

# The Mathematics of Neural Networks

*19 Feb 2018 @ UCSY, Yangon, Myanmar*

Ye Kyaw Thu  
Researcher,  
Okayama Prefectural University, Okayama, Japan

Visiting Researcher,  
Waseda University, Tokyo, Japan

# Abstract of this Tutorial

- Machine Learning မှာ အရေးကြီးတဲ့ လက်ရှိ သုတေသန ပညာရှင်တွေ focus လုပ်နေကြတဲ့ Neural Network ရဲ့ concept ကို အကြမ်းမျဉ်း နားလည် စေရန်
- ဒီ tutorial မှာ ပါတဲ့ Slide တွေအားလုံးကို နားလည်ရင် Feed Forward Neural Network ရဲ့ အလုပ်လုပ်တဲ့ ပုံကို နားလည်ပြီ
- Error Function, Activation Function ဆိုတာကိုလည်း အိုက်ဒီယာ ရသွားပါလိမ့်မယ်
- ကိုယ်တိုင် image ဒေတာကိုပြင်ပြီး၊ ကျွန်တော်ပြင်ပေးထားတဲ့ shell scripts, python code တွေကိုပါ နားလည်အောင် ကြိုးစားရင်၊ run ကြည့်ရင် ရှိပြီးသား caffe model ကနေ feature ထုတ်တာကိုရော၊ နောက်ပြီးတော့ image classification လုပ်တဲ့ အခါမှာ လက်တွေ့ပြင်ဆင်ရမဲ့ preprocessing အလုပ်တွေ ကိုပါ အတွေ့အကြံရပါလိမ့်မယ်

# Table of Content

- A simple predicting machine
- Classification
- Human Brain
- Activation Function
- Introduction to Neural Network
- Matrix multiplication
- Make your own NN
- Q&A

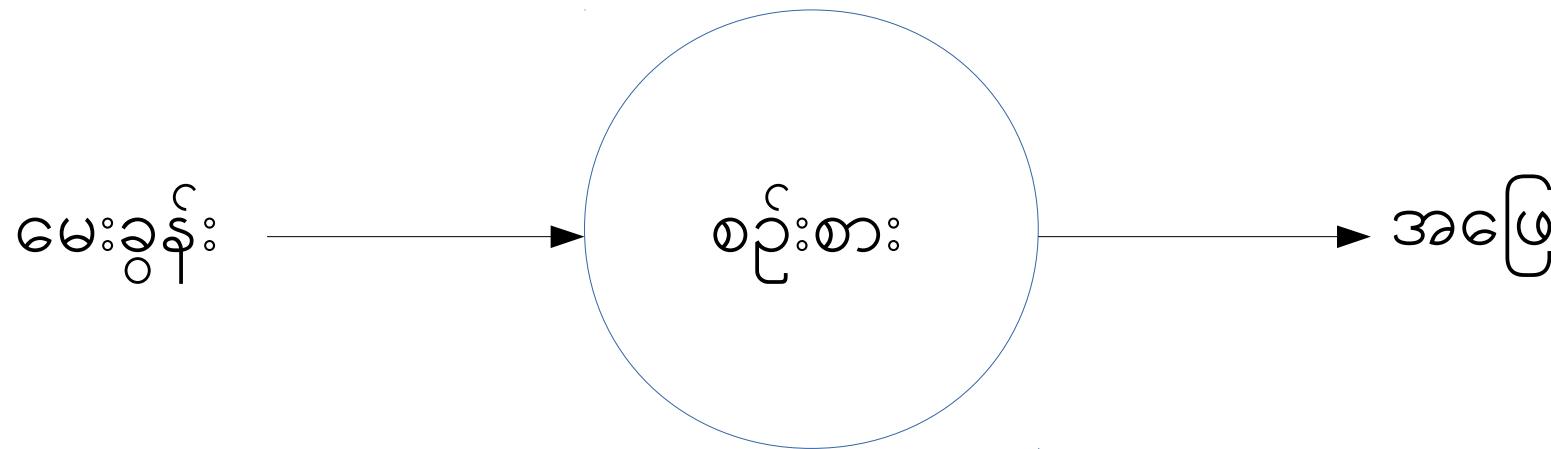
# A Simple Predicting Machine

- လက်ရှိ ကွန်ပျိုတာတွေရဲ့ အလုပ်လုပ်ပုံ နဲ့ လူရဲ့ ခီးနှောက် ကို အကြမ်းမျဉ်း နှိုင်းယှဉ်ကြည့်ရအောင်
- $24,567,890 \times 30,193,840$  ကို တွက်ပေးပါ (ကွန်ပျိုတာက အားသာ)
- Totoro ဆိုတဲ့ အရှပ်ကိုရွေးပါ  
(လူတွေ အတွက်က လွယ်ပေမဲ့ ကွန်ပျိုတာအတွက်က မလွယ်)



\* Image recognition က ဘယ်အရှပ်က ဘာဆိုတဲ့ knowledge ရှိဖို့ လိုအပ်တယ်!

# A Simple Predicting Machine



# A Simple Predicting Machine



# A Simple Predicting Machine



# A Simple Predicting Machine



ဆိုကြပါစွဲ

ဘယ်လိုတွက်ရသလဲ ဆိုတဲ့ ဖော်မြှုလာ ကို မသိဘူး။  
သိတေက နာရီ နဲ့ မိနစ်ရဲ့ ဆက်စပ်မှုက ရှိတယ်ဆိုတေကိုပဲ။  
(အဲဒါကို အင်္ဂလိပ်လို့ က linear relationship လို့ ခေါ်တယ်)

# A Simple Predicting Machine



အဲဒါကို သချုပ်ဖြေဆုံးမှုလုပ်များ ချရေးကြည့်ရင်

$$\text{နာရီ} = \text{မိနစ်} \times C$$

ကျွန်ုတ်တိုက အဲဒီ  $C$  (constant) ကို မသိတာ

# A Simple Predicting Machine



နောက်ထပ် ကျွန်ုင်တော်တို့ သိတာက ဘယ်နှစ်နာရီမှာ မိနစ်  
ဘယ်လောက် ဆိုတဲ့ example data တစ်ချို့။

(i.e. X and y)

ညပမာ ဒေဝါ	နာရီ (x)	မိနစ် (y)
၁	၀	၀
J	၃	၁၈၀

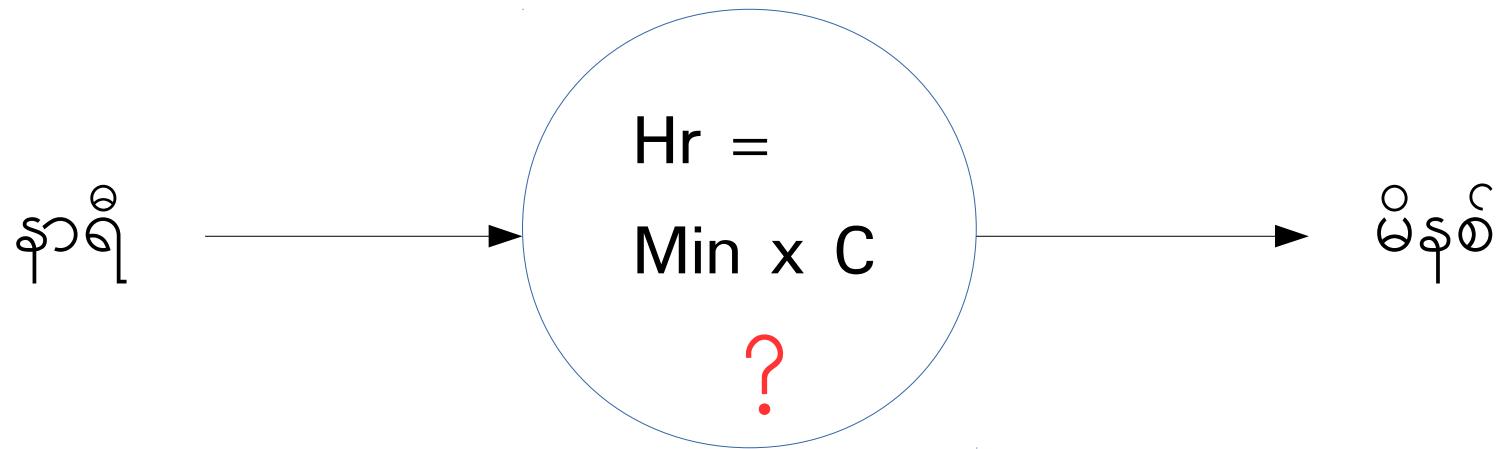
# A Simple Predicting Machine



ကျဉ်တော်တိုက သိရမှာ၊ သိချင်တာက အဲဒီ constant (C)

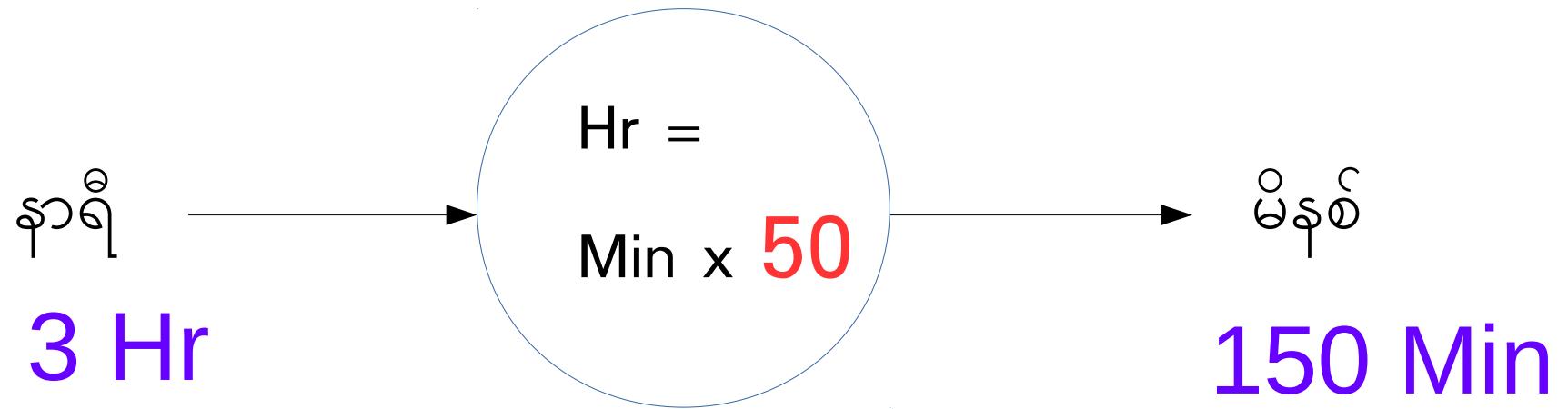
ကောင်းပြီ။  
ဘယ်လို တွက်ကြမလဲ။

# A Simple Predicting Machine



Constant (C) ကို တန်ဖိုး တစ်ခုခု ကျပ်န်း (Random) သတ်မှတ်  
လိုက်ပြီးတော့ တွေ့ကြတယ်။  
ရလာတဲ့ အဖြေ (Output) ကို အဖြေမှန် (Truth Answer) နဲ့  
တိုက်စစ်ပြီးတော့၊ C ကို အတိုးအလျှော့ လုပ်တဲ့ နည်း

# A Simple Predicting Machine



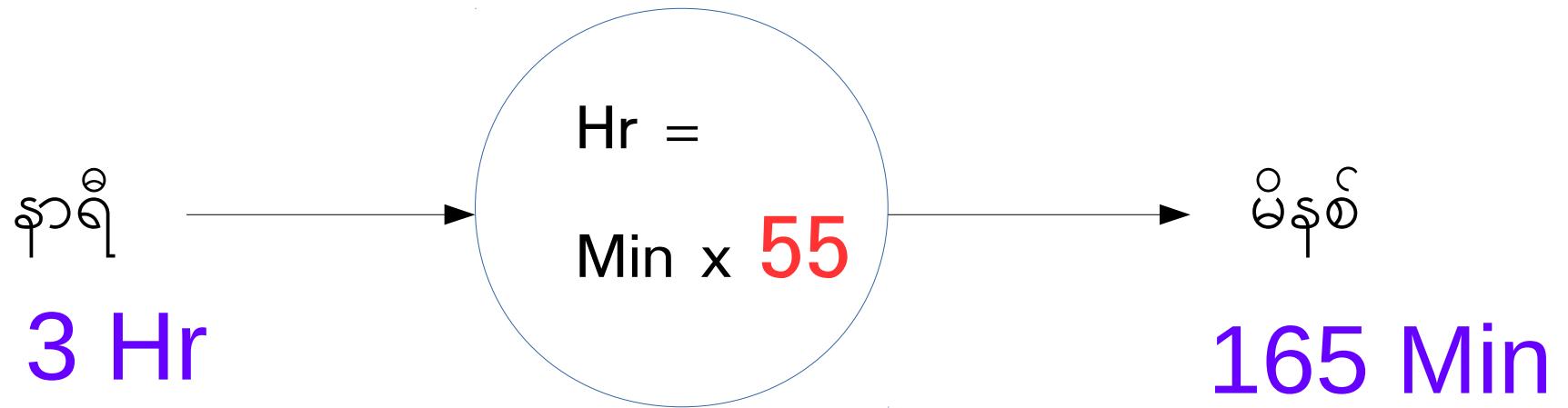
ကျော်တော်တို့ သိထားတဲ့ အဖြေမှန်နဲ့ တိုက်ကည့်ရင်

Error = Truth Answer – Calculated Output

$$= 180 - 150$$

$$= 30$$

# A Simple Predicting Machine



ကျွန်ုတ်တို့ သိတော်တဲ့ အဖြေမှန်နဲ့ တိုက်ကည့်ရင်

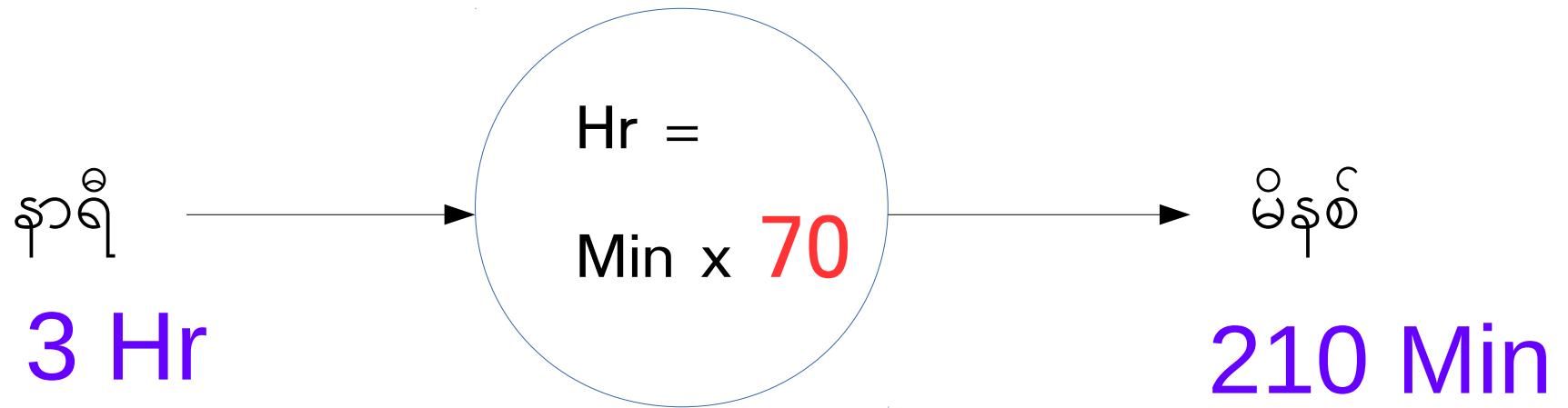
Error = Truth Answer – Calculated Output

$$= 180 - 165$$

15

# Progress!!!

# A Simple Predicting Machine



# ကျွန်ုပ်တော်ထို့ သိတော်းတဲ့ အဖြေမှန်နဲ့ ထိုက်ကည့်ရင်

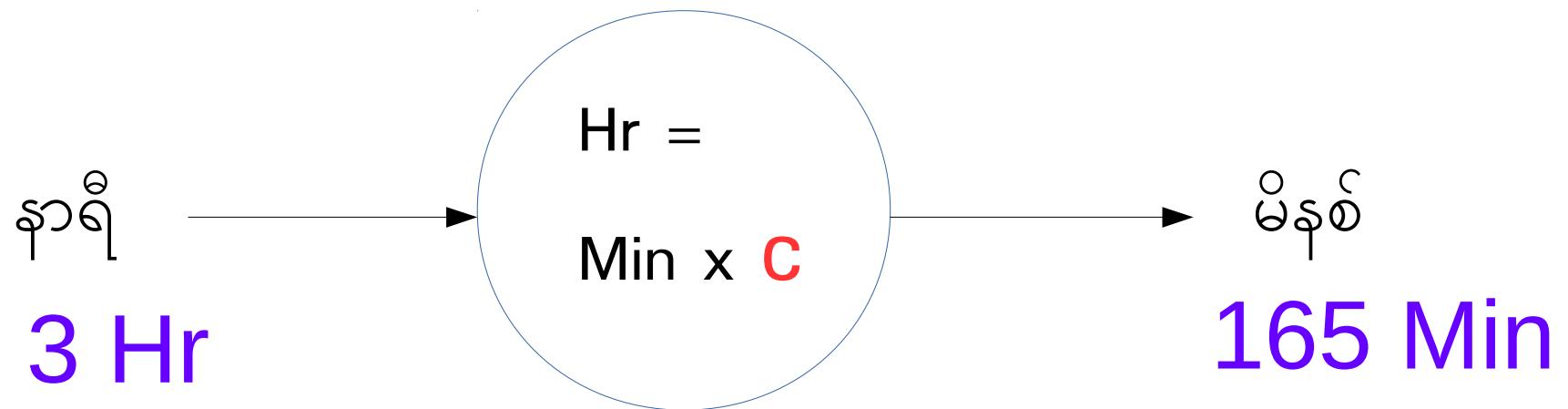
Error = Truth Answer – Calculated Output

$$= 180 - 210$$

$$= -30$$

# Overshot!!!

# A Simple Predicting Machine



\*\*\* Constant “c” က သေခါတယ် between 55 and 70

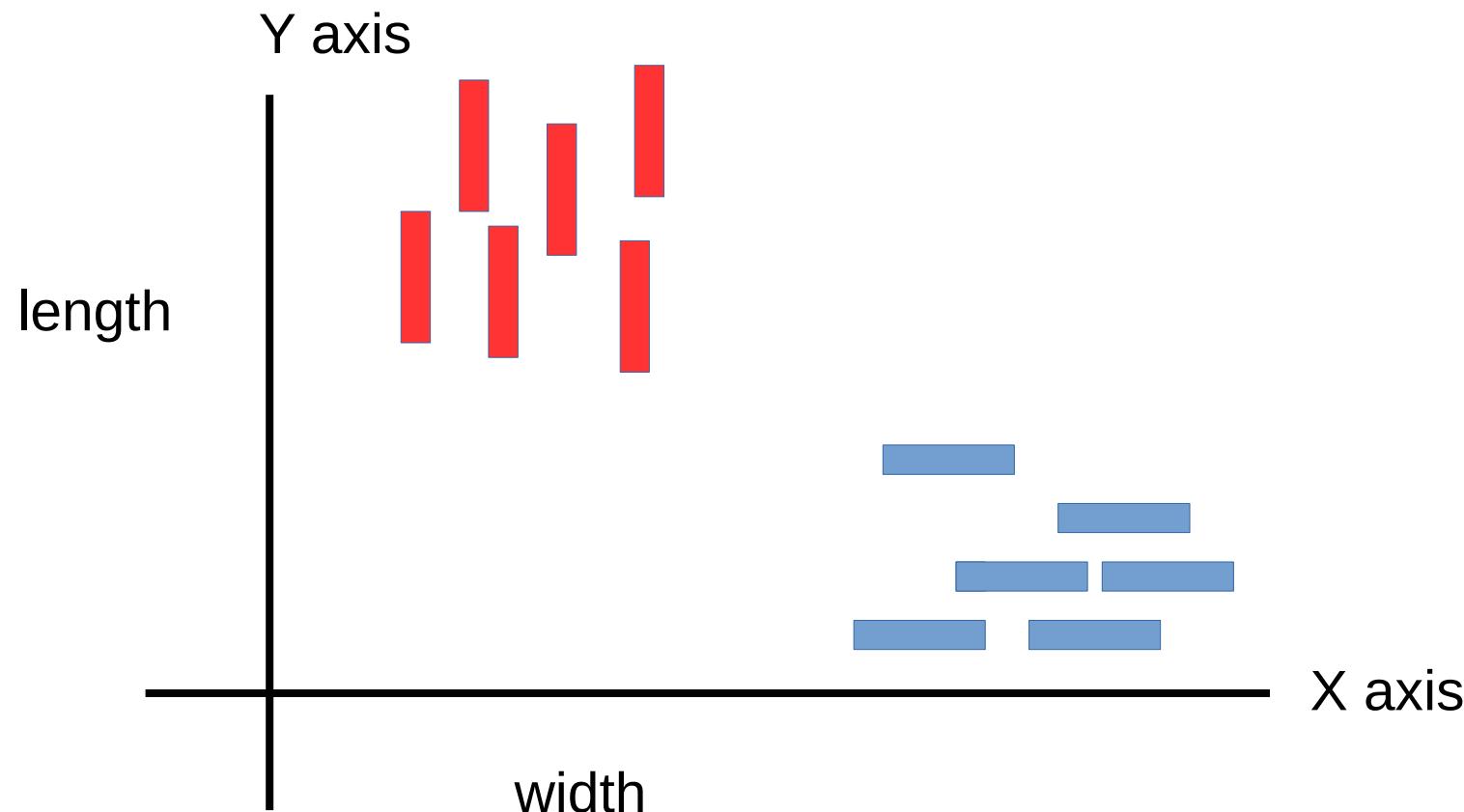
အဲဒါကြောင့်  $C=55$  ကနေ နည်းနည်းစီ တိုးသွားရင် အဖြောက်များပြီ။

