Self-Pretrainable In-situ Normalizer For Deep Learning Error Function

Speaker: Jyun-Xin Ke

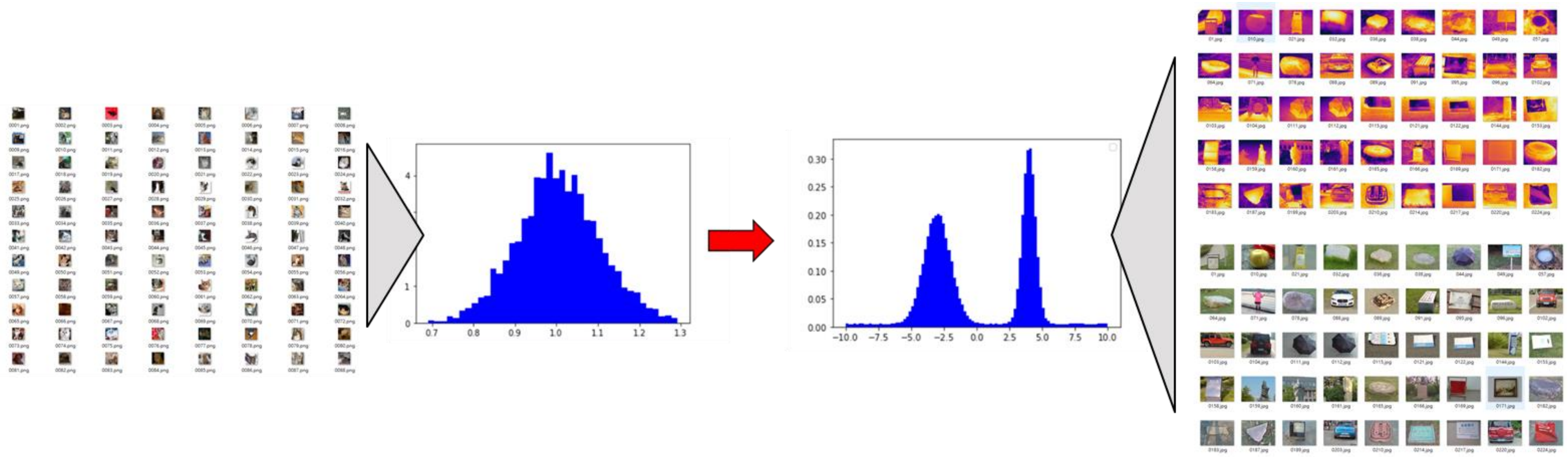
Advisor: Tsung-Chu Huang

Outline

- Introduction
- Previous Work
- Error Function
- Online Fine-Tuning Method
- Offline Sampling Ranking Method
- Experiment for SPINDLE
- Experiment for KDE-SPINDLE
- Conclusions

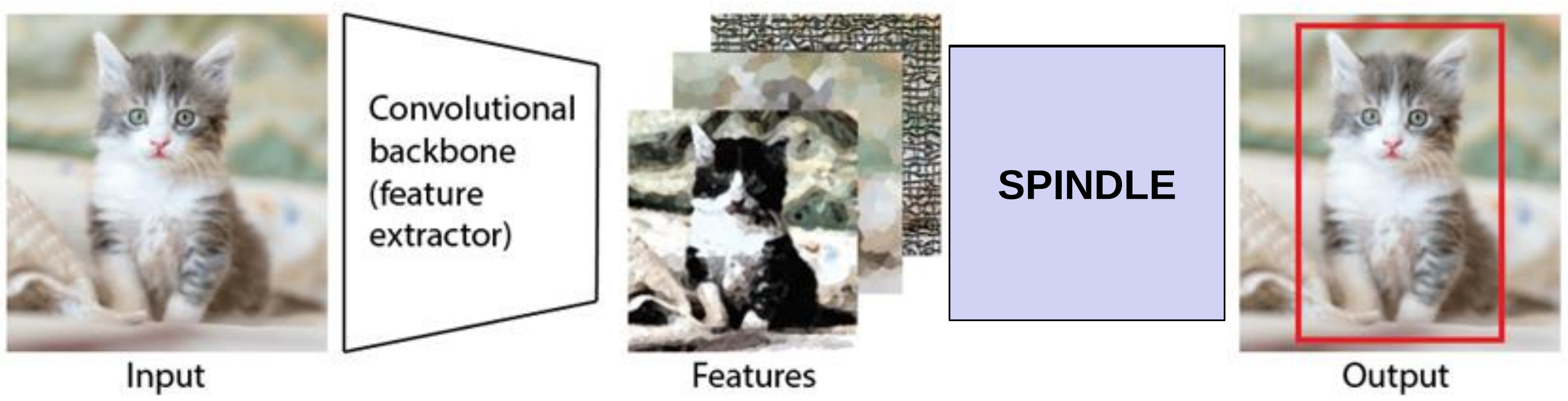
Introduction

- Neural networks have become the mathematical model in the field of AI, and have been applied in the consumer electronics.
- Since the data come from different datasets, the input probability density Function (pdf) of activation function is no longer normal distribution.

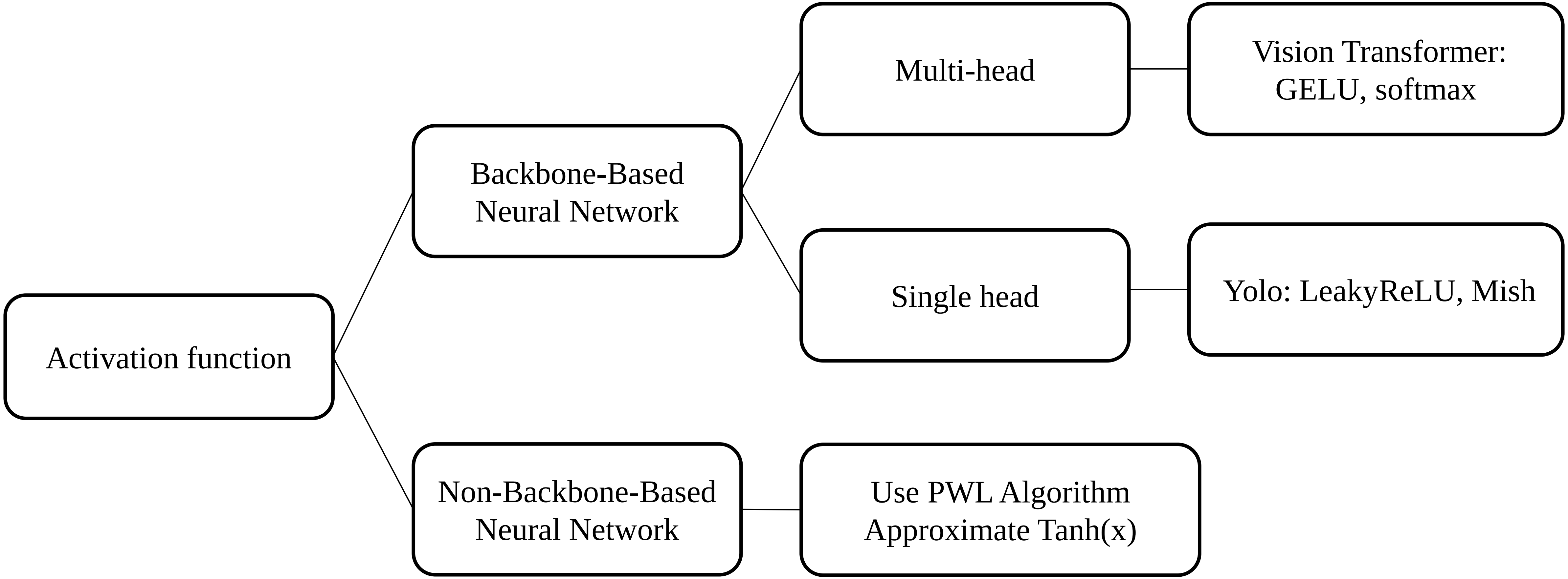


Introduction

- With the expansion of information, neural networks have more layers and higher training costs each year.
- Backbone neural networks can reduce training time by utilizing pre-trained convolutional networks.



Previous Work



Previous Work

- Backbone-Based Neural Network

1. Multi-head Attention

Multi-head Attention is a module for attention mechanisms which runs through an attention mechanism several times in parallel.

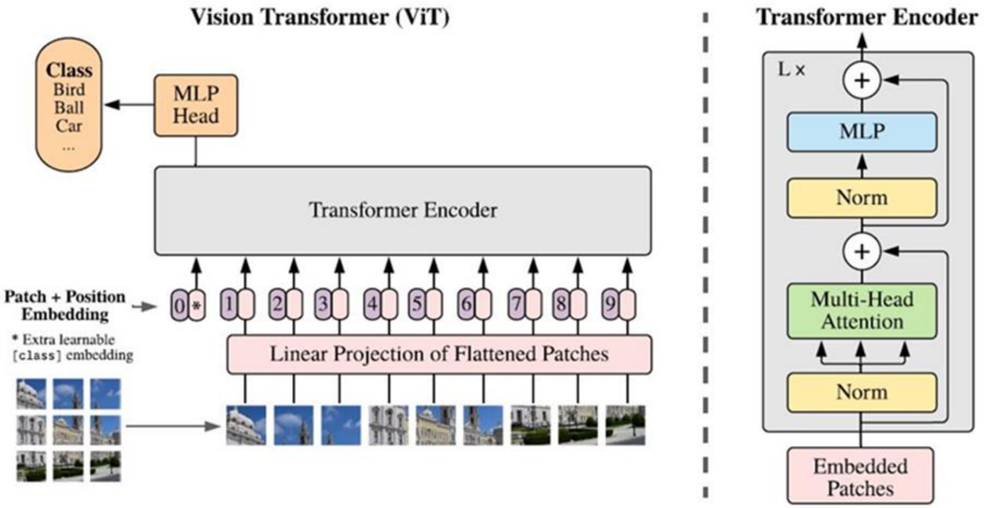


Fig.1 Vision Transformer[17]

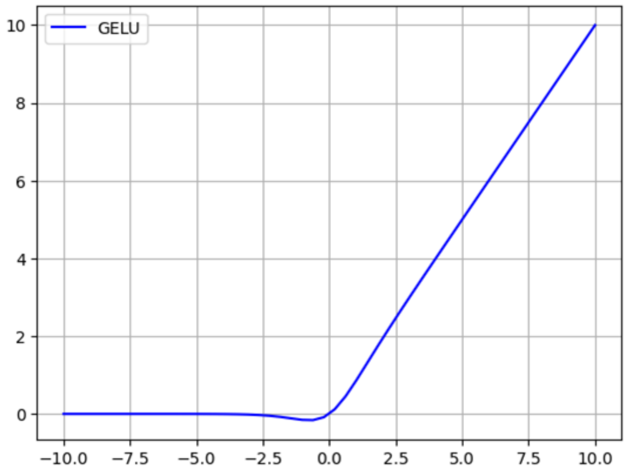


Fig.2 GELU

Previous Work

- Backbone-Based Neural Network

2. Single Head Detector

A single head refers to a <u>single set of output</u> units in the network.

These output units are responsible for <u>generating predictions</u> or outputs for a <u>specific task</u>.

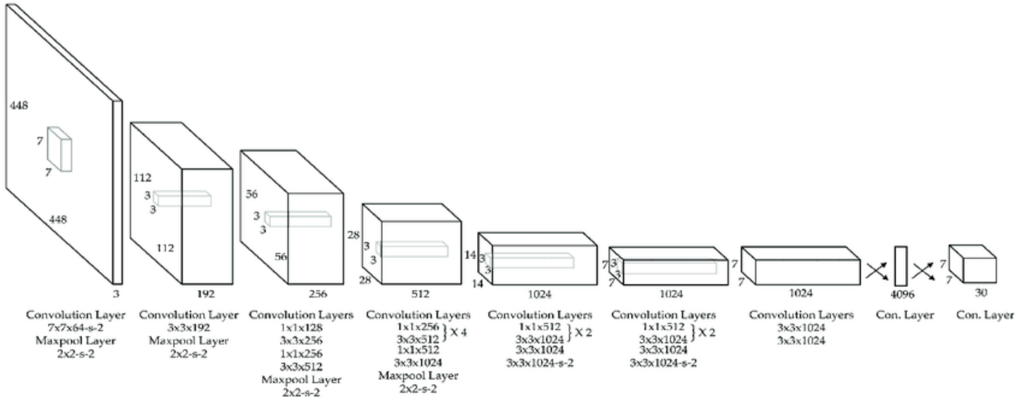


Fig.1 Yolo[20] model

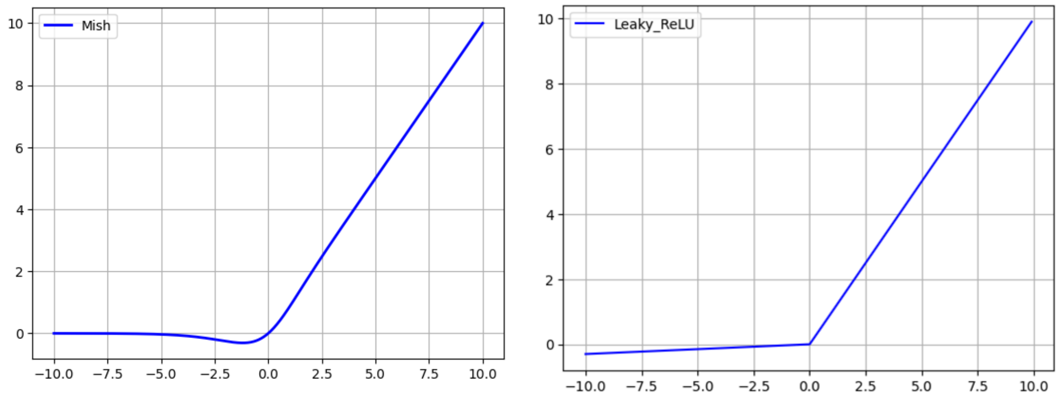


Fig.2 Mish & LeakyReLU

Previous Work

- Backbone Classifier

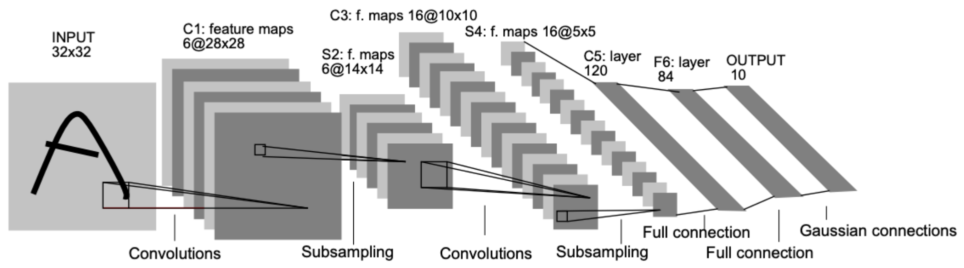


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Fig.1 LeNet-5[9] Model

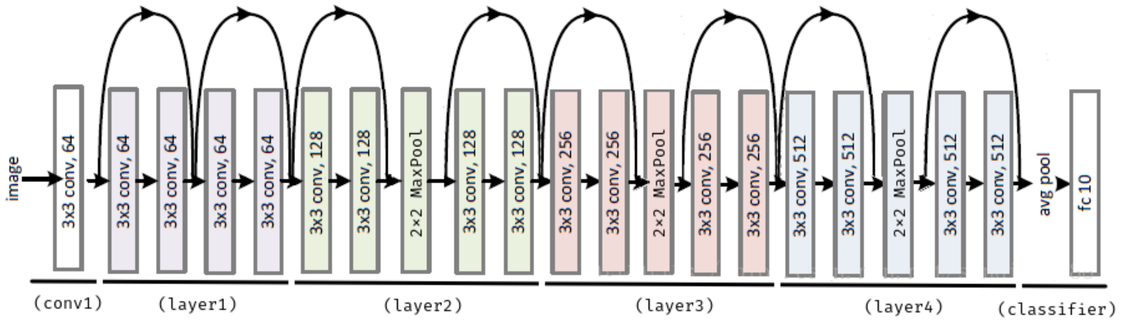


Fig.3 ResNet-18[11] Model

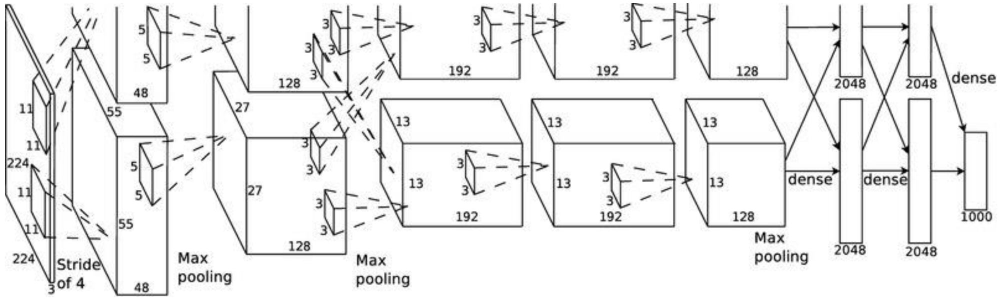


Fig.2 AlexNet[10] Model

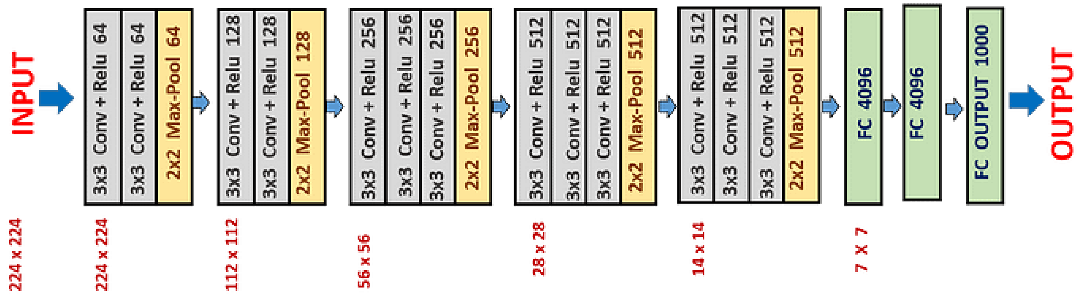


Fig.4 VGG-16[12] Model

Previous Work

- Non-Backbone-Based Neural Network

The piecewise linear function (PWL) is constructed using segments or pieces to calculate different patterns of features.

