KAN Kolmogorov Arnold Network Note

研究任務

1. 閱讀 KAN Kolmogorov-Arnold Network 論文
2. 設計 KAN 網路系統階層式架構 IDEF0
3. 設計 KAN 網路系統每個功能模組離散事件建模 Grafcet
4. 以 MIAT 方法論合成每個 Grafcet 控制器電路
5. 以 ChatGPT 合成每個 Grafcet Datapath 電路
6. FPGA 整合驗證

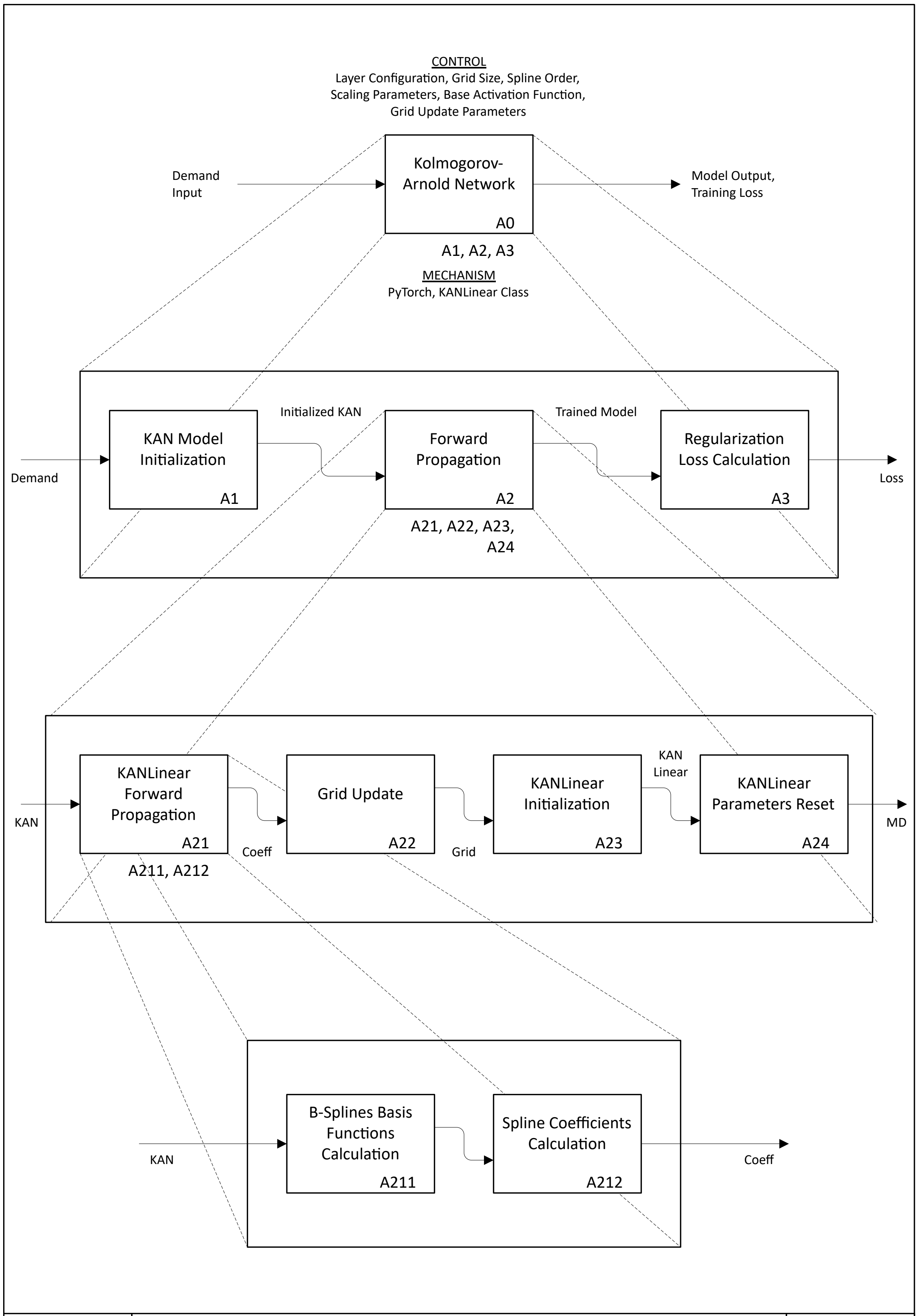
Attention

KAN 之研究任務須於六月底完成，作為實習機會的前置條件。

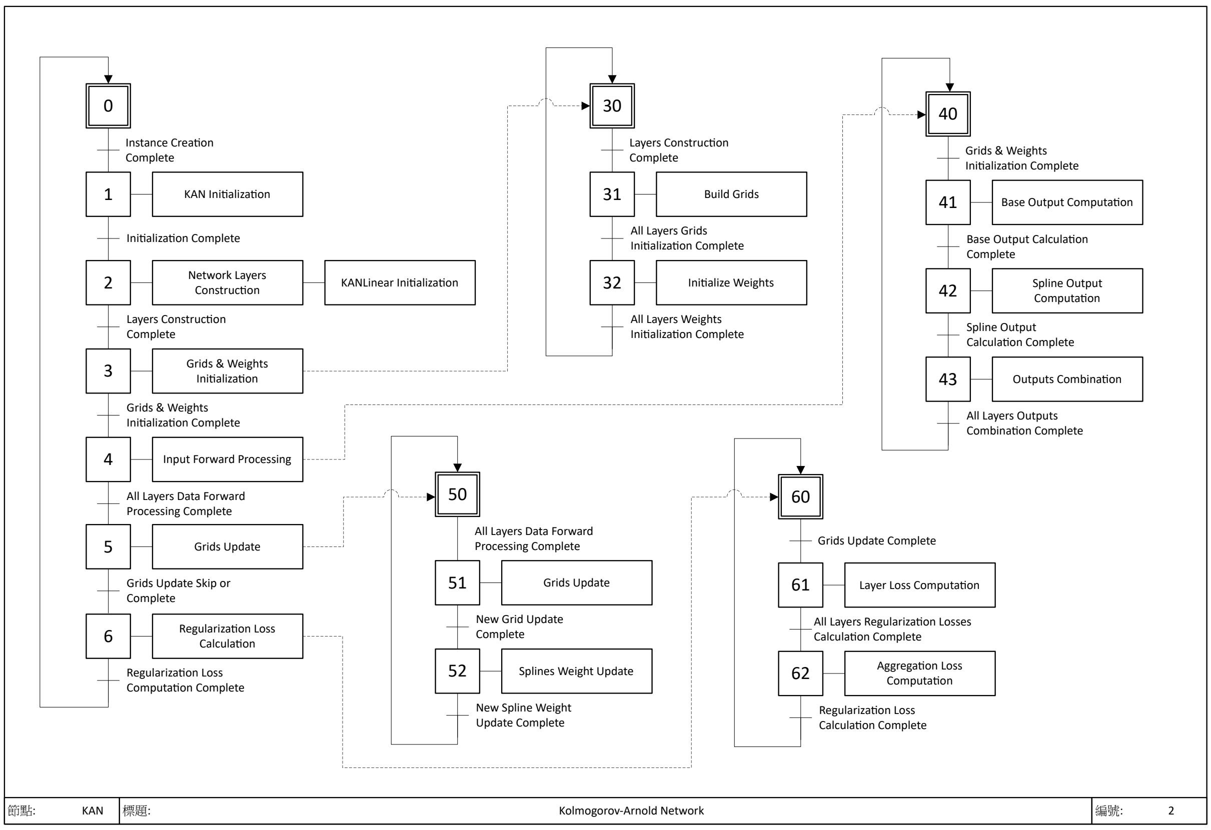
開放原始碼

- Code: https://github.com/KindXiaoming/pykan
- Reference: https://arxiv.org/abs/2404.19756

設計階層式架構 IDEF0


CONTROL
Layer Configuration, Grid Size, Spline Order, Scaling Parameters, Base Activation Function, Grid Update Parameters
Demand Input
Kolmogorov-Arnold Network
A0
Model Output, Training Loss
A1, A2, A3
MECHANISM
PyTorch, KANLinear Class
Demand
KAN Model Initialization
A1
Initialized KAN
Forward Propagation
A2
Trained Model
Regularization Loss Calculation
A3
Loss
A21, A22, A23, A24
KAN
KANLinear Forward Propagation
A21
Coeff
Grid Update
A22
Grid
KANLinear Initialization
A23
KAN Linear
KANLinear Parameters Reset
A24
MD
A211, A212
KAN
B-Splines Basis Functions Calculation
A211
Spline Coefficients Calculation
A212
Coeff

節點:	KAN	標題:	Kolmogorov-Arnold Network	編號:	1


0
Instance Creation Complete
1
KAN Initialization
Initialization Complete
2
Network Layers Construction
KANLinear Initialization
Layers Construction Complete
3
Grids & Weights Initialization
Grids & Weights Initialization Complete
4
Input Forward Processing
All Layers Data Forward Processing Complete
5
Grids Update
Grids Update Skip or Complete
6
Regularization Loss Calculation
Regularization Loss Computation Complete
30
Layers Construction Complete
31
Build Grids
All Layers Grids Initialization Complete
32
Initialize Weights
All Layers Weights Initialization Complete
40
Grids & Weights Initialization Complete
41
Base Output Computation
Base Output Calculation Complete
42
Spline Output Computation
Spline Output Calculation Complete
43
Outputs Combination
All Layers Outputs Combination Complete
50
All Layers Data Forward Processing Complete
51
Grids Update
New Grid Update Complete
52
Splines Weight Update
New Spline Weight Update Complete
60
Grids Update Complete
61
Layer Loss Computation
All Layers Regularization Losses Calculation Complete
62
Aggregation Loss Computation
Regularization Loss Calculation Complete
節點: KAN
標題: Kolmogorov-Arnold Network
編號: 2