# A Simple Predicting Machine

- ကွန်ပျိုတာမှာ က input ရယ် output ရယ် calculation လုပ်တဲ့  
အပိုင်းရယ် ဆိုပြီး ရုပိုင်းရှိ
- အခါးလိုပဲ Neural Network မှာလည်း အတူတူပါပဲ
- Calculation က ဘယ်လိုလုပ်တယ်ဆိုတာကို မသိရင်၊ input နဲ့  
output ကိုကြည့်ပြီး estimation လုပ်ကြည့်လို့ ရနိုင်
- မော်ဒယ် တစ်ခုအနေနဲ့ ပြောရရင်၊ အဖြေမှန် နဲ့ estimate လုပ်ပြီး  
တွက်လို့ ရတဲ့ ရလဒ်ကို နှင့်ယုဉ်၊ ရလာတဲ့ error value ကို ကြည့်  
ပြီး၊ အခါး မော်ဒယ်ရဲ့ parameter ကို tuning လုပ်လို့ ရတယ်

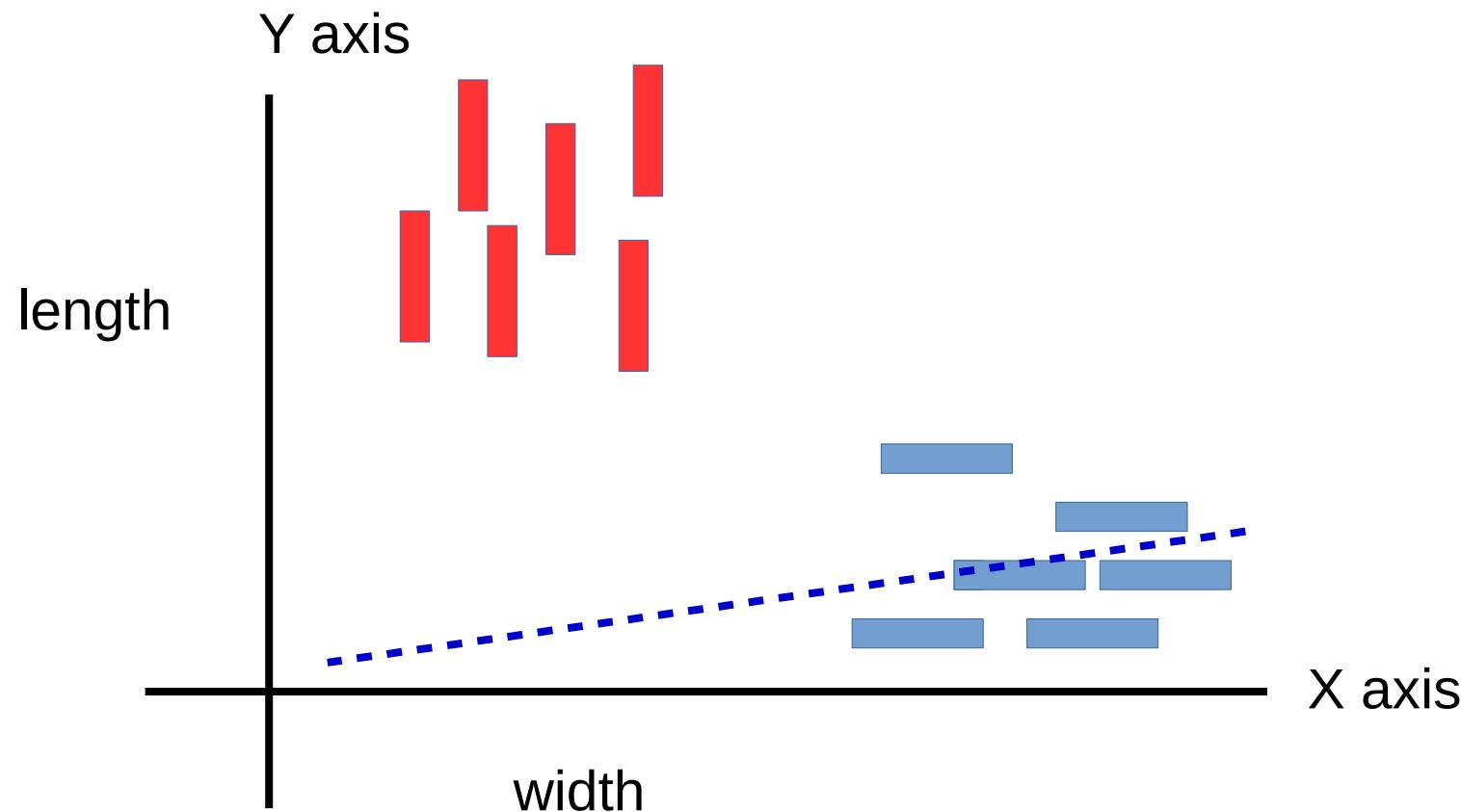
# Classification

- Machine Learning በ Classification  
እኩል: Predicting Problem ስለ



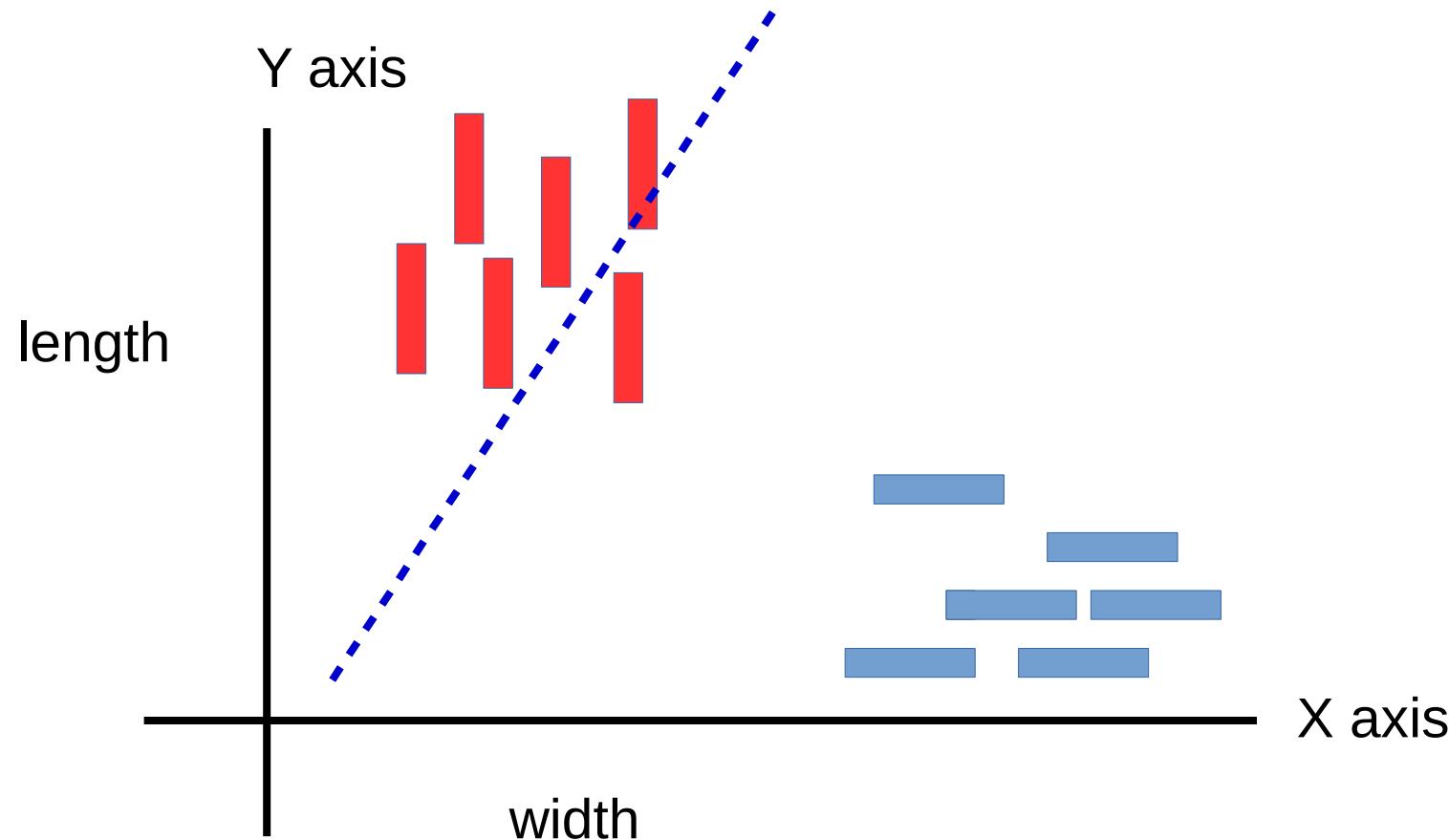
# Classification

- လွယ်လွယ်ရှင်းပြရရင် အပိုစုနစ်စုကို ခွဲ့စားတဲ့ လိုင်းကိုချေဘာ



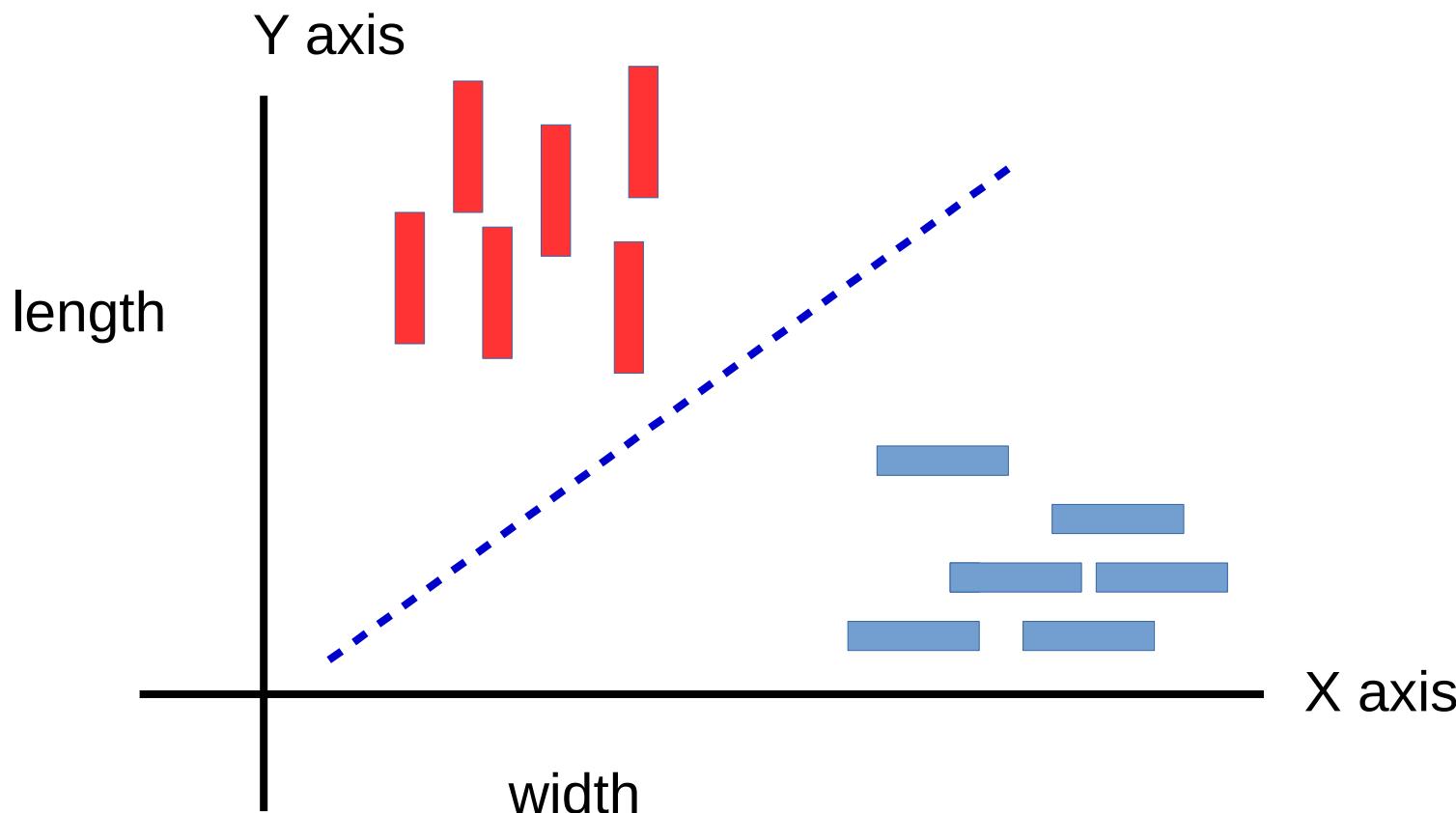
# Classification

- ရှေ့မှာ သင်ပေးခဲ့တဲ့ နာရီ-မိနစ် ပြဿနာလိုပါပဲ



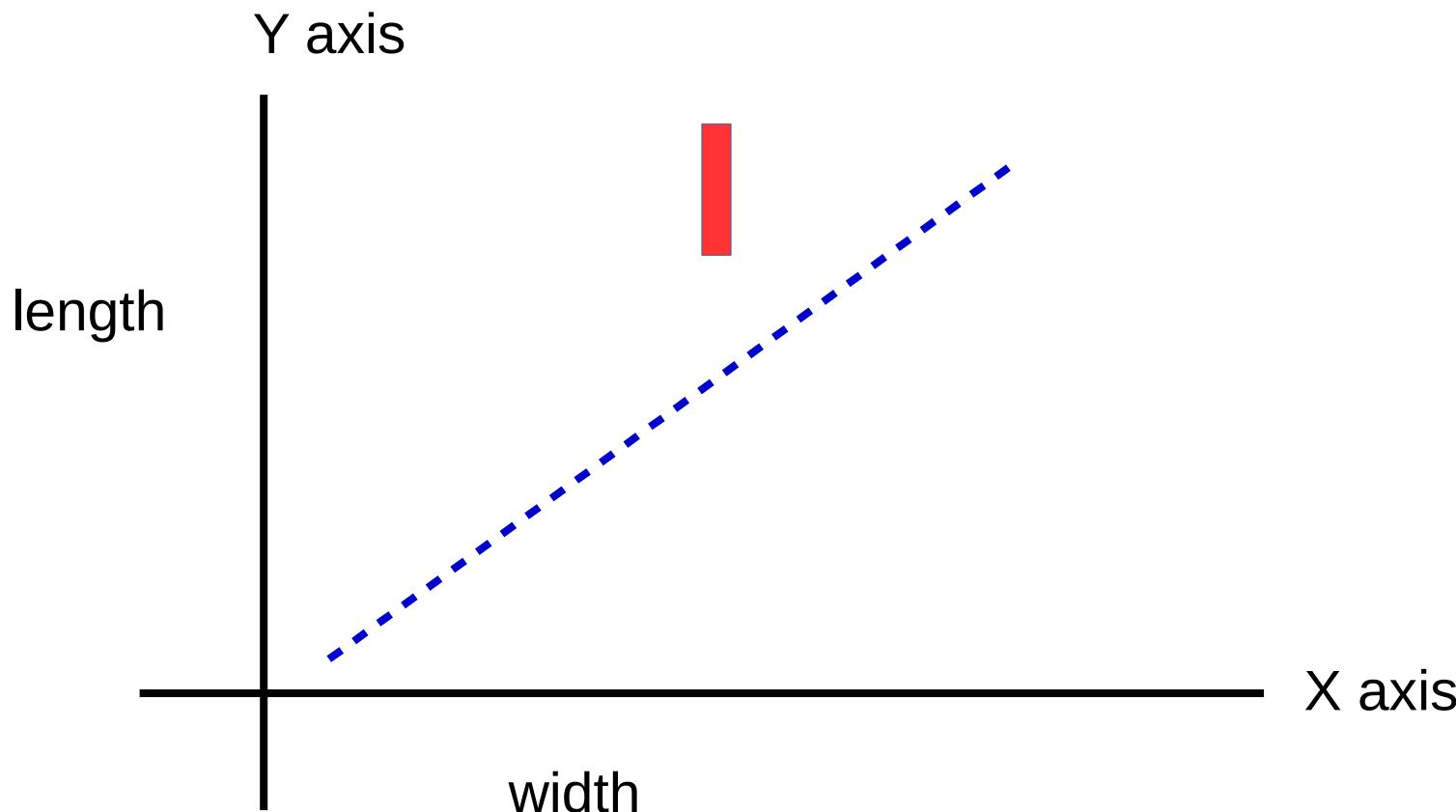
# Classification

- လိုင်းကို error ကိုတွက်ကြည့်ပြီး၊ slope ခဲ့တန်ဖိုးကို ပြောင်း၊ ပြန်နေရာချ
- အဲဒါကို ထပ်ခါထပ်ခါလှပ်
- နောက်ဆုံး training data ခဲ့အပ်စုနှစ်စုကို မှန်မှန်ကန်ကန်ခွဲလို့ ရတဲ့အထိ



# Classification

- အခြေဖော်ပို့ training data မှာ မပါတဲ့ object ကို input လုပ်ပြီး မှန်ကန် တဲ့ အုပ်စွဲပေးနိုင်သလား စစ်တာ (သို့) ဘယ်လောက် % မှန်မှန်ကန်ကန် အုပ်စွဲပေးနိုင်သလဲ စစ်တာကို testing လုပ်တယ်လို့ ခေါ်။ Evaluation လုပ်တယ်လို့ ခေါ်ပါတယ်။



