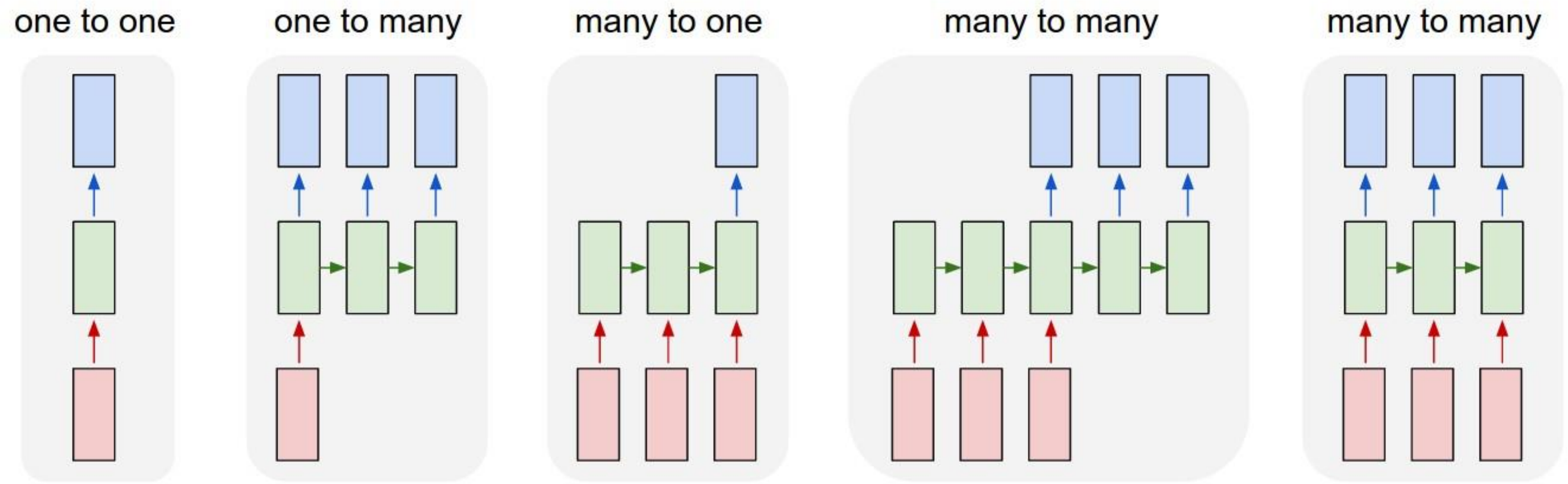
Softmax

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

```
z = seq(0,5,by=0.2)

softmax <- exp(z)/sum(exp(z))

plot(z,softmax,type='l')
```



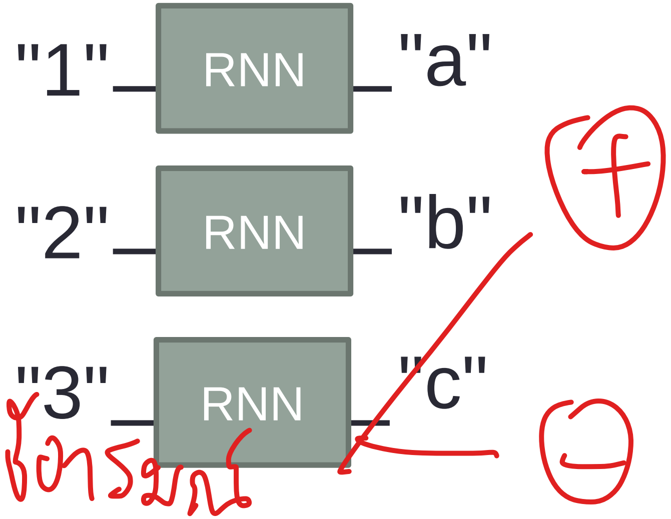
RNNs



RNN usages

12 5 4 7 2 ε "A"

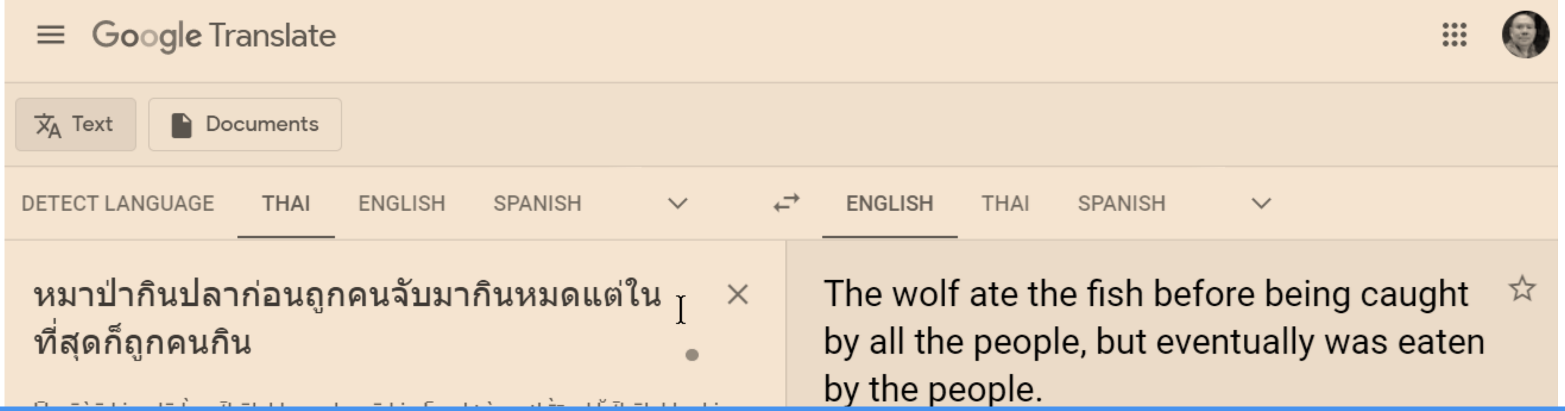
- Mapping sequence X and Y
- Sentiment analysis การวิเคราะห์คำที่เป็น โดยคนที่ใช้อารมณ์
  - Analyzing text that written by human to understand the level of emotion .



- Machine Translation

แปลภาษาสื่ออารมณ์

การใช้คำ



หมาป่ากินปลาก่อนถูกคนจับมากินหมดแต่ในที่สุดก็ถูกคนกิน

The wolf ate the fish before being caught by all the people, but eventually was eaten by the people.

อดีตนานไม่ได้

RNN

ข้อดี

Advantages

- RNN can learn the difference between of the input sequence.
- Model size does not depend on the length of the input sequence.

ขนาดก็ไม่โต ตาม Inpt

- Weight and some parameters passing by time.

ตามเวลา

เรียนรู้ ด.ต่างของ Input ได้

ข้อเสีย

Disadvantages

- RNN cannot bring the output from the future or long pass return to train in the model.
- RNN does not support skipping in some point of the time in the training process.

ข้ามบางเวลา บาง จุดไม่ได้

- Most paper told that RNN uses high computation time for taining.

การจัดส่ง

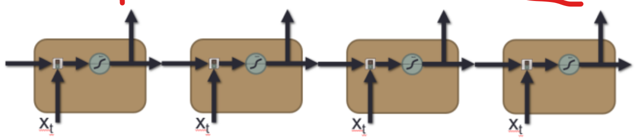
- RNN suffers from Vanishing gradient problem

Vanishing gradient problem

คือ ข้อมูลถูกที่แทรก (noise) ทำให้เกิด ข้อผิดพลาด

- Discover by Joseph 1991
- The situation that a deep multilayers RNN is inability to learn in a long sequence that related long data sequence.

The man who talks with her dog is her friend.



The man who buy the chocolate is her friend.

RNN ไม่ดี

- Solving แก้โดย
  - Weight initialization management ช่วยลด (ทำ by pass, Gammar)
  - The usage of LSTM from by the many research paper told LSTM can reduce 49% of transcription error.
  - Echo state network

Echo state network (ESN)

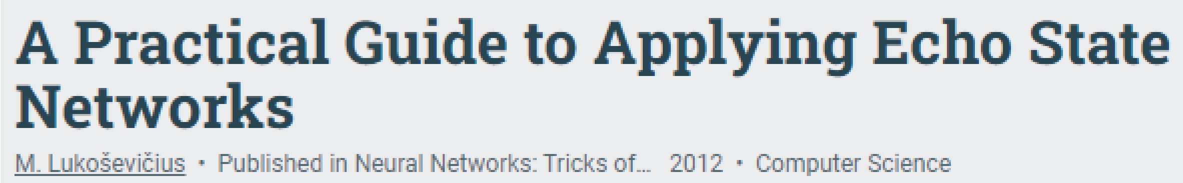
คือ weight กระจายของ State

- ESN is a new recurrent neural network designed to reduce the difficulties of training in the conventional RNN.
- Most RNN (with usually 1 percent connectivity) with a sparsely connected hidden layer.

ที่เหลือ ศูน.

- ESN is characterized by an outsized reservoir converting the input file to a high-dimensional dynamic state space, which may be the “echo” of recent input history.

ค่าน้อยมาก



A Practical Guide to Applying Echo State Networks

M. Lukoševičius • Published in Neural Networks: Tricks of... 2012 • Computer Science

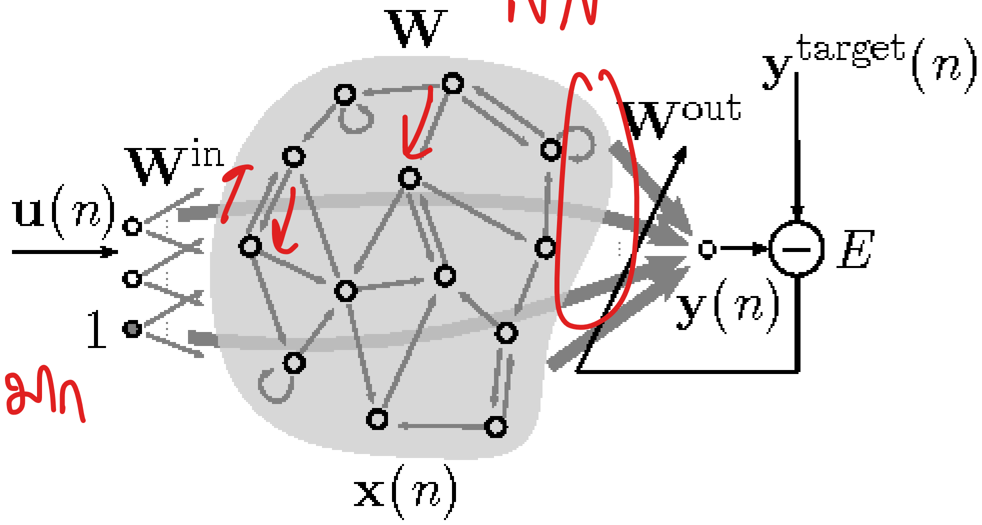
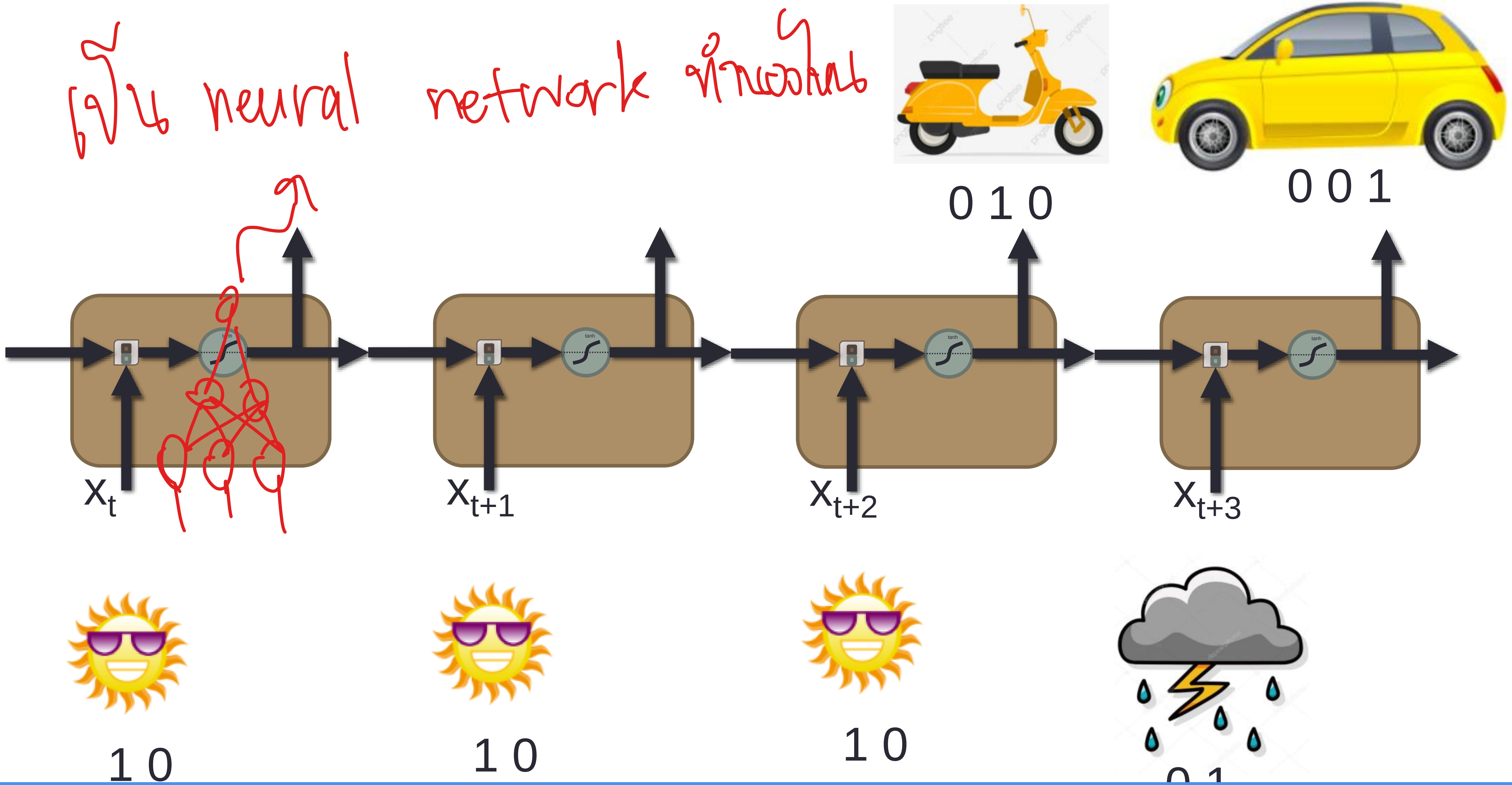