However, each time a different curve is used, it is necessary to redesign the piecewise function.

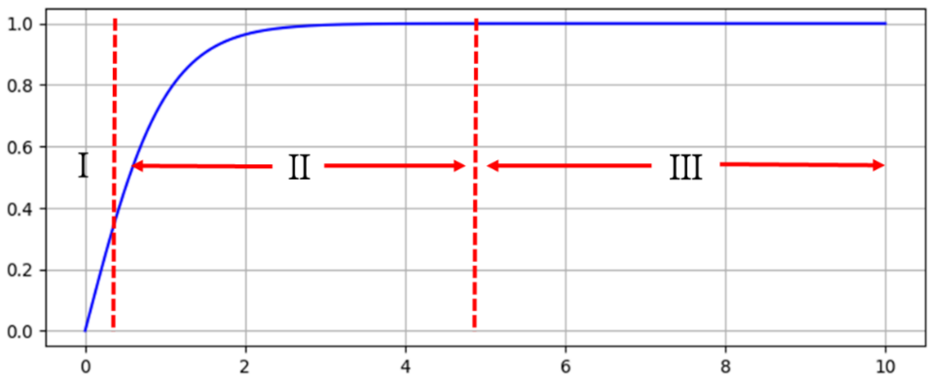


Fig.1 PWL tanh function

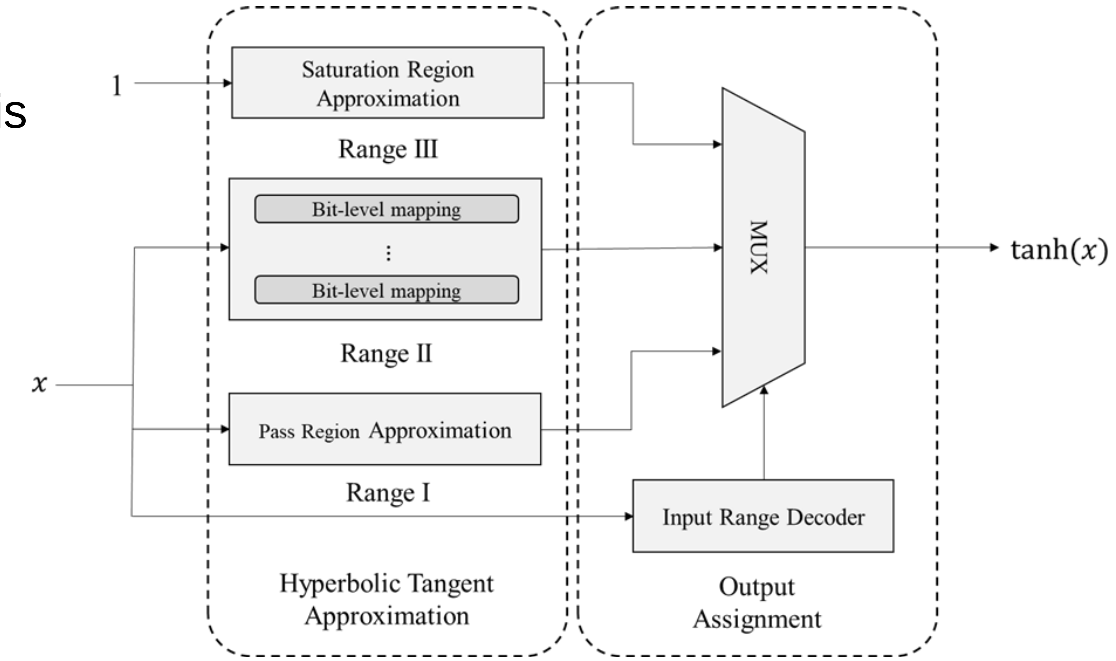


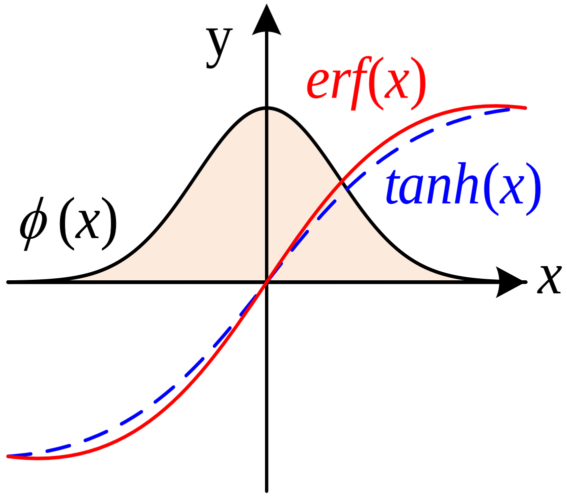
Fig.2 PWL hardware circuit[24]

Error Function

- What is Error Function?

According to the law of large numbers, the distribution of features tends to approach a normal distribution.

Therefore, the cumulative probability distribution function(CDF) of the activation function will be an error function.

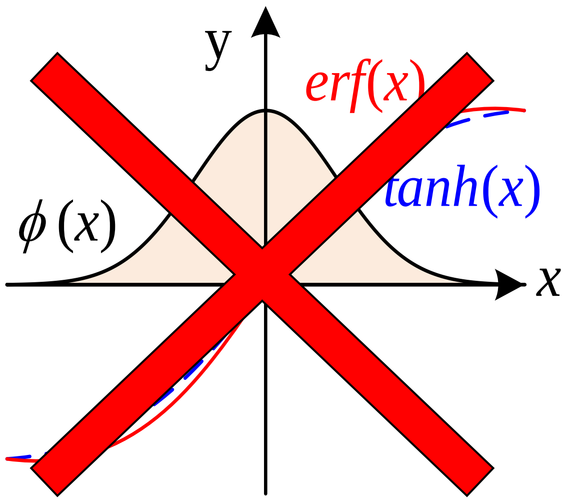


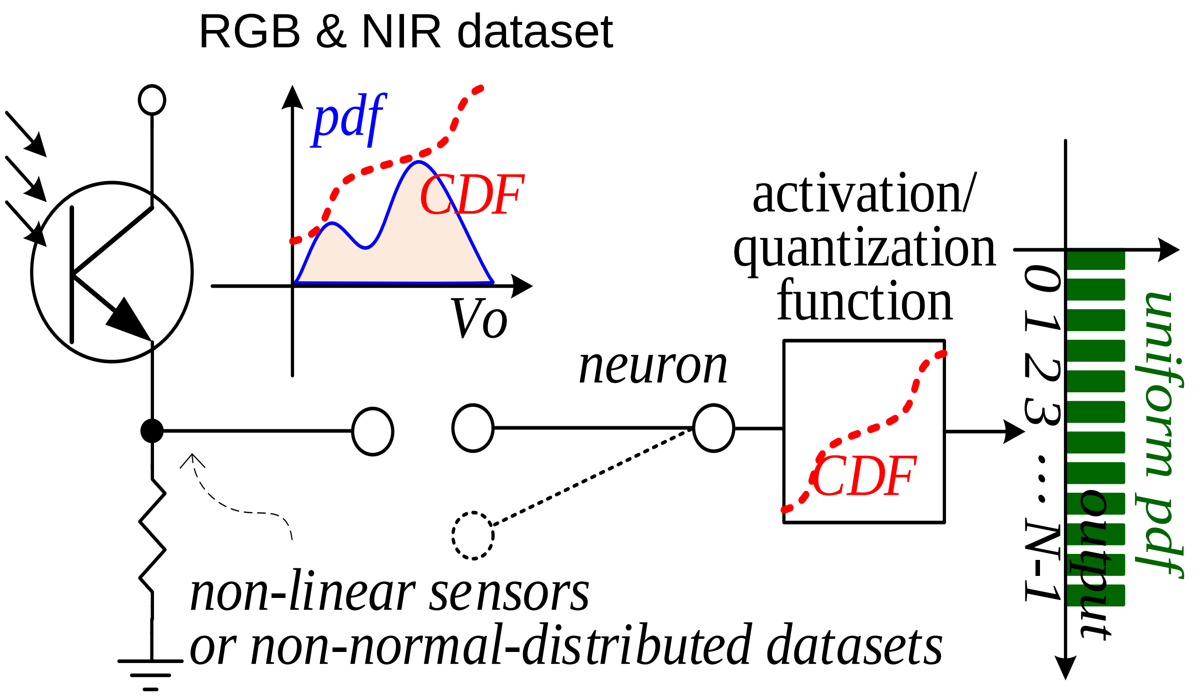
$$\mathrm{erf}(x) = \frac{2}{\sigma\sqrt{2\pi}} \int_0^x e^{-\frac{(t-\mu)^2}{2\sigma}} dt$$

$$CDF(x) = \int_{-\infty}^{x} PDF(t)dt$$

Error Function

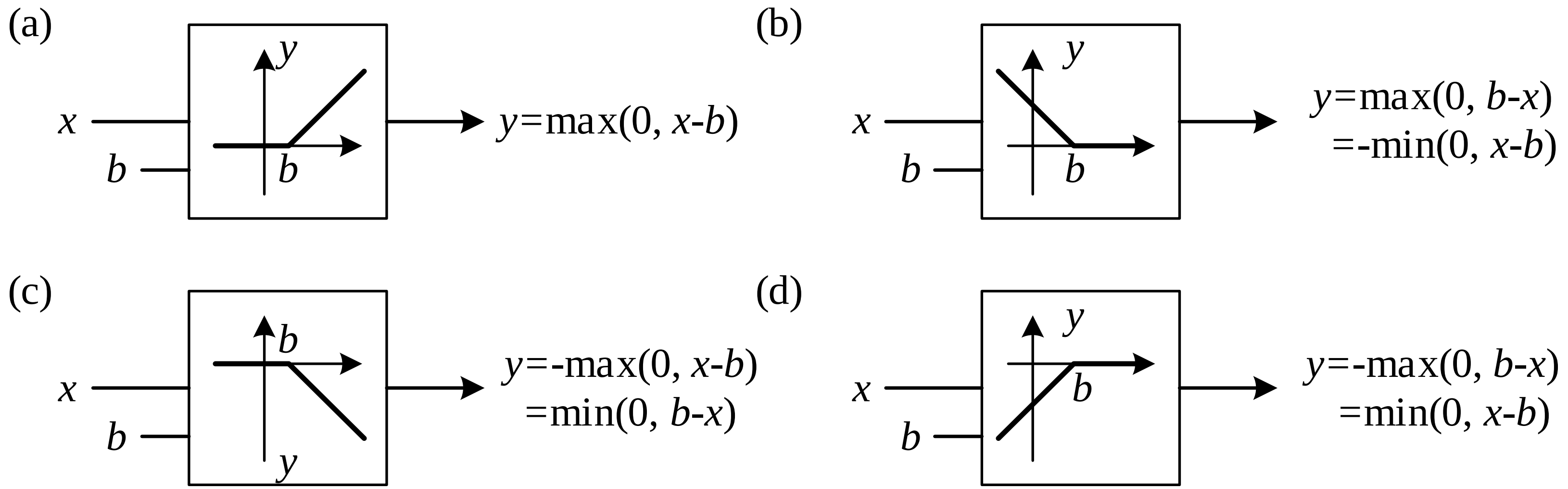
- What is Error Function?





Online Fine-tuning Method

- Four kinds of ReLUs



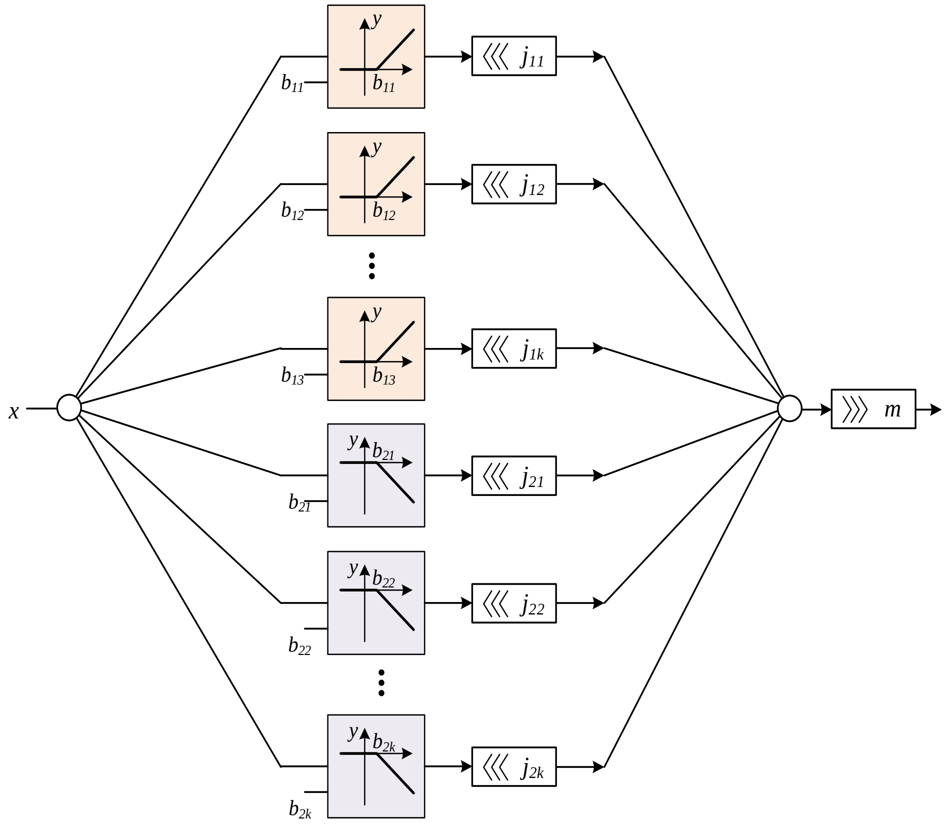
$$ReLU(s_x, s_y, x) = s_y \cdot \max(0, s_x \cdot (x - bias))$$