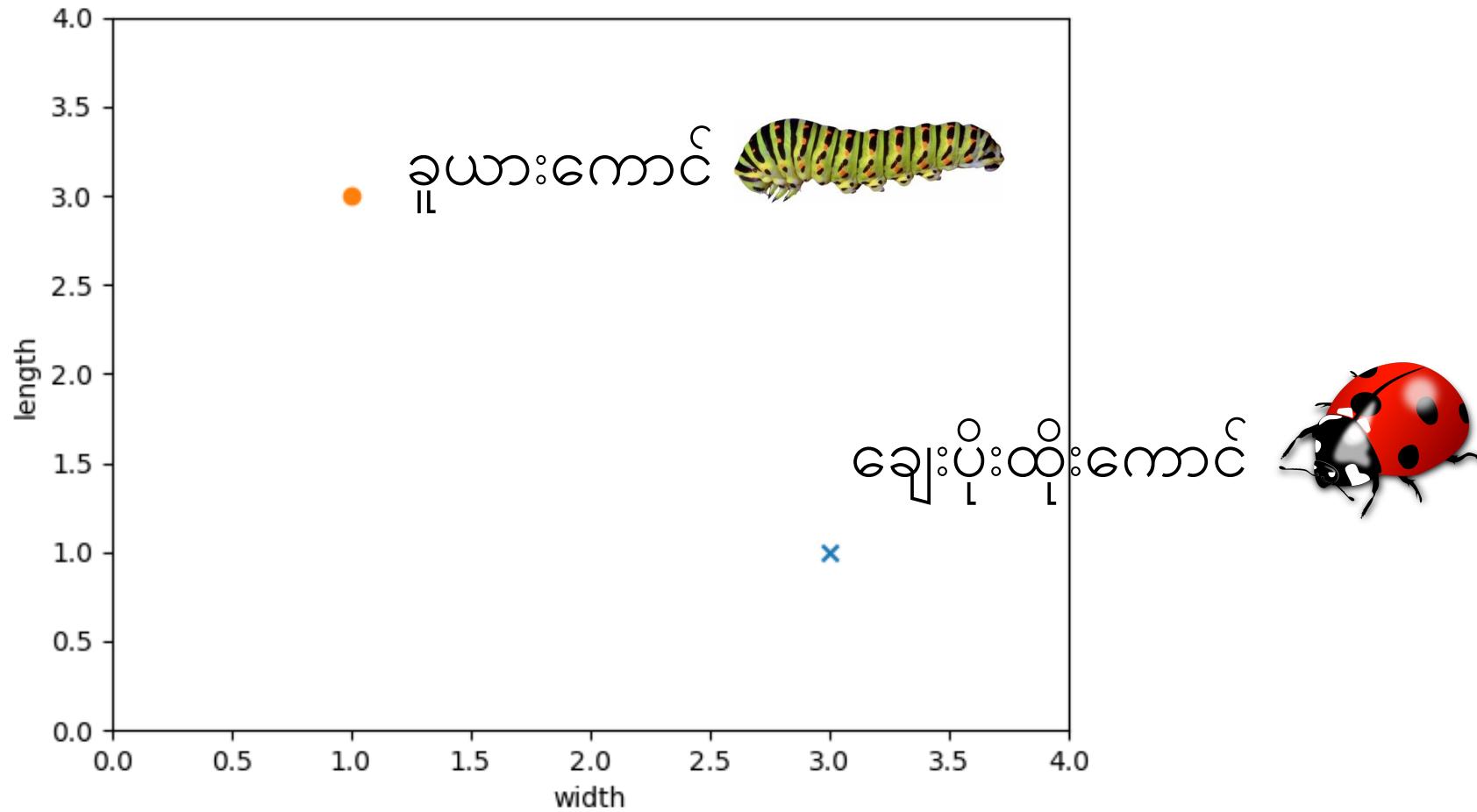
# Classification

- ဒီအထိက Classification ကို Prediction နဲ့ နှင်းယူဉ်ပြီး  
သဘောတရားကို သိအောင် အကြမ်း ရှင်းပြခဲ့
- လက်တွေ့မှာက အဲဒီ slop ကိုဘယ်လိုတွေက်သလဲ ဆိုတာက  
အရေးကြီးတယ်
- ဥပမာ ခယားကောင် နဲ့ ချေးပိုးထိုးကောင် ကို classification လုပ်  
ရအောင်

Example	Width	Length	Bug
1	3.0	1.0	ခယားကောင်
2	1.0	3.0	ချေးပိုးထိုးကောင်

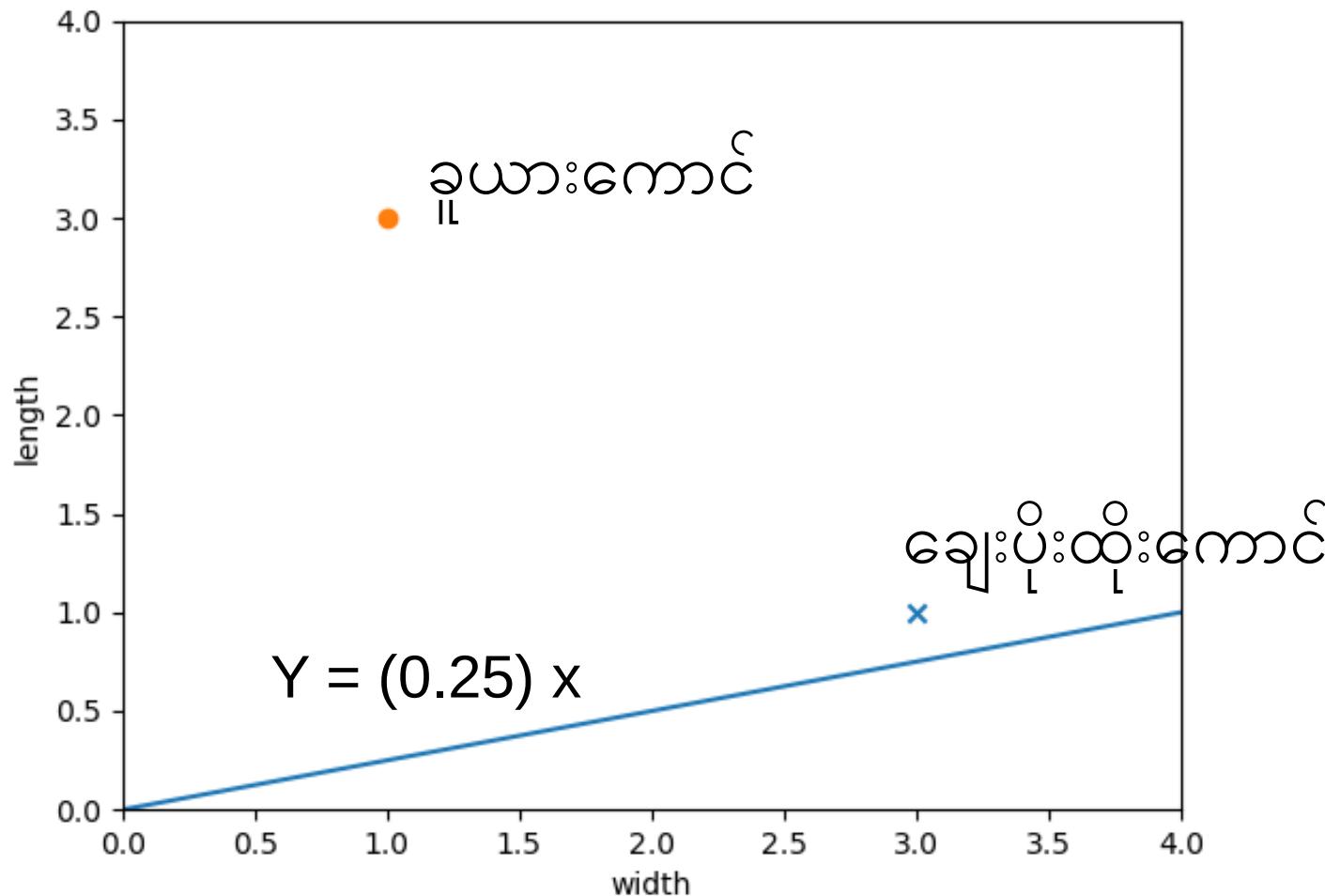
# Classification

- ရှိနေတဲ့ ဒေတာကို ဂရုံး မှာ နေရာချုပည့်ရအောင်



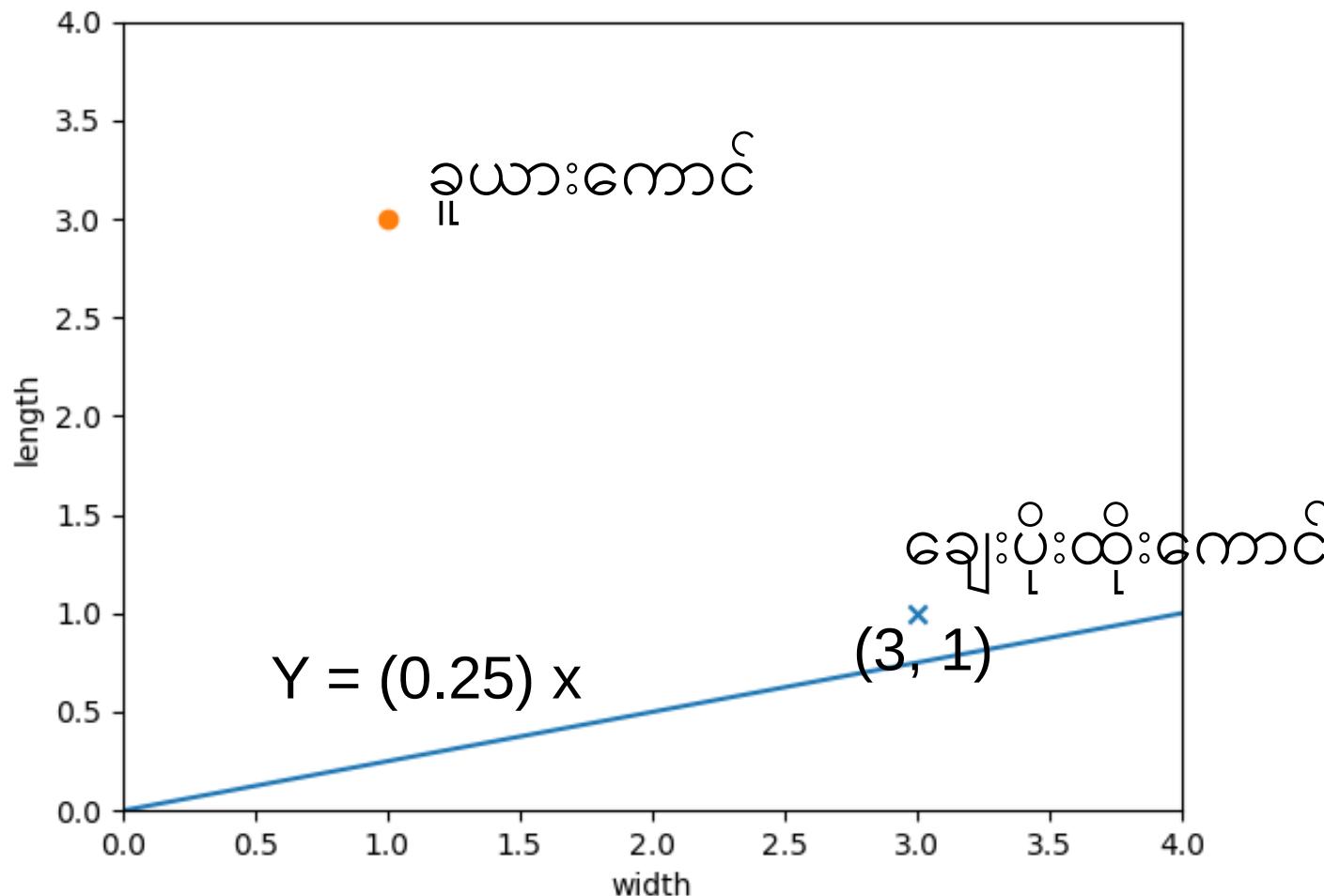
# Classification

- Linear Function ကိုသုံးလိုရတယ်
- $Y = Ax$ ,  $A$  ကို 0.25 ထားကြည့်ရင်
- $A$  ကို slope control လုပ်တယ်



# Classification

- ချေးပိုးထိုးကောင်က  $x=3.0$ ,  $y=1.0$
- ငါတို့ရတဲ့ လက်ရှိအဖြောက်  $y = Ax$ ,  $= (0.25) \times (3.0) = 0.75$
- တစ်ခုသတိထားရမှာက၊ ငါတို့က  $y=1.0$  လုပ်လို့ မရဘူး

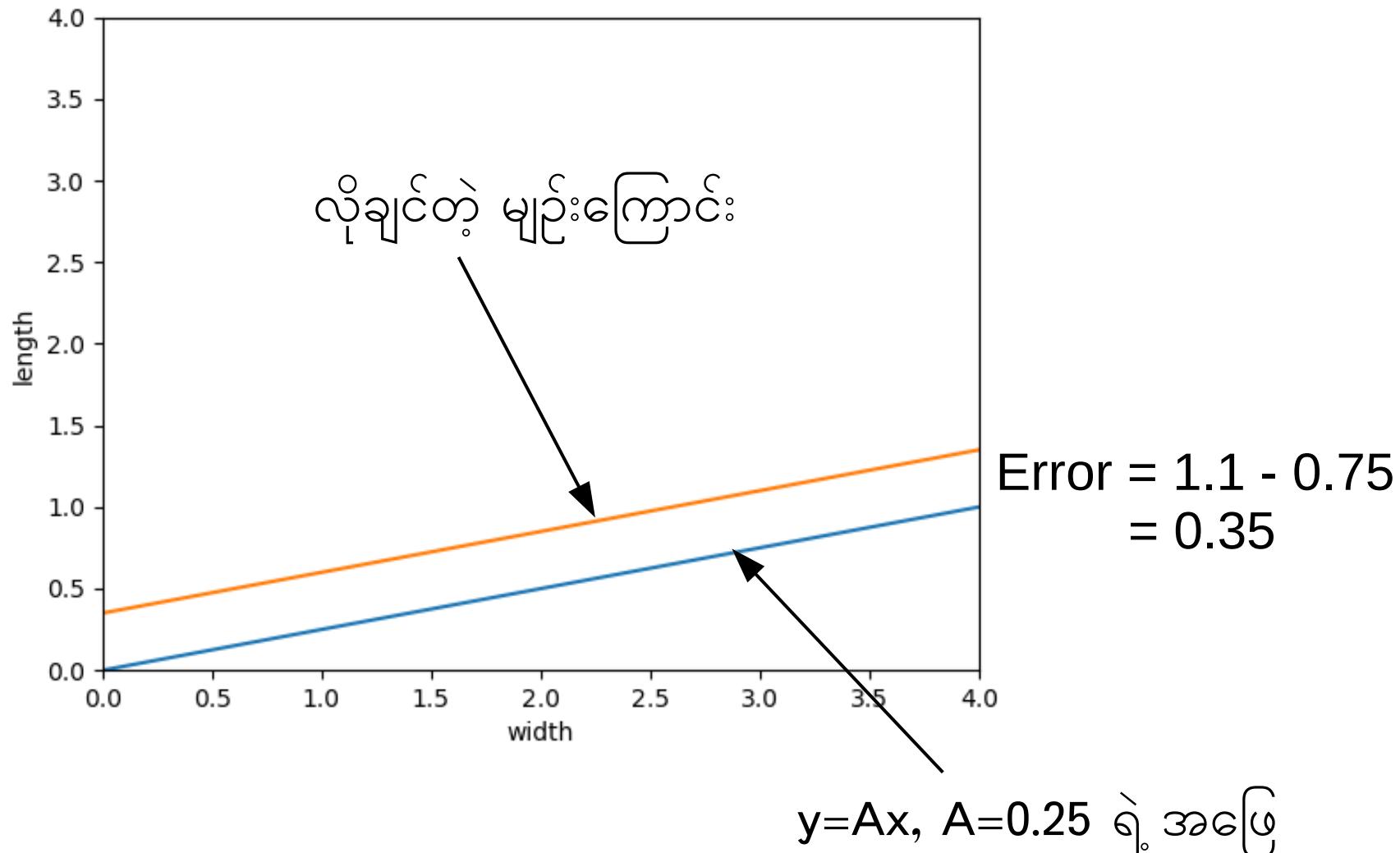


# Classification

- လိုချင်တဲ့ y value ကို 1.0 မထေားပဲ 1.1 ထေားမယ်
- ဘုရားကြောင်းလဲ ဆိုတော့ ချွေးပိုစိုးကောင်ရဲ့ အထက်မှာ  
မျဉ်းကြောင်းကို ရှိစေချင်တာမိုလို
- Error Function က တွေကိုပုံအတူတူပဲ
- $\text{Error} = (\text{လိုချင်တဲ့အဖွဲ့} - \text{Algorithm ရဲ့ ရလဒ်})$   
 $= 1.1 - 0.75 = 0.35$

# Classification

- မြင်သအောင် ဂရဖ်မှာ မျဉ်းကြောင်း ပြည့်စုံကို နှင့်ယဉ်ကည့်ရအောင်

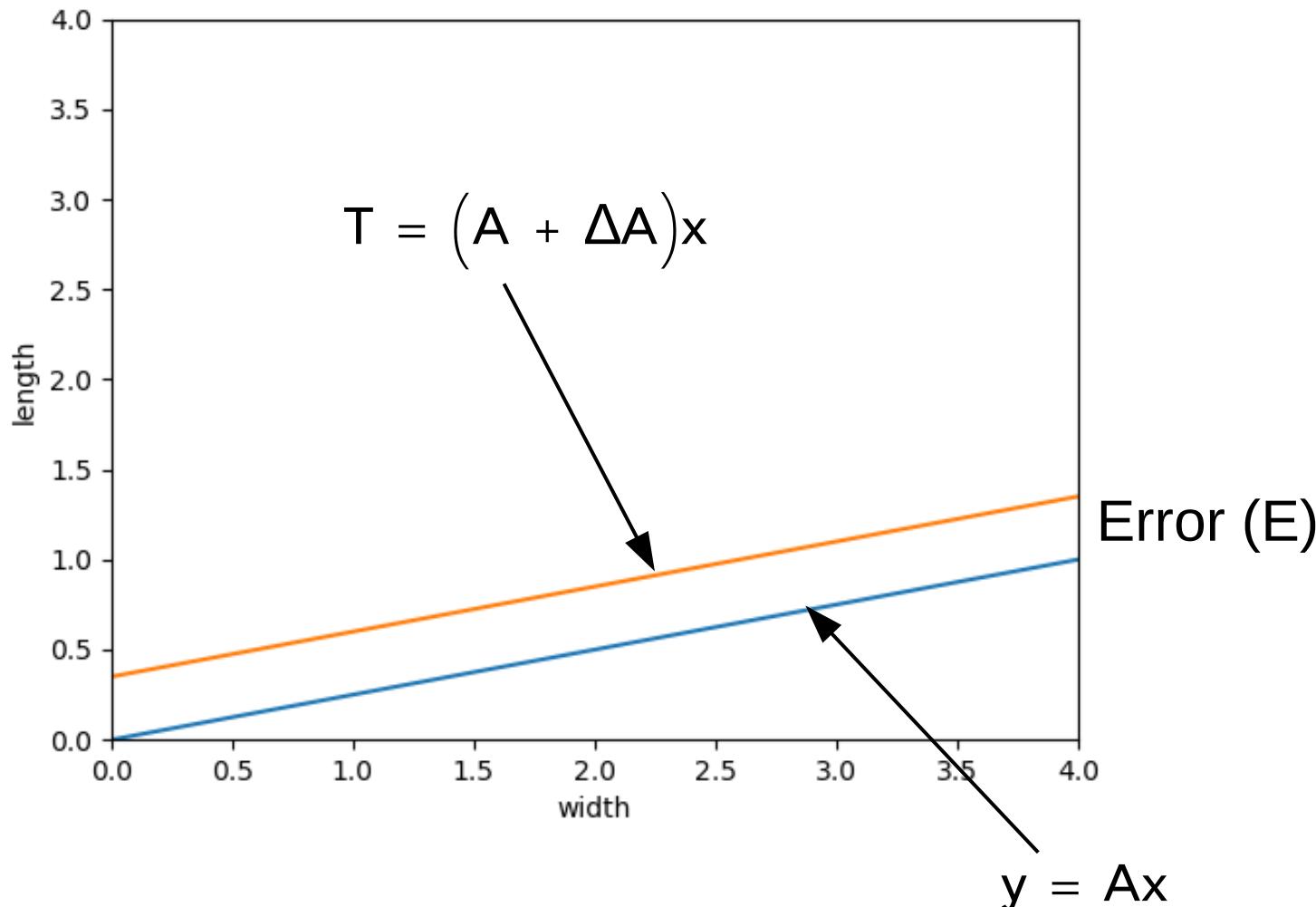


# Classification

- အရေးကြီးတဲ့ မေးခွန်းက Error ကနေ ပိုကောင်းတဲ့ parameter A ကို ဘယ်လို လုပ်ယူကြမလဲ ဆိုတဲ့ ကိစ္စ
- အသေးစိတ်စဉ်းစားကြည့်ရင် Error (E) နဲ့ parameter A နဲ့ က ဘယ်လို ချိတ်ဆက်မှု ရိုနေတာလဲ ဆိုတာပဲ
- သုံးခဲ့တဲ့ linear function က  $y = Ax$  မှာ A အတွက်ကျွန်တော်တို့ ထားခဲ့တဲ့ initial value နဲ့ ရလာတဲ့ y က မှားနေတယ်ဆိုတာ
- သိနေတာက လိုချင်တဲ့ target value (t) ကိုရပိုအတွက် အဲဒီ A ကို နည်းနည်း စိုးအတိုးအလျှော့လုပ်သွားရမယ်ဆိုတာကို
- အဲဒီကို သချာ equation အဖြစ် ချေရေးရင် အောက်ပါအတိုင်း  
$$t = (A + \Delta A) x$$