Python 模擬驗證

- 重構後之 Kolmogorov-Arnold Network

```python
import math

import torch
import torch.nn.functional as F


class KANLinear(torch.nn.Module):
    def __init__(
            self,
            in_features,
            out_features,
            grid_size=5,
            spline_order=3,
            scale_base=1.0,
            scale_spline=1.0,
            enable_standalone_scale_spline=True,
            base_activation=torch.nn.SiLU,
            grid_eps=0.02,
            grid_range=[-1, 1],
    ):
        super(KANLinear, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.grid_size = grid_size
        self.spline_order = spline_order

        # 構建網格點
        self.grid = self.build_grid(grid_range, grid_size,
spline_order)

        # 初始化基礎權重和樣條權重
        self.base_weight, self.spline_weight,
self.spline_scaler = self.initialize_weights(
            out_features, in_features, grid_size,
spline_order, scale_base, scale_spline,
enable_standalone_scale_spline
        )

        self.scale_base = scale_base
        self.scale_spline = scale_spline
        self.enable_standalone_scale_spline =
enable_standalone_scale_spline
        self.base_activation = base_activation()
        self.grid_eps = grid_eps

    def build_grid(self, grid_range, grid_size,
spline_order):
        h = (grid_range[1] - grid_range[0]) / grid_size
        grid = (
            (
                torch.arange(-spline_order, grid_size +
spline_order + 1) * h
                + grid_range[0]
            )
            .expand(self.in_features, -1)
            .contiguous()
        )
        return grid

    def initialize_weights(self, out_features, in_features,
grid_size, spline_order, scale_base, scale_spline,
                           enable_standalone_scale_spline):
        base_weight =
torch.nn.Parameter(torch.Tensor(out_features, in_features))
        spline_weight = torch.nn.Parameter(
            torch.Tensor(out_features, in_features,
grid_size + spline_order)
        )
        if enable_standalone_scale_spline:
            spline_scaler = torch.nn.Parameter(
                torch.Tensor(out_features, in_features)
            )
        else:
            spline_scaler = None
        torch.nn.init.kaiming_uniform_(base_weight,
a=math.sqrt(5) * scale_base)
        torch.nn.init.kaiming_uniform_(spline_weight,
a=math.sqrt(5) * scale_spline)
        if enable_standalone_scale_spline:
            torch.nn.init.kaiming_uniform_(spline_scaler,
a=math.sqrt(5) * scale_spline)
        return base_weight, spline_weight, spline_scaler

    def b_splines(self, x: torch.Tensor):
        bases = self.calculate_b_spline_bases(x)
        return bases.contiguous()

    def calculate_b_spline_bases(self, x: torch.Tensor):
        grid: torch.Tensor = (
            self.grid
        )  # (in_features, grid_size + 2 * spline_order +
1)
```

```
        x = x.unsqueeze(-1)
        bases = ((x ⩾ grid[:, :-1]) & (x < grid[:,
1:])).to(x.dtype)
        for k in range(1, self.spline_order + 1):
            bases = (
                            (x - grid[:, : -(k + 1)])
                            / (grid[:, k:-1] - grid[:, : -
(k + 1)])
                            * bases[:, :, :-1]
                    ) + (
                            (grid[:, k + 1:] - x)
                            / (grid[:, k + 1:] - grid[:, 1:
(-k)])
                            * bases[:, :, 1:]
                    )
        return bases

    def curve2coeff(self, x: torch.Tensor, y:
torch.Tensor):
        A = self.b_splines(x).transpose(
            0, 1
        )  # (in_features, batch_size, grid_size +
spline_order)
        B = y.transpose(0, 1)  # (in_features, batch_size,
out_features)
        solution = torch.linalg.lstsq(
            A, B
        ).solution  # (in_features, grid_size +
spline_order, out_features)
        result = solution.permute(
            2, 0, 1
        )  # (out_features, in_features, grid_size +
spline_order)
        return result.contiguous()

    @property
    def scaled_spline_weight(self):
        if self.enable_standalone_scale_spline:
            return self.spline_weight *
```