Monotonic increasing function

$$ReLU(1, s_y, x) = \mathcal{L}(s_y, x - bias) = s_y \cdot \max(0, (x - bias))$$

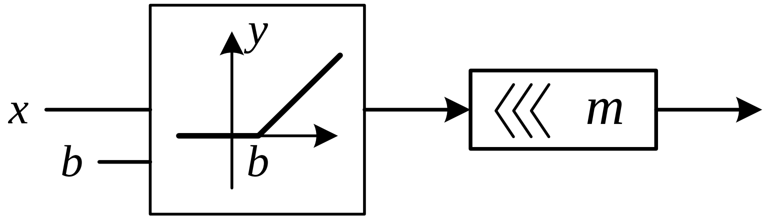
Online Fine-tuning Method

- SPINDLE architecture



$$ReLU(1, s_y, x) = \mathcal{L}(s_y, x - bias) = s_y \cdot \max(0, (x - bias))$$

Given a 2^m slopes



$$y = 2^m \times \max(0, x - b)$$

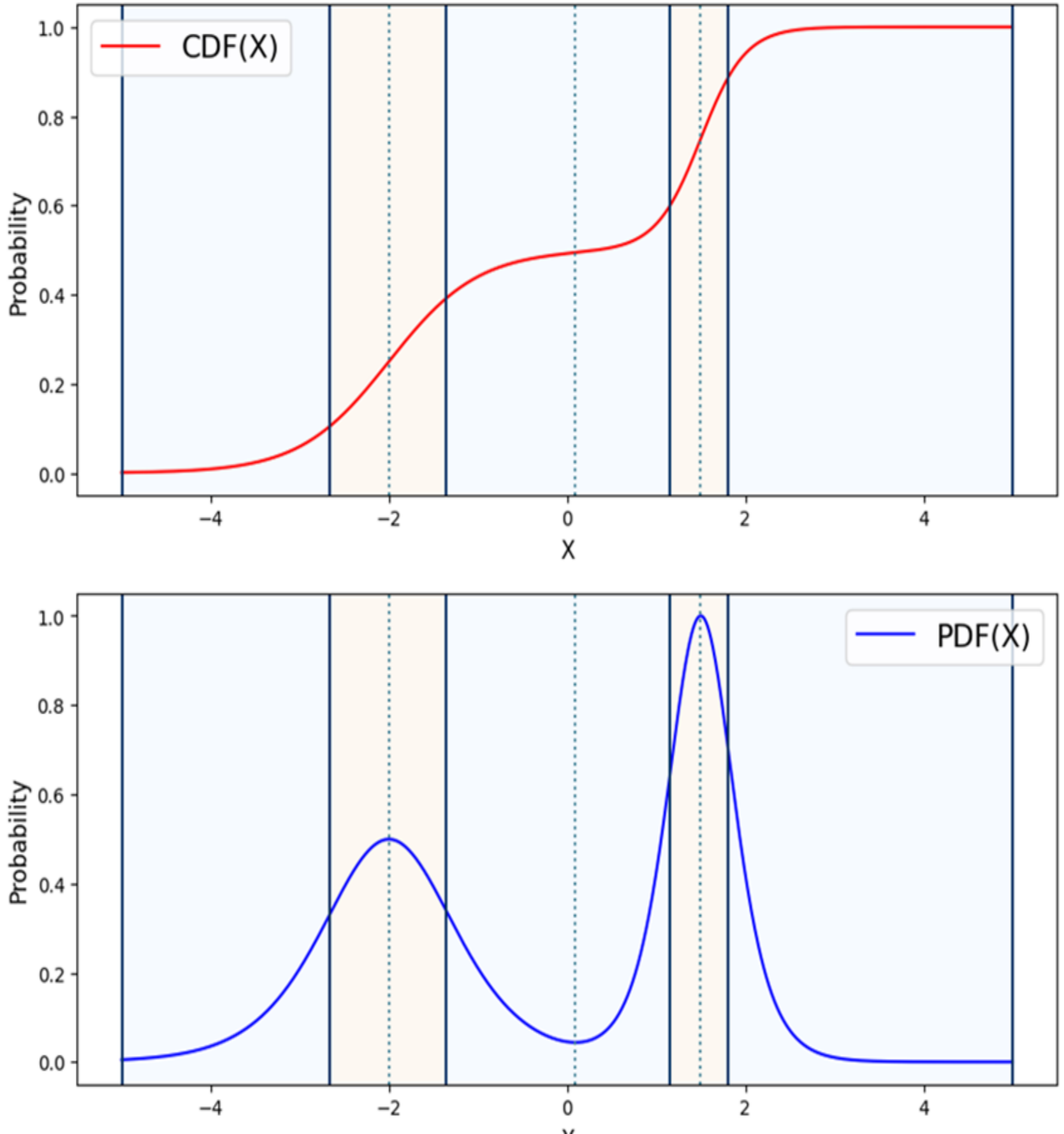
Shifter replace multiplier

$$y = \left(\sum_{i=1}^{k} \sum_{j=0}^{n-1} \sum_{s_{y\in\{\pm 1\}}} \mathcal{L}(s_y, x - b_{ij}) \lll j\right) \ggg m$$

Online Fine-tuning Method

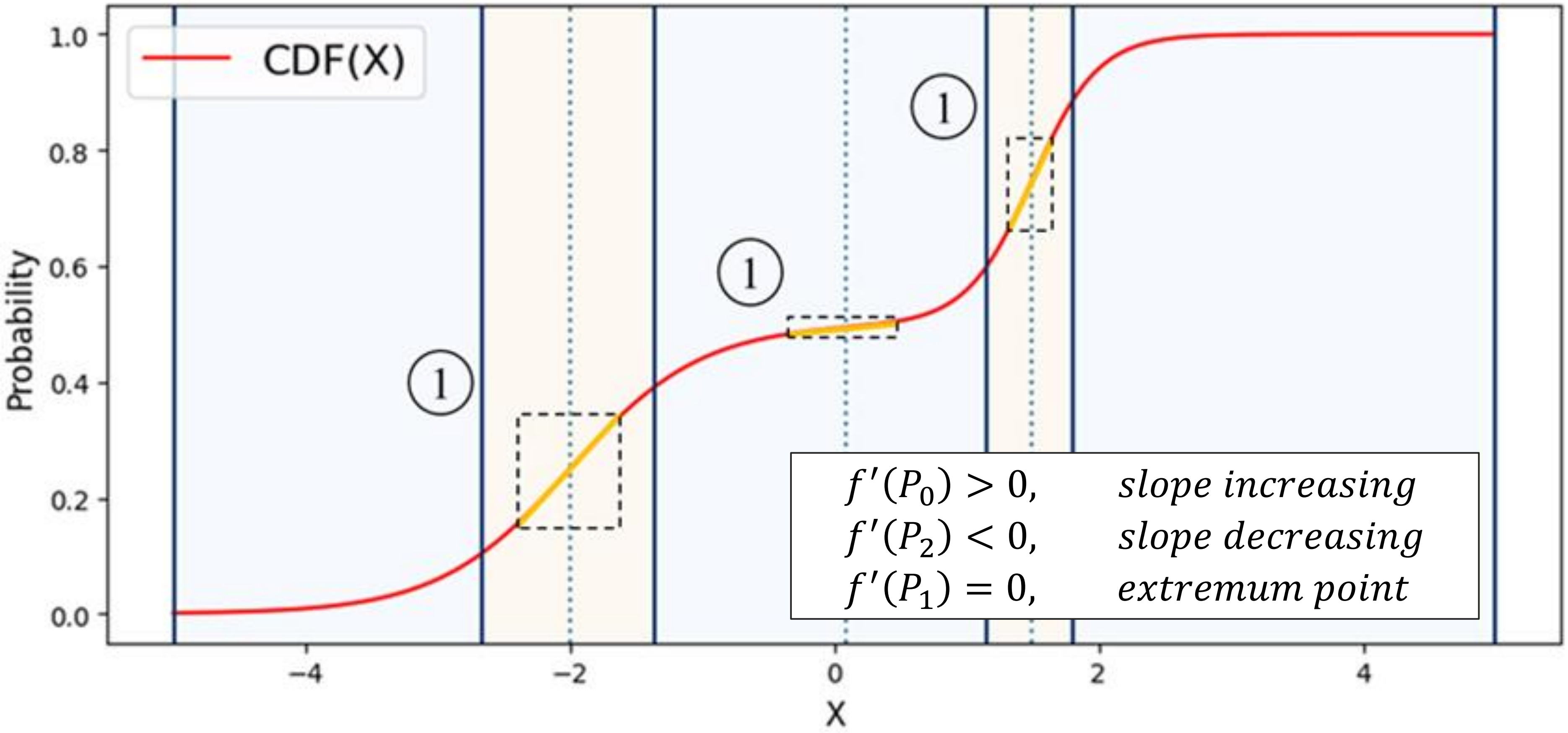
- Binary Search LS-PWL

$$CDF(x) = \int_{-\infty}^{x} PDF(t)dt$$



Online Fine-tuning Method

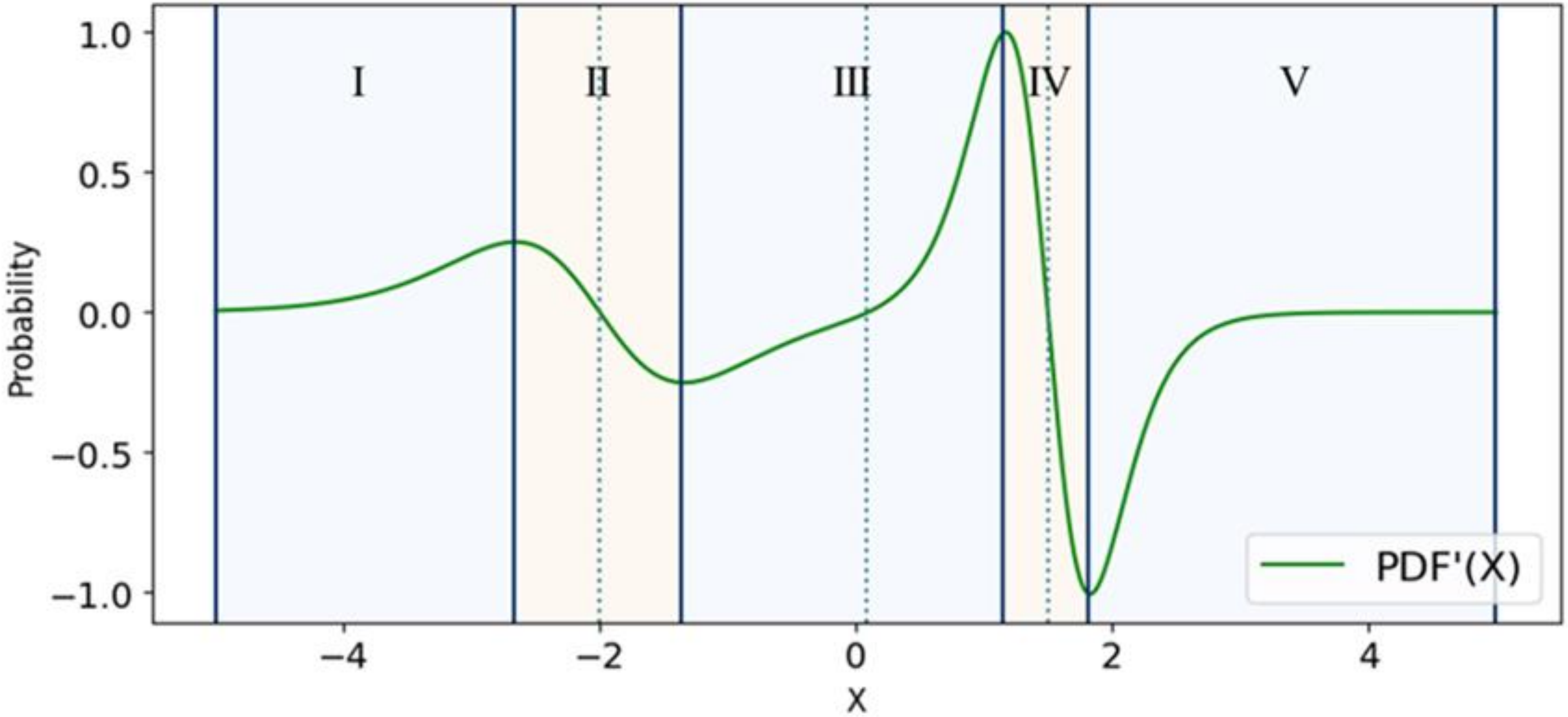
- Binary Search Light-Slope Piece-Wise Line



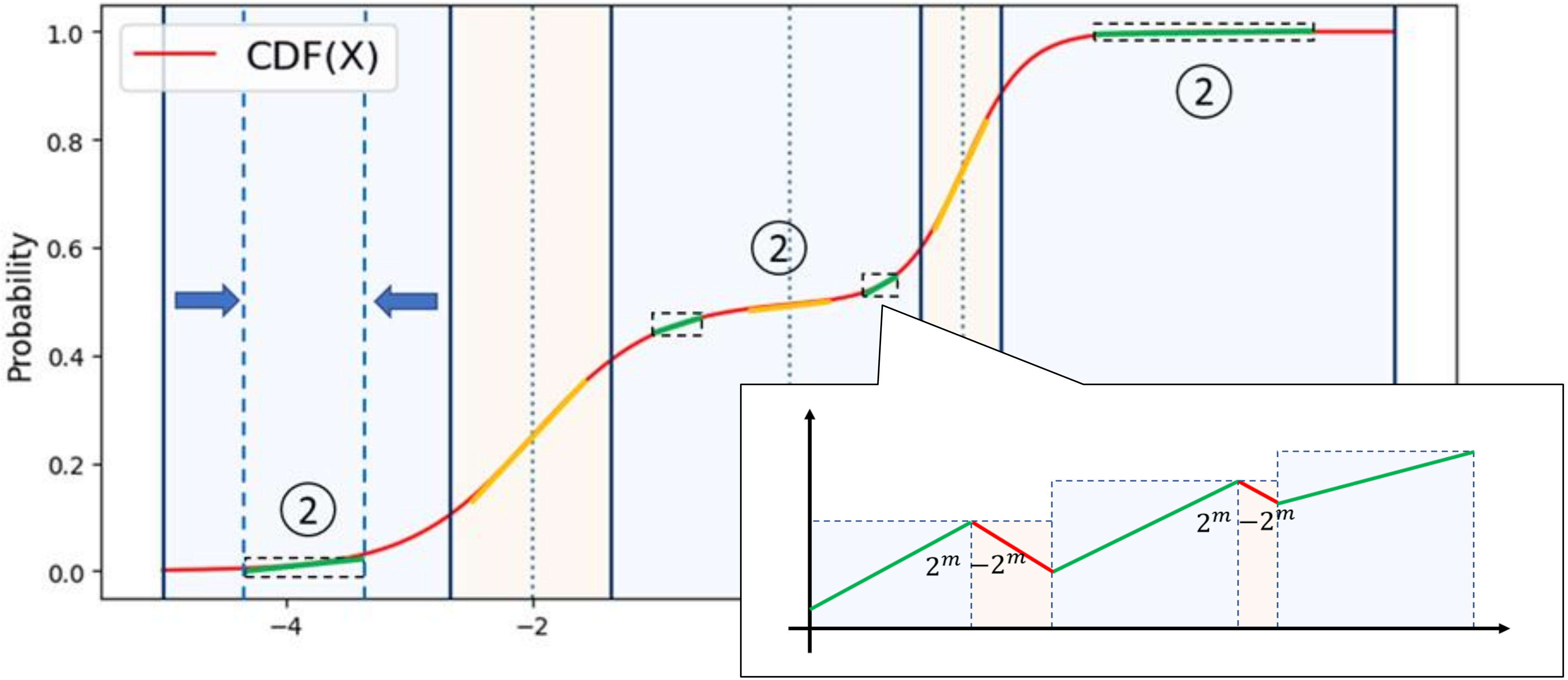
$f'(P_0) > 0,$ *slope increasing*
$f'(P_2) < 0,$ *slope decreasing*
$f'(P_1) = 0,$ *extremum point*