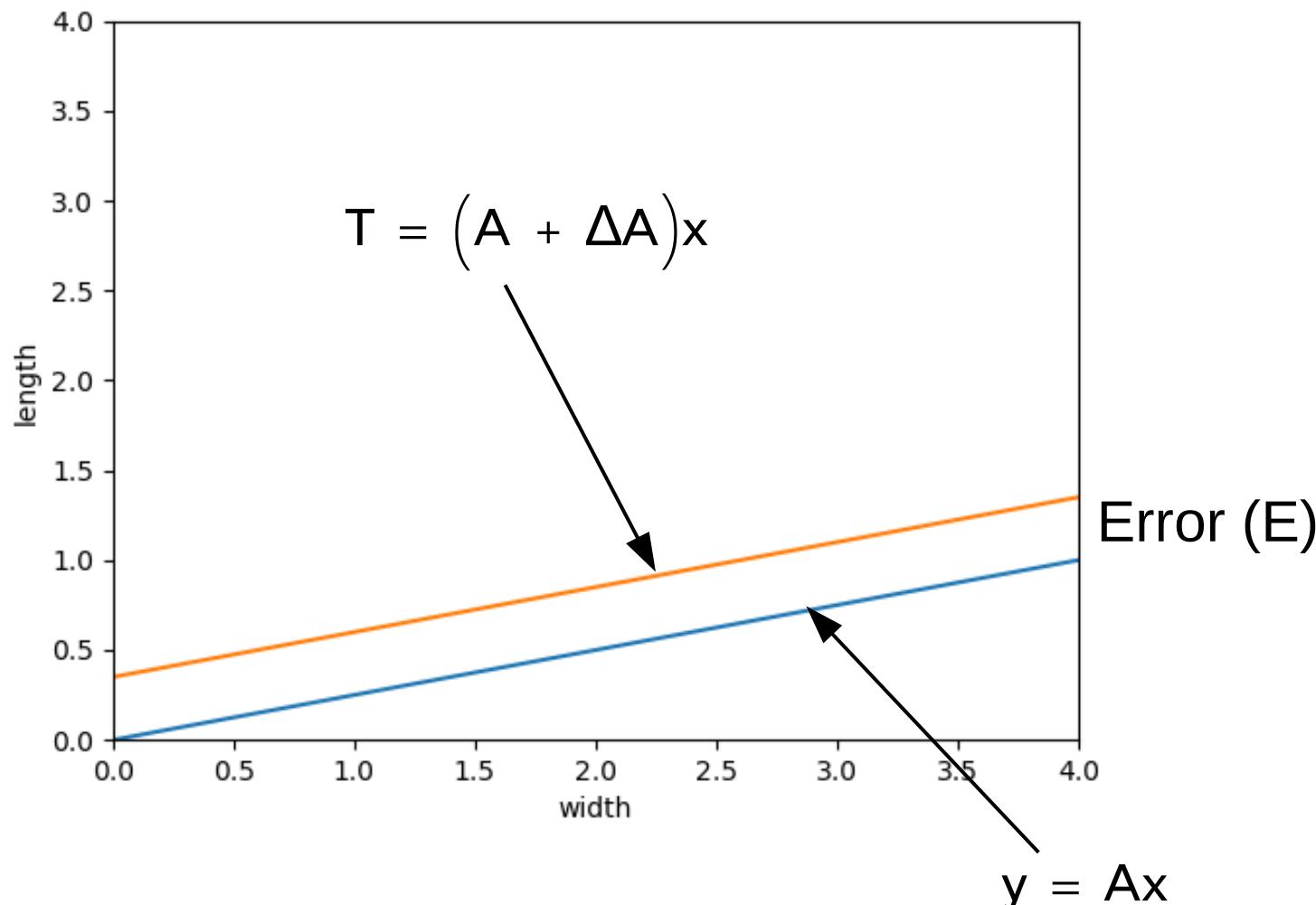
# Classification

- မြင်သာအောင် ဂရဖိန့် ပြမယ်ဆိုရင်



# Classification

- ရှေ့က နာရိ-မိနစ် ပြဿနာမျာလိုပဲ Error (E) =  $t - y$   
 $t - y = (A + \Delta A)x - Ax$



# Classification

- Error ( $E$ ) =  $t - y$

$$t - y = (A + \Delta A)x - Ax$$

ပိုပြီးတော့ ရှင်းအောင်ရေးမယ်ဆိုရင် အောက်ပါအတိုင်း

- $E = t - y$

$$E = Ax + (\Delta A)x - Ax$$

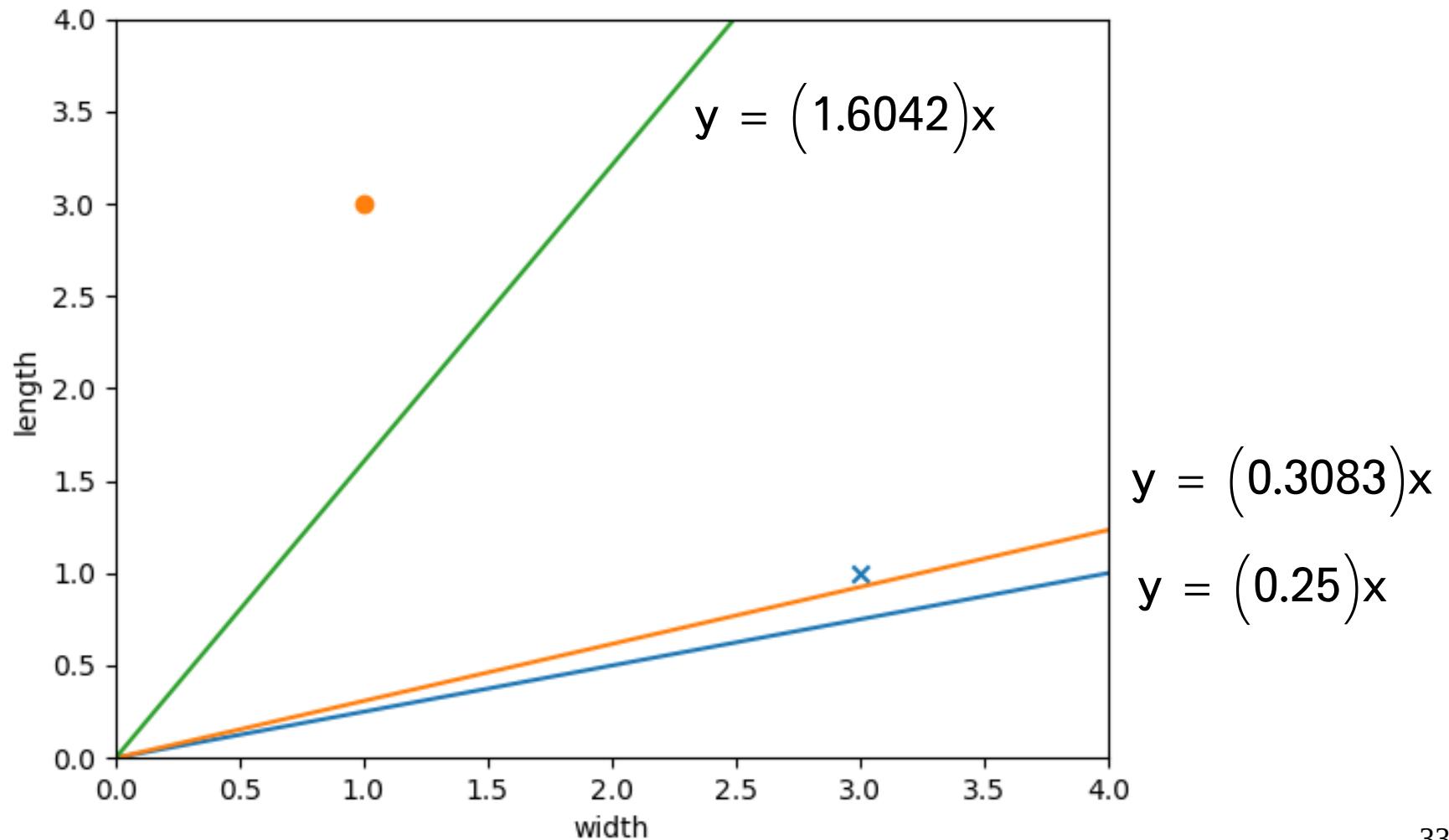
$$E = (\Delta A)x$$

That's remarkable!!!

$E \propto \Delta A$  ရဲ့ relationship က အရမ်းကို  
နားလည်ရလွယ်တယ်။ မှားများမှားနေသလား။

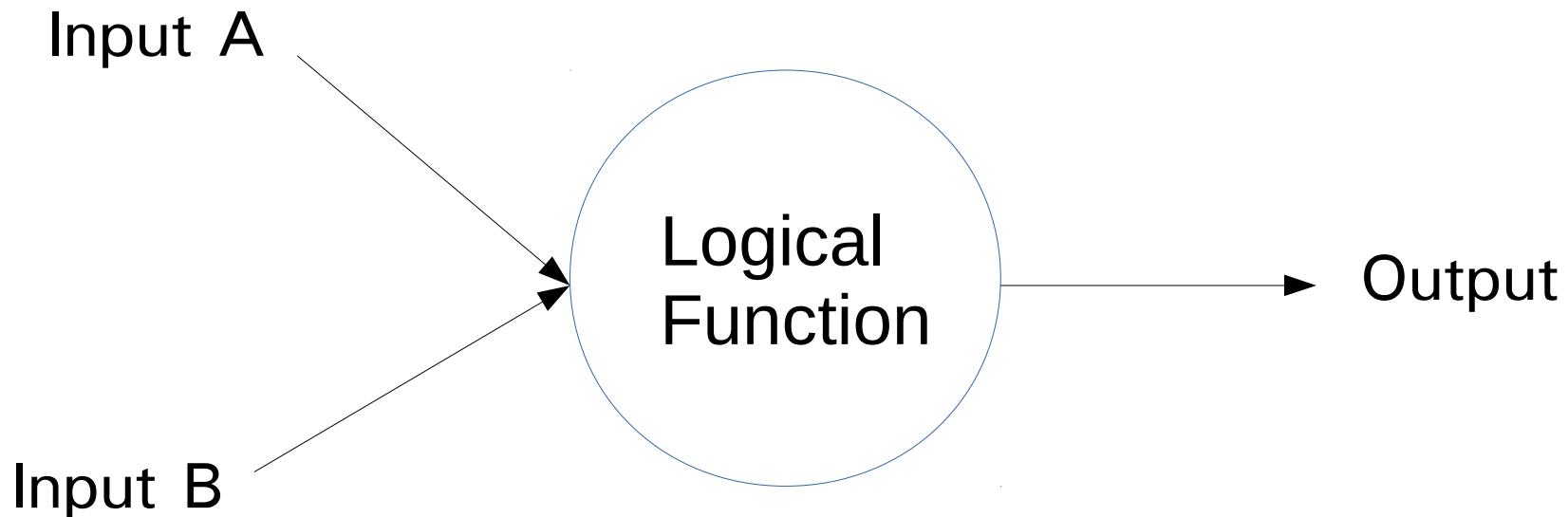
# Classification

- ဒဲ algorithm ကိုသုံးပြုတော့ A တန်ဖိုးကို အတိုးအလျော့လှပ်သွားရင်နဲ့ classification လုပ်တယ်



# Classification

- တကယ့်လက်တွေ့လောကမှာ classifier တစ်ခုတည်နဲ့က မပြောလည်တဲ့  
ပြဿနာတွေချည်းပဲ
- Logical function နဲ့ ဥပမာ ပေးပြီး predicting machine ဆောက်ကြည့်  
ရအောင်



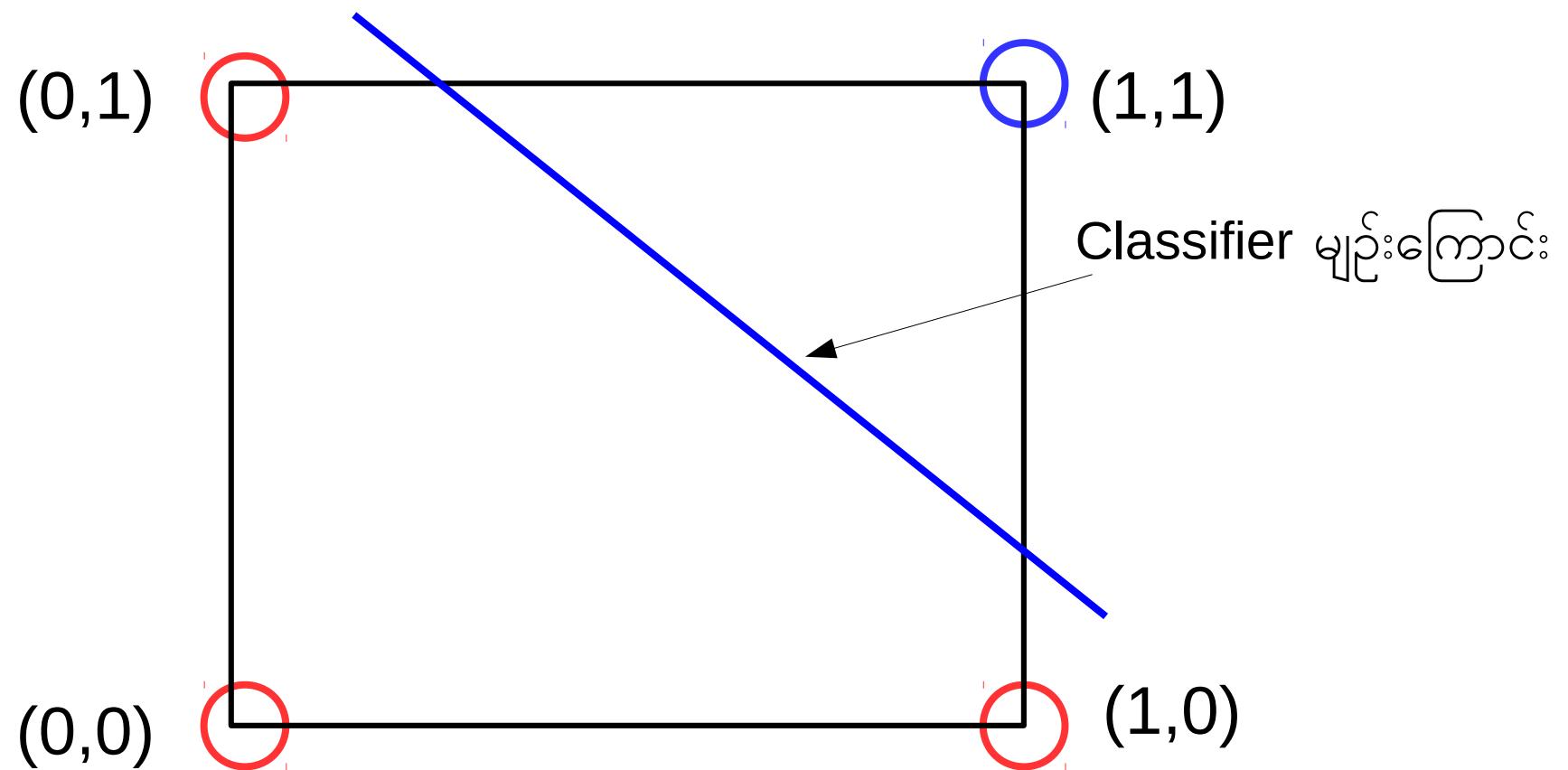
# Classification

- Logical AND  $\wedge$  OR

Input A	Input B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

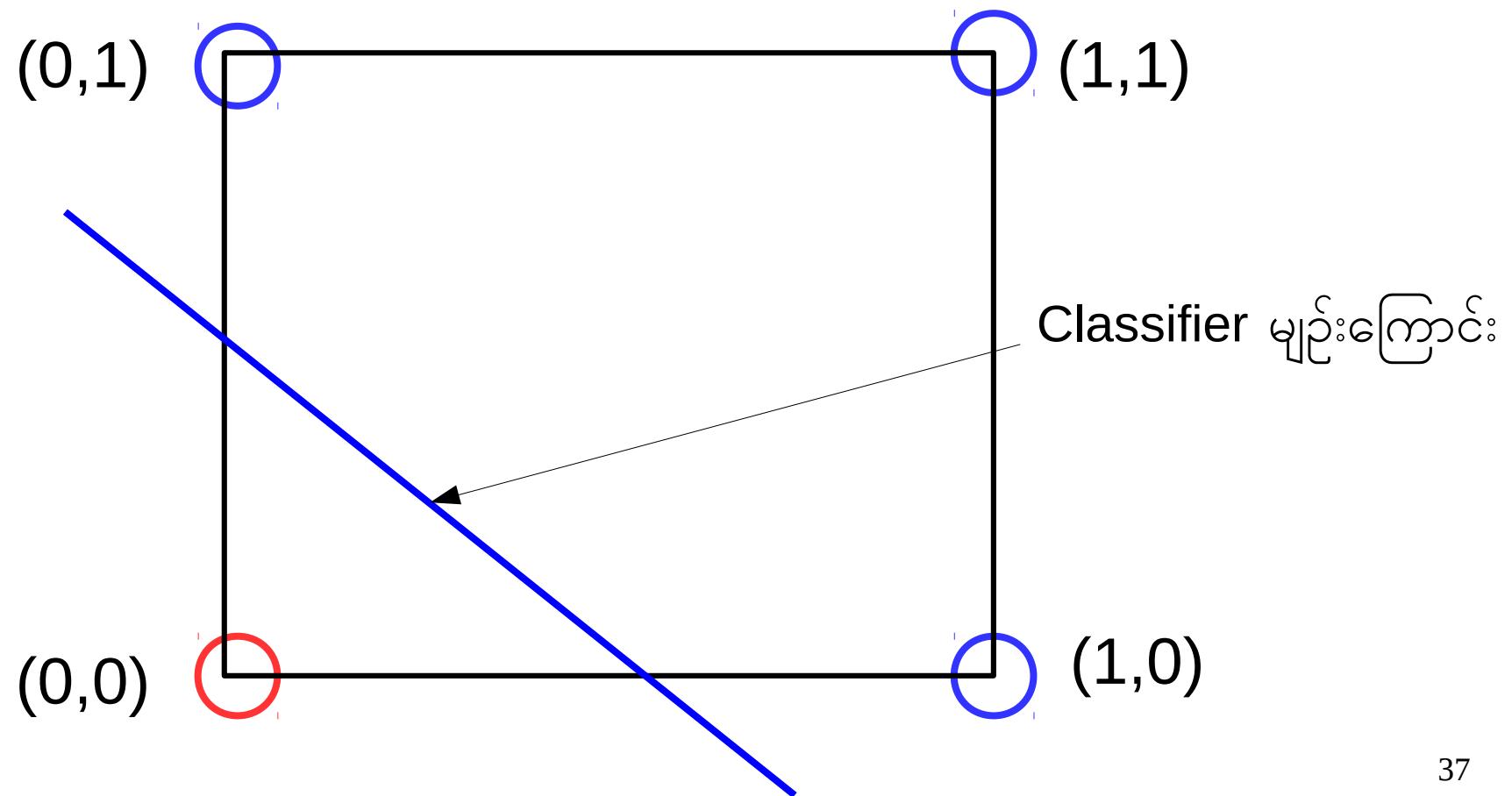
# Classification

- Logical AND အတွက် Classifier ကိစ္စားစားကည့်ရင်



# Classification

- Logical Boolean OR အတွက် Classifier ကိစ္စုံးစားကြည့်ရင်
- ဒီနေရာမှာက သချုပ်အသေးစိတ်မသွားတော့ဘူး၊ အိုက်ဒီယာကိုပဲ ရဖော်ပေါ်ထောင်