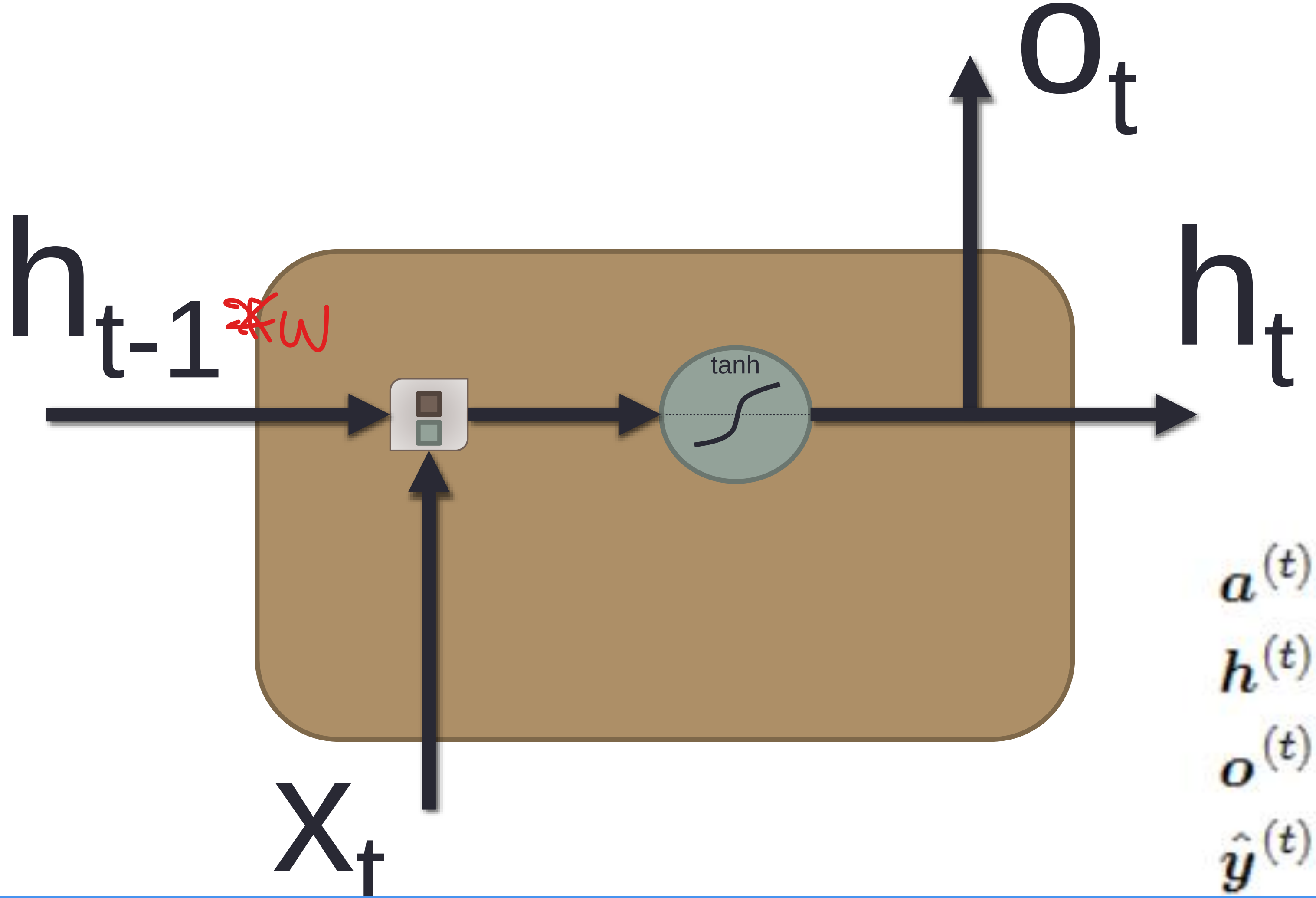
Fig. 1: An echo state network.

neural network = non linear

เป็น neural network ที่ทอดไหม



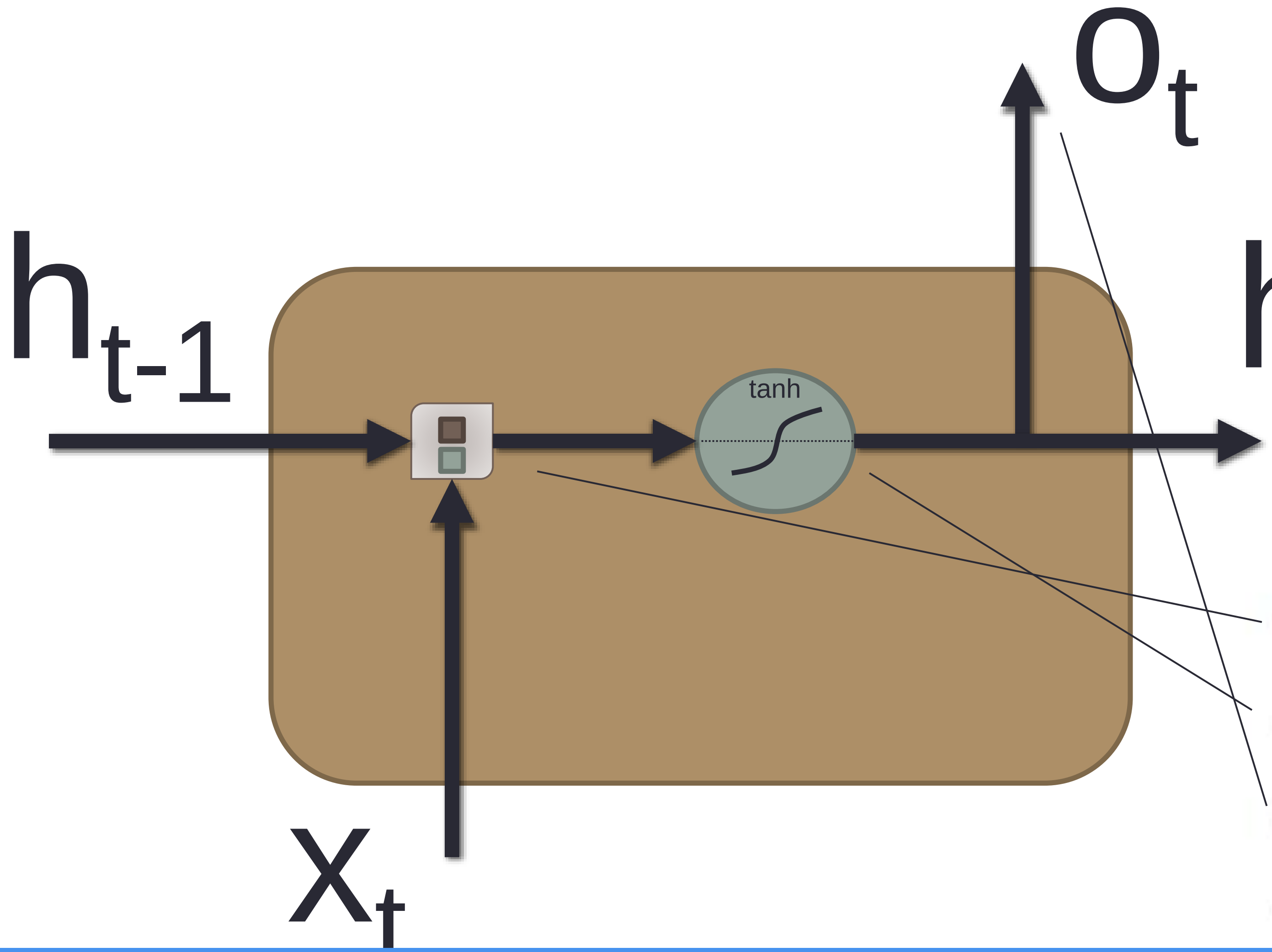
Equation in Recurrent Neural Network



Weight: W,U,V
bias: b,c

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Equation in Recurrent Neural Network