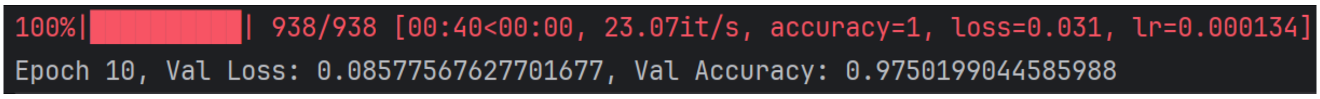
```
self.spline_scaler.unsqueeze(-1)
        else:
            return self.spline_weight

    def forward(self, x: torch.Tensor):
        base_output = self.compute_base_output(x)
        spline_output = self.compute_spline_output(x)
        return base_output + spline_output

    def compute_base_output(self, x: torch.Tensor):
        return F.linear(self.base_activation(x),
self.base_weight)

    def compute_spline_output(self, x: torch.Tensor):
        return F.linear(
            self.b_splines(x).view(x.size(0), -1),

self.scaled_spline_weight.view(self.out_features, -1),
        )

    @torch.no_grad()
    def update_grid(self, x: torch.Tensor, margin=0.01):
        batch = x.size(0)

        splines = self.b_splines(x)  # (batch, in, coeff)
        splines = splines.permute(1, 0, 2)  # (in, batch,
coeff)
        orig_coeff = self.scaled_spline_weight  # (out, in,
coeff)
        orig_coeff = orig_coeff.permute(1, 2, 0)  # (in,
coeff, out)
        unreduced_spline_output = torch.bmm(splines,
orig_coeff)  # (in, batch, out)
        unreduced_spline_output =
unreduced_spline_output.permute(
            1, 0, 2
        )  # (batch, in, out)

        x_sorted = torch.sort(x, dim=0)[0]
        grid_adaptive = x_sorted[
            torch.linspace(
                0, batch - 1, self.grid_size + 1,
dtype=torch.int64, device=x.device
            )
        ]

        uniform_step = (x_sorted[-1] - x_sorted[0] + 2 *
margin) / self.grid_size
        grid_uniform = (
                torch.arange(
                    self.grid_size + 1,
dtype=torch.float32, device=x.device
                ).unsqueeze(1)
                * uniform_step
                + x_sorted[0]
                - margin
        )

        grid = self.grid_eps * grid_uniform + (1 -
self.grid_eps) * grid_adaptive
        grid = torch.concatenate(
            [
                grid[:1]
                - uniform_step
                * torch.arange(self.spline_order, 0, -1,
device=x.device).unsqueeze(1),
                grid,
                grid[-1:]
                + uniform_step
                * torch.arange(1, self.spline_order + 1,
device=x.device).unsqueeze(1),
            ],
            dim=0,
        )

        self.grid.copy_(grid.T)
        self.spline_weight.data.copy_(self.curve2coeff(x,
unreduced_spline_output))
```

```
    def regularization_loss(self,
regularize_activation=1.0, regularize_entropy=1.0):
        l1_fake = self.spline_weight.abs().mean(-1)
        regularization_loss_activation = l1_fake.sum()
        p = l1_fake / regularization_loss_activation
        regularization_loss_entropy = -torch.sum(p *
p.log())
        return (
                regularize_activation *
regularization_loss_activation
                + regularize_entropy *
regularization_loss_entropy
        )


class KAN(torch.nn.Module):
    def __init__(
            self,
            layers_hidden,
            grid_size=5,
            spline_order=3,
            scale_base=1.0,
            scale_spline=1.0,
            base_activation=torch.nn.SiLU,
            grid_eps=0.02,
            grid_range=[-1, 1],
    ):
        super(KAN, self).__init__()
        self.grid_size = grid_size
        self.spline_order = spline_order

        # 構建 KAN 的層
        self.layers = self.build_layers(
            layers_hidden, grid_size, spline_order,
scale_base, scale_spline, base_activation, grid_eps,
grid_range
        )
    def build_layers(self, layers_hidden, grid_size,
spline_order, scale_base, scale_spline, base_activation,
grid_eps,
                     grid_range):
        layers = torch.nn.ModuleList()
        for in_features, out_features in zip(layers_hidden,
layers_hidden[1:]):
            layers.append(
                KANLinear(
                    in_features,
                    out_features,
                    grid_size=grid_size,
                    spline_order=spline_order,
                    scale_base=scale_base,
                    scale_spline=scale_spline,
                    base_activation=base_activation,
                    grid_eps=grid_eps,
                    grid_range=grid_range,
                )
            )
        return layers

    def forward(self, x: torch.Tensor, update_grid=False):
        for layer in self.layers:
            if update_grid:
                layer.update_grid(x)
            x = layer(x)
        return x

    def regularization_loss(self,
regularize_activation=1.0, regularize_entropy=1.0):
        return sum(

layer.regularization_loss(regularize_activation,
regularize_entropy)
            for layer in self.layers
        )
```