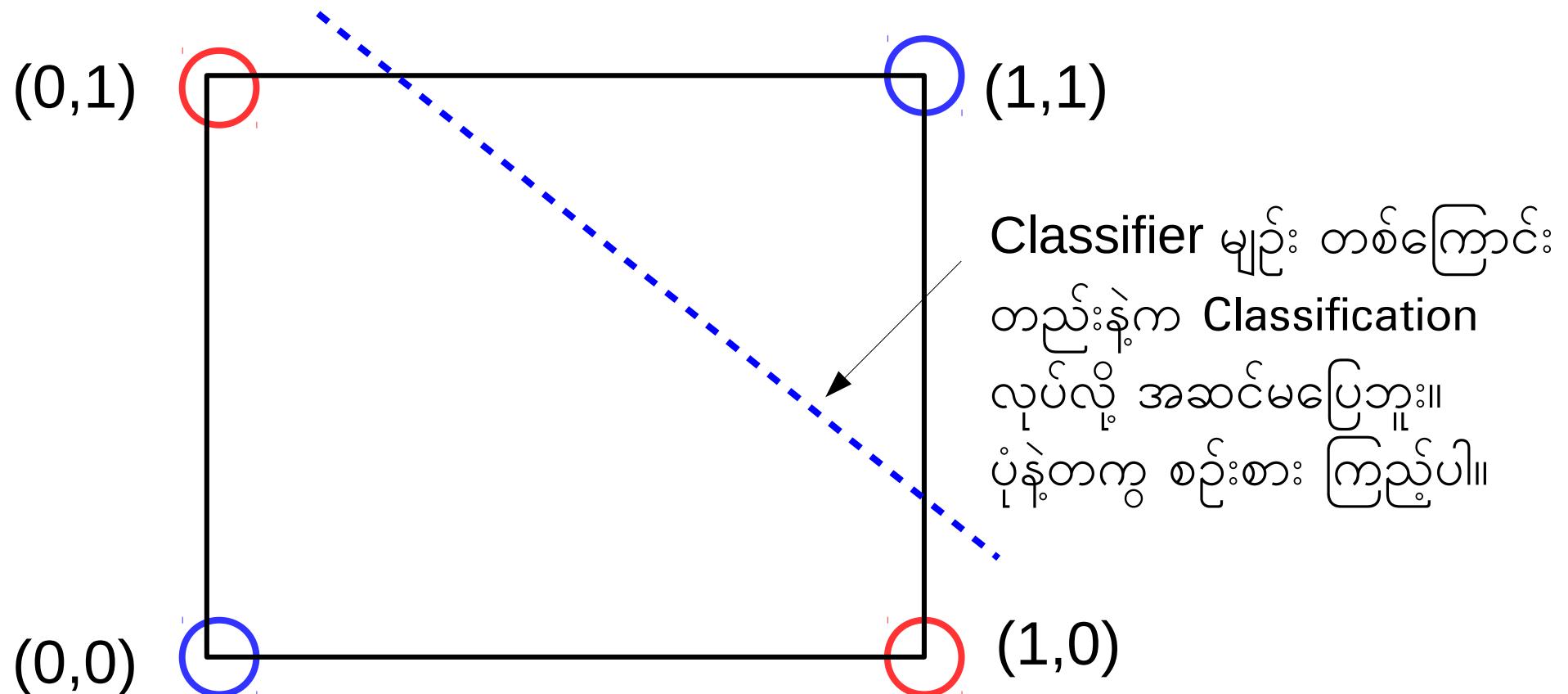
# Classification

- Logical XOR (eXclusive OR), input A နဲ့ input B နှစ်ခုမှာ တစ်ခုခဲ့က true ဆိုရင် true ဖြစ်တယ်
- ဒါပေမဲ့ နှစ်ခုစလုံးက false သို့ true ဖြစ်နေရင် XOR ရဲ့ အဖွဲ့က false ဖြစ်တယ်

Input A	Input B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

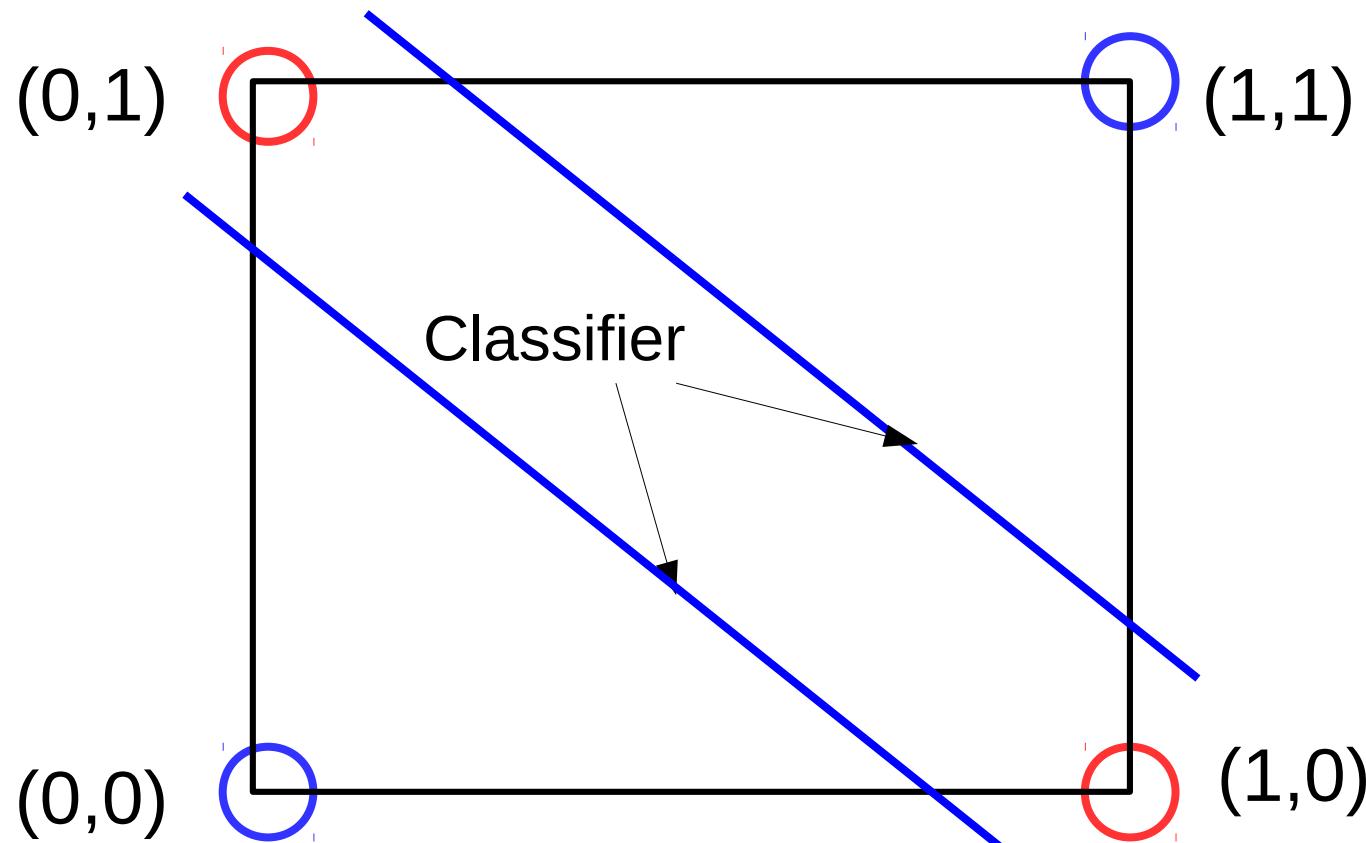
# Classification

- Logical XOR အတွက် Classifier ကိစ်းစားကြည့်ရင်



# Classification

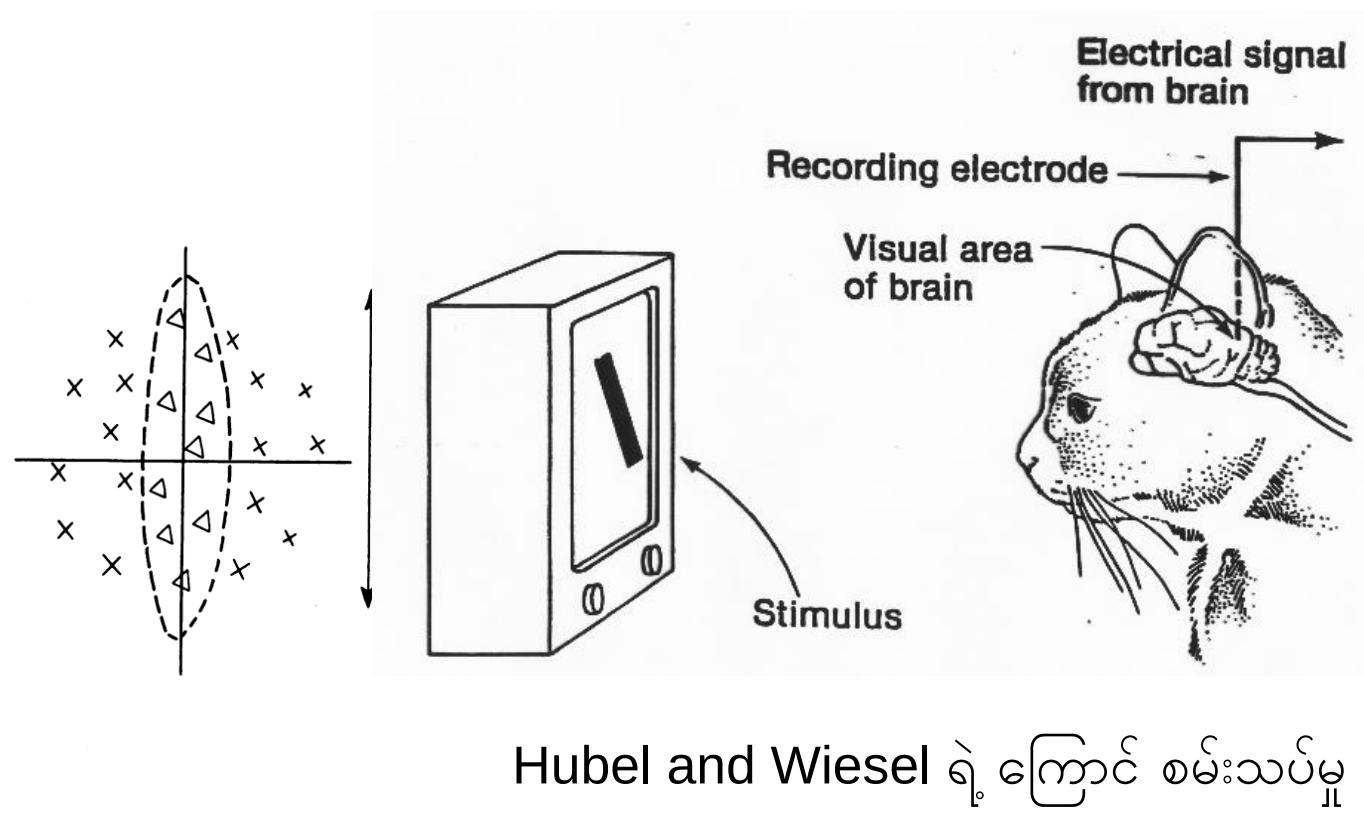
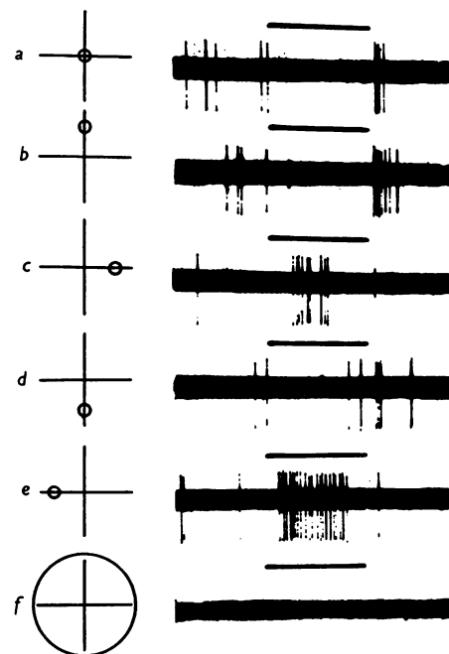
- ဒီပြဿနာအတွက်က multiple classifier ကိုသုံးပြုးရင်းလိုရတယ်။ ဒီနေရာမှာတော့ classifier နှစ်ခုသုံးရင် ရပါ။



# Classification

- အခုအချင်အထိ ပြောခဲ့တေတွက Error Function သို့ Loss Function ရဲ့ concept
- Machine learning တွေမှာ သုံးနောက်တဲ့ Classification ဆိုတဲ့ concept
- နည်းနည်း detail ပြောမယ်ဆိုရင် သုံးခဲ့တေက linear classifier ( $y=mx+b$ ), ငယ်ငယ်တူနှုန်းက သင်ခဲ့က်တဲ့ equation of line ပါ
- လက်တွေ့ပြသာအများစုအတွက် multiple classification လုပ်ဖို့ လိုအပ်တယ်
- နောက်တမျိုးပြောရရင် non-linear classification နဲ့မှ ရတယ်

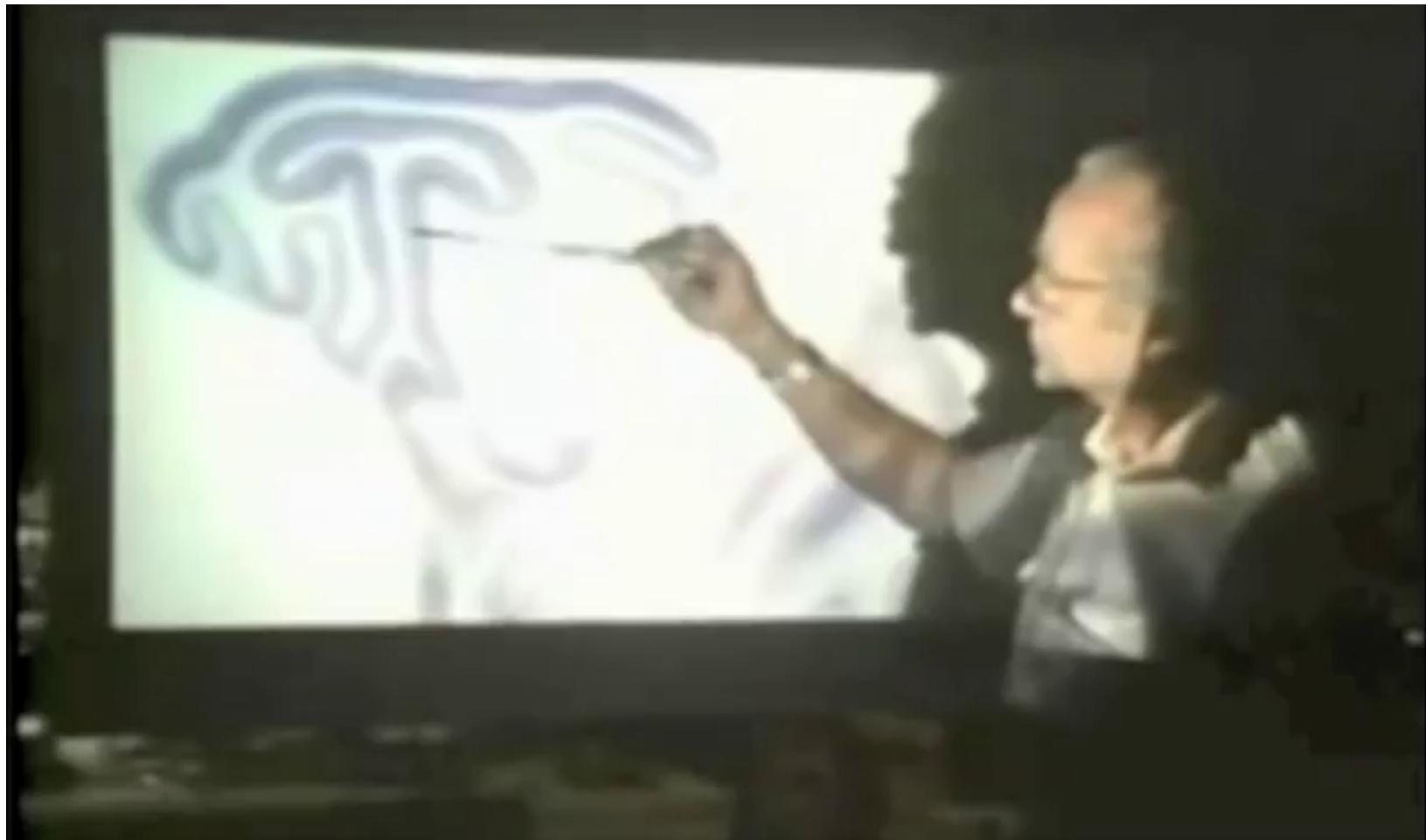
# Human Brain (Cat Experiment)



Hubel and Wiesel ရဲ့ ကြောင် စမ်းသပ်မှု

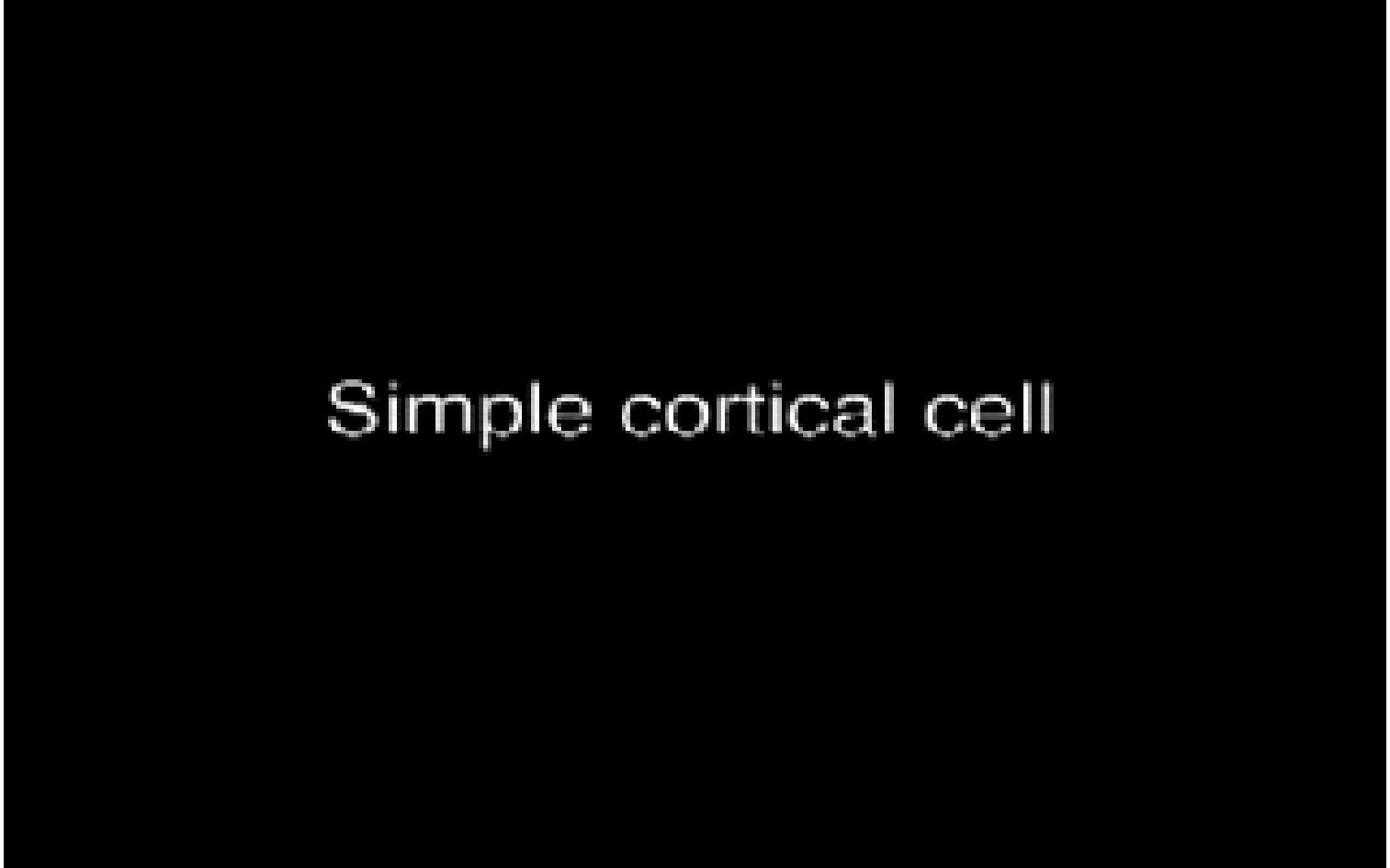
\* Responses of a cell in the cat's striate cortex