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

GATED RECURRENT UNIT

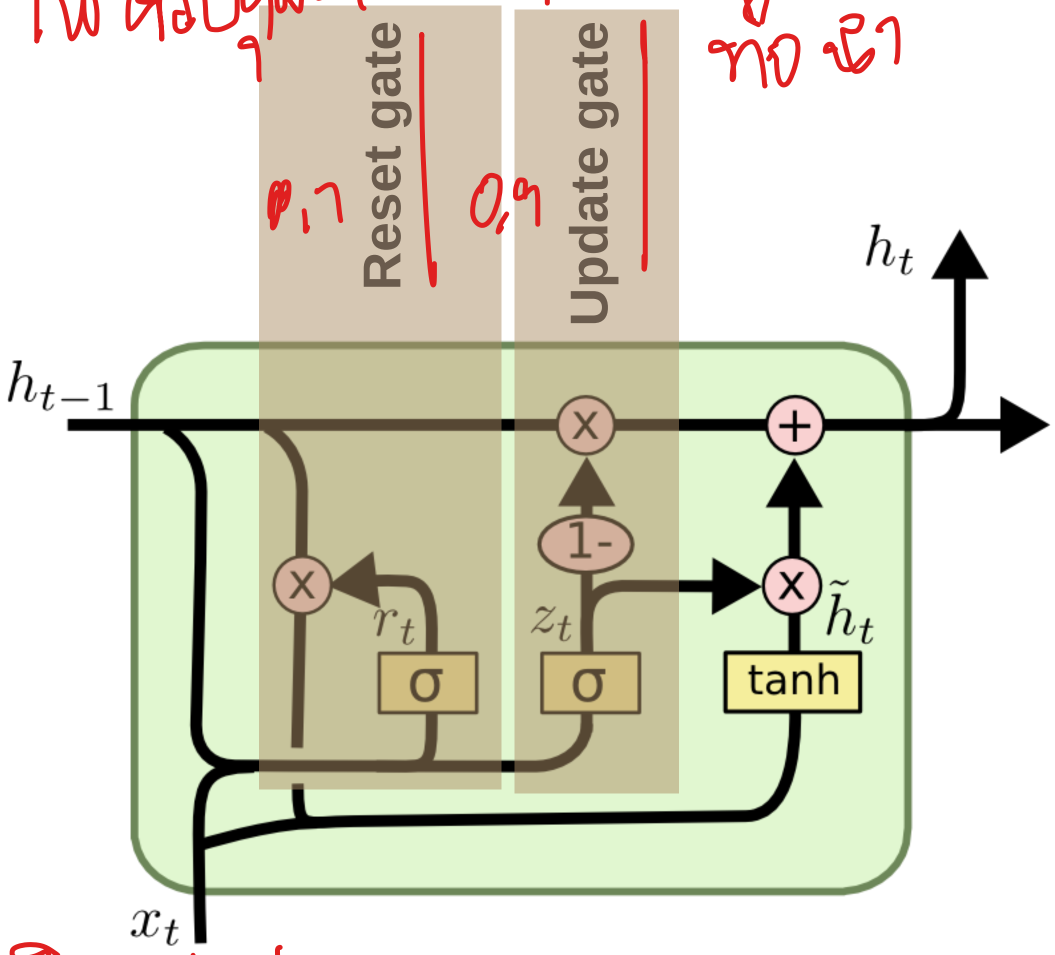
What is GRU

ปิดเปิด ทิศทาง ให้ควบคุมการไหลได้

ท่อ น้ำ

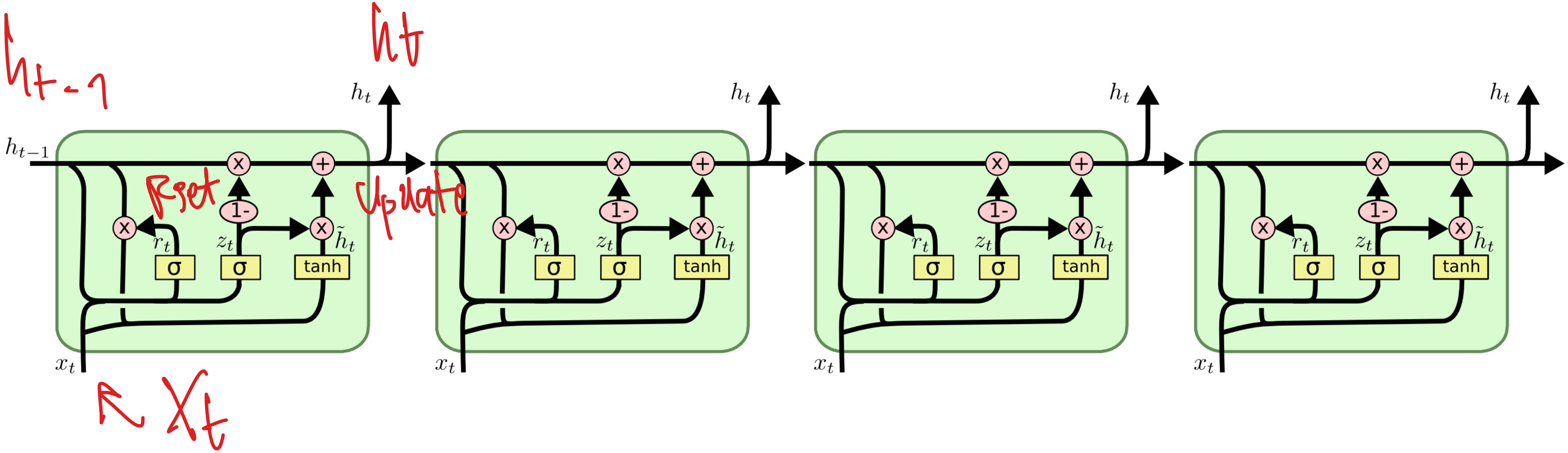
- GRU works as the water tap (door) to control
  - Size of the UPDATE (bring data from the past to direct process in the current)
  - Size of RESET (delete the past data processing only current data)
- The research claimed that GRU can reduce vanishing gradient problem.

วัดออกเป็น ตัวเลข (ทดลองต่อ)



ปิด/เปิดเส้อมีการคูณ...

Data flow in GRU



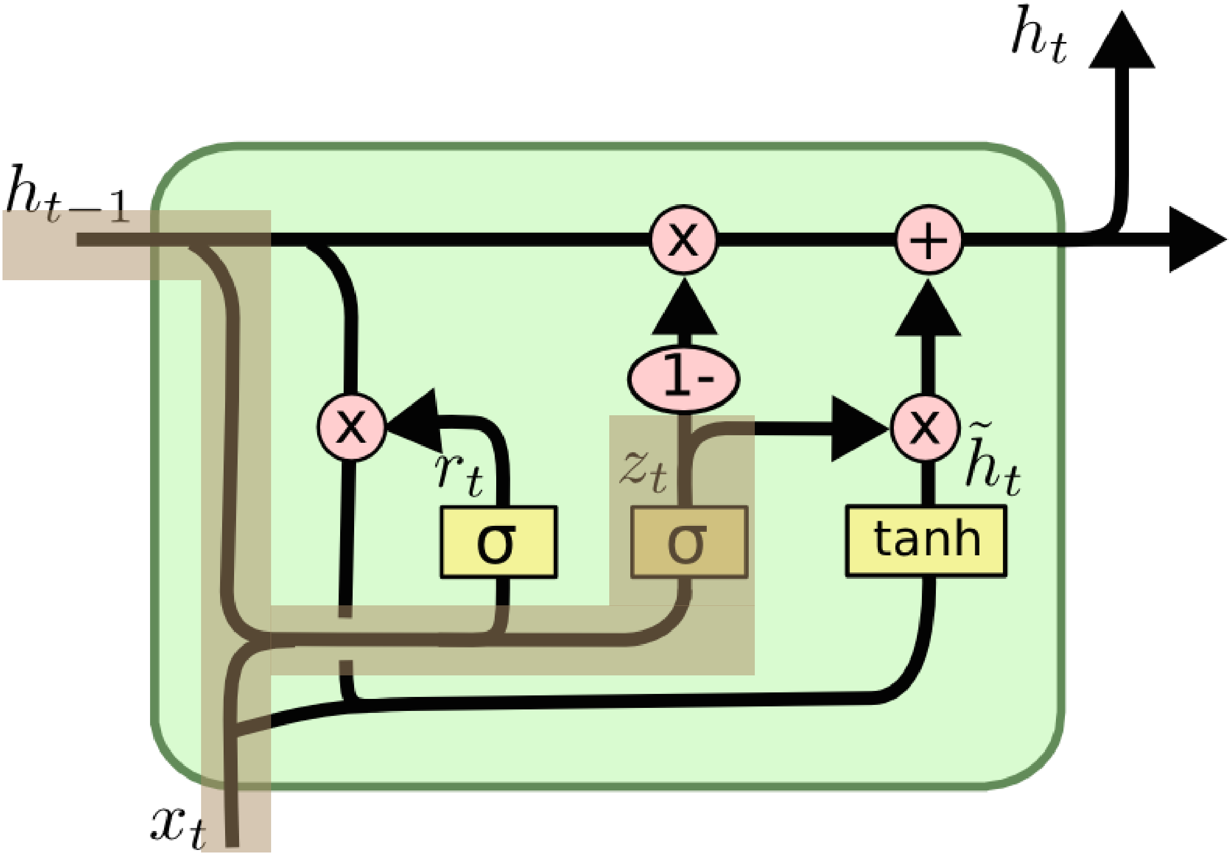
Step1: Update gate (open / close)

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

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

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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



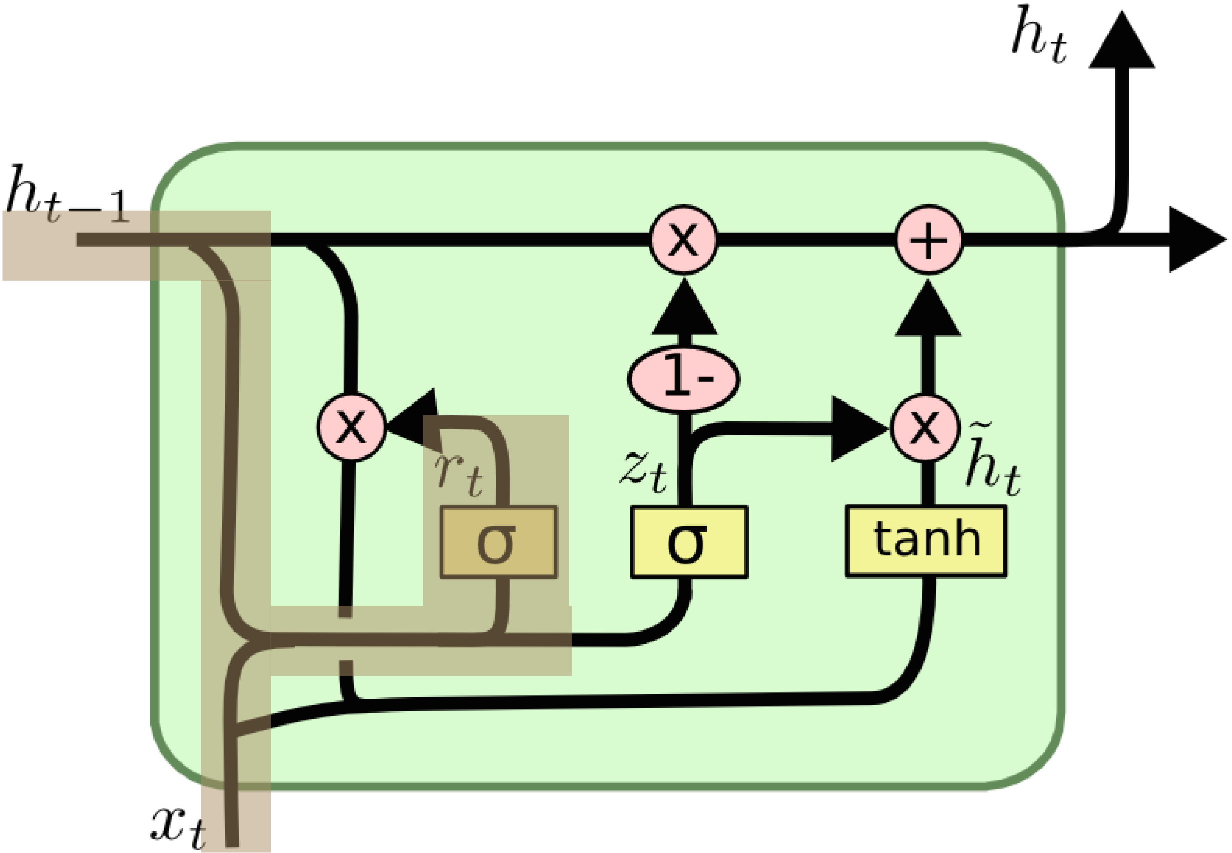
Step2: Reset gate (open / close)

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

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

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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



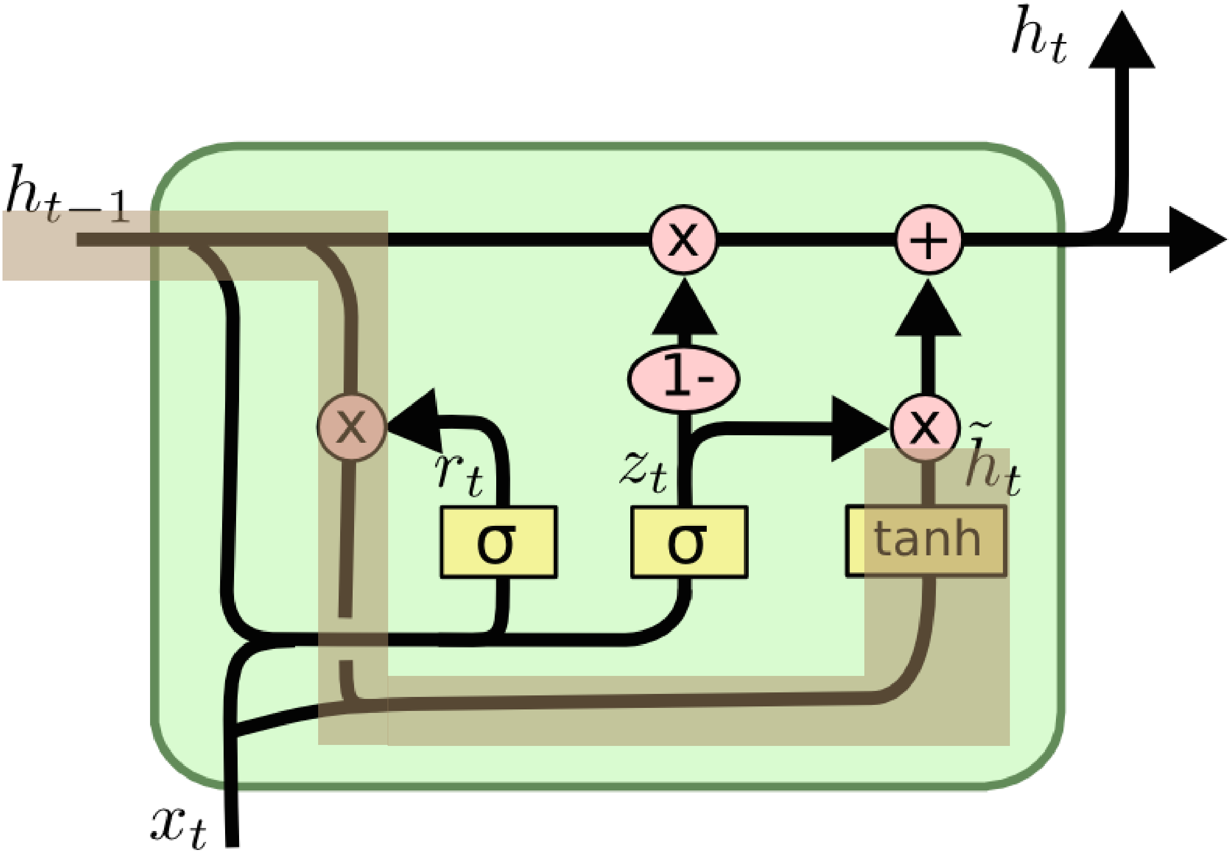
Step3: Current memory

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

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

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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