- KAN 測試（MNIST）

```python
from EfficientKAN import KAN

# Train on MNIST
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from tqdm import tqdm

# Load MNIST
transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.5,),
(0.5,))]
)
trainset = torchvision.datasets.MNIST(
    root="./data", train=True, download=True,
transform=transform
)
valset = torchvision.datasets.MNIST(
    root="./data", train=False, download=True,
transform=transform
)
trainloader = DataLoader(trainset, batch_size=64,
shuffle=True)
valloader = DataLoader(valset, batch_size=64,
shuffle=False)

# Define model
model = KAN([28 * 28, 64, 10])
device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
model.to(device)
# Define optimizer
optimizer = optim.AdamW(model.parameters(), lr=1e-3,
weight_decay=1e-4)
# Define learning rate scheduler
scheduler = optim.lr_scheduler.ExponentialLR(optimizer,
gamma=0.8)

# Define loss
criterion = nn.CrossEntropyLoss()
for epoch in range(10):
    # Train
    model.train()
    with tqdm(trainloader) as pbar:
        for i, (images, labels) in enumerate(pbar):
            images = images.view(-1, 28 * 28).to(device)
            optimizer.zero_grad()
            output = model(images)
            loss = criterion(output, labels.to(device))
            loss.backward()
            optimizer.step()
            accuracy = (output.argmax(dim=1) ==
labels.to(device)).float().mean()
            pbar.set_postfix(loss=loss.item(),
accuracy=accuracy.item(), lr=optimizer.param_groups[0]
['lr'])

    # Validation
    model.eval()
    val_loss = 0
    val_accuracy = 0
    with torch.no_grad():
        for images, labels in valloader:
            images = images.view(-1, 28 * 28).to(device)
            output = model(images)
            val_loss += criterion(output,
labels.to(device)).item()
            val_accuracy += (
                (output.argmax(dim=1) ==
labels.to(device)).float().mean().item()
            )
    val_loss /= len(valloader)
    val_accuracy /= len(valloader)

    # Update learning rate
    scheduler.step()

    print(
        f"Epoch {epoch + 1}, Val Loss: {val_loss}, Val
Accuracy: {val_accuracy}"
    )

# Print model weights
print("Trained Model Weights:")

for i, layer in enumerate(model.layers):
    print(f"Layer {i + 1}:")
    print("Spline Weights:")
    print(layer.spline_weight)
    print("Base Weights:")
    print(layer.base_weight)
    print()

# Save model weights (need to create KAN instance then
"torch.load")
torch.save(model.state_dict(), "kan_mnist_weights.pth")
# Save the entire model (just get with "torch.load")
torch.save(model, "kan_mnist_model.pth")
```

- Model 訓練結果（MNIST）



```
100%|██████████| 938/938 [00:40<00:00, 23.07it/s, accuracy=1, loss=0.031, lr=0.000134]
Epoch 10, Val Loss: 0.08577567627701677, Val Accuracy: 0.9750199044585988
```

有達到預期的準確度，單輪訓練也有訓練出 `Accuracy = 1` 的終極情況，總的來說代表調整後的代碼是正確工作的

- KAN 測試（x * y）

```
import torch
import torch.nn as nn
from tqdm import tqdm

from EfficientKAN import KAN
```

```
def test_mul():
    kan = KAN([2, 3, 3, 1], base_activation=nn.Identity)
    optimizer = torch.optim.LBFGS(kan.parameters(),
lr=0.001)

    with tqdm(range(200)) as pbar:
        for i in pbar:
            loss, reg_loss = None, None

            def closure():
                optimizer.zero_grad()
                x = torch.rand(1024, 2)
                y = kan(x, update_grid=(i % 20 == 0))
                assert y.shape == (1024, 1)
                nonlocal loss, reg_loss
                u = x[:, 0]
                v = x[:, 1]
                loss =
nn.functional.mse_loss(y.squeeze(-1), u * v)
                reg_loss = kan.regularization_loss(1, 0)
                (loss + 1e-5 * reg_loss).backward()
                return loss + reg_loss

            optimizer.step(closure)
            pbar.set_postfix(mse_loss=loss.item(),
reg_loss=reg_loss.item())

    for layer in kan.layers:
        print(layer.spline_weight)

    torch.save(kan, 'model/kan_multiple_model.pth')
    torch.save(kan.state_dict(),
"model/kan_multiple_weights.pth")

    # Test the trained model
    test_model(kan)
```

```
def test_model(model):
    model.eval()
    with torch.no_grad():
        test_x = torch.rand(1024, 2)
        test_y = model(test_x)
        u = test_x[:, 0]
        v = test_x[:, 1]
        expected_y = u * v
        test_loss =
nn.functional.mse_loss(test_y.squeeze(-1), expected_y)
        print(f"Test Loss: {test_loss.item():.4f}")


test_mul()
```

- Model 訓練結果 (x * y)

```
100%|██████████| 200/200 [01:03<00:00,  3.13it/s, mse_loss=0.0938, reg_loss=9.12e-7]
```

方法論合成 Grafcet 控制器電路

- 模型量化與權重輸出 (輸出符合 Quartus 的 Weights，並動態進行 Scaling 以避免 Quantize 完變成零)

```
import torch
import numpy as np

# Load the trained model
model = torch.load('model/kan_multiple_model.pth')
model.eval()

# Quantize the weights to fixed-point format with dynamic
scaling
def quantize_weight(weight, scale):
    return (weight * scale).round().int().numpy()
```

```
# Convert to 16-bit signed hex
def to_hex_str(arr):
    return [format(x & 0xFFFF, '04X') for x in arr]

# Function to find dynamic scale
def find_dynamic_scale(weight, target_range=32767):
    max_val = torch.max(torch.abs(weight)).item()
    if max_val == 0:
        return 1
    return target_range / max_val

# Extract and quantize weights
layer_base_weights = []
layer_spline_weights = []
for i, layer in enumerate(model.layers):
    # Print statistics of weights before quantization
    print(f"Layer {i} base_weight min:
{layer.base_weight.min()}, max: {layer.base_weight.max()}")
    print(f"Layer {i} spline_weight min:
{layer.spline_weight.min()}, max:
{layer.spline_weight.max()}")

    # Determine dynamic scales
    base_scale = find_dynamic_scale(layer.base_weight)
    spline_scale = find_dynamic_scale(layer.spline_weight)

    print(f"Layer {i} base_scale: {base_scale}")
    print(f"Layer {i} spline_scale: {spline_scale}")

    # Quantize weights with dynamic scales
    base_weight = quantize_weight(layer.base_weight,
scale=base_scale).flatten()
    spline_weight = quantize_weight(layer.spline_weight,
scale=spline_scale).flatten()

    # Check if spline weights are being quantized to zero
    if np.all(spline_weight == 0):
        print(f"Warning: All spline weights in layer {i}
are quantized to zero.")
```

```
    # Save weights to files
    with open(f'weight2/base_weight_layer_{i}.txt', 'w') as
f:
        f.write('\n'.join(to_hex_str(base_weight)))
    with open(f'weight2/spline_weight_layer_{i}.txt', 'w')
as f:
        f.write('\n'.join(to_hex_str(spline_weight)))
```