# Human Brain (Cat Experiment)



Hubel and Wiesel Cat Experiment:  
<https://www.youtube.com/watch?v=IOHayh06LJ4>

# Human Brain (Cat Experiment)



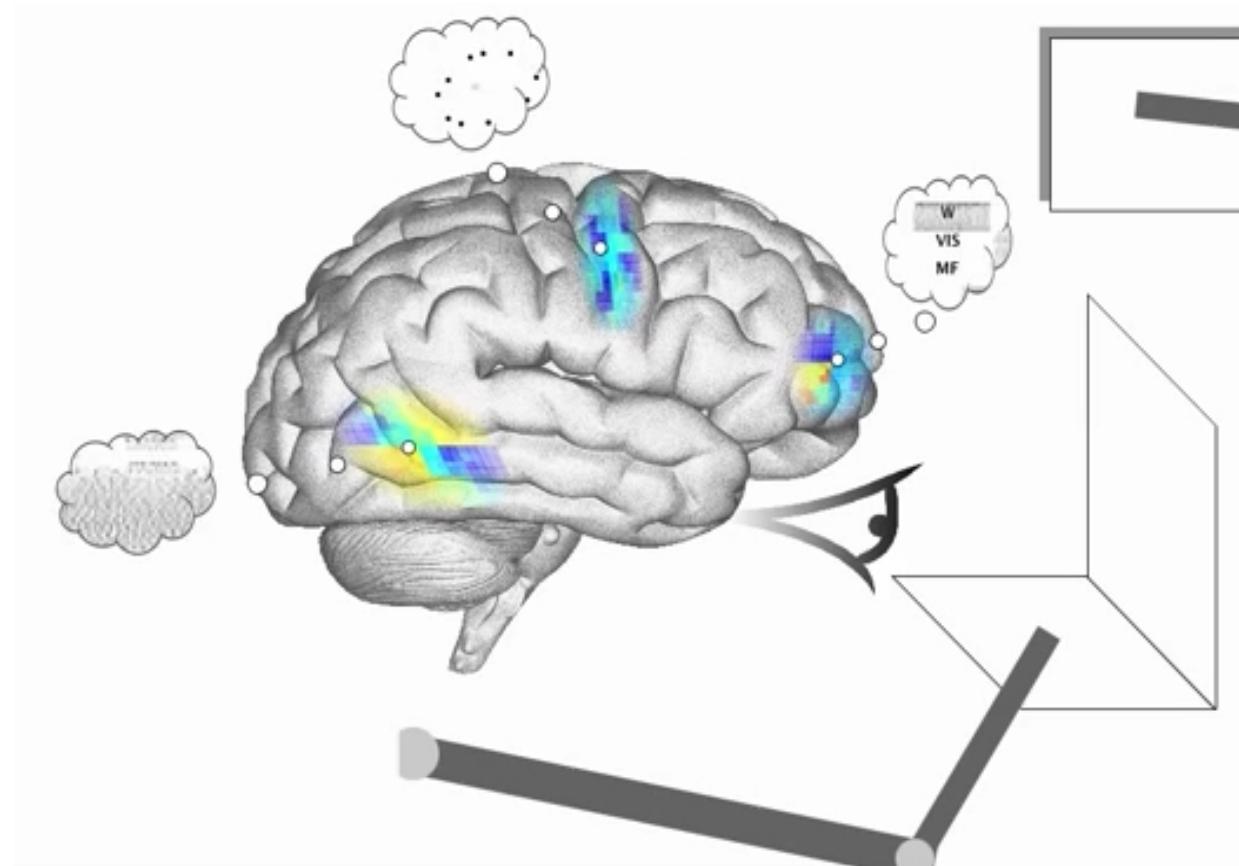
Simple cortical cell

Hubel & Wiesel's demonstration of simple, complex and hypercomplex cells in the cat's visual cortex:

<https://www.youtube.com/watch?v=jw6nBWo21Zk>

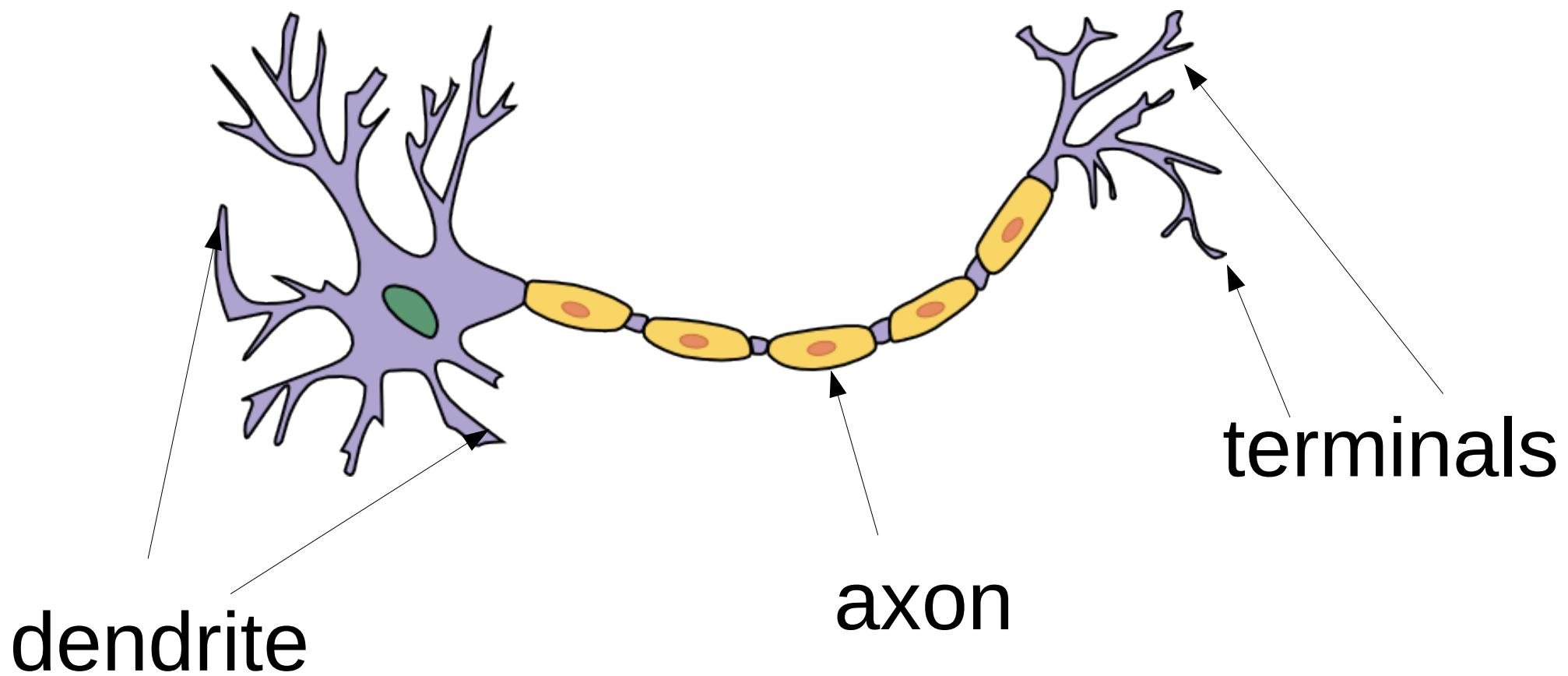
# Human Brain (Spaun)

- Will we ever simulate the brain?
- ဦးနှောက် နဲ့ ပတ်သက်ပြီး အမျိုးမျိုးသော လွှဲလာမှုတွေကို လုပ်နေကြတဲ့ အထူက Spaun Brain Simulation ကို ကြည့်ရအောင် (<https://xchoo.github.io/spaun2.0/videos.html>)



# Human Brain

- လူ့နွောက်ရဲ အခြေခံအကျဆုံးဖြစ်တဲ့ neuron ဆိတာက

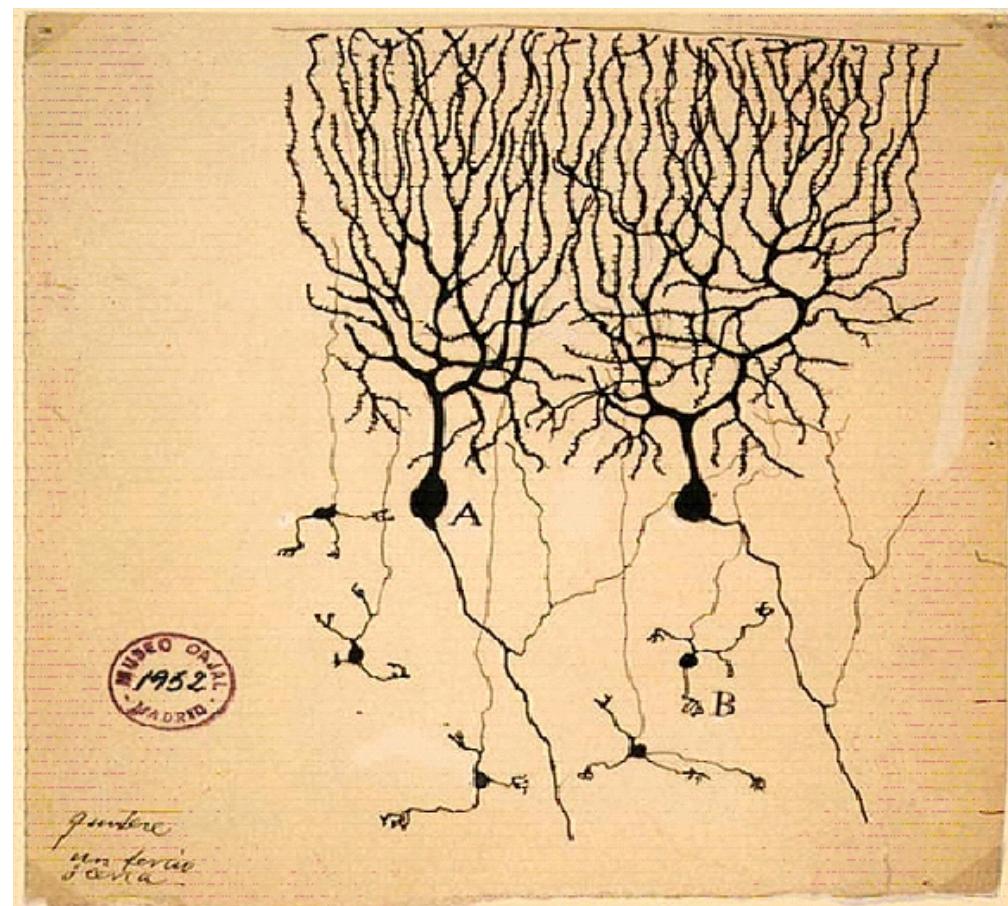


# Human Brain

- Neuron က ပုံစံမျိုးစုံ အနေနဲ့ ရှိတယ်
- Electrical signal ကို စီးဆင်းစေတယ် (one end to the other), dendrites ကနေ axon တွေကို ဖြတ်ပြီးတော့ terminal ကို ရောက်တဲ့ အထိ
- Terminal တွေကနေမှ နောက်ထပ် တဲ့ အခြား neuron တစ်ခုစွဲ ကို လက်ဆင့်ကမ်းတယ်
- ဒီလိုပုံစံနဲ့ ပါတို့ရဲ့ ဦးနောက်တွေက အလင်း၊ အသံ၊ ထိတွေမှု၊ အပူချိန် စောတွေကို ခံစားသိရှိနိုင်တယ်

# Human Brain

- စပိန်လူမျိုး neuroscientist ဖြစ်တဲ့ Santiago Ramón y Cajal
- The Father of modern Neuroscience (1906 မှာ နိုဗယ်ဆုရရှိ)
- ညာဘက်က ပုံက သူက သင္တေဇ် မှာ ဆွဲခဲ့တဲ့ ပင်ဂွင်း ဦးနောက်ရဲ့ neuron ပုံ

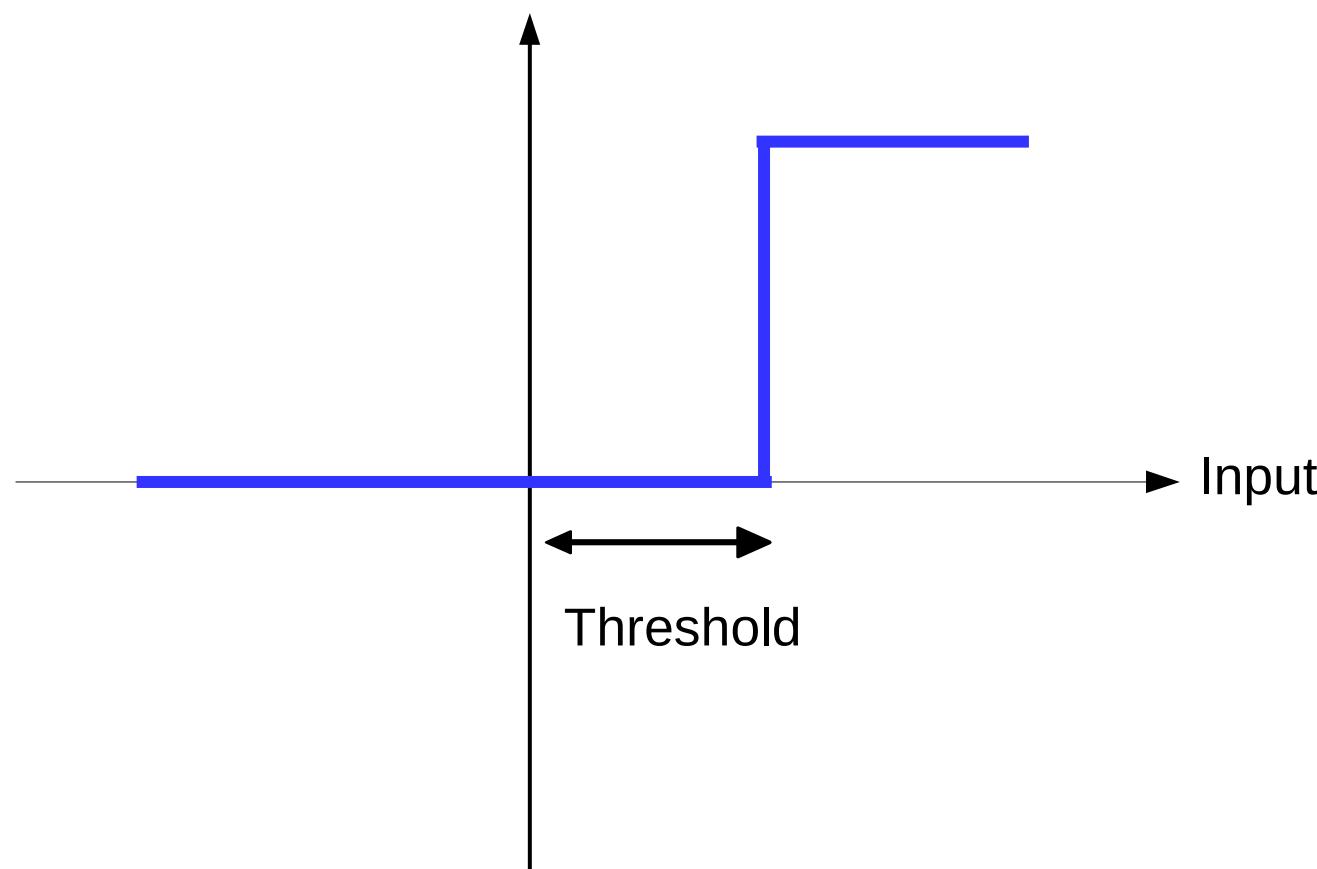


# Human Brain

- လုပ်းနွောက်က တအားစိတ်ဝင်စားပိုကောင်းတယ်
- ကျွန်ုတ်တို့ မသိသေးတာတွေအများကြီး
- အခု ပြန်ပြီးတော့ ခေတ်ထလာတဲ့ Neural Network (Deeplearning) က ဦးနွောက်ရဲ့ အလုပ်လုပ်တဲ့ ပုံစံကို အခြေခံထား၊ အတူခိုးထားတာ
- ဥပမာ electrical signal က input ဝင်လာတိုင်း တောက်လျှောက်စီး ပေးနေတာမဟုတ်ဘူး (threshold နဲ့ ထိန်းထားတယ်)၊ အဲဒါက Neural Network မှာ ပြောနေကြတဲ့ Activation function ပါပဲ

# Activation Function

- Step Function ကလည်း neuron ရဲ့ အလုပ်လုပ်ပုံလိုပါပဲ
- Threshold input အထိရောက်ပြီဆိုတာနဲ့ output စလုပ်တယ်



# Activation Function

- Step Function ၊ Python ၊ coding လုပ်ကည့်ရအောင်

```
import numpy as np
import matplotlib.pyplot as plt

def step_function(input_value):
    return np.array(input_value > 2, dtype=np.int)

input_value = np.arange(-5.0, 5.0, 0.1)
output_value = step_function(input_value)

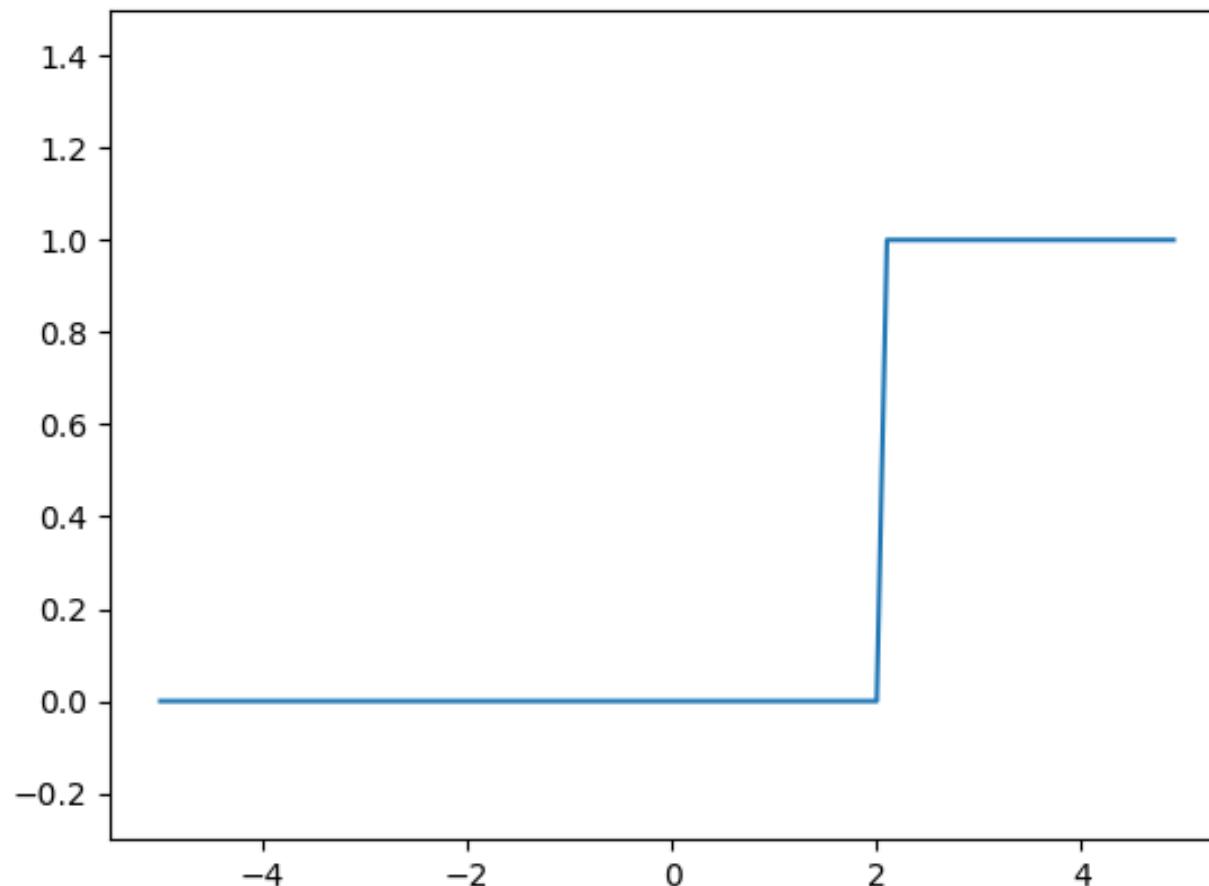
plt.plot(input_value, output_value)
plt.ylim(-0.3, 1.5)
plt.savefig('step-function.png')

plt.show()
```