Online Fine-tuning Method

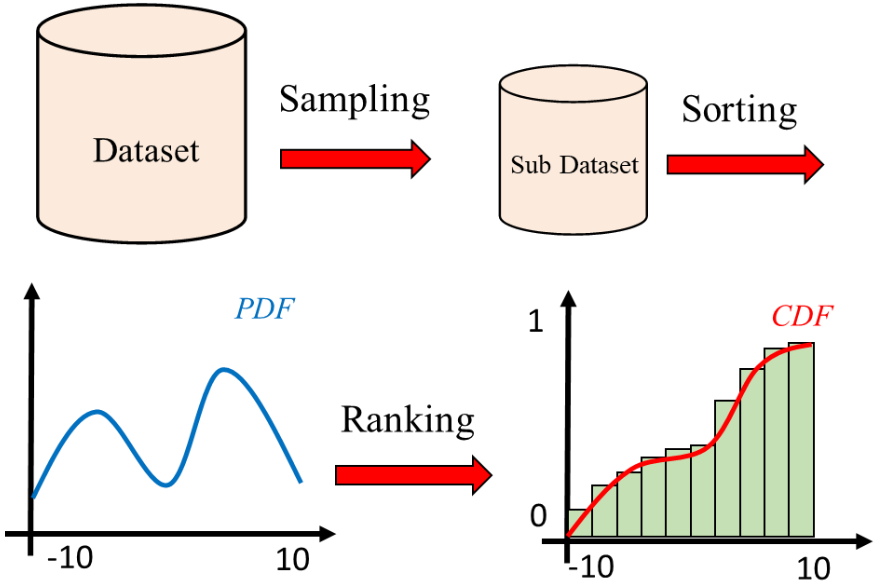
- Binary Search Light-Slope Piece-Wise Line

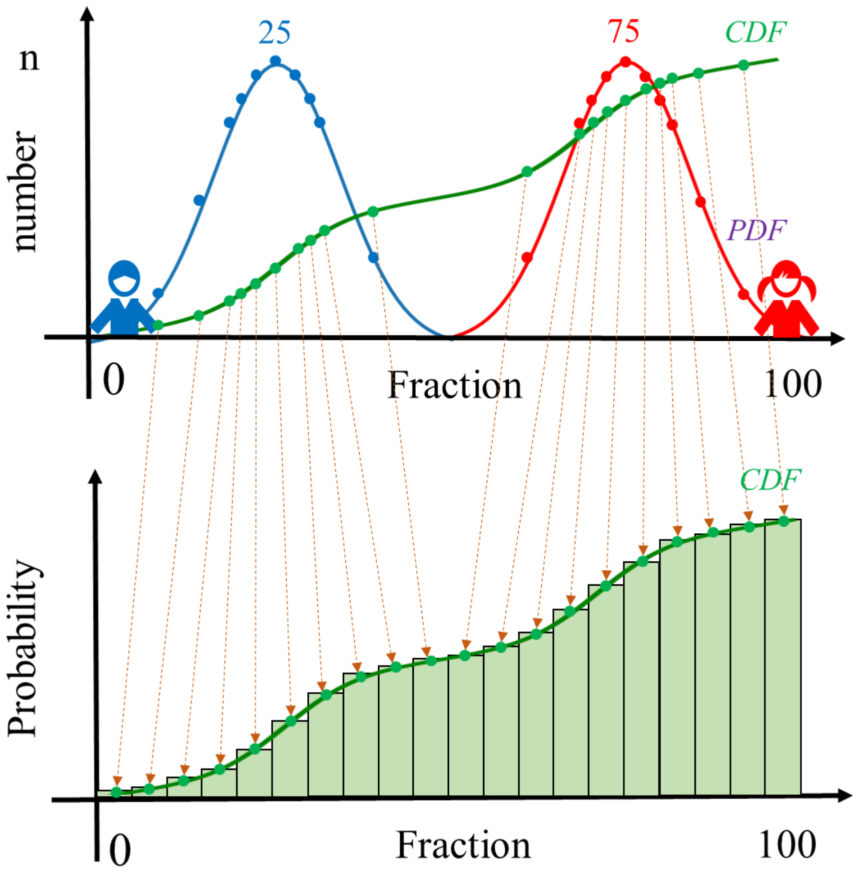


- Binary Search Light-Slope Piece-Wise Line



- Sampling Method
  1. **Sampling Ranking Method**





Offline Sampling Ranking Method

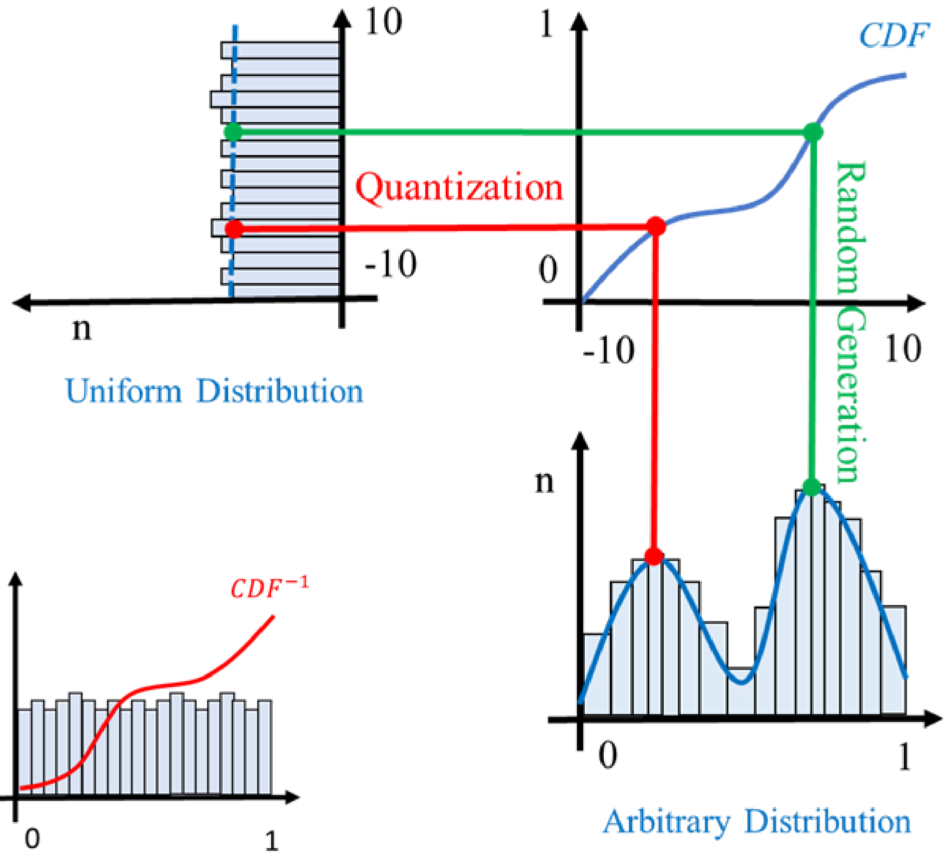
- Sampling Method

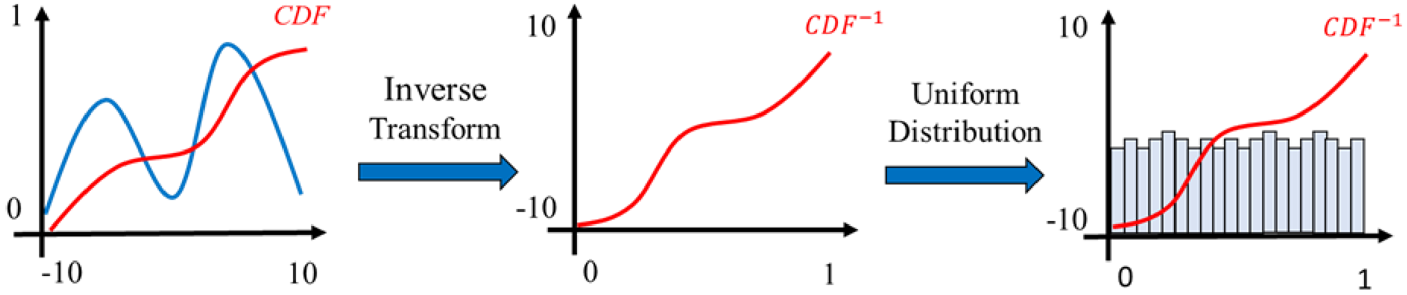
2. Inverse Transform Sampling

$$CDF(x) = P(X \leq x) = \int_{-\infty}^{\infty} PDF(t)dt$$

$$U = CDF(X)$$

$$X = CDF^{-1}(U)$$

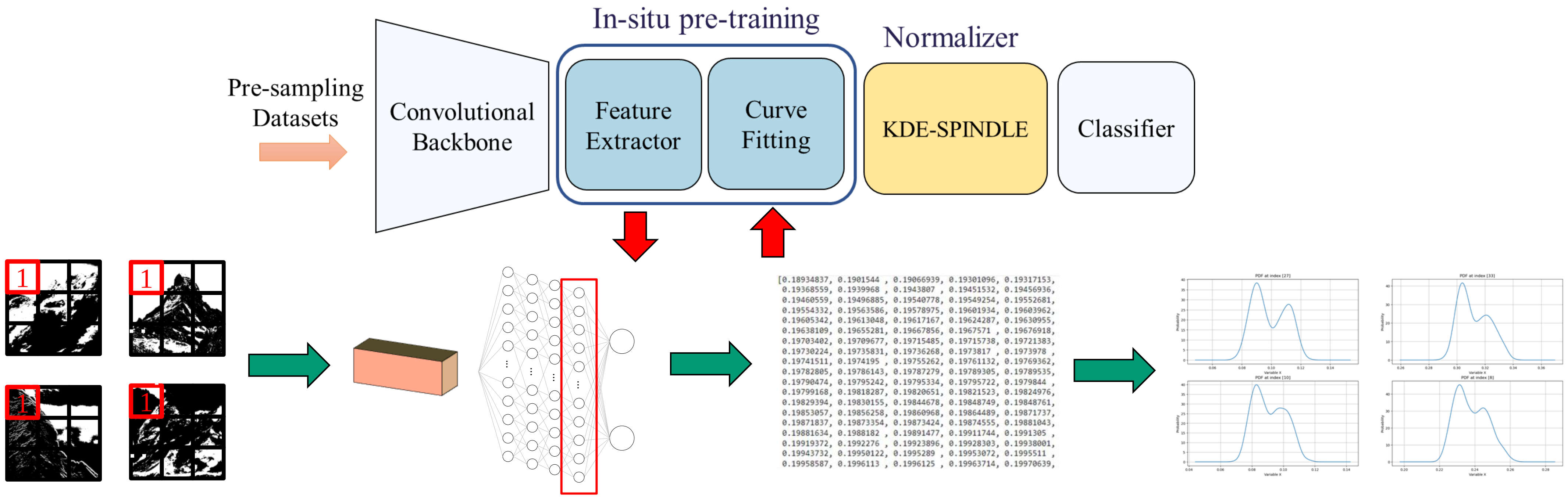




Offline Sampling Ranking Method

- Feature extractor

In computer vision, a feature extractor takes an image as input and identifies key visual patterns or characteristics that are useful for subsequent analysis or classification.



(a) Feature Map (b) Feature Extraction (c) Feature Flattening (d) Feature Distribution Map

Offline Sampling Ranking Method

- Kolmogorov-Smirnov test

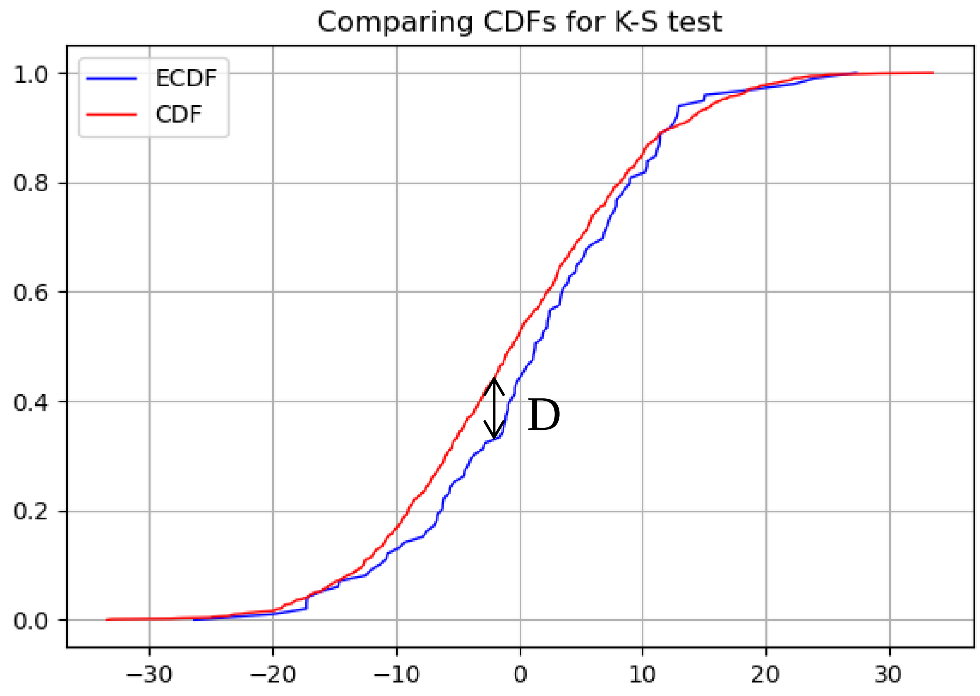
The Kolmogorov-Smirnov test is used to decide if a sample comes from a population with a specific distribution.

Given a sample $x_1, x_2, \ldots, x_n$ of i.i.d, random variables with distribution function F,

$$F(x) = P(X \leq x)$$

Here F_n is a normal distribution

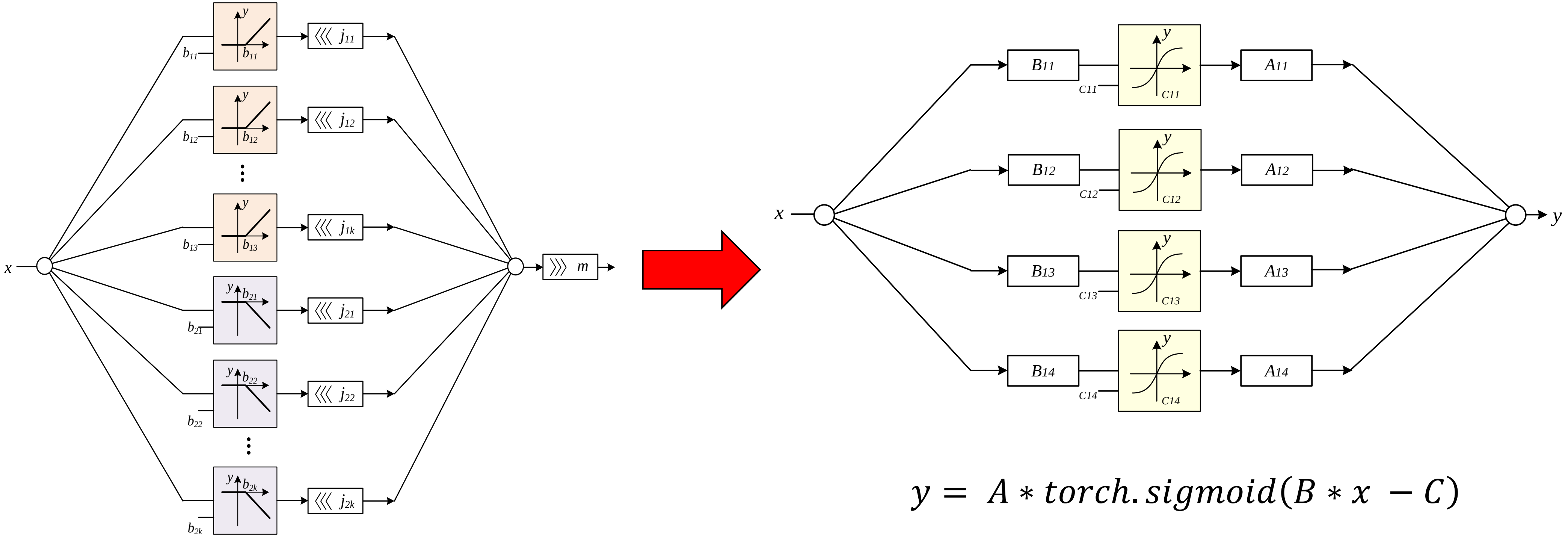
$$D = \max_{1 \leq i \leq n}(\,|F_n(x_i) - F(x_i)|\,)$$