⚠ Warning

我 Quartus Prime 是選擇用實驗室的板子來進行預設的 FPGA 規格，下列的設計是可以用的（經過其他的驗證程式確認），但因為使用了大量的 PIN 腳，而 MAX10 預設的 PIN 腳不夠用，所以沒有辦法編譯成 MAX10 可用的電路規格，可能會需要優化電路設計，或是更換為比較簡易的測試資料（第一輪採用 MNIST Dataset 進行測試，且 KAN 的架構為 [28 * 28, 64, 10]）。

- 方法論重構後之 KANLayer 實現（KANLayer.v）

```
module KANLayer #(
    parameter IN_FEATURES = 784,
    parameter OUT_FEATURES = 64,
    parameter SCALE = 256,   // Quantization scale factor
    parameter BASE_WEIGHT_FILE =
"C:/intelFPGA_lite/18.1/KAN/base_weight_layer_0.txt",
    parameter SPLINE_WEIGHT_FILE =
"C:/intelFPGA_lite/18.1/KAN/spline_weight_layer_0.txt"
)(
    input wire clk,
    input wire reset,
    input wire [7:0] in_data [0:IN_FEATURES-1],  // Input
data
    output reg [7:0] out_data [0:OUT_FEATURES-1] // Output
data
);
```

```
    // Weights stored in on-chip memory (BRAM)
    reg signed [15:0] base_weights
[0:OUT_FEATURES*IN_FEATURES-1];
    reg signed [15:0] spline_weights
[0:OUT_FEATURES*IN_FEATURES-1];

    // Load weights from memory (initialization)
    initial begin
        $readmemh(BASE_WEIGHT_FILE, base_weights);
        $readmemh(SPLINE_WEIGHT_FILE, spline_weights);
    end

    // Output registers
    reg signed [31:0] base_output [0:OUT_FEATURES-1];
    reg signed [31:0] spline_output [0:OUT_FEATURES-1];
    reg signed [31:0] total_output [0:OUT_FEATURES-1];

    integer i, j;

    // Forward pass
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
                base_output[i] <= 0;
                spline_output[i] <= 0;
                total_output[i] <= 0;
            end
        end else begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
                base_output[i] <= 0;
                spline_output[i] <= 0;
                for (j = 0; j < IN_FEATURES; j = j + 1)
begin
                    base_output[i] <= base_output[i] +
in_data[j] * base_weights[i*IN_FEATURES + j];
                    spline_output[i] <= spline_output[i] +
in_data[j] * spline_weights[i*IN_FEATURES + j];
                end
                total_output[i] <= (base_output[i] +
spline_output[i]) / SCALE;  // Combine and scale the
outputs
                out_data[i] <= total_output[i][15:8];  //
Convert to 8-bit output
            end
        end
    end
endmodule
```