# Activation Function

>python ./step-function.py

- ဒေသကိပ် step-function.png ပုံကိုရမယ်



# Activation Function

- Sigmoid Function က လွယ်လွယ်ရှင်းပြရရင် step function ကို ပိုပီးတော့ သဘာဝကြအောင် ပြောင်းထားတဲ့ (smooth curve) function ပါပဲ
- Neural Network တွေမှာ sigmoid function ကို တော်တော်လေးကို အသုံးပြုခဲ့ကြပါတယ်
- Sigmoid function ကို logistic function လိုလဲခေါပါတယ်

$$y = \frac{1}{1 + e^{-x}}$$

# Activation Function

- Sigmoid Function ၊ Python ၊ coding လုပ်ကြည့်ရအောင်

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid_function(x):
    return 1 / (1 + np.exp(-x))

x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid_function(x)

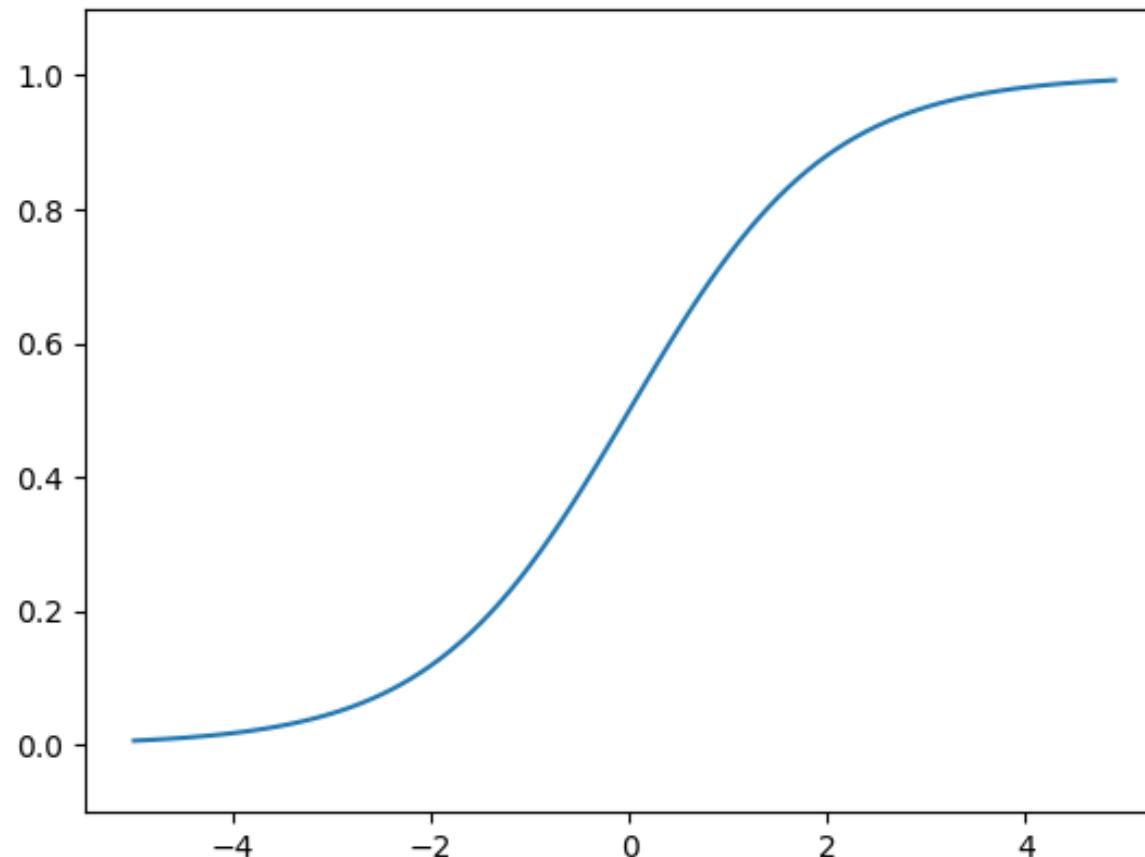
plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.savefig('sigmoid-function.png')

plt.show()
```

# Activation Function

>python ./sigmoid-function.py

- အောက်ပါ sigmoid-function.png ပုံကိုရမယ်



# Activation Function

- sigmoid\_function ဆိတိကို numpy array ကို input လုပ်တဲ့ အခါမှာလည်း မှန်မှန်ကန်ကန် အလုပ်လုပ်ပေးပါတယ်

```
x = np.array([-1.0, 2.0, 4.0])
sigmoid_function(x)
```

```
array([ 0.26894142,  0.88079708,  0.98201379])
```

# Activation Function

- Numpy ရဲ့ broadcasting ကို သိတော်ယူ

```
x2 = np.array([1.0, 2.0, 3.0])  
1.0 + x2
```

```
array([ 2.,  3.,  4.])
```

```
2.0 * x2
```

```
array([ 2.,  4.,  6.])
```

```
2.0 / x2
```

```
array([ 2.          ,  1.          ,  0.66666667])
```

# Activation Function

- လက်ရှိမှာ Neural Network အတွက် အသုံးအများဆုံး activation function ကိုပြောပါဆိုရင် ReLU
- တကယ်တမ်းက သူရဲ့ အလုပ်လုပ်ပုံက ရှင်းပါတယ်

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

# Activation Function

- ReLu (Rectified Linear Unit) Function ကို Python ပုံ  
coding လုပ်ကည့်ရအောင်

```
import numpy as np
import matplotlib.pyplot as plt

def ReLU_function(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = ReLU_function(x)

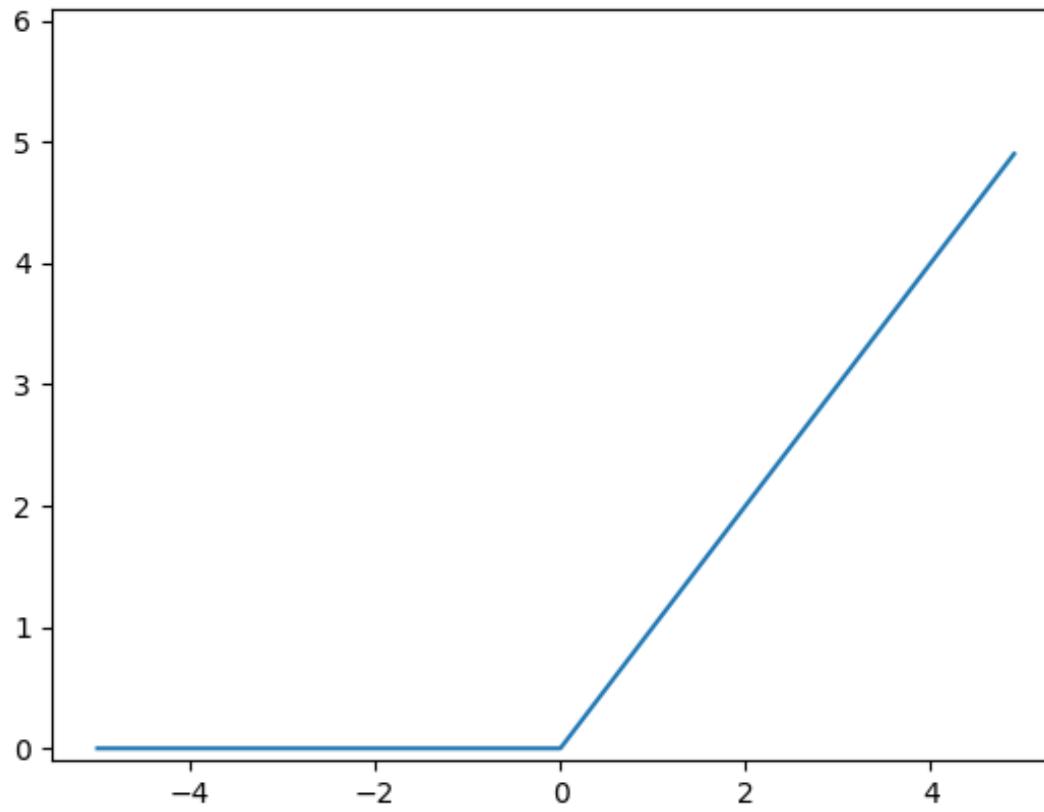
plt.plot(x, y)
plt.ylim(-0.1, 6.1)
plt.savefig('ReLU-function.png')

plt.show()
```

# Activation Function

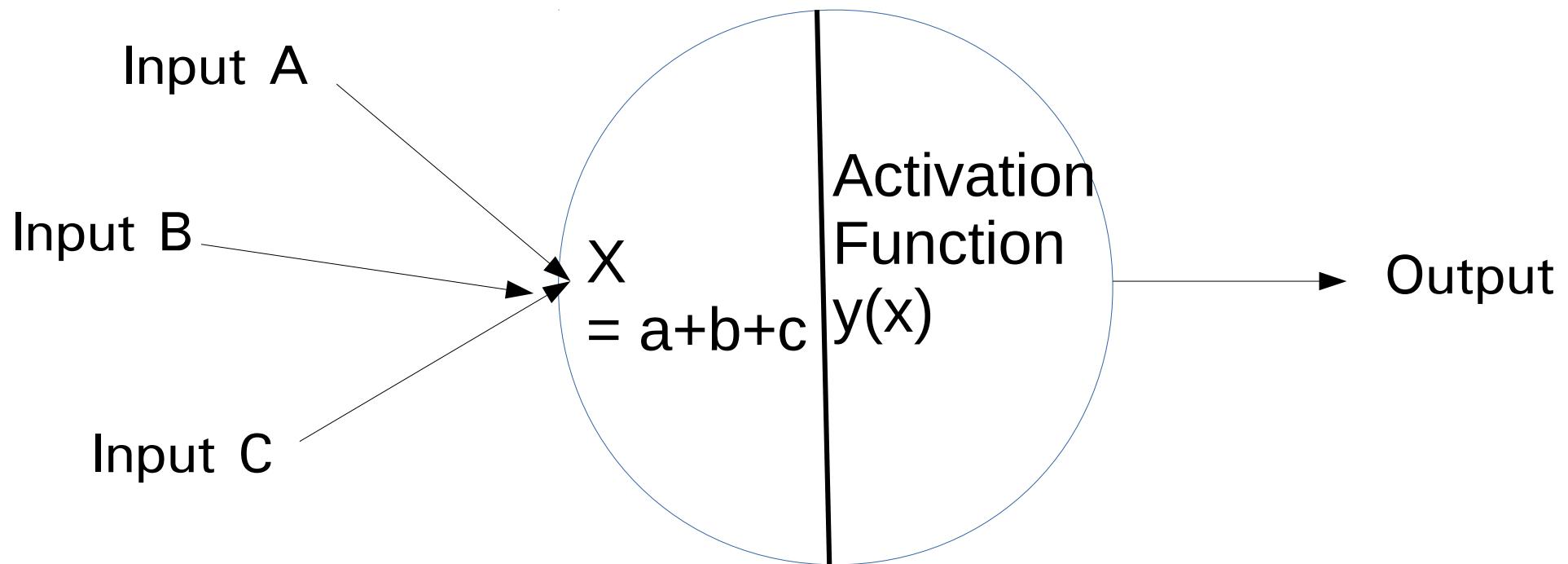
>python ./relu-function.py

- အောက်ပါ ReLU-function.png ပုံကိုရမယ်



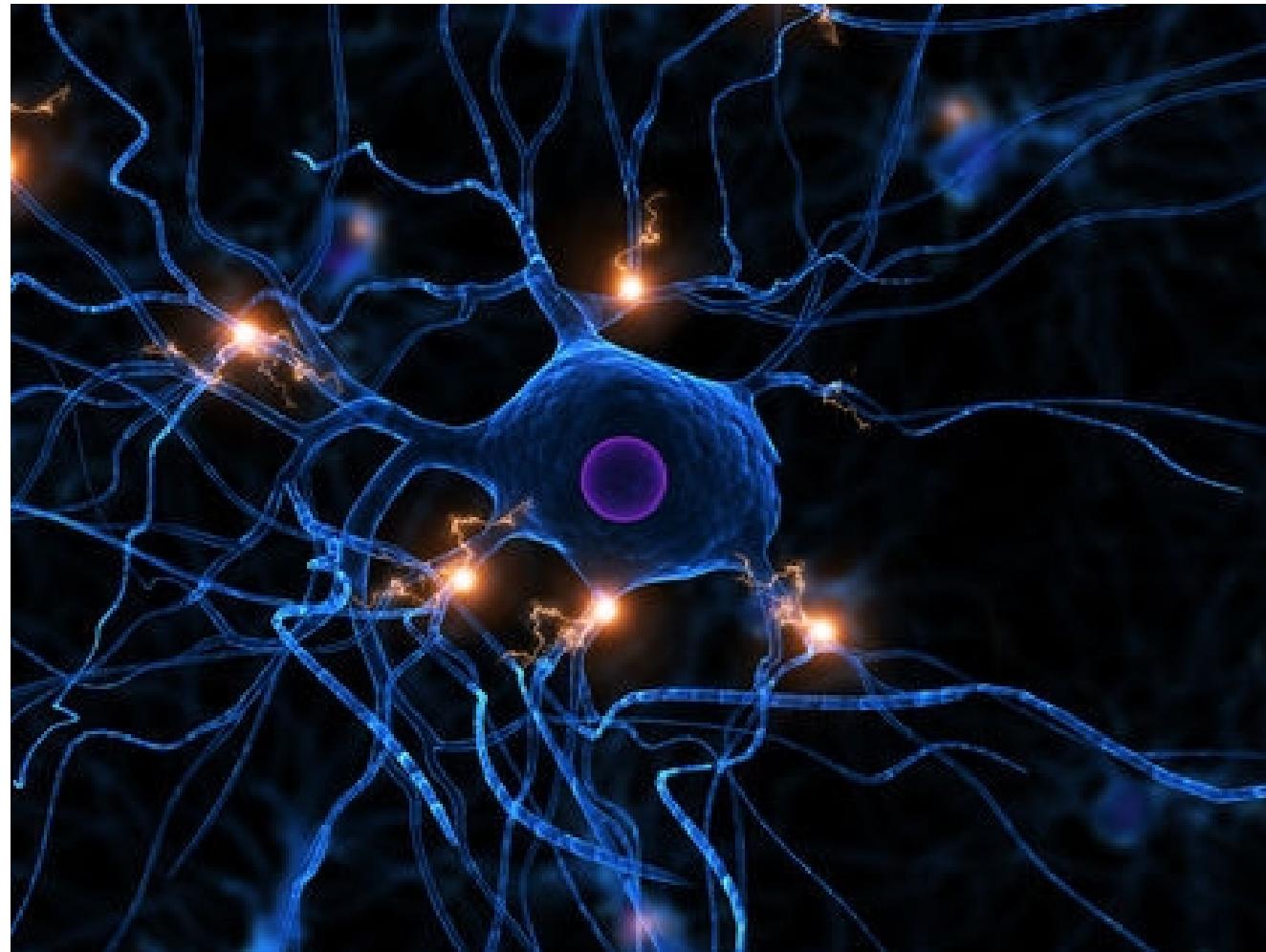
# Neural Network

- တကယ်က neural network ရဲ့ concept က အခမ်းကိုရှင်းထယ်



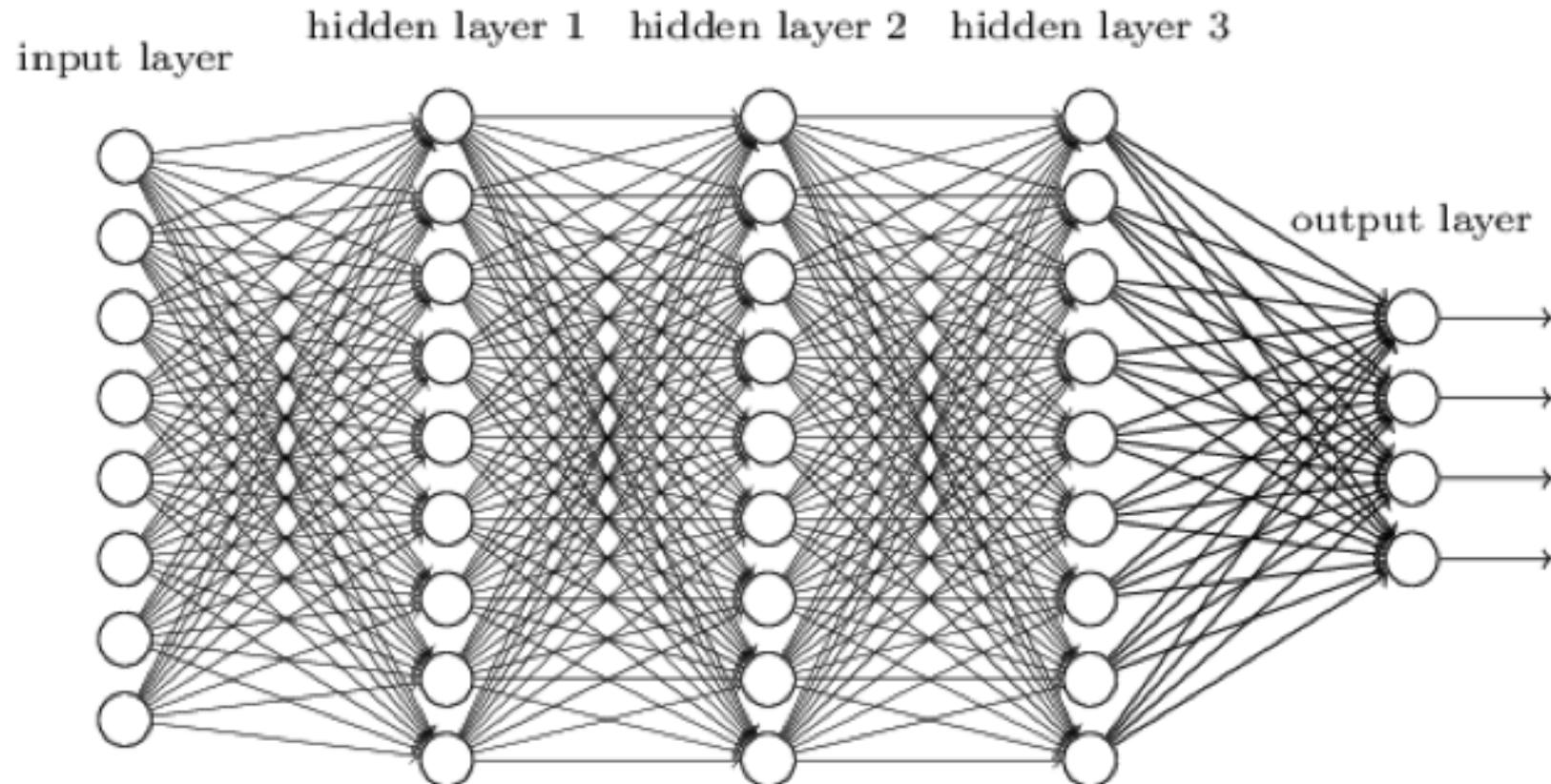
# Neural Network

- တကယ့် လျှိုးနွာက်မှုလည်း neuron တွေက အများကြီး ရှိသလိုပဲ

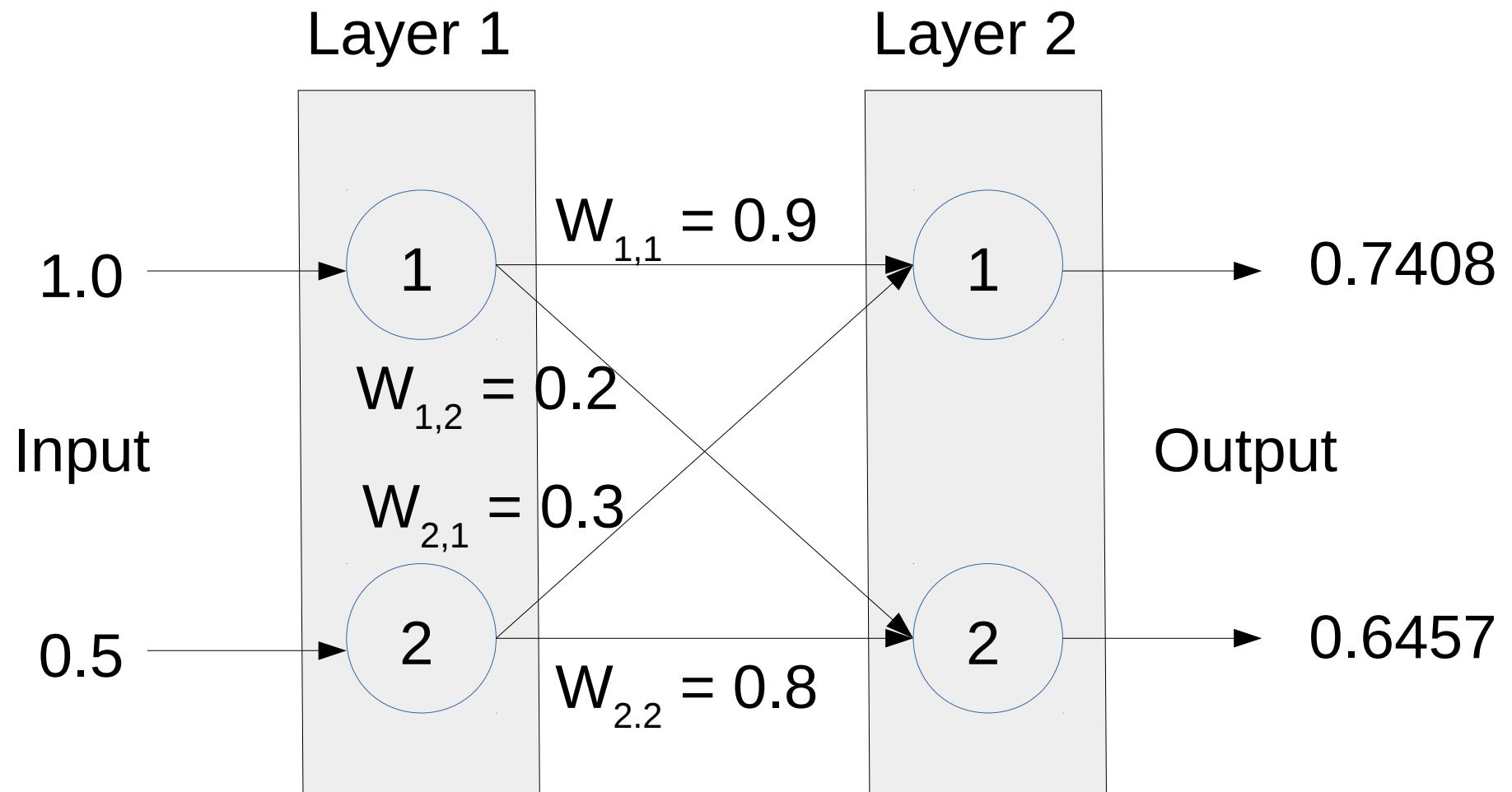


# Neural Network

- Artificial Neural Network မှတည်း neuron တွေ  
layer တွေက အများကြီးပဲ



# Neural Network



# Neural Network

- ခုနက input နဲ့ output ပဲရှိတဲ့ 1 layer neural network  
ကို sigmoid function ကို သုံးပြီး တွေက်ထားတယ်
- ပထမ layer ကိုတွေက် ကြည့်ရင်

$$\begin{aligned}x &= (1^{\text{st}} \text{ neuron output} \times \text{weight}) + (2^{\text{nd}} \text{ neuron output} \times \text{weight}) \\&= (1.0 * 0.9) + (0.5 * 0.3) \\&= 0.9 + 0.15 \\&= 1.05\end{aligned}$$

# Neural Network

- ၃တို့ယ layer အတွက်က

$$\begin{aligned}x &= (1^{\text{st}} \text{ neuron output} \times \text{weight}) + (2^{\text{nd}} \text{ neuron output} \times \text{weight}) \\&= (1.0 * 0.2) + (0.5 * 0.8) \\&= 0.2 + 0.4 \\&= 0.6\end{aligned}$$

# Neural Network

- ပြီးတော့မှ sigmoid function ဆိုကို pass လုပ်တယ်
- output layer ရဲ့ neuron 1 အတွက်က

$$y = \frac{1}{1 + e^{-x}}$$

$$y = 1 / (1 + 0.3499) = 1 / 1.3499 = 0.7408$$

- Output layer ရဲ့ neuron 2 အတွက်က

$$y = 1 / (1 + 0.5488) = 1 / 1.5488 = 0.6457$$

- အခု တွက်ခဲ့တာက feed forward calculation ပါပဲ

# Matrix Multiplication

- Matrix ḡdot multiplication

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$(1 \times 5) + (2 \times 7) = 19$$

$$(3 \times 5) + (4 \times 7) = 43$$

# Matrix Multiplication

- Matrix က တကယ်ကို အသုံးဝင်ပါတယ်
- Numpy library နဲ့ Python coding လုပ်ကြည့်ရအောင်