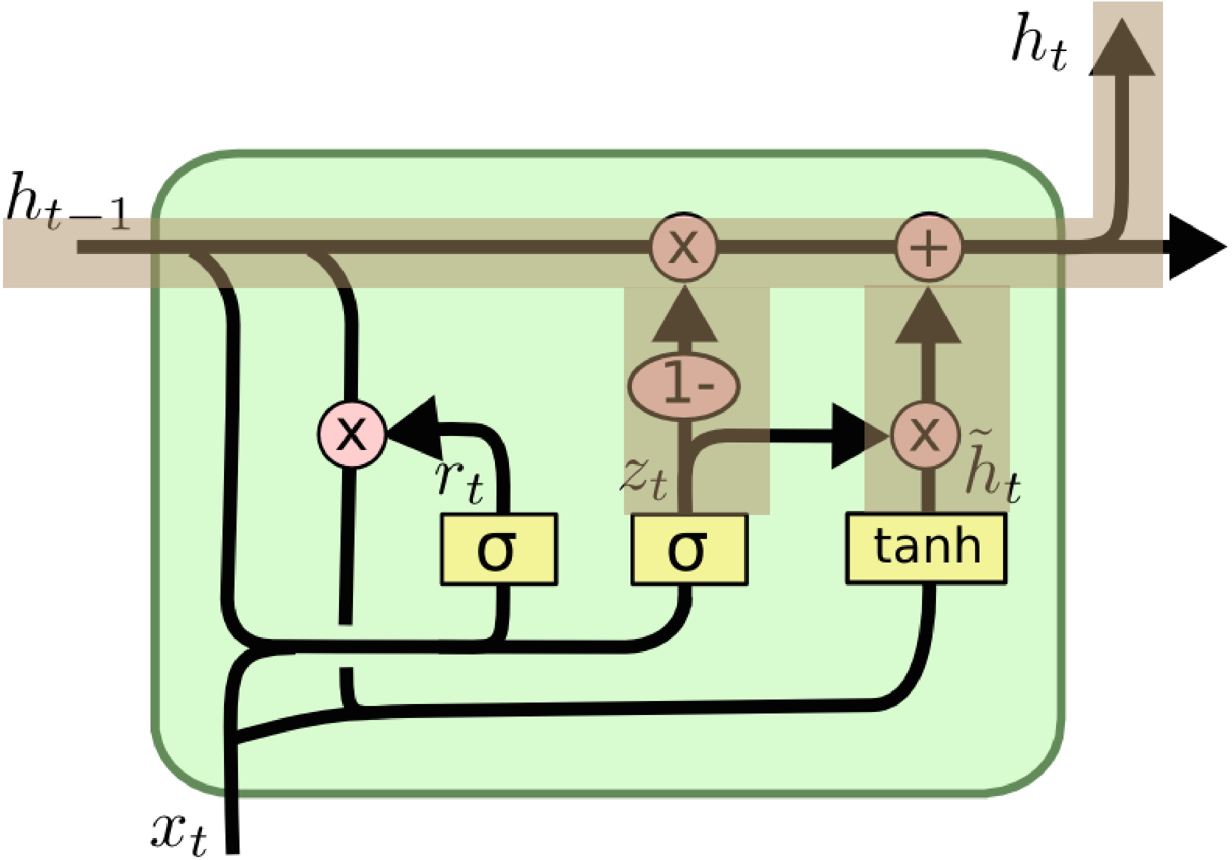
Step4: Final memory at current time

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

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

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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



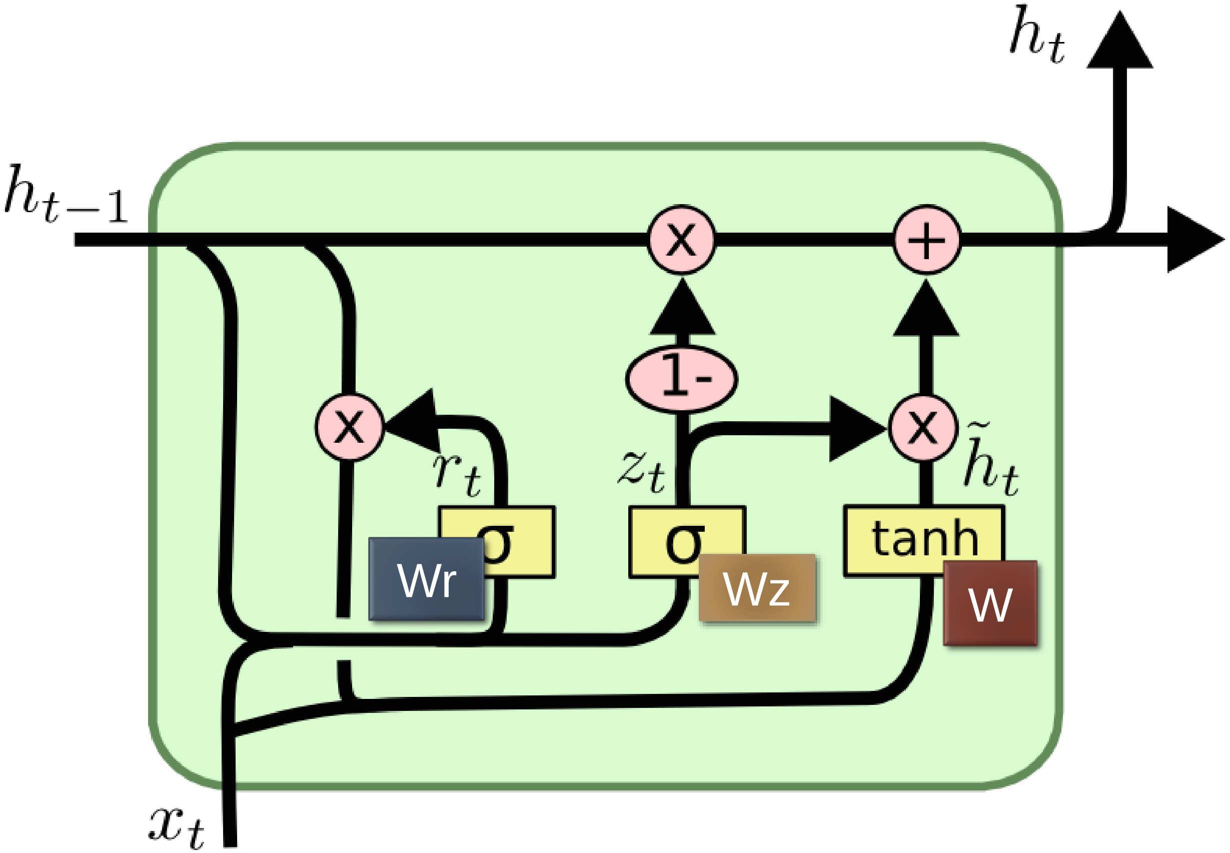
Weight location

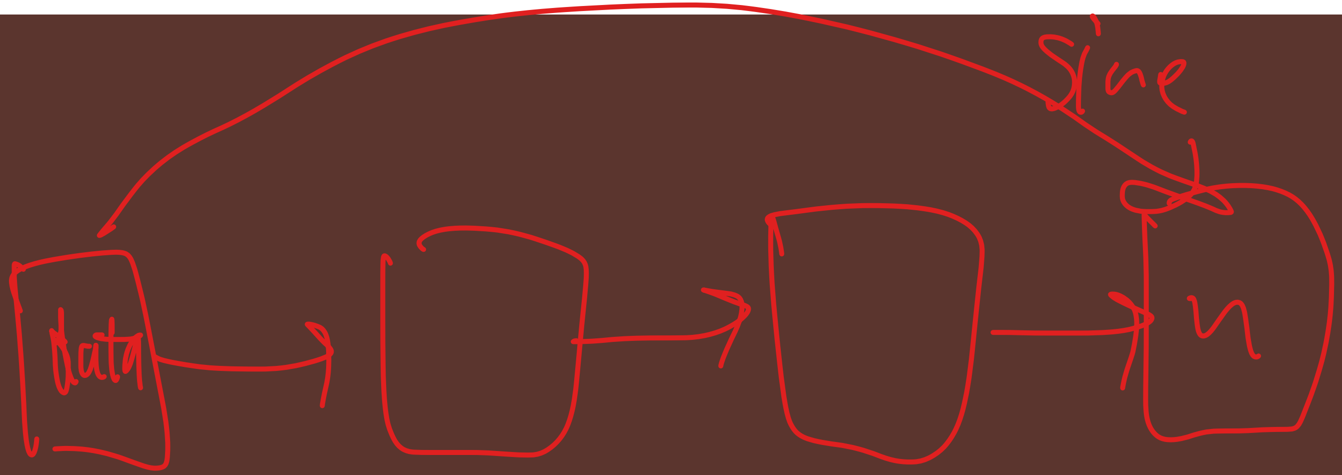
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

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

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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





LONG SHORT-TERM MEMORY

RNN suffer from

เกิดในRNN

- Vanishing gradient problem is unstable behavior when training a deep neural network.
- Information update in the RNN parameter becomes smaller and smaller after long running.

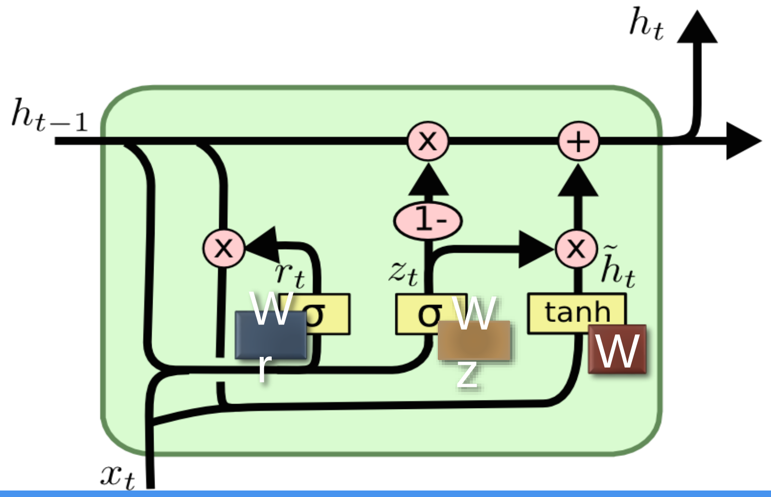
A problem with training networks with many layers (e.g. deep neural networks) is that the gradient diminishes dramatically as it is propagated backward through the network. The error may be so small by the time it reaches layers close to the input of the model that it may have very little effect. As such, this problem is referred to as the "vanishing gradients" problem.