Offline Sampling Ranking Method

- KDE-SPINDLE

KDE means Kernel Density Estimation, we use custom sigmoid as the kernel function to replace the ReLU unit.



$$y = A * torch.sigmoid(B * x - C)$$

Experiment for SPINDLE

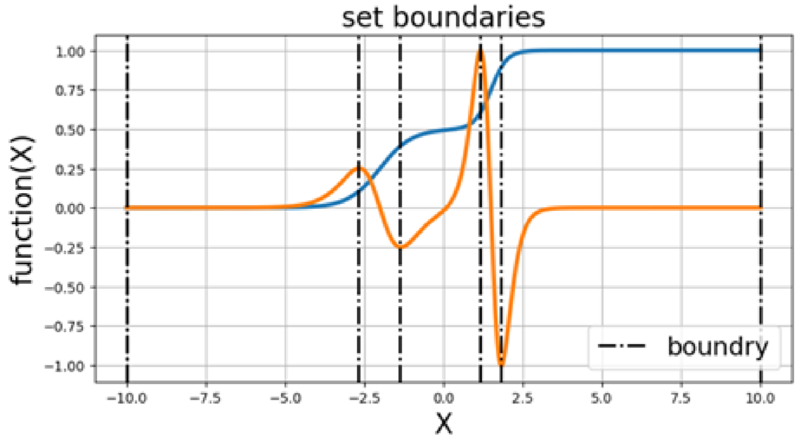
- BLS-PWL

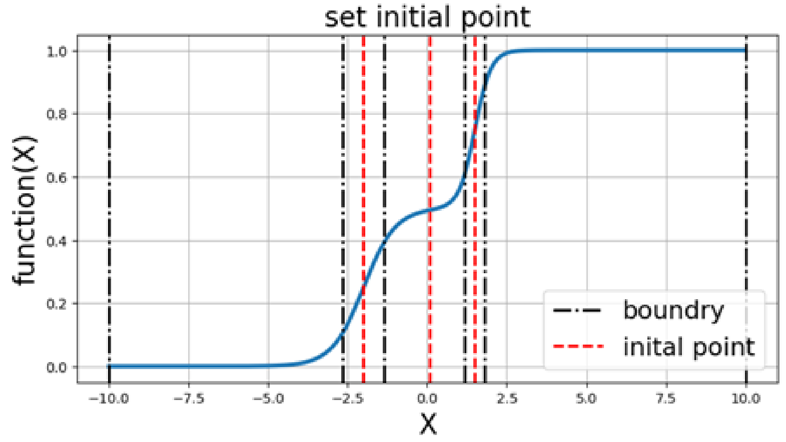
```
Algorithm  Dichotomy LS-PWL

def PWL(m,bias,X):
        return (2**m) * X + bias
def intercept(X):
        return CDF(X)- PWL(m,bias,X)

dl <- gradient(CDF(X))
d2 <- gradient(gradient(CDF(X)))
(In order to find ALL blocks and start points)

for All blocks in PWL(slope from -m to m):
        if PWL (L side + R side) < error rate
                output result(err,m,c,bias,nL,nR)
        else
                do another PWL
        if all PWL > error rate
                do range / 2
        else
                output all PWL(err,m,c,bias,nL,nR)
```

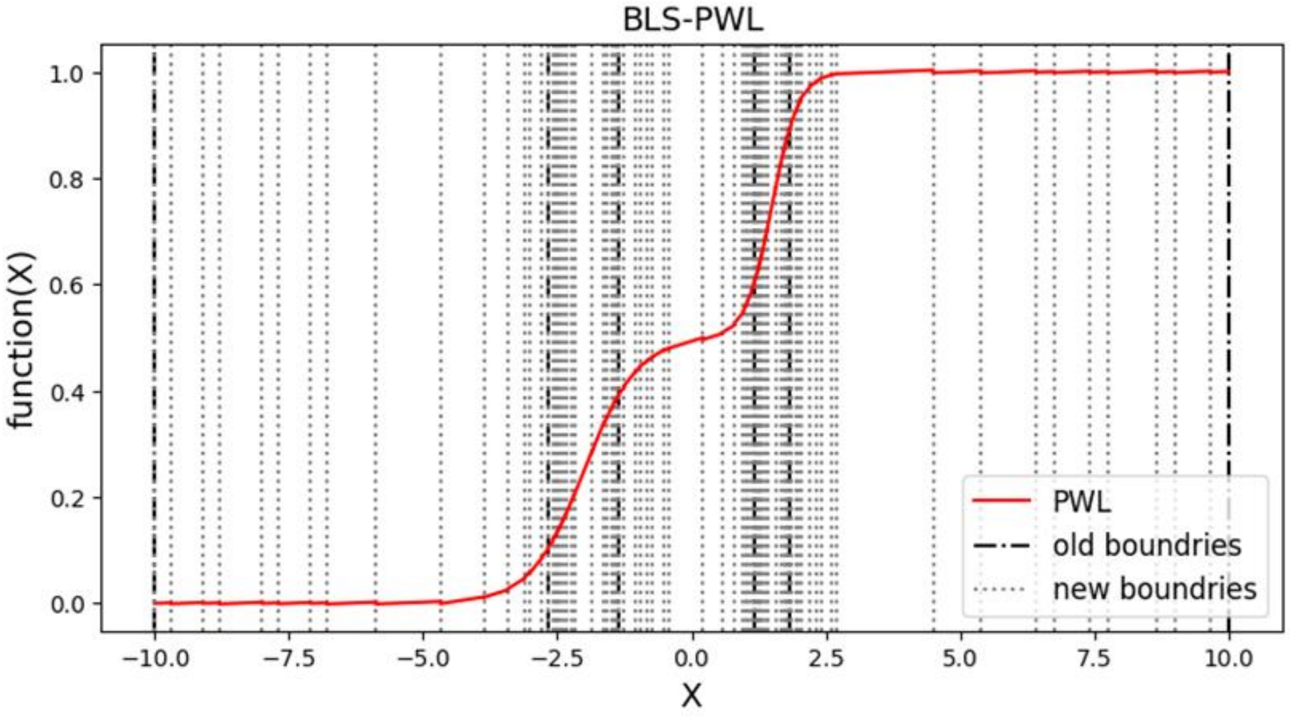




Experiment for SPINDLE

- BLS-PWL

```
 [ -2.      -2.015    0.75    -2.1788  -1.8512]
 [ -2.      -1.7438   0.7486  -1.8512  -1.6363]
 [ -2.      -1.6056   0.7452  -1.6363  -1.5749]
 [ -2.      -1.5346   0.7423  -1.5749  -1.4943]
 [ -2.      -1.4675   0.7387  -1.4943  -1.4406]
 [ -3.      -1.4305   0.5576  -1.4406  -1.4204]
 [ -3.      -1.4003   0.5593  -1.4204  -1.3801]
 [ -3.      -1.3701   0.5608  -1.3801  -1.36   ]]
------------------------------
| Area: 3 Range: -1.36 ~ 1.16 |
------------------------------
######
ans:
[[ -8.      -9.8497   0.0385 -10.      -9.6994]
 [ -8.      -9.3987   0.0367  -9.6994  -9.0981]
 [ -8.      -8.9477   0.035   -9.0981  -8.7974]
 [ -8.      -8.3966   0.0328  -8.7974  -7.9957]
 [ -8.      -7.8454   0.0307  -7.9957  -7.6951]
```



Experiment for SPINDLE

$$y = \text{logistic}(X) \ldots\ldots (1)$$
$$y = 0.5 * \text{logistic}(x + 2) + 0.5 * \text{logistic}(2 * x - 3) \ldots\ldots (2)$$
$$y = 0.1 * \text{logistic}(X + 2) + 0.2 * \text{logistic}(2 * X - 3) + 0.3 * \text{logistic}(X - 5) \ldots\ldots (3)$$

Table.1 BLS-PWL

BLS-PWL		Equation(1)			Equation(2)			Equation(3)		
PDF Type		Unimodal			Bimodal			Multimodal		
Target error		0.01	0.005	0.001	0.01	0.005	0.001	0.01	0.005	0.001
Cost	PWL	23	33	163	17	35	137	19	31	129
Cost	ReLU	46	66	326	34	70	274	38	62	258

Table.2 LS-PWL

LS-PWL[25]		Equation(1)			Equation(2)			Equation(3)		
PDF Type		Unimodal			Bimodal			Multimodal		
Target error		0.01	0.005	0.001	0.01	0.005	0.001	0.01	0.005	0.001
Cost	PWL	7	11	32	NA	NA	NA	NA	NA	NA

Experiment for SPINDLE

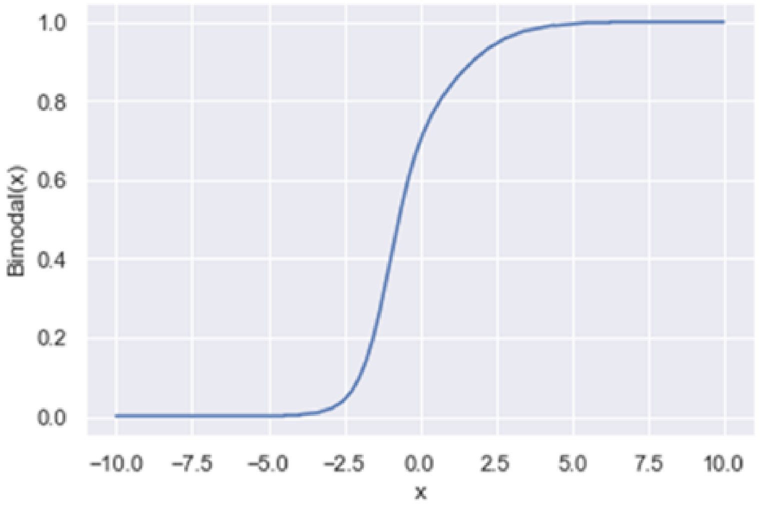


Fig.1 LS-PWL[25] bimodal function

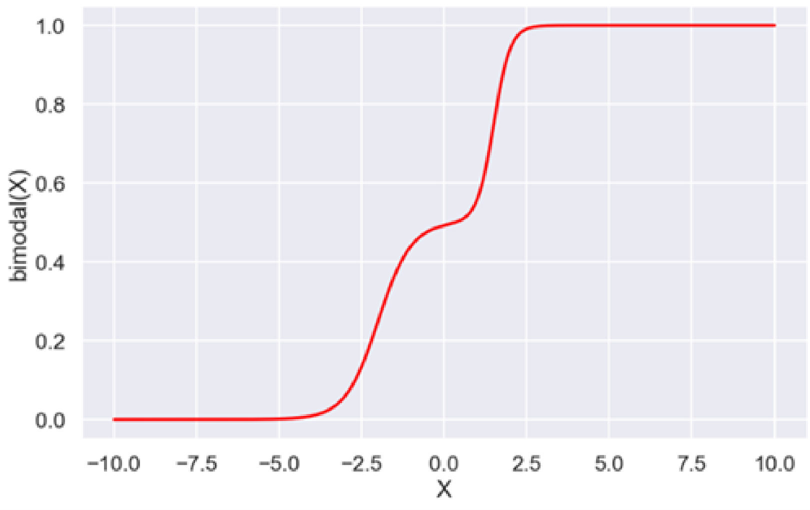


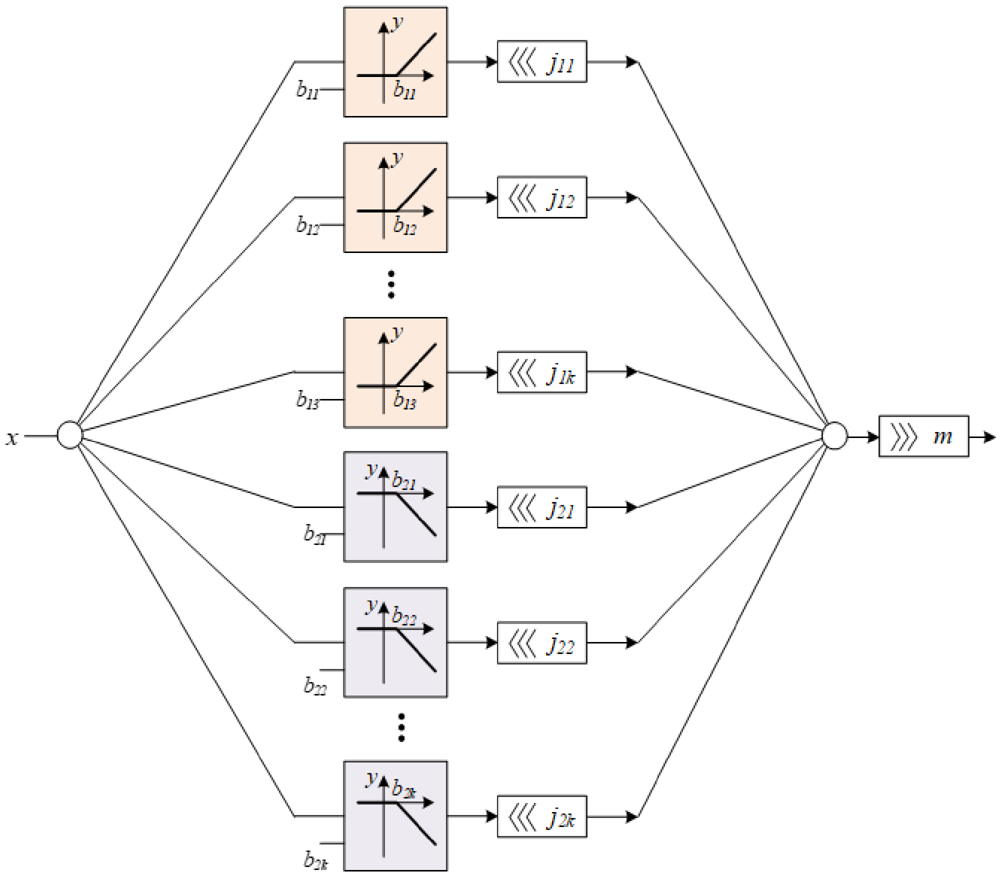
Fig.2 BLS-PWL bimodal function

Table. Compare with pervious work

	LS-PWL[25]	BLS-PWL
Fine-Tune	No	Yes
Hardware circuit	Yes	Yes
PWL Cost	Low	high
Function	Only Unimodal distribution	Any distribution

Experiment for SPINDLE

- SPINDLE




```python
def ReLU(s, x, b):
    return s* max(0, x-b)

def spindle(s, x, b, L, R):
    n = s.size
    y = 0.0
    for i in range(n):
        y += ReLU(s[i], x, b[i]) * 2 ** L[i]
    y = y / (2 ** R)
    return y
```

Fig.1 SPINDLE.ipynb (Python code)

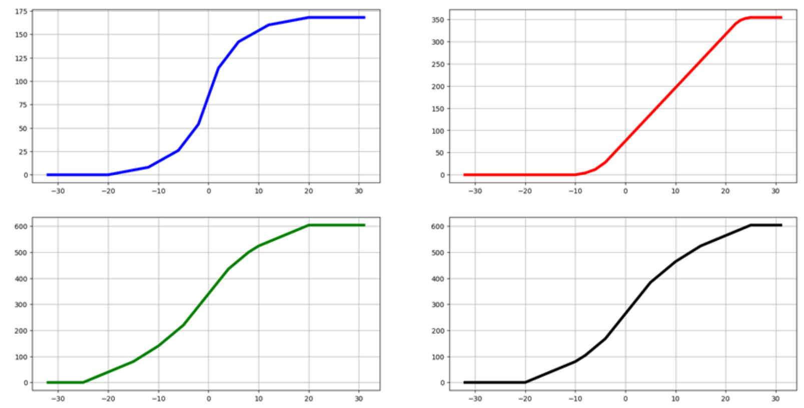


Fig.2 Four kinds of activation (Golden data)