Success

更換測試資料以求可以將電路設計容納進去（學習 `mulitplication`：`x * y`，`KAN` 的架構為 `[2, 3, 3, 1]`）

- 方法論重構後之 `KANLayer` 實現（`KANLayer.v`）

```
module KANLinear #(parameter IN_FEATURES = 2, OUT_FEATURES
= 3, integer LAYER_NUM = 0) (
    input clk,
    input reset,
    input [15:0] data_in [IN_FEATURES-1:0],
    output reg [15:0] data_out [OUT_FEATURES-1:0]
);

    reg [15:0] base_weight [OUT_FEATURES*IN_FEATURES-1:0];
    reg [15:0] spline_weight [OUT_FEATURES*IN_FEATURES-
1:0];
    reg [15:0] base_weight_2d [OUT_FEATURES-1:0]
[IN_FEATURES-1:0];
    reg [15:0] spline_weight_2d [OUT_FEATURES-1:0]
[IN_FEATURES-1:0];

    integer i, j;

    initial begin
        if (LAYER_NUM == 0) begin
```

```
$readmemh("C:/intelFPGA_lite/18.1/KAN2/base_weight_layer_0.
txt", base_weight);

$readmemh("C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_
0.txt", spline_weight);
        end else if (LAYER_NUM == 1) begin

$readmemh("C:/intelFPGA_lite/18.1/KAN2/base_weight_layer_1.
txt", base_weight);

$readmemh("C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_
1.txt", spline_weight);
        end else if (LAYER_NUM == 2) begin

$readmemh("C:/intelFPGA_lite/18.1/KAN2/base_weight_layer_2.
txt", base_weight);

$readmemh("C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_
2.txt", spline_weight);
        end

        for (i = 0; i < OUT_FEATURES; i = i + 1) begin
            for (j = 0; j < IN_FEATURES; j = j + 1) begin
                base_weight_2d[i][j] = base_weight[i *
IN_FEATURES + j];
                spline_weight_2d[i][j] = spline_weight[i *
IN_FEATURES + j];
            end
        end
    end

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
                data_out[i] <= 16'd0;
            end
        end else begin
            for (i = 0; i < OUT_FEATURES; i = i + 1) begin
```

```
                    data_out[i] <= 16'd0;
                    for (j = 0; j < IN_FEATURES; j = j + 1)
begin
                        data_out[i] <= data_out[i] +
base_weight_2d[i][j] * data_in[j];
                    end
                    if (data_out[i] < 16'd0) begin
                        data_out[i] <= 16'd0;
                    end
                end
            end
        end

endmodule
```

ChatGPT 合成 Grafcet Datapath 電路

Warning

我 Quartus Prime 是選擇用實驗室的板子來進行預設的 FPGA 規格，下列的設計是可以用的（經過其他的驗證程式確認），但因為使用了大量的 PIN 腳，而 MAX10 預設的 PIN 腳不夠用，所以沒有辦法編譯成 MAX10 可用的電路規格，可能會需要優化電路設計，或是更換為比較簡易的測試資料（第一輪採用 MNIST Dataset 進行測試，且 KAN 的架構為 [28 * 28, 64, 10]）。

- 方法論重構後之完整 Network 實現（KAN.v）

```
module KAN (
    input wire clk,
    input wire reset,
    input wire [7:0] in_data [0:783],   // 28x28 = 784
pixels, 8-bit each
```

```verilog
    output wire [7:0] out_data [0:9]   // 10 classes, 8-bit
each
);
    // Internal signals for each layer
    wire [7:0] layer1_out [0:63];
    wire [7:0] layer2_out [0:9];

    // Instantiate layers
    KANLayer #(
        .IN_FEATURES(784),
        .OUT_FEATURES(64),

.BASE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/base_weight_l
ayer_0.txt"),

.SPLINE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/spline_weig
ht_layer_0.txt")
    ) layer1 (
        .clk(clk),
        .reset(reset),
        .in_data(in_data),
        .out_data(layer1_out)
    );

    KANLayer #(
        .IN_FEATURES(64),
        .OUT_FEATURES(10),

.BASE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/base_weight_l
ayer_1.txt"),

.SPLINE_WEIGHT_FILE("C:/intelFPGA_lite/18.1/KAN/spline_weig
ht_layer_1.txt")
    ) layer2 (
        .clk(clk),
        .reset(reset),
        .in_data(layer1_out),
        .out_data(layer2_out)
    );
    // Connect the final output
    assign out_data = layer2_out;
endmodule
```

✅ Success

更換測試資料以求可以將電路設計容納進去（學習 mulitplication：x * y，KAN 的架構為 [2, 3, 3, 1]）

- 方法論重構後之完整 Network 實現（KAN.v）

```
module KAN #(parameter IN_FEATURES = 2, L1_FEATURES = 3,
L2_FEATURES = 3, OUT_FEATURES = 1) (
    input clk,
    input reset,
    input [15:0] input_data [IN_FEATURES-1:0],
    output [15:0] output_data
);

    wire [15:0] layer1_out [L1_FEATURES-1:0];
    wire [15:0] layer2_out [L2_FEATURES-1:0];
    wire [15:0] layer3_out [OUT_FEATURES-1:0];

    KANLinear #(.IN_FEATURES(IN_FEATURES),
.OUT_FEATURES(L1_FEATURES), .LAYER_NUM(0)) layer1 (
        .clk(clk),
        .reset(reset),
        .data_in(input_data),
        .data_out(layer1_out)
    );

    KANLinear #(.IN_FEATURES(L1_FEATURES),
.OUT_FEATURES(L2_FEATURES), .LAYER_NUM(1)) layer2 (
        .clk(clk),
        .reset(reset),
        .data_in(layer1_out),
        .data_out(layer2_out)
    );

    KANLinear #(.IN_FEATURES(L2_FEATURES),
.OUT_FEATURES(OUT_FEATURES), .LAYER_NUM(2)) layer3 (
        .clk(clk),
        .reset(reset),
        .data_in(layer2_out),
        .data_out(layer3_out)
    );

    assign output_data = layer3_out[0];

endmodule
```