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

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

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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



Long Short-Term Memory (LSTM)

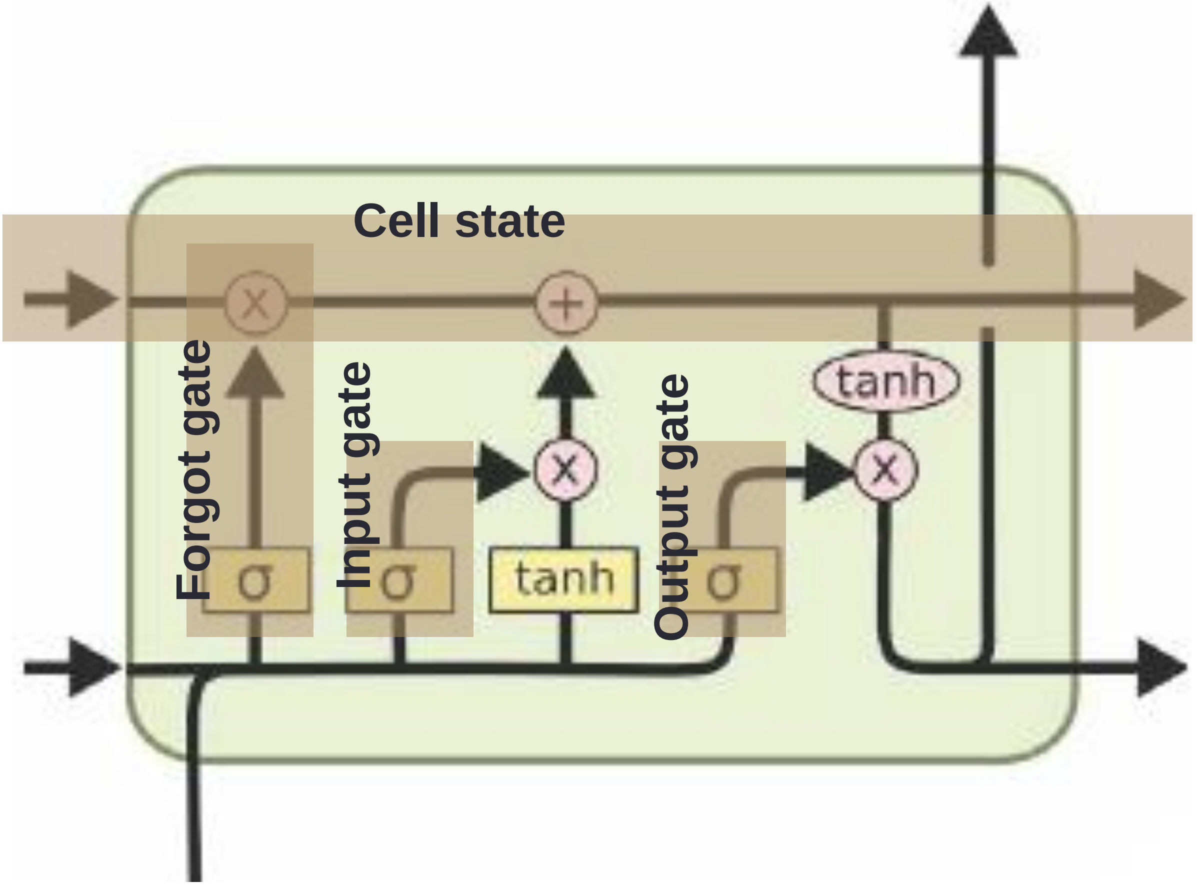
- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- LSTM is designed as a memory cell.
- Information gets into the cell whenever its "write" gate is on.
- The information stays in the cell so long as its "keep" gate is on.
- Information can be read from the cell by turning on its "read" gate.

Each part in LSTM

- Cell state
- Forgot gate
- Input gate
- Output gate

H Fast
(Short)

C Slow
(Long)



Each part in LSTM

W U V = weight

$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

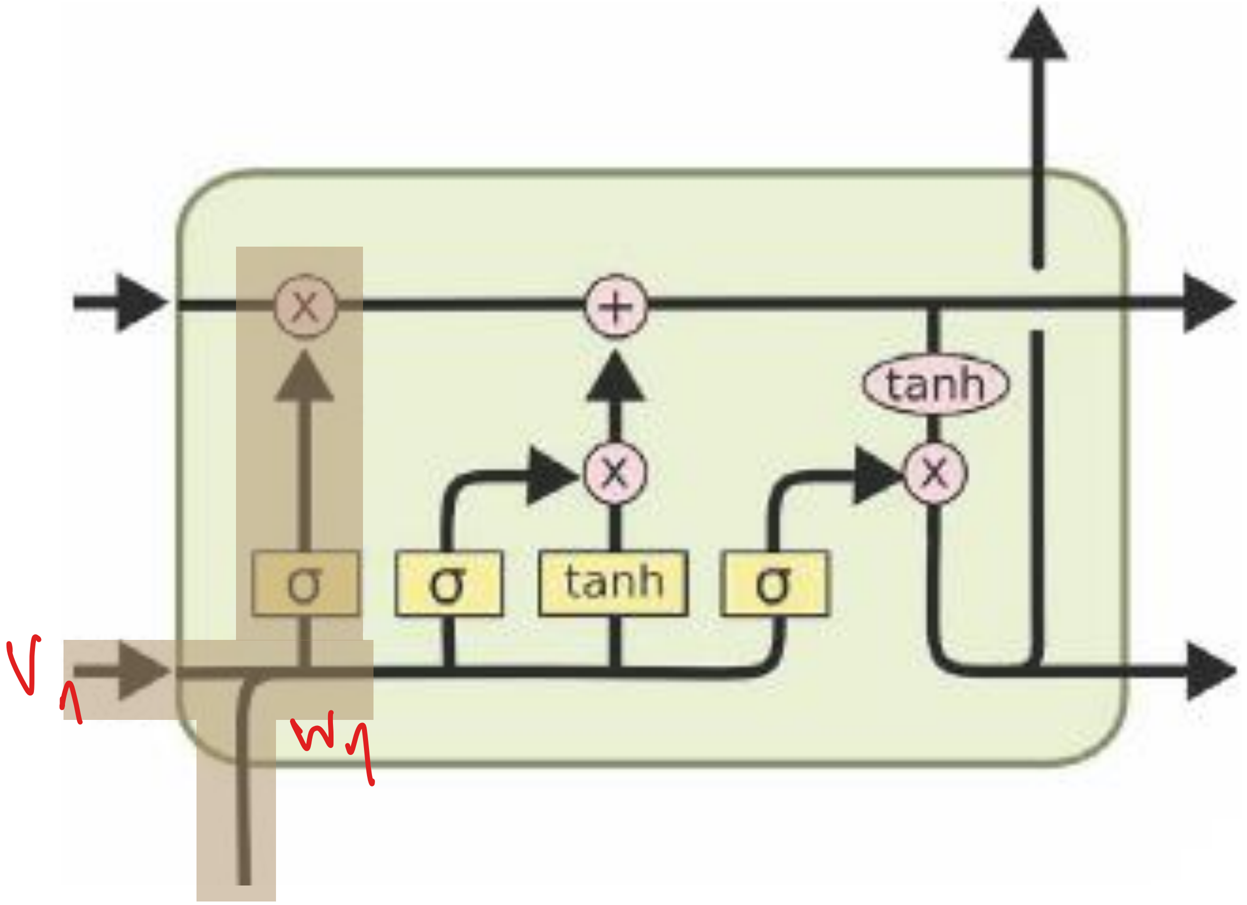
$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j$$

$$\tilde{c}_t^j = \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}\right)^j$$

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

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

$$h_t^j = o_t^j \tanh\left(c_t^j\right)$$



Each part in LSTM

$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

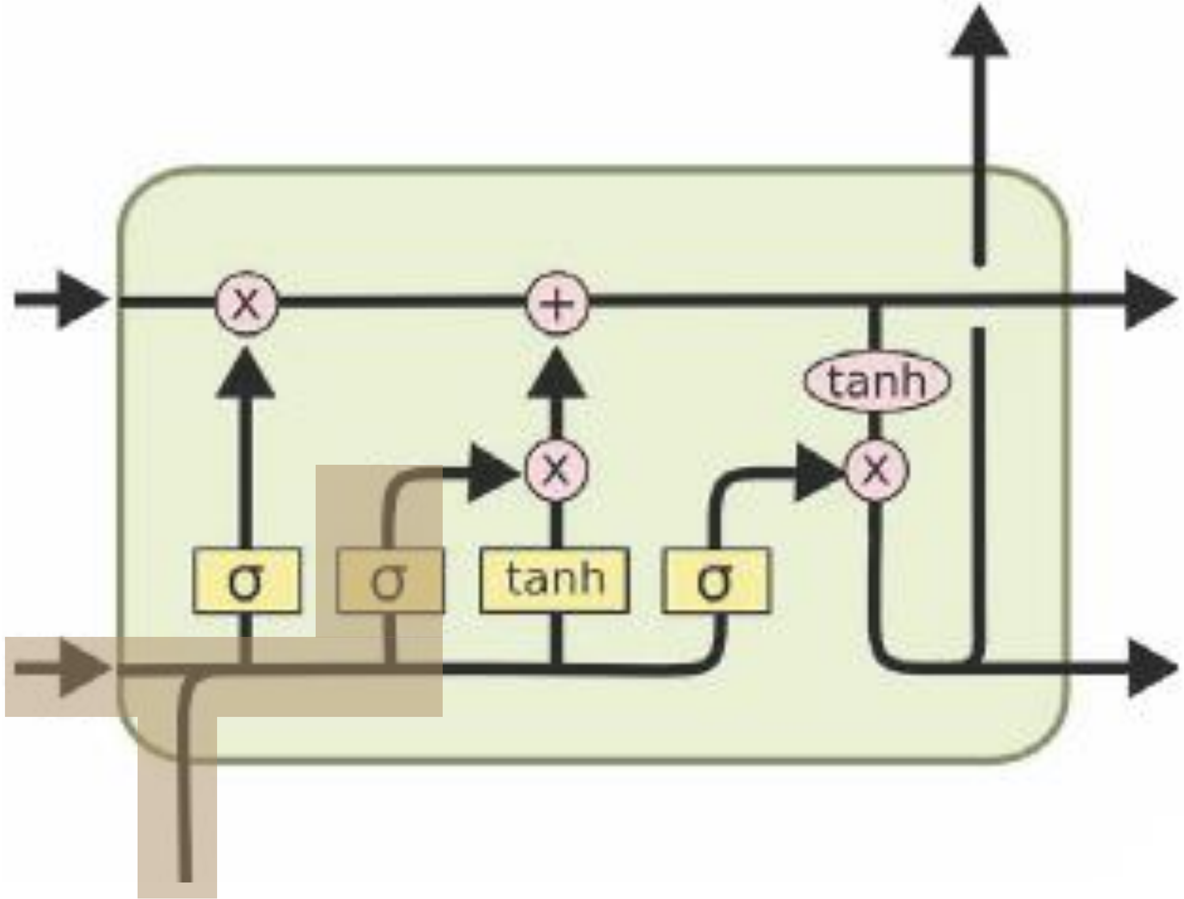
$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j$$

$$\tilde{c}_t^j = \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}\right)^j$$

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

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

$$h_t^j = o_t^j \tanh\left(c_t^j\right)$$



Each part in LSTM

$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

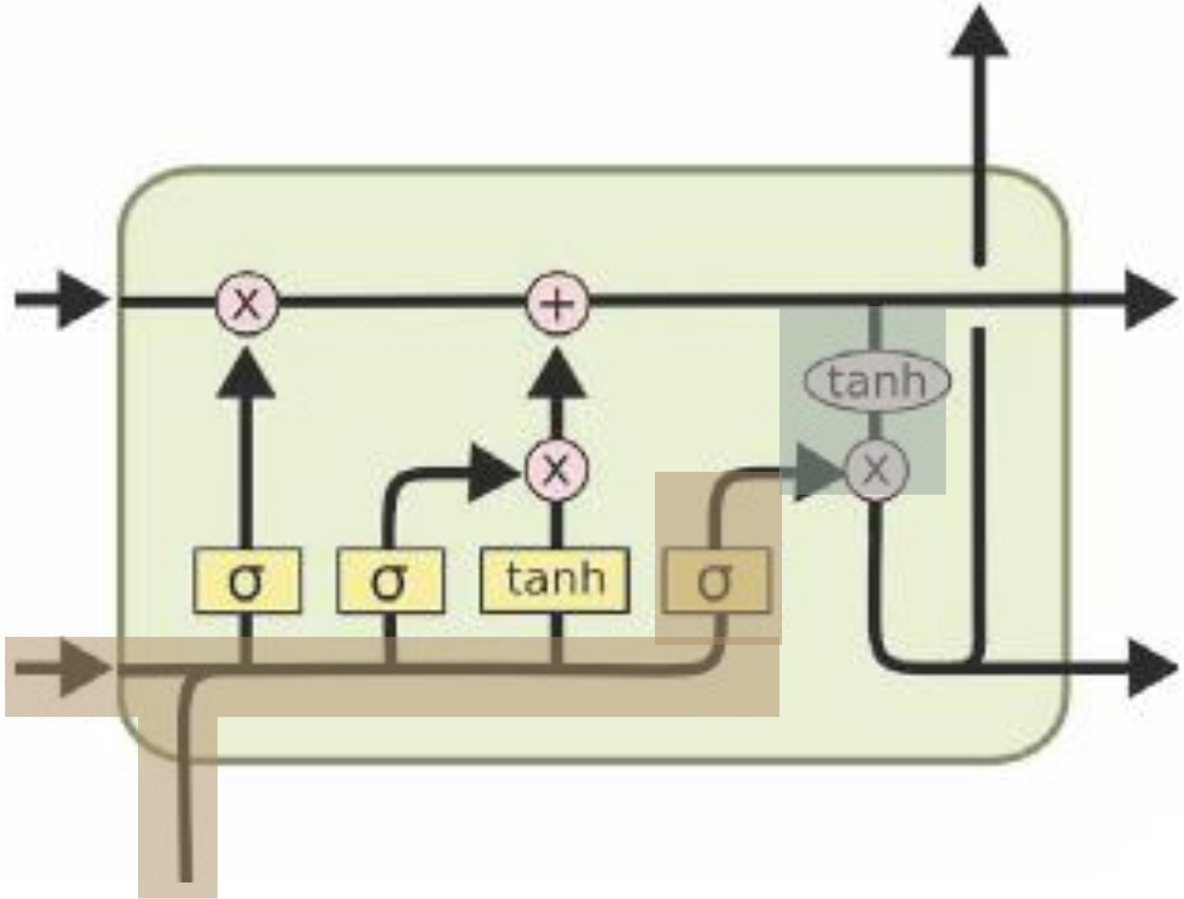
$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j$$

$$\tilde{c}_t^j = \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}\right)^j$$