```
In [3]: matrix_A = np.array([[1, 2], [3, 4]])
matrix_A.shape
```

Out[3]: (2, 2)

```
In [5]: matrix_B = np.array([[5, 6], [7, 8]])
matrix_B.shape
```

Out[5]: (2, 2)

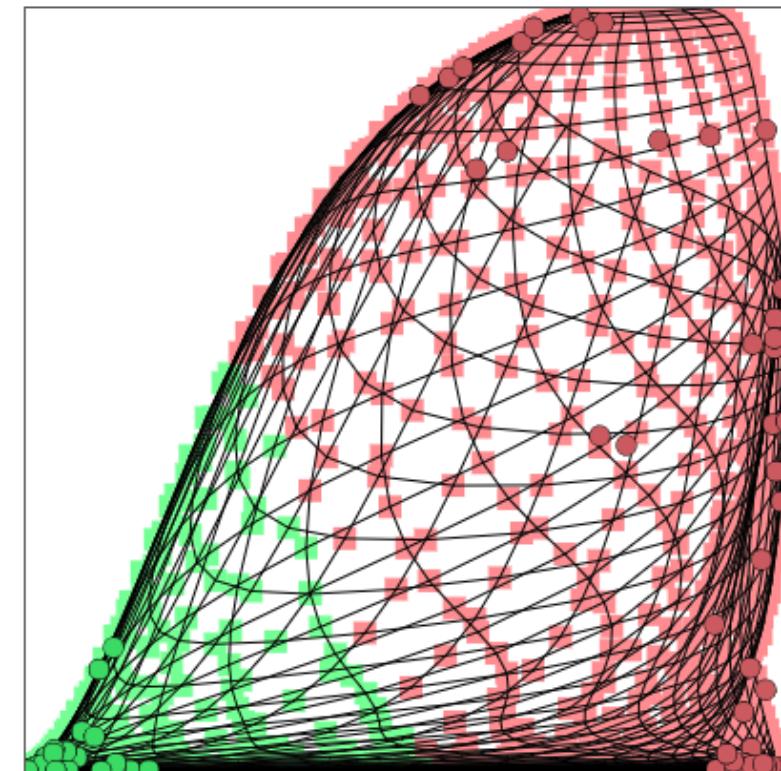
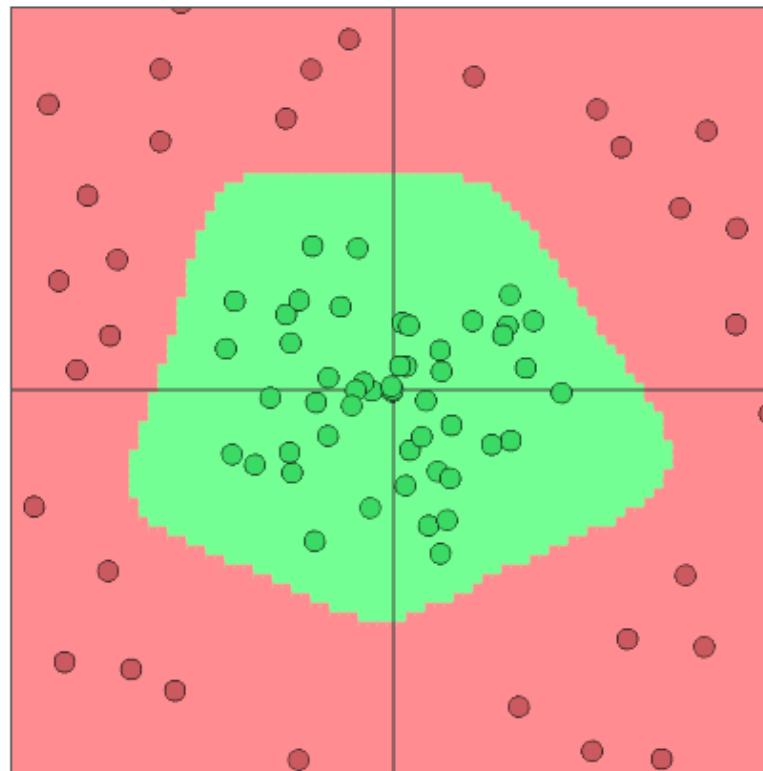
---

```
In [6]: np.dot(matrix_A, matrix_B)
```

Out[6]: array([[19, 22],
[43, 50]])

# Useful Links

- ConvnetJS က 63တာ အမျိုးအစားတွေ (circle, spiral, random, ...) မတူတဲ့ အပေါ်မှာ activation function တွေ ဖြစ်တဲ့ ဥပမာ tanh, ReLu တို့က classification လုပ်တဲ့ ပုံကို 2D space မှာ visualization လုပ်ပြတဲ့ website ပါ။
- coding ကိုပါ ဝင်ပြောင်းကြည့်ပါ။ နားလည်သွားပါလိမ့်မယ်။



# Make Your Own NN

- Multi-layer Perceptron (MLP) က simple ဖြစ်တဲ့ Neural Network ပါပဲ
- Python programming library တစ်ခုဖြစ်တဲ့ `sklearn.neural_network` ကိုသုံးပြီး `Image Classification` ကို ဒေတာပြင်တာက အစာ `feature` ထုတ်ပြုး `classification` လုပ်တဲ့ အဆင့်အားလုံးကို လက်တွေ့လုပ်ကြည့်ကြရအောင်။

# Make Your Own NN

- မန်မာ fingerspelling ထလုံးတွေထဲက "ကၣ္း" ကနေ "c" အထိ ထလုံး ၅လုံးကို ပိုဘိုင်းဖုန်းနဲ့ စတ်ပုံရှိက်ယူတယ်



က

ခ

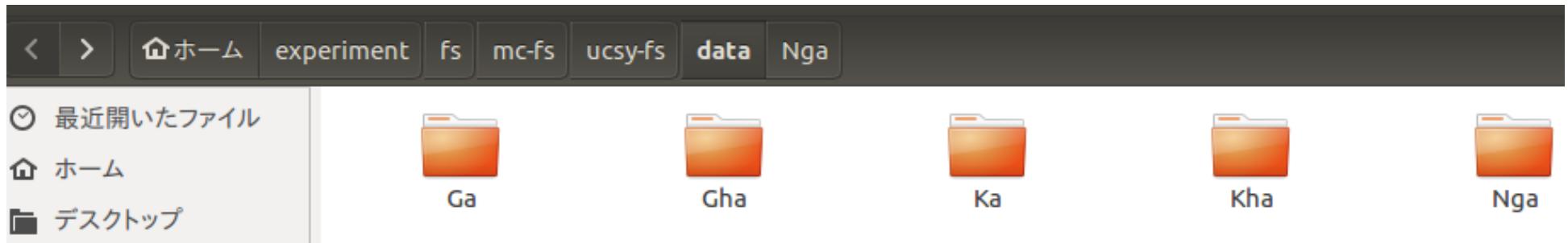
ဂ

ဃ

ၣ္း

# Make Your Own NN

- ပြီးတော့ ဖိုလ်ဒါ ၅၁ အောက်မှာ ပုံတွေကို ခွဲသိမ်းပါတယ်။



- bvlc\_reference\_caffenet ရဲ့ မော်ဒယ်ကိုသုံးပြီး Feature extraction လုပ်မှာ မိမိ အောက်ပါ ဖိုင်သုံးဖိုင် လိုအပ်ပါတယ်

bvlc\_reference\_caffenet.caffemodel (weight တွေကိုသိမ်းထားတဲ့ဖိုင်)

deploy.prototxt (Neural Network ရဲ့ ဒီဇိုင်း (သို့) architecture ဖိုင်)

ilsvrc\_2012\_mean.npy (numpy ဖိုင် format)

# Make Your Own NN

- အောက်ပါ ပရိုဂရမ်တွေကို သုံးပြီး image classification ကို လက်တွေ့လုပ်ကြည့်ကြရအောင်
- extract-feature.sh (for feature extraction)
- add-label.sh (for adding y labels, combining all features as one file for training)
- train-test-MLP-clf.py (for training image classifier and testing)

# Make Your Own NN

- How to run:

```
>./extract-feature.sh ./data/
```

```
>./add-label.sh ./data/
```

- အထက်ပါ ပရိုဂရမ် ပုံချက အဆင်ပြည့် run ပြီးသွားရင် ./data ပိုလ်ဒါအောက်မှာ

data.feature (feature ပိုင်) နဲ့

id-name.txt (id နဲ့ y label အတိအကျင့်ပိုင်) ကို ရမယ်။

- Training, testing က အောက်ပါ command နဲ့ run လို့ ရတယ်

```
>python ./train-test-MLP-clf.py ./data/data.feature
```

# Make Your Own NN

- `MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(20,20,20), random_state=1, max_iter=300, verbose=300)` နဲ့ run ကည့်တဲ့အခါ ရတဲ့ ရလဒ် :

```
Tit = total number of iterations
Tnf = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip = number of BFGS updates skipped
Nact = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F = final function value

* * *

      N      Tit      Tnf      Tnint      Skip      Nact      Projg          F
82885      32      34          1          0          0   5.319D-05   1.352D-05
      F =  1.35169632681228974E-005

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL

Cauchy              time 0.000E+00 seconds.
Subspace minimization time 0.000E+00 seconds.
Line search          time 0.000E+00 seconds.

Total User time 0.000E+00 seconds.

MLP Cross Validation Score: [0.9          0.75          0.65          0.9
  0.875      0.75      0.8125      0.75      ]
                           precision    recall    f1-score    support

      1          0.70          0.64          0.67          11
      2          0.89          0.89          0.89           9
      3          1.00          0.75          0.86          16
      4          0.80          0.73          0.76          11
      5          0.58          0.85          0.69          13

avg / total       0.80          0.77          0.77          60
```

# Useful Links

- Visualization of deploy.prototext :

<http://ethereon.github.io/netscope/#/editor>

```
1 name: "CaffeNet"
2 layer {
3   name: "data"
4   type: "Input"
5   top: "data"
6   input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227 } }
7 }
8 layer {
9   name: "conv1"
10  type: "Convolution"
11  bottom: "data"
12  top: "conv1"
13  convolution_param {
14    num_output: 96
15    kernel_size: 11
16    stride: 4
17  }
18 }
19 layer {
20   name: "relu1"
21   type: "ReLU"
22   bottom: "conv1"
23   top: "conv1"
24 }
25 layer {
26   name: "pool1"
27   type: "Pooling"
28   bottom: "conv1"
29   top: "pool1"
30   pooling_param {
31     pool: MAX
32     kernel_size: 3
--}
```

Json format ဖြစ်တဲ့  
deploy.prototxt ပိုင်ကို  
Netscope ဆိုတဲ့ online editor  
မှာ paste လုပ်ပြီး ကြည့်ရင်  
neural network ခဲ့  
architecture သိမဟုတ် ဒေဝါယံ  
ကို visualization လုပ်ပေးလိုမ့်  
မယ်

# Useful Links

- Visualization of deploy.prototext :

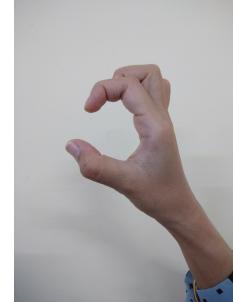
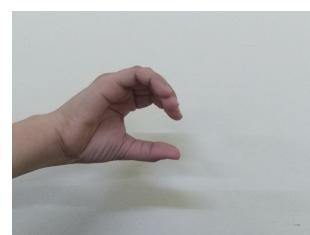
The screenshot shows the netscope editor interface. On the left, a code editor displays the `deploy.prototxt` file for the CaffeNet architecture. On the right, a diagram titled "CaffeNet" illustrates the network flow. The diagram consists of a vertical sequence of nodes: `data` (green), `conv1`/`relu1` (red), `pool1` (yellow), `norm1` (yellow), `conv2`/`relu2` (red), `pool2` (yellow), `norm2` (yellow), and `conv3`/`relu3` (red). Arrows indicate the flow from `data` through `conv1`, then to `pool1`, then to `norm1`, then to `conv2`, then to `pool2`, then to `norm2`, and finally to `conv3`. The `deploy.prototxt` code in the editor is as follows:

```
164 top: "fc6"
165 }
166 layer {
167   name: "drop6"
168   type: "Dropout"
169   bottom: "fc6"
170   top: "fc6"
171   dropout_param {
172     dropout_ratio: 0.5
173   }
174 }
175 layer {
176   name: "fc7"
177   type: "InnerProduct"
178   bottom: "fc6"
179   top: "fc7"
180   inner_product_param {
181     num_output: 4096
182   }
183 }
184 layer {
185   name: "relu7"
186   type: "ReLU"
187   bottom: "fc7"
188   top: "fc7"
189 }
190 layer {
191   name: "drop7"
192   type: "Dropout"
193   bottom: "fc7"
194   top: "fc7"
195   dropout_param {
196     dropout_ratio: 0.5
197   }
198 }
199 layer {
200   name: "fc8"
201   type: "InnerProduct"
202   bottom: "fc7"
203   top: "fc8"
204   inner_product_param {
205     num_output: 1000
206   }
207 }
208 layer {
```

စောစောက **Myanmar**  
fingerspelling image  
classification မှာ သုံးခဲတဲ့  
deploy.prototxt ဖြင်ကို  
netscope editor မှာ  
copy/paste လုပ်ရင် မြင်ရမဲ့ ပုံ

# Make Your Own NN

- အခု လုပ်ခဲ့ကြတဲ့ မြန်မာ fingerspelling စာလုံးတွေရဲ့ image classification မှာ ရိုက်ခဲ့ကြ၊ သုံးခဲ့ကြတဲ့ ပုံတွေက လက်ပုံတွေချည်းပဲ ဖြစ်ပေမဲ့ လက်တွေမှာက signer ရဲ့ ခန္ဓာကိုယ် ပါတဲ့ ပုံတွေလည်း ဖြစ်ချင်ဖြစ်မယ်။ အဲဒီအခါမှာတော့ လက်အပိုင်းကိုပဲ ဖြတ်ယူရတဲ့ အလုပ် (image segmentation) လိုအပ်တာမျိုးလည်း ရှိနိုင်တယ်။
- နောက်တစ်ခုက စာလုံးအတူတူကိုပဲ ဘယ်သန်၊ ညာသန် ကိစ္စမျိုး၊ ဘယ်လက်နဲ့ပြ၊ ညာလက်နဲ့ ပြရင် ပုံက အောက်ပါအတိုင်း ဆန့်ကျင်ဘက် ဖြစ်တဲ့ ကိစ္စတွေ၊ နောက်ပြီးတော့ ဘယ်ညာ တူရင်တောင်မှ လက်အနေအထား မတူတာတွေလည်း ရှိတယ်။



# References

- Make Your Own Neural Network by Tariq Rashid
- The Human Brain Project, a report to the EU commission:  
[https://ec.europa.eu/research/participants/portal/doc/call/h2020/fe/tflag-1-2014/1595110-6pilots-hbp-publicreport\\_en.pdf](https://ec.europa.eu/research/participants/portal/doc/call/h2020/fe/tflag-1-2014/1595110-6pilots-hbp-publicreport_en.pdf)
- A science of the brain, an introduction for young students, British Neuroscience Association, European Dana Alliance for the Brain:  
<http://brain.mcmaster.ca/BrainBee/Neuroscience.Science.of.the.Brain.pdf>
- Wiki of Santiago Ramón y Cajal:  
[https://en.wikipedia.org/wiki/Santiago\\_Ram%C3%B3n\\_y\\_Cajal](https://en.wikipedia.org/wiki/Santiago_Ram%C3%B3n_y_Cajal)
- How to build a brain with Python:  
<https://www.youtube.com/watch?v=7hvpoLKJHOw>

# References

- Hubel & Wiesel's breakthrough experiment and discoveries on the how the visual cortex works:  
<https://www.nobelprize.org/mediaplayer/index.php?id=1605>
- Papers:  
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/pdf/jphysiol01298-0128.pdf>  
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/pdf/jphysiol01247-0121.pdf>
- Nengo Neural Simulator:  
<https://www.nengo.ai/>
- Artificial Brains  
<http://www.artificialbrains.com/spaun>

# References

- ConvnetJS:  
<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>
- Netscope:  
<http://ethereon.github.io/netscope/quickstart.html>