FPGA 整合驗證

Success

更換測試資料以求可以將電路設計容納進去（學習 mulitplication：x * y，KAN 的架構為 [2, 3, 3, 1]）

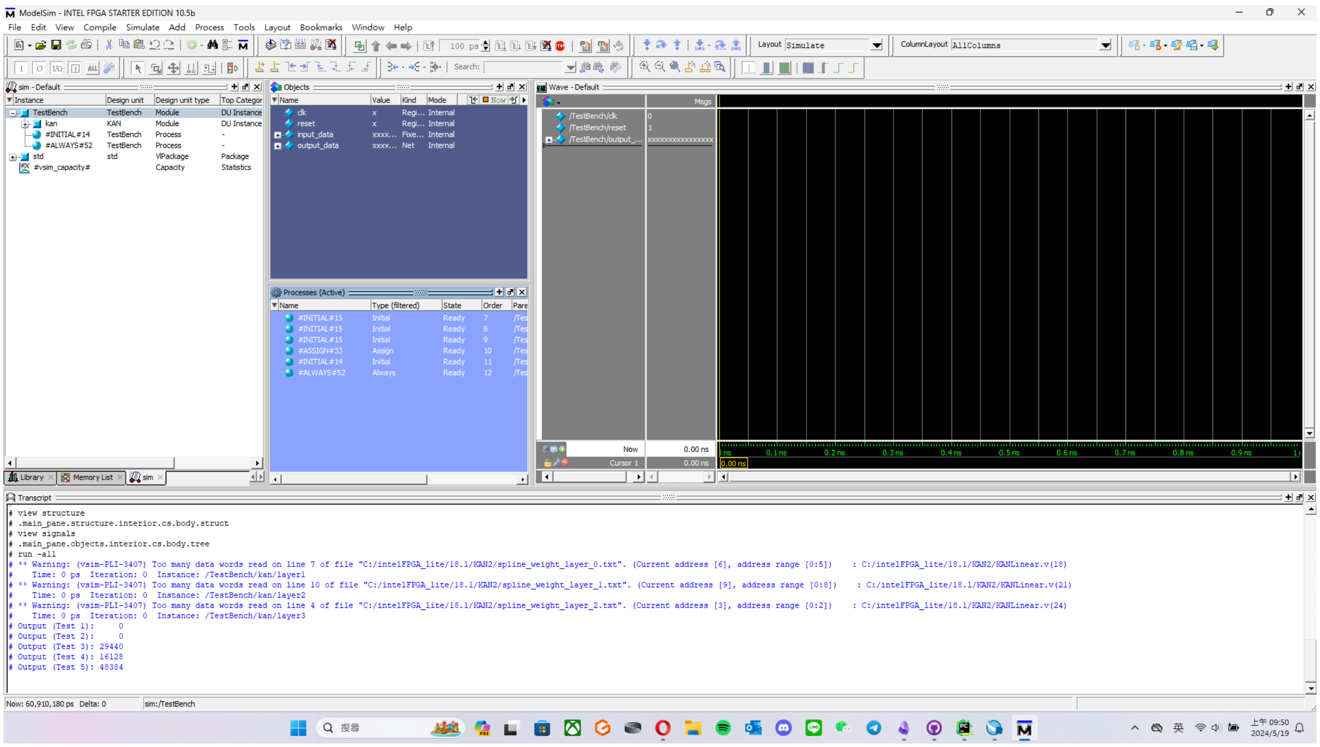
- Testbench 實現（Testbench.v）

```
module TestBench;
    reg clk;
    reg reset;
    reg [15:0] input_data [0:1];
    wire [15:0] output_data;

    KAN #(.IN_FEATURES(2), .L1_FEATURES(3),
.L2_FEATURES(3), .OUT_FEATURES(1)) kan (
        .clk(clk),
        .reset(reset),
        .input_data(input_data),
        .output_data(output_data)
    );

    initial begin
        clk = 0;
        reset = 1;
        input_data[0] = 16'd0;
        input_data[1] = 16'd0;
        #10 reset = 0;

        // test data 1
        input_data[0] = 16'd50;
        input_data[1] = 16'd30;
        #10;
        $display("Output (Test 1): %d", output_data);

        // test data 2
        input_data[0] = 16'd100;
        input_data[1] = 16'd200;
        #10;
        $display("Output (Test 2): %d", output_data);

        // test data 3
        input_data[0] = 16'd150;
        input_data[1] = 16'd250;
        #10;
        $display("Output (Test 3): %d", output_data);

        // test data 4
        input_data[0] = 16'd75;
        input_data[1] = 16'd125;
        #10;
        $display("Output (Test 4): %d", output_data);

        // test data 5
        input_data[0] = 16'd175;
        input_data[1] = 16'd225;
        #10;
        $display("Output (Test 5): %d", output_data);
    end

    always #5 clk = ~clk;

endmodule
```

⚠ Warning

硬體設計完仿真的結果如下，誤差老實說非常大，進行排查之後確認是權重的問題，因為 PyTorch 讀取 Quantize 後的權重結果一樣糟糕，可能會需要考慮導入可以計算小數的硬體來解決問題。

- ModelSim 仿真結果



```
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Warning: (vsim-PLI-3407) Too many