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

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

$$h_t^j = o_t^j \tanh\left(c_t^j\right)$$



Each part in LSTM

$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

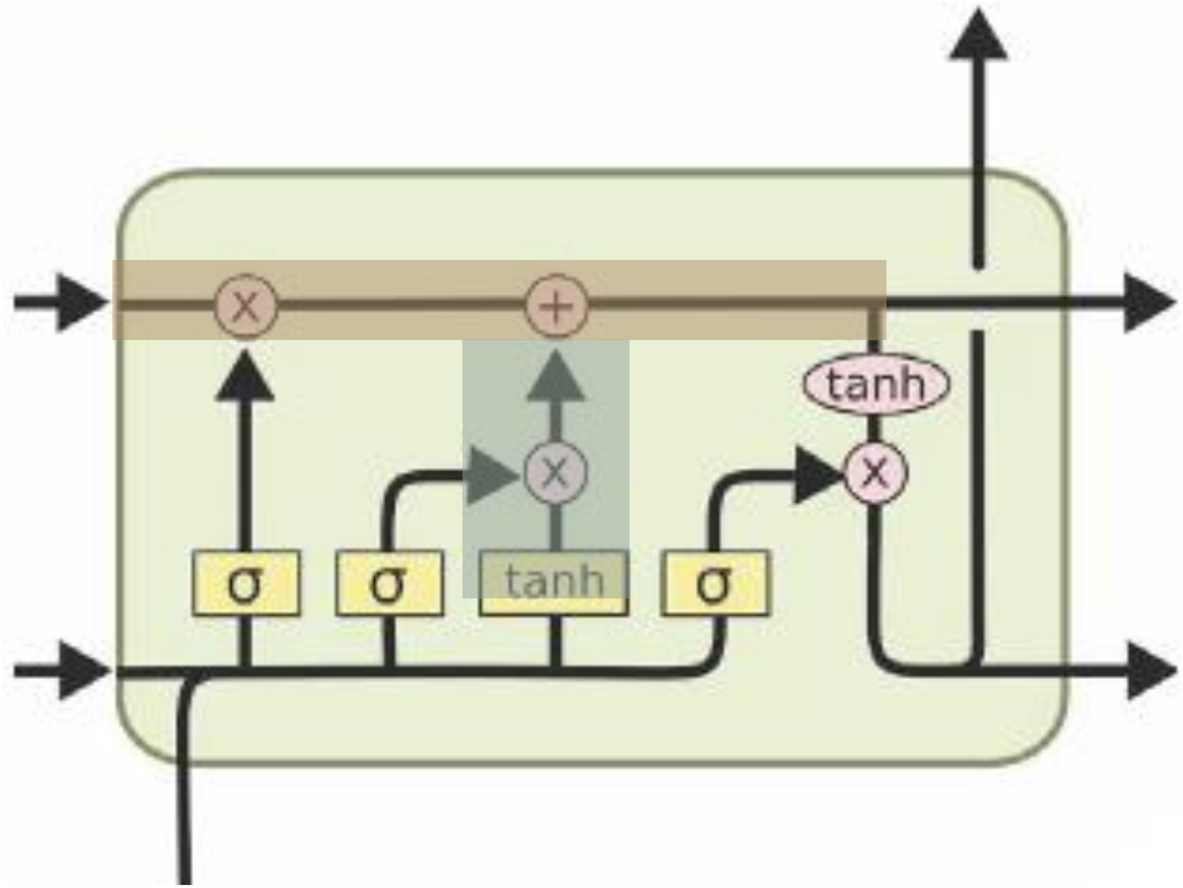
$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j$$

$$\tilde{c}_t^j = \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}\right)^j$$

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

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

$$h_t^j = o_t^j \tanh\left(c_t^j\right)$$



Weight location

$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

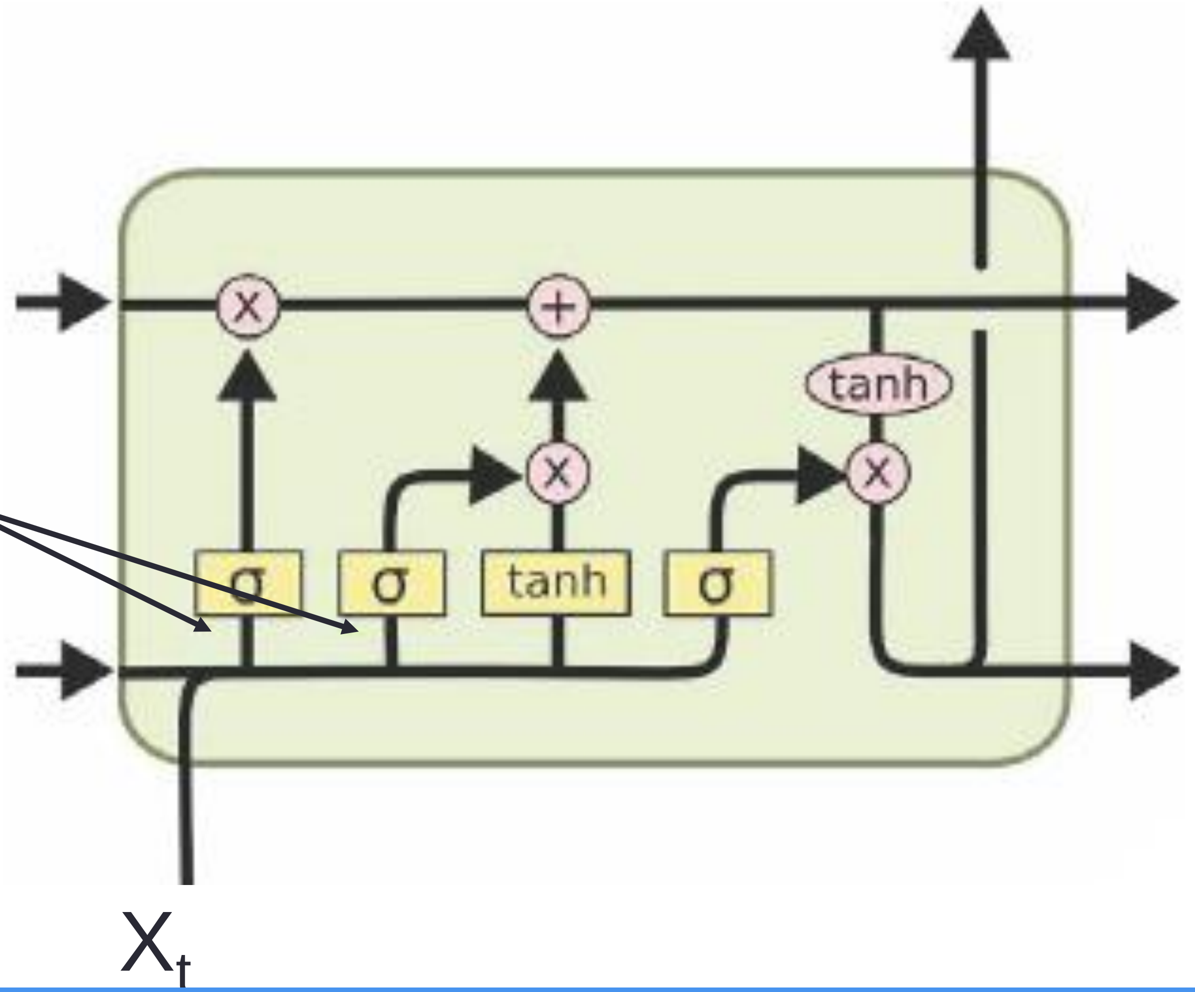
$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j$$

$$\tilde{c}_t^j = \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}\right)^j$$

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

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

$$h_t^j = o_t^j \tanh\left(c_t^j\right)$$



Weight location

$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

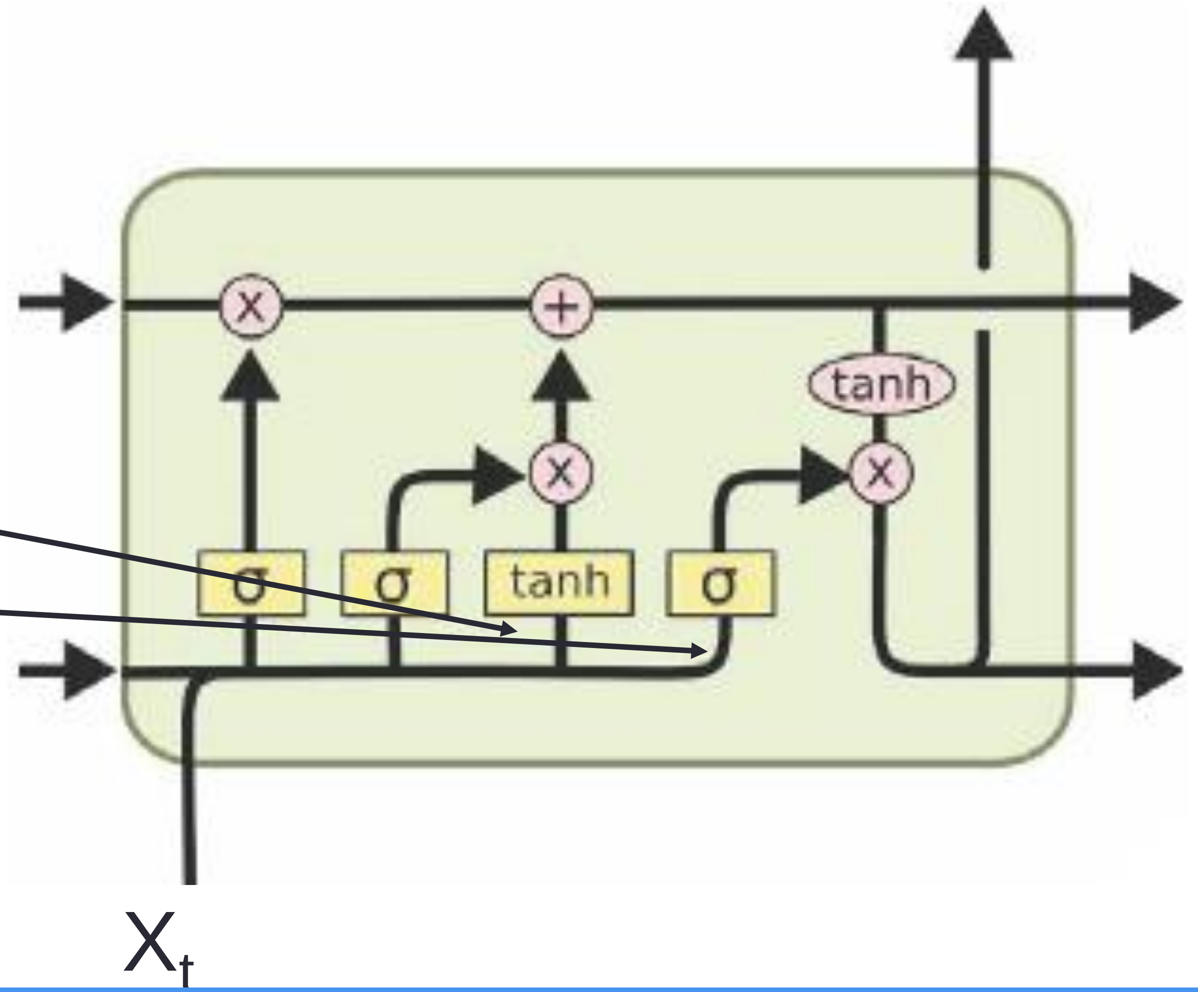
$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j$$

$$\tilde{c}_t^j = \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}\right)^j$$