Experiment for SPINDLE

- SPINDLE

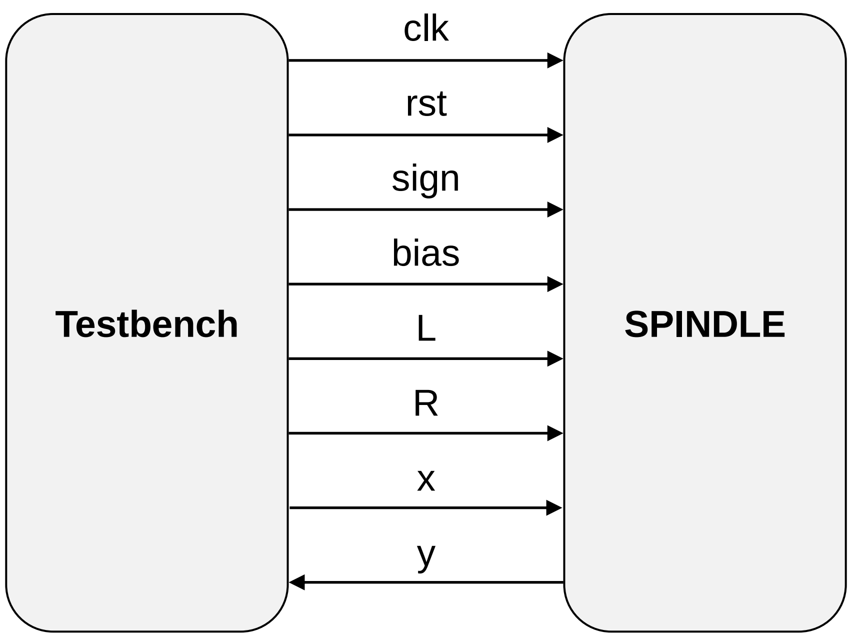


Fig.1 block diagram

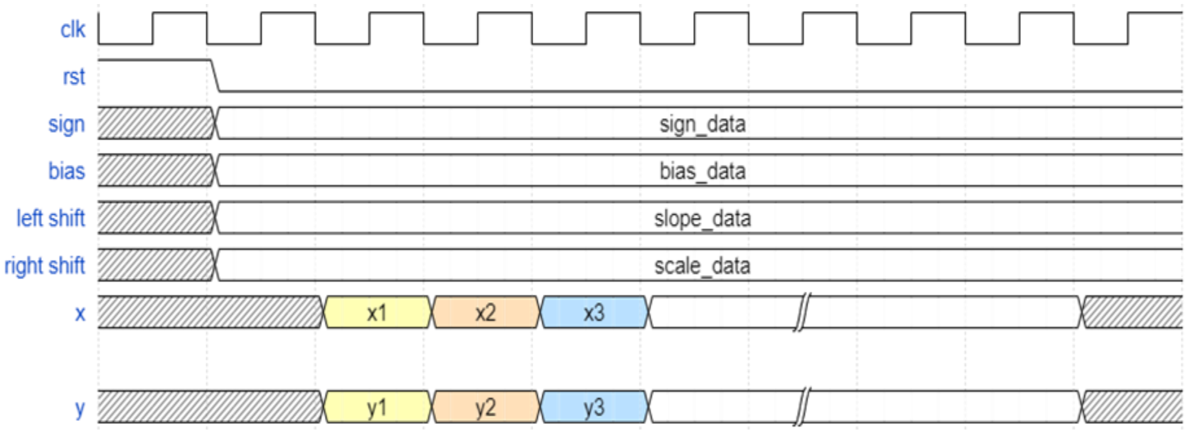


Fig.2 timing diagram

Experiment for SPINDLE

- SPINDLE

spindle.v × Workspace Trust

users > m1053012 > Desktop > spindle > verilog > spindle.v > {} spindle

```verilog
// wire [9:0] max_temp[0:7];
wire [9:0] sign_max[0:7];
wire [9:0] return_max[0:7];
wire [9:0] Relu;
reg [9:0] com_result[0:7];
integer k;

wire [7:0] i_data_b_two_s_complement[0:7];
genvar gen_i;
generate
  for (gen_i = 0; gen_i < 8; gen_i = gen_i + 1) begin : gen_two_s_complement
    assign i_data_b_two_s_complement[gen_i] = ~i_data_b[gen_i] + 1;
  end
endgenerate
always @(*) begin
  for (k = 0; k < 8; k = k + 1) begin

    case ({
      i_data_x[7], i_data_b[k][7]
    })
      2'b00:   com_result[k] = i_data_x > i_data_b[k] ? i_data_x - i_data_b[k] : 0;
      2'b01:   com_result[k] = i_data_x + i_data_b_two_s_complement[k];
      2'b10:   com_result[k] = 0;
      2'b11:   com_result[k] = i_data_x > i_data_b[k] ? i_data_x - i_data_b[k] : 0;
      default: com_result[k] = i_data_x > i_data_b[k] ? i_data_x - i_data_b[k] : 0;
    endcase
  end
end
genvar gen_x;
generate
  for (gen_x = 0; gen_x < 8; gen_x = gen_x + 1) begin : max_gen
    assign max[gen_x] = com_result[gen_x];
    assign sign_max[gen_x] = i_data_s[gen_x] ? max[gen_x] : ~max[gen_x] + 1;  // 1 pos
    assign return_max[gen_x] = sign_max[gen_x] << i_data_L[gen_x];
  end
endgenerate
assign Relu   = return_max[0] + return_max[1] + return_max[2] + return_max[3] + retur
assign temp_y = (Relu >> i_data_R);
```

Fig.1 SPINDLE.v (Verilog Code)

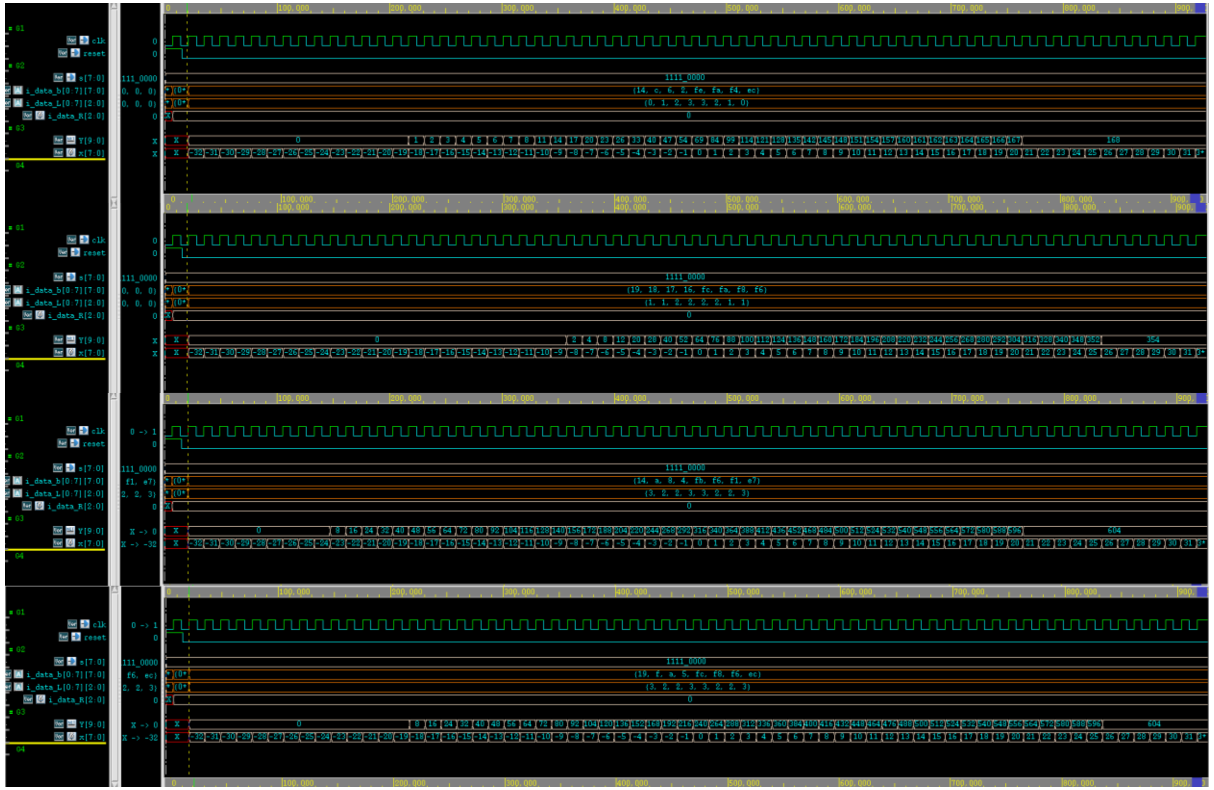
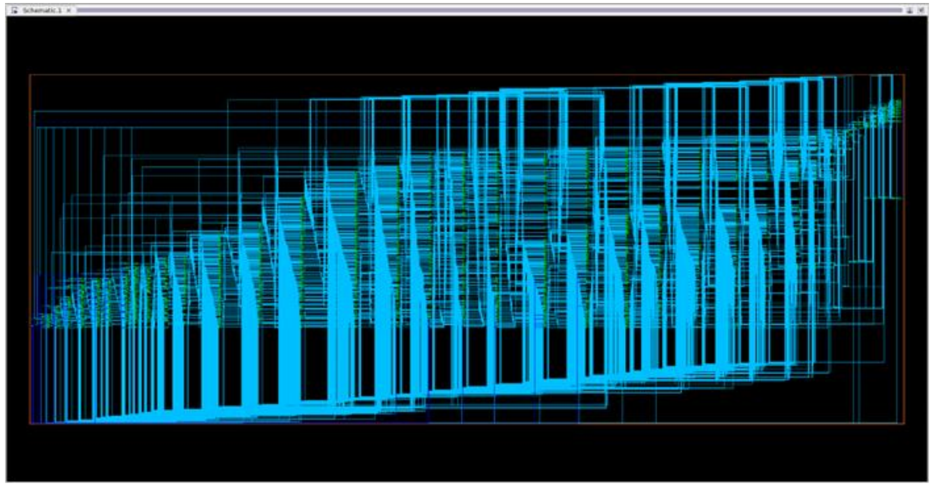
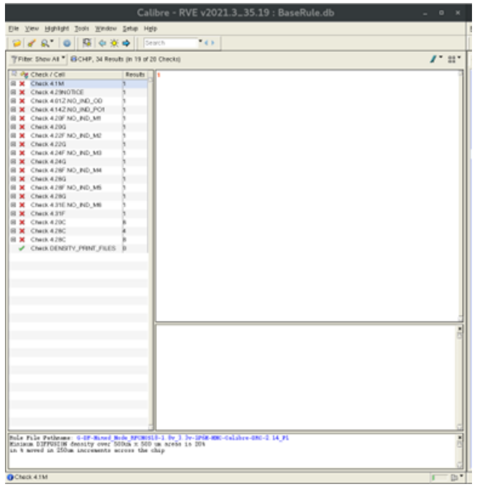
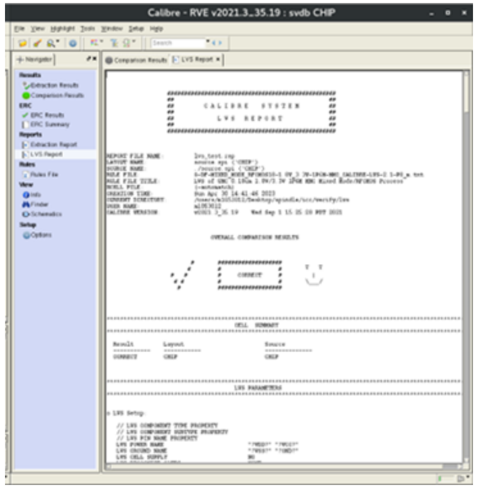


Fig.2 Four Output Wave View

Experiment for SPINDLE







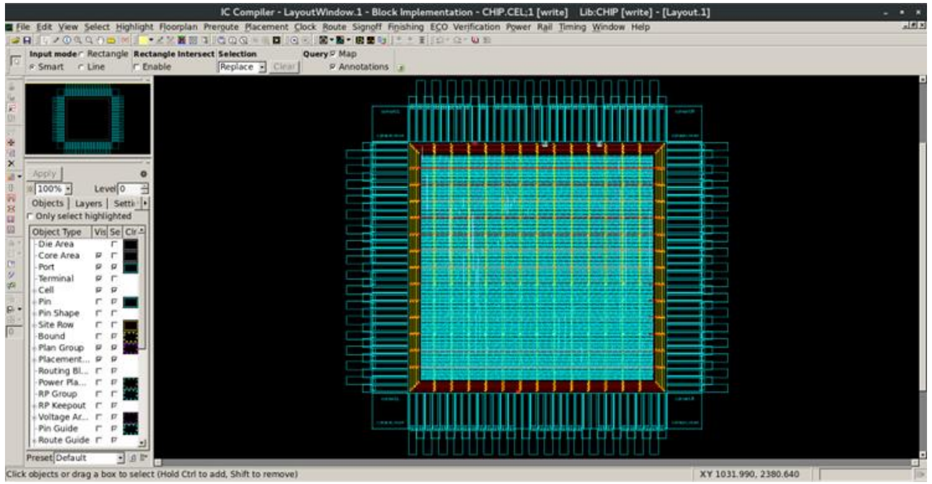
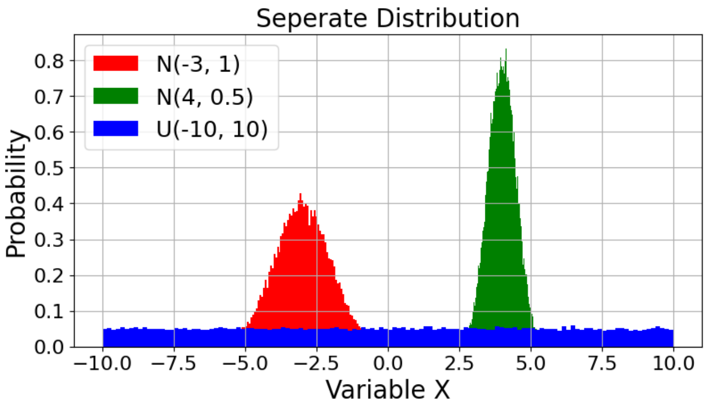


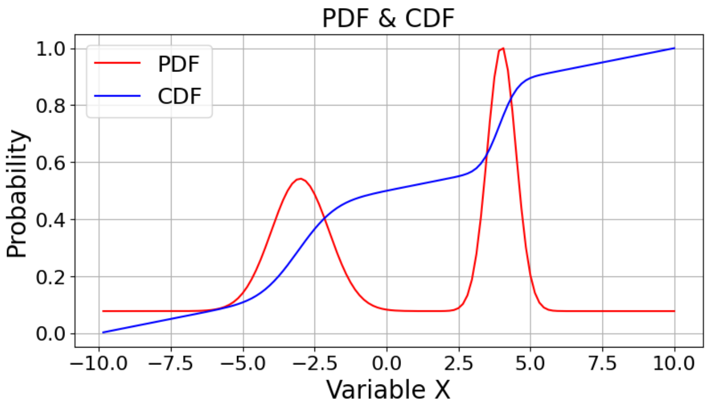
Table. CHIP PAT

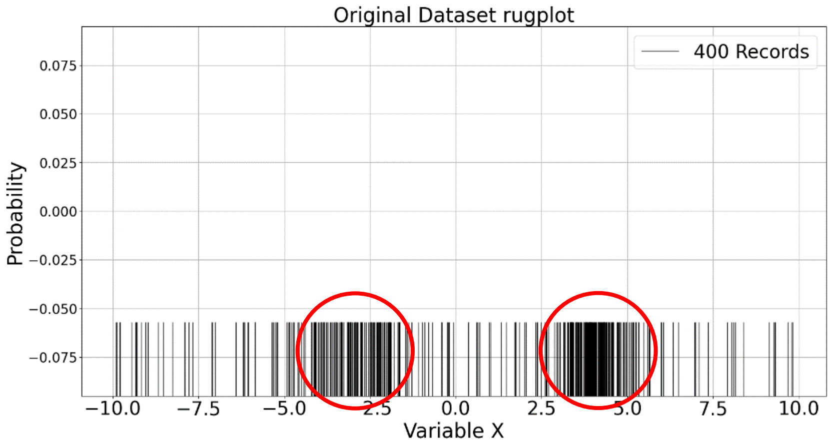
	SPINDLE	
function	Any kind of activation (Fine-Tune)	
stage	Per-sim	Post-sim
Power(mW)	4.4151	5.4151
Area(μm^2)	991848.318711	997419.837070
Timing(ns)	15.24	16.13