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

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

$$h_t^j = o_t^j \tanh\left(c_t^j\right)$$



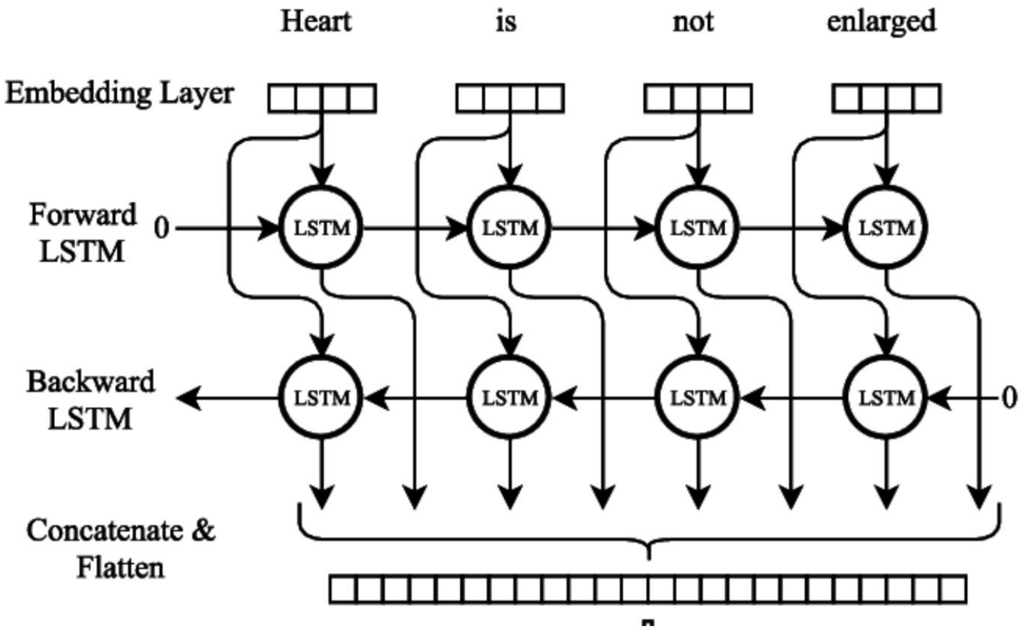
Bidirectional LSTM

- A Bidirectional LSTM, or biLSTM, is a sequence processing model that consists of two LSTMs:

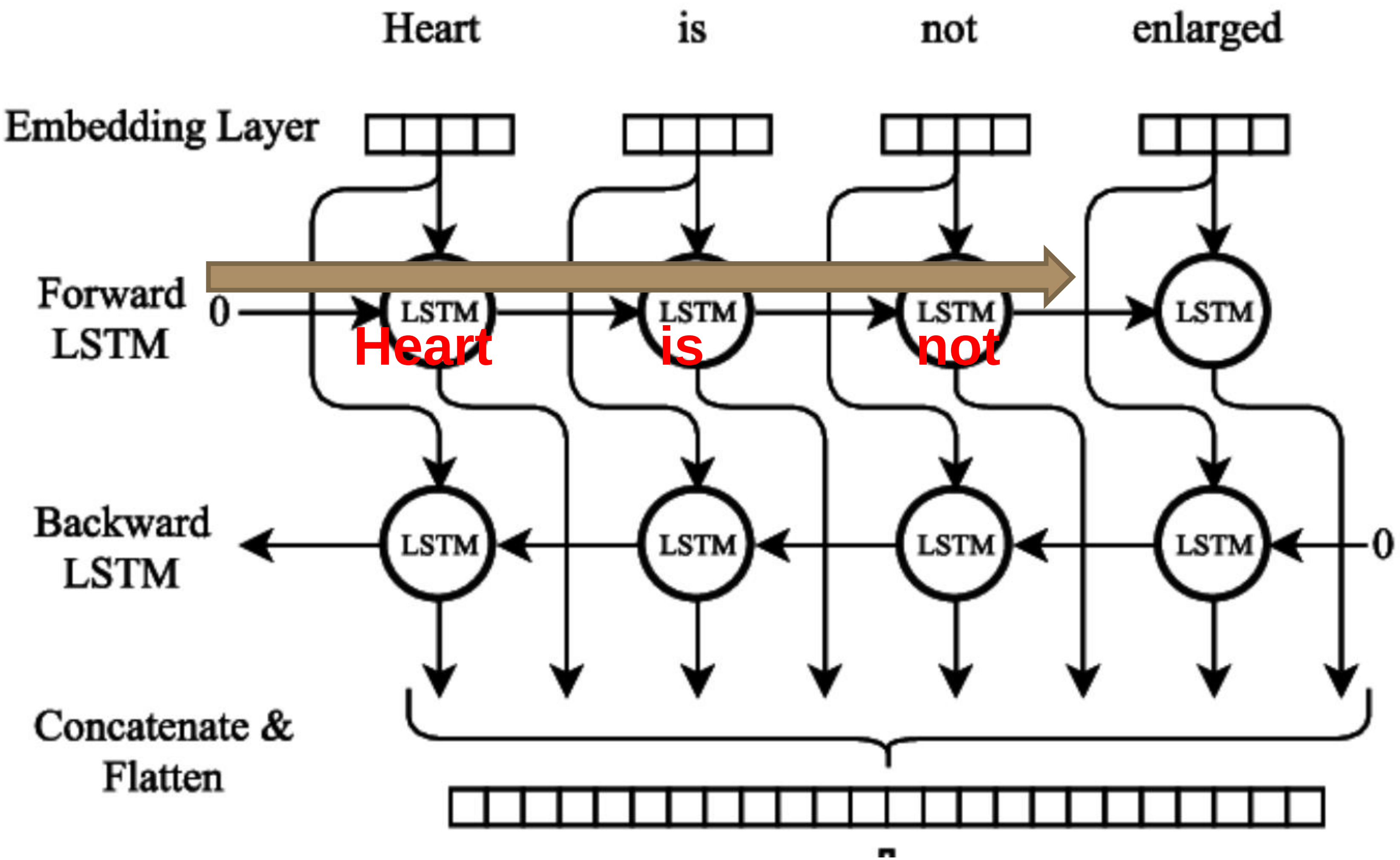
  1) Forward direction (left to right)

  2) Backwards direction (right to left)

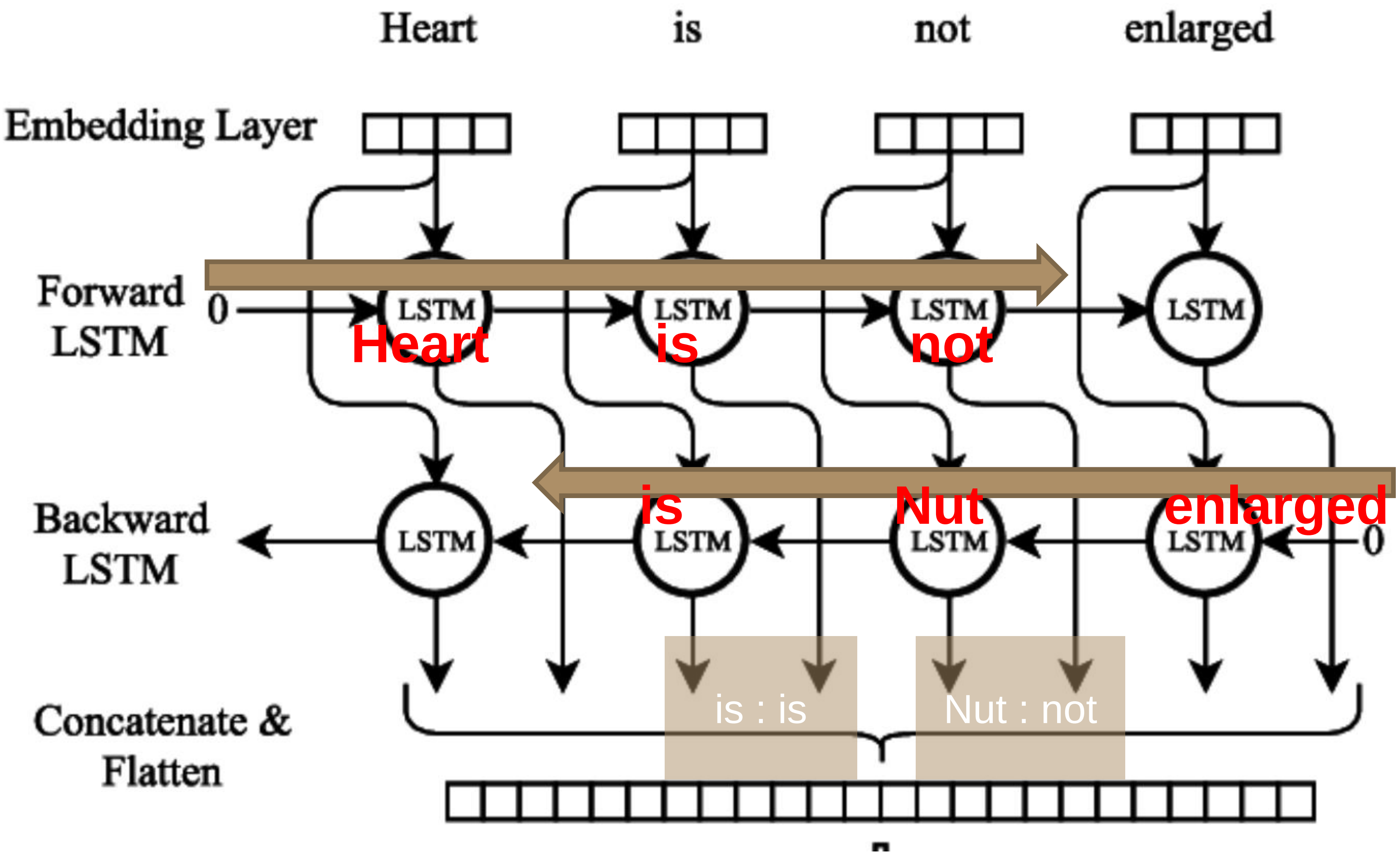
- BiLSTMs effectively increase the amount of information available to the network, by improving the context to the algorithm immediately.



Bidirectional LSTM



Bidirectional LSTM



RNN()

```
trainr(Y, X, network_type="..")
```

rnn

The function trainr() has a network-type parameter changing the model.

"rnn" = recurrent neural network. default

"gru" = gate recurrent unit

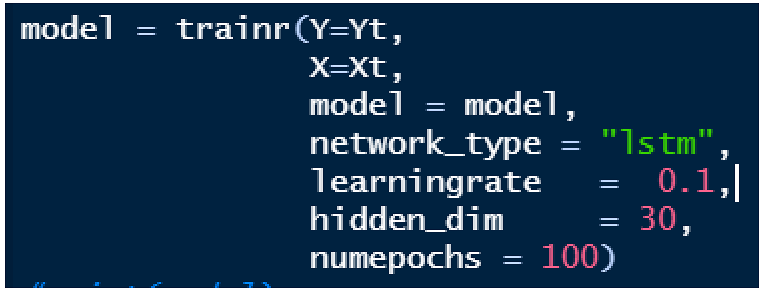
"lstm" = long short-term memory

#both gru and lstm are experimental and development.

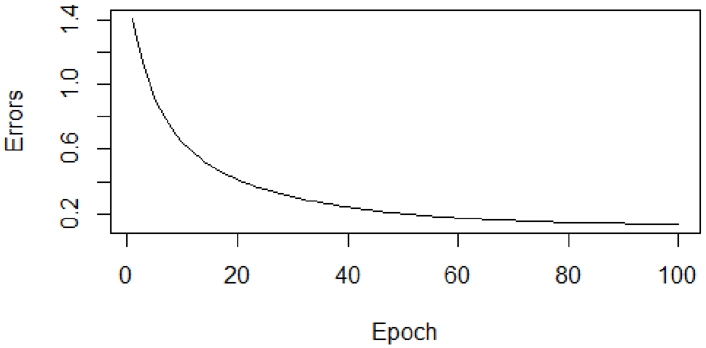
Activity 9.1 Compare RNN, GRU, and LSTM on trainr()

```
# Activity 9.1 Test RNN, GRU and LSTM
# BY supakit@it.kmitl.ac.th
rm(list=ls())
#install.packages("rnn")
library("rnn")
packageVersion("rnn") #latest  1.4.0

x1 = 1:(5*4*3)
x2 = 101:(100+5*4*3)
y  = 201:(200+5*4*3)
mx1 =  matrix(x1,ncol=4)
mx2 =  matrix(x2,ncol=4)
my =   matrix(y,ncol=4)

X = array( c(mx1,mx2), dim=c(dim(mx1),2) )
Y = array( c(my), dim=c(dim(my),1) )
dim(X);X
dim(Y);Y

Xt = X/261
Yt = Y/261

model = NULL
model = trainr(Y=Yt,
               X=Xt,
               model = model,
               network_type = "rnn",  #rnn  gru  lstm
               sigmoid = "logistic",  #logistic  tanh
               use_bias = FALSE,        #TRUE  FALSE
               learningrate   =  0.1,
```

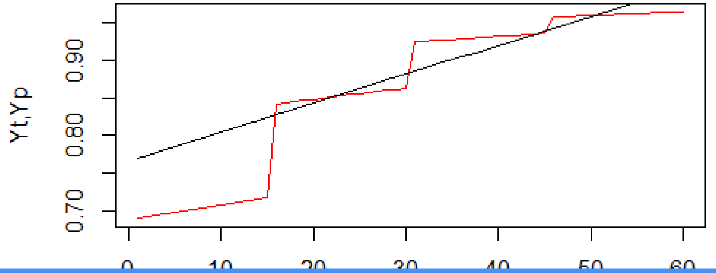
```
maxiter = model$numepochs
Yp = predictr(model, Xt)

par(mfrow=c(2,1))
strConf = sprintf("%s lr:%1.2f hdim:%d
sig:%s",model$network_type,
     model$learningrate, model$hidden_dim, model$sigmoid)
plot(colMeans(model$error[,1:maxiter]),type='l',xlab='Epoch'
,
     main = strConf, ylab='Errors')
plot(as.vector(Yp), col = 'red', type='l',
     main = "Actual vs Predicted on Training Data Set",
     ylab = "Yt,Yp")
lines(as.vector(Yt), type = 'l', col = 'black')
```

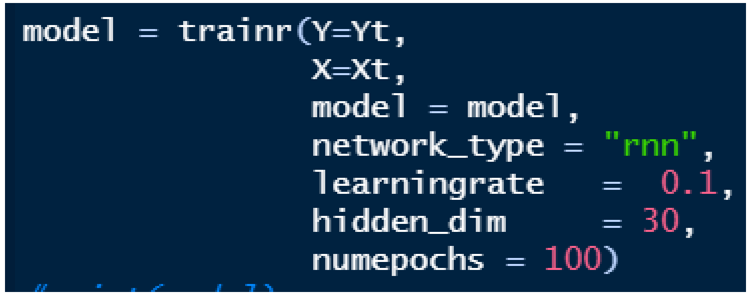


```
model = trainr(Y=Yt,
               X=Xt,
               model = model,
               network_type = "lstm",
               learningrate   =  0.1,
               hidden_dim     = 30,
               numepochs = 100)
```

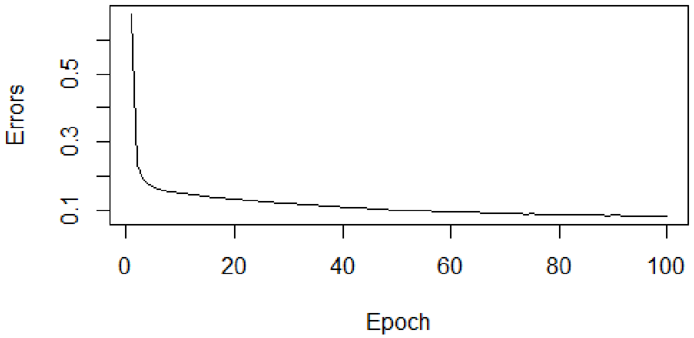


Actual vs Predicted on Training Data Set

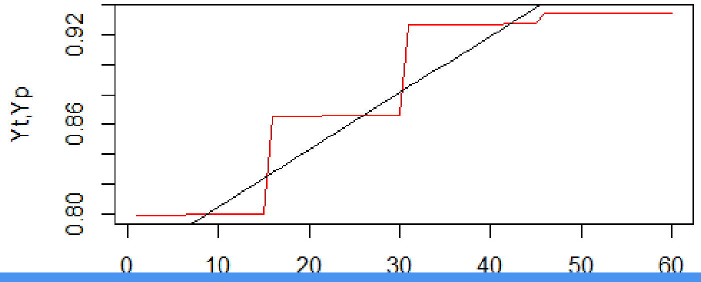


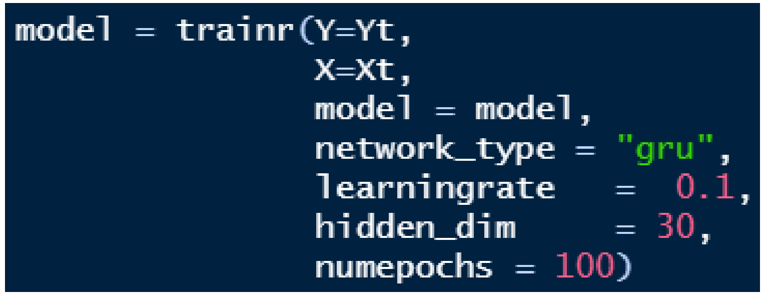


```
model = trainr(Y=Yt,
               X=Xt,
               model = model,
               network_type = "rnn",
               learningrate   =  0.1,
               hidden_dim     = 30,
               numepochs = 100)
```

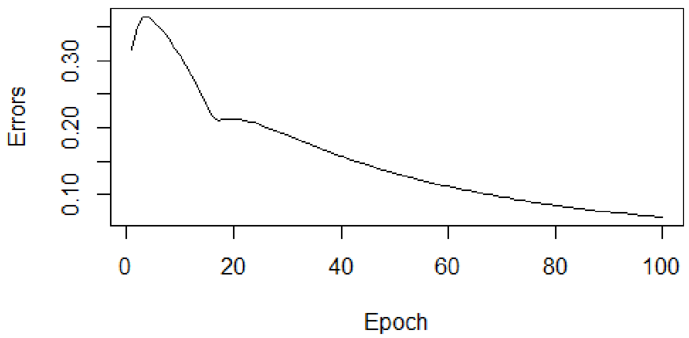


Actual vs Predicted on Training Data Set

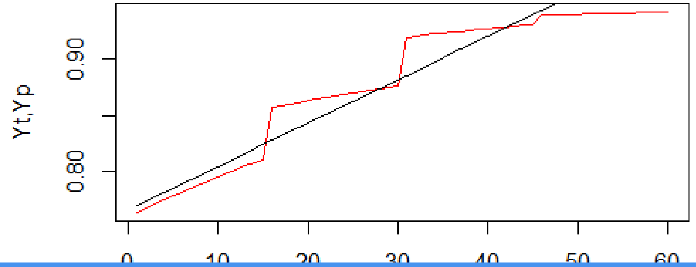




```
model = trainr(Y=Yt,
               X=Xt,
               model = model,
               network_type = "gru",
               learningrate   =  0.1,
               hidden_dim     = 30,
               numepochs = 100)
```



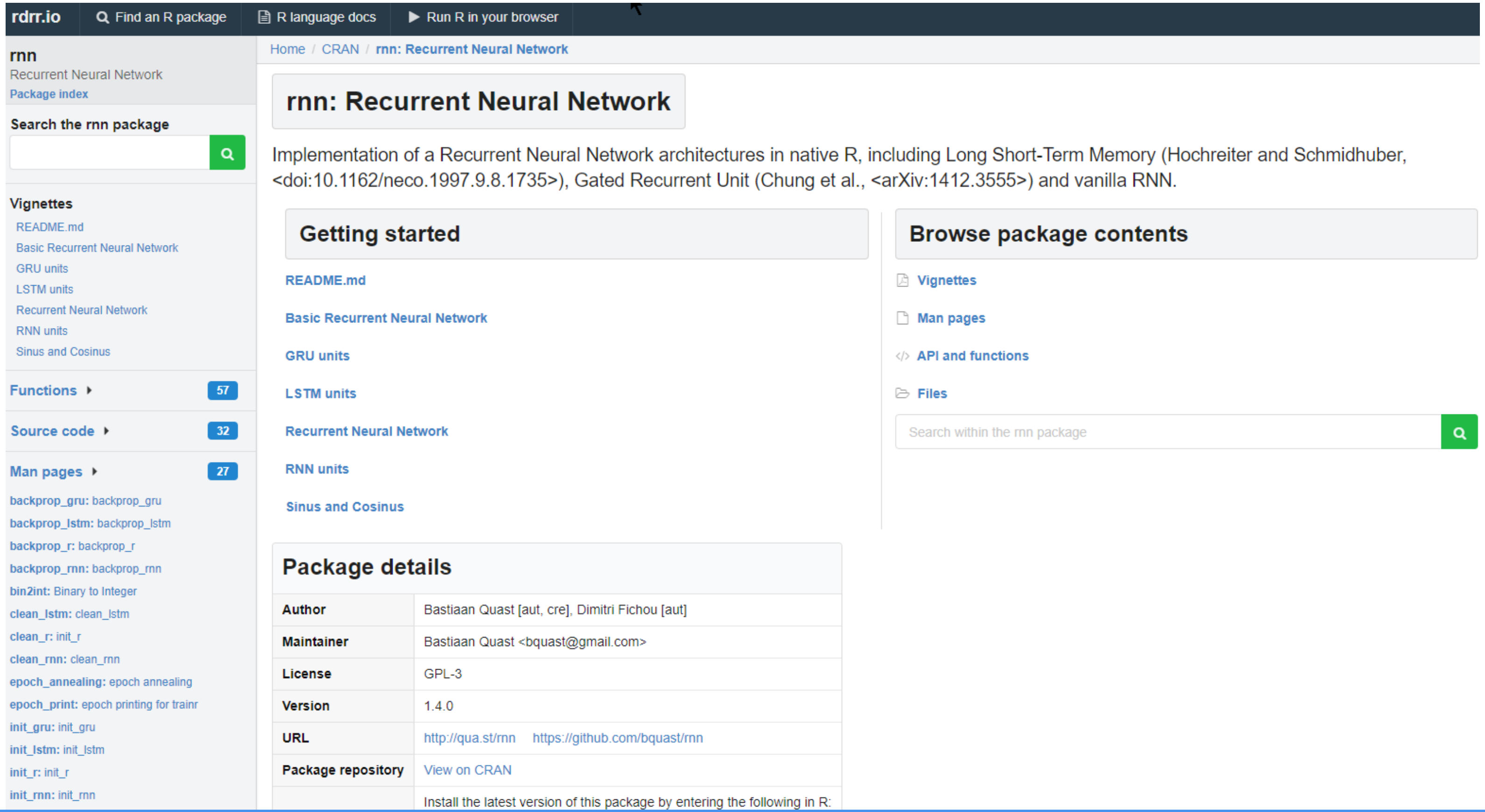
Actual vs Predicted on Training Data Set



DEVELOPMENT TRAINR()

Can I develop or modify the Trainr()?

- resource: https://rdrr.io/cran/rnn/src/R/trainr.R



rnn
Recurrent Neural Network
Package index

Search the rnn package

Vignettes

- README.md
- Basic Recurrent Neural Network
- GRU units
- LSTM units
- Recurrent Neural Network
- RNN units
- Sinus and Cosinus

Functions 57

Source code 32

Man pages 27

- backprop_gru: backprop_gru
- backprop_lstm: backprop_lstm
- backprop_r: backprop_r
- backprop_rnn: backprop_rnn
- bin2int: Binary to Integer
- clean_lstm: clean_lstm
- clean_r: init_r
- clean_rnn: clean_rnn
- epoch_annealing: epoch annealing
- epoch_print: epoch printing for trainr
- init_gru: init_gru
- init_lstm: init_lstm
- init_r: init_r
- init_rnn: init_rnn

rnn: Recurrent Neural Network

Implementation of a Recurrent Neural Network architectures in native R, including Long Short-Term Memory (Hochreiter and Schmidhuber, <doi:10.1162/neco.1997.9.8.1735>), Gated Recurrent Unit (Chung et al., <arXiv:1412.3555>) and vanilla RNN.

Getting started

- README.md
- Basic Recurrent Neural Network
- GRU units
- LSTM units
- Recurrent Neural Network
- RNN units
- Sinus and Cosinus

Browse package contents

- Vignettes
- Man pages
- API and functions
- Files

Search within the rnn package

Package details

Author	Bastiaan Quast [aut, cre], Dimitri Fichou [aut]
Maintainer	Bastiaan Quast <bquast@gmail.com>
License	GPL-3
Version	1.4.0
URL	http://qua.st/rnn https://github.com/bquast/rnn
Package repository	View on CRAN
	Install the latest version of this package by entering the following in R:

Activity 9.2

- Open NNDL_activity9_2.r

EXAMPLE LSTM IN C LANGUAGE

Activity 9.3

Let's compile and run code in activity 9.3 on Visual studio.

```
int largest_number = (pow(2, binary_dim));

double sigmoid(double x)

double dsigmoid(double y)

double dtanh(double y)

void int2binary(int n, int *arr)

void winit(double w[], int n)
```

```
class RNN
{
public:
        RNN();
        virtual ~RNN();
        void train();
public:
        double W_I[innode][hidenode];
        double U_I[hidenode][hidenode];
        double W_F[innode][hidenode];
        double U_F[hidenode][hidenode];
        double W_O[innode][hidenode];
        double U_O[hidenode][hidenode];
        double W_G[innode][hidenode];
        double U_G[hidenode][hidenode];
        double W_out[hidenode][outnode];
        double *x;
        double *y;
};
```

Summary