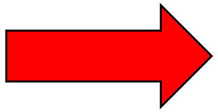
Experiment for KDE-SPINDLE

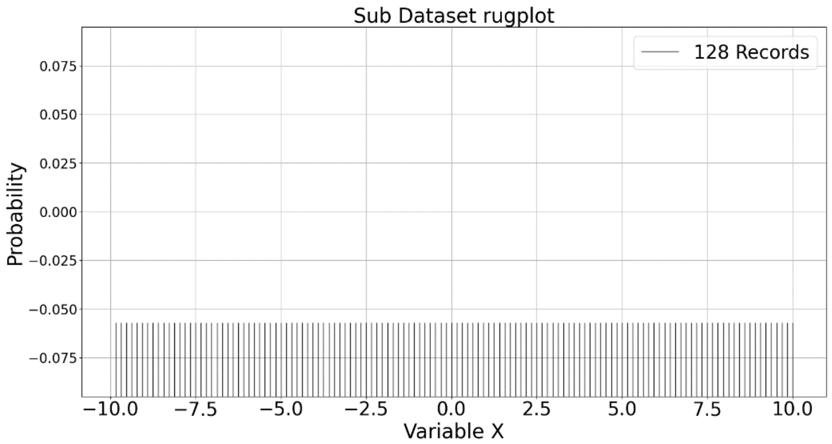
- Sampling Ranking Method





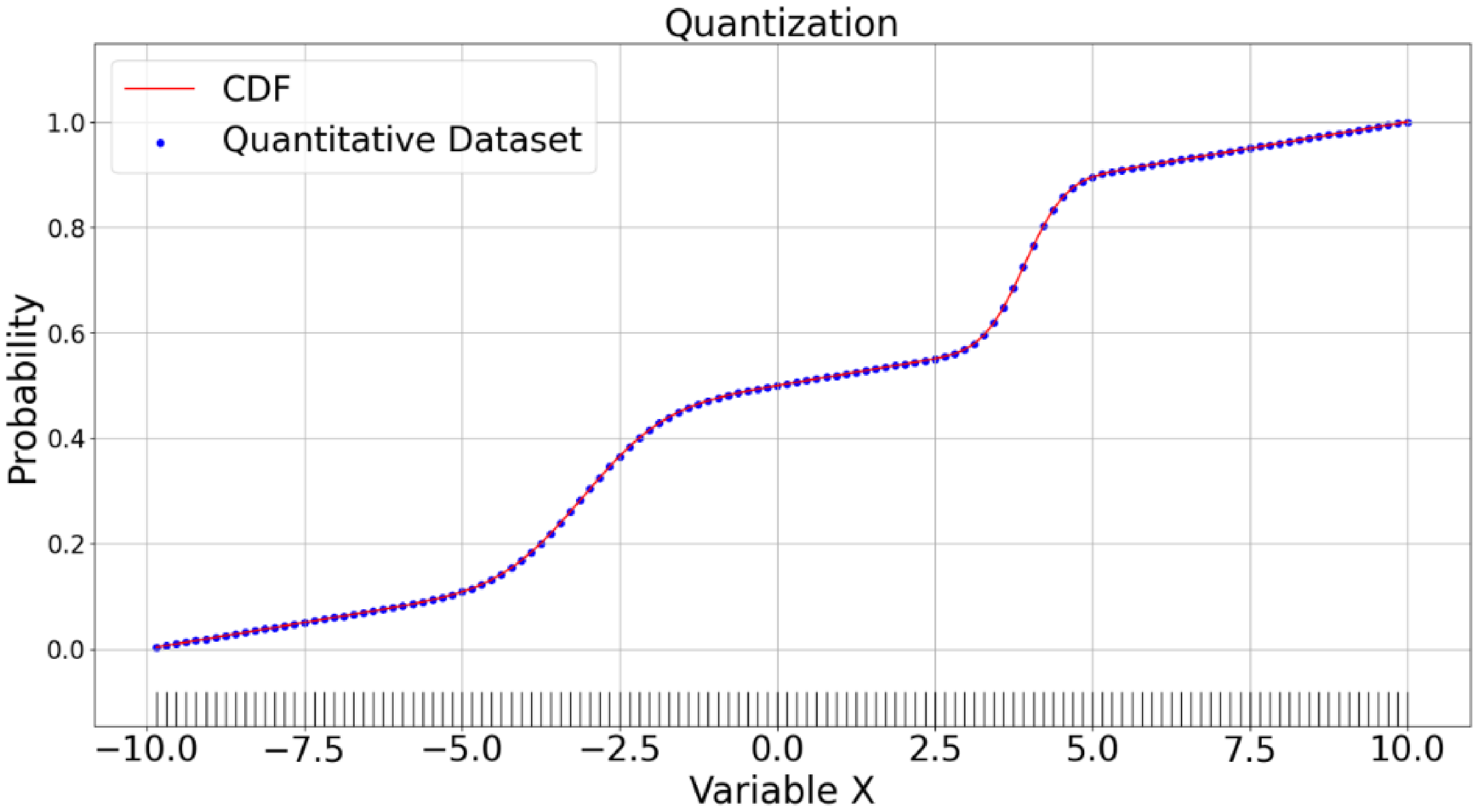






Experiment for KDE-SPINDLE

- Sampling Ranking Method



Experiment for KDE-SPINDLE

- Inverse Transform Sampling

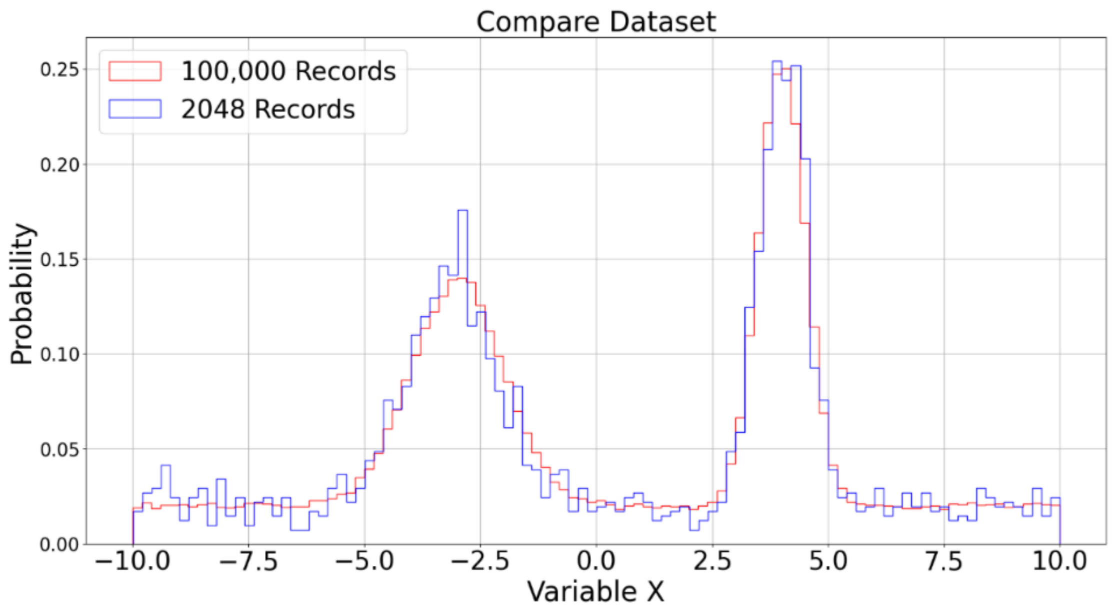


Fig.1 Two dataset PDF

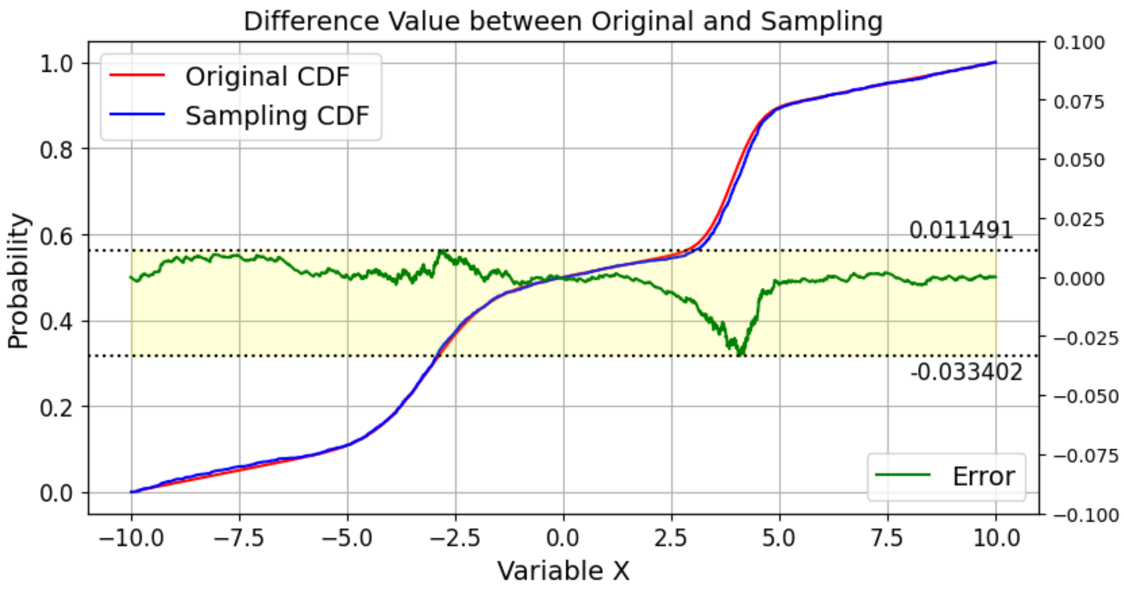


Fig.2 Two dataset CDF & difference value

Experiment for KDE-SPINDLE

- Three kinds of backbone neural network

Fig.1 nirscene1[2] dataset

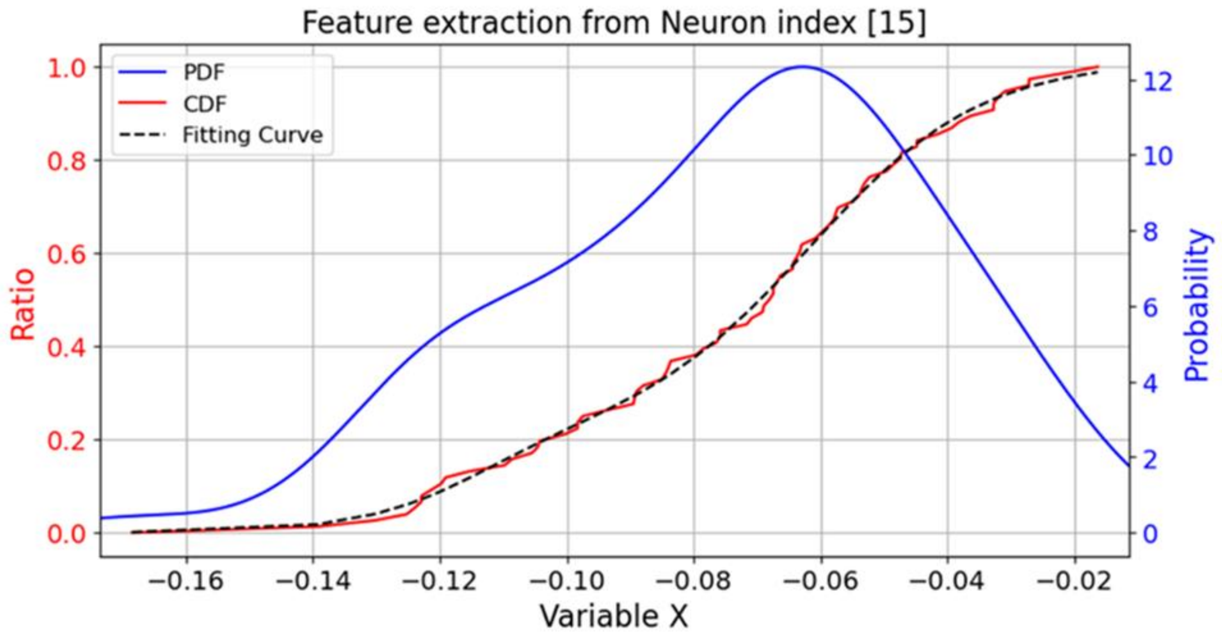


Fig.2 non-normal distribution feature map (VGG-16)

Experiment for KDE-SPINDLE

Table.1 Feature Extraction Parameters

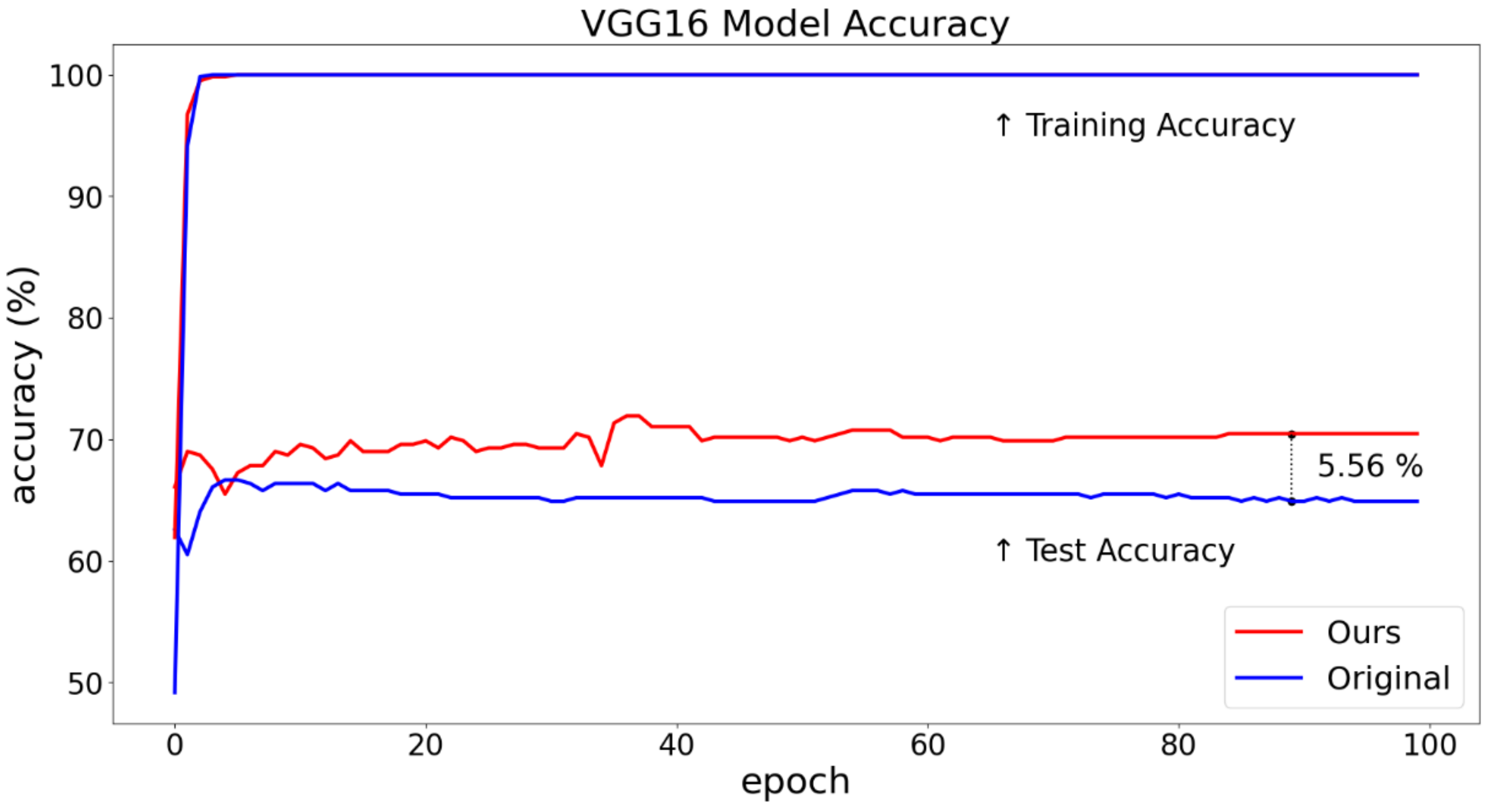
Model	AlexNet[10]	ResNet-18[11]	VGG-16[12]
Dataset	nirscene1[2]	nirscene1[2]	nirscene1[2]
Kernel Unit	$y = A * torch.sigmoid(B * x - C)$		
Kernel Number	2	2	2
Variable Value	[0.2, 522.6, -26.6] [0.8, 788.3, -41.5]	[0.3, 79, -33.1] [0.7, 84.8, -32.7]	[0.8, 36.8, 2.3] [0.2, 51.7, 5.9]

Table.2 Compare with Test Accuracy

Dataset		nirscene1[2]					
Model		AlexNet[10]		ResNet-18[11]		VGG-16[12]	
		Orig.	Ours	Orig.	Ours	Orig.	Ours
Input	(C,H,W)	(3,256,256)		(3,224,224)		(3,224,224)	
Batch size		8		8		8	
Train / Test num.		612 / 342		612 / 342		612 / 342	
Layer		27		73		40	
Learning Rate		1e-5	1e-5	1e-4	5e-5	1e-5	1e-5
Epoch		500		100		100	
Total Params		11.24 M		44.69M		119.61 M	
Params size	(MB)	182.25		106.27		675.7	
Accuracy	(%)	38.89	**42.4**	41.8	**42.4**	66.67	**71.92**

Experiment for KDE-SPINDLE

- Three kinds of backbone neural network



Experiment for KDE-SPINDLE

- Three kinds of custom datasets

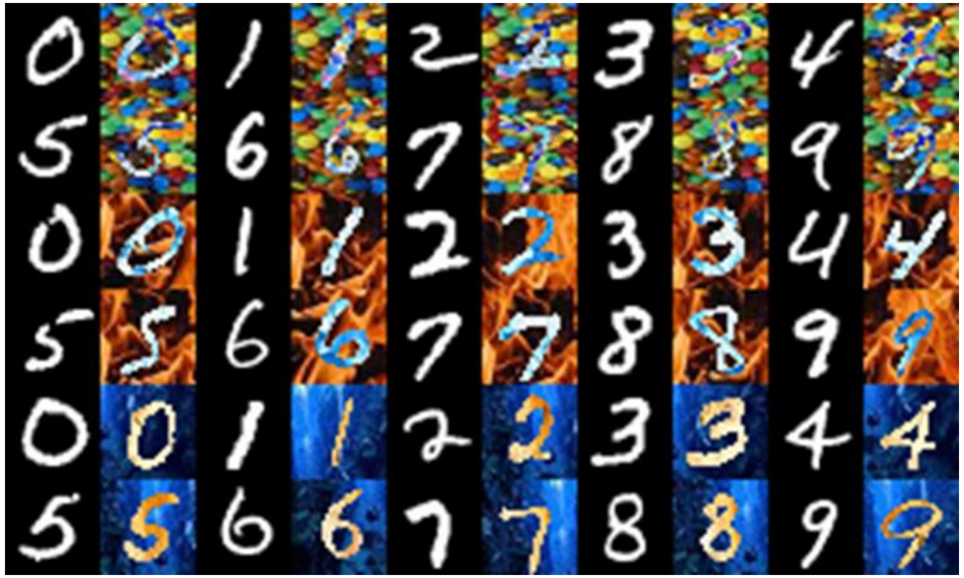


Fig.1 PolyMNIST[27] dataset

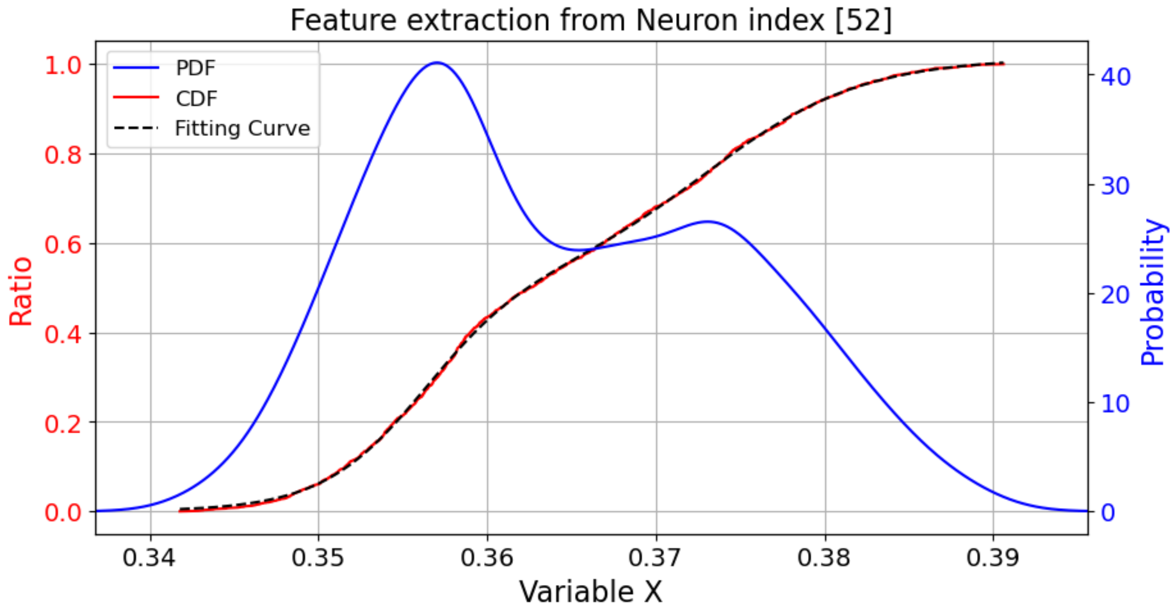


Fig.2 non-normal distribution feature map (M0-dataset)

Experiment for KDE-SPINDLE

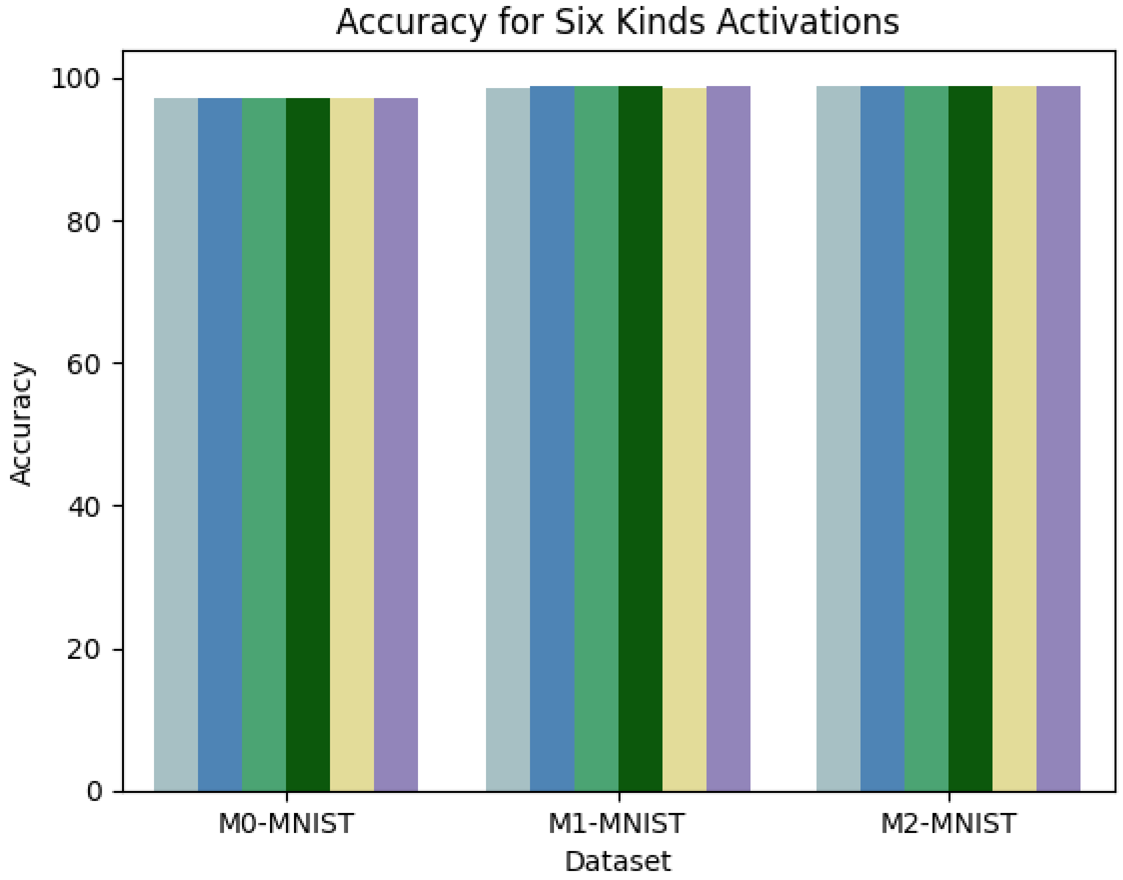
Table.1 Feature Extraction Parameters

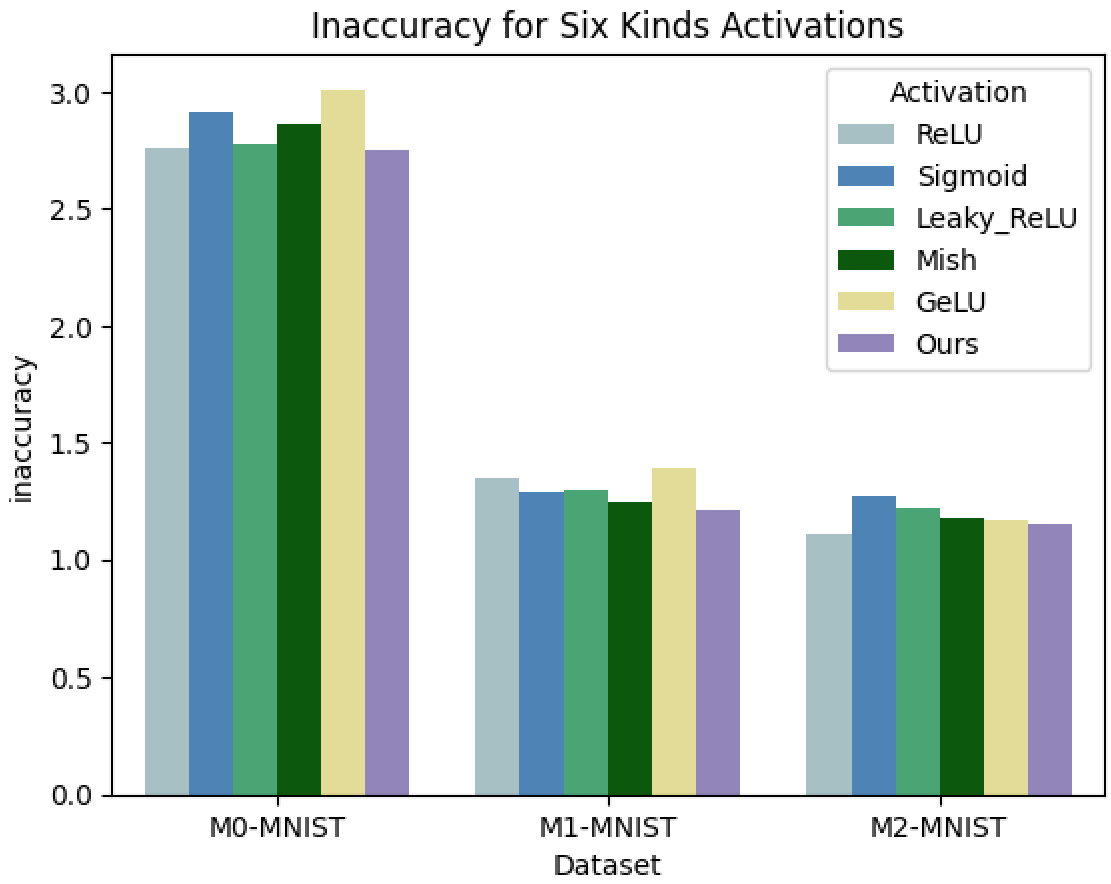
Model	LeNet-5[9]		
Dataset	M0-MNIST	M1-MNIST	M2-MNIST
Background	Candy	Fire	Sea
Kernel Unit	$y = A * torch.sigmoid(B * x - C)$		
Kernel Number	2	3	2
Variable Value	[0.5, 522.6, -26.6] [0.8, 788.3, -41.5]	[0.2, 173.1, -17.9] [0.5, 138, -12.9] [0.3 188.5 -15.9]	[0.5, 185.3, -23.3] [0.5, 121, -16.2]

Table.2 Compare with Test Accuracy

Kind		Base		Backbone			Ours
Activation		ReLU [13]	Sigmoid [15]	L-ReLU [19]	Mish [19]	GeLU [21]	SPINDLE
M0	Accuracy	97.24	97.09	97.22	97.14	96.99	97.25
	Inaccuracy	2.76	2.91	2.78	2.86	3.01	**2.75**
M1	Accuracy	98.65	98.71	98.7	98.75	98.61	98.79
	Inaccuracy	1.35	1.29	1.3	1.25	1.39	**1.21**
M2	Accuracy	98.89	98.73	98.78	98.82	98.83	98.85
	Inaccuracy	**1.11**	1.27	1.22	1.18	1.17	**1.15**

Experiment for KDE-SPINDLE





Experiment for KDE-SPINDLE

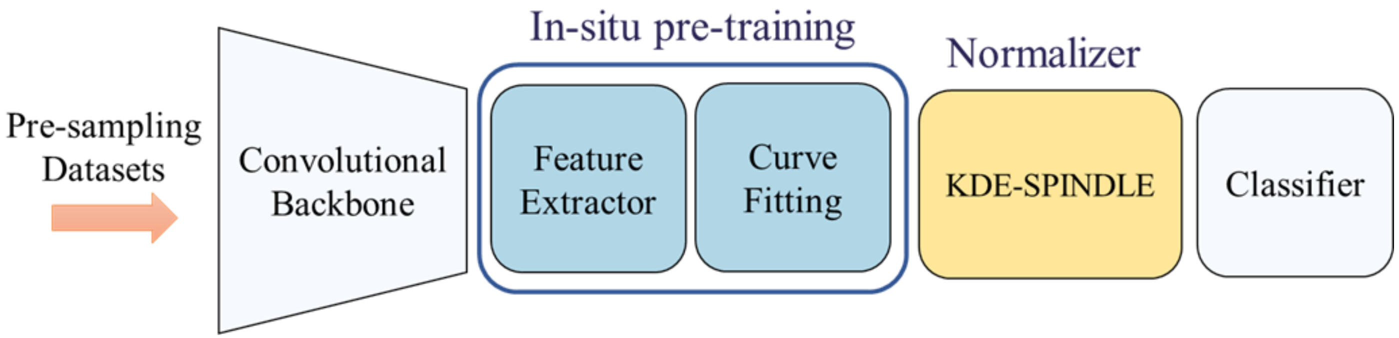


Fig.1 Self Pre-trainable In-situ Normalizer for Deep Learning Error function

Table.1 Ranking of the six activation functions

Kind		Base		Backbone			Ours
Activation		ReLU [13]	Sigmoid [15]	L-ReLU [19]	Mish [19]	GeLU [21]	SPINDLE
M0	Rank	II	V	III	IV	VI	**I**
M1	Rank	V	III	IV	II	VI	**I**
M2	Rank	**I**	VI	V	IV	III	II
Total Rank		II	V	IV	III	VI	**I**

Conclusions

- ✓ This paper is to improve the dataset with non-normal distributions. Basic activation functions are unable to achieve accurate normalization.
- ✓ This design must be applicable to the backbone network.

1. In Online SPINDLE, we propose BLS-PWL algorithm that can generate initial points and boundary , and introduce a fine-tuning feature.
2. In Offline KDE-SPINDLE, we implemented an in-situ design that allows us to change the shape of the activation function through pre-sampling.
3. We propose two methods to demonstrate the quantization characteristics of the sampled dataset.
4. Our approach exhibits higher accuracy compared to various backbone networks and existing activation functions.

Thanks for your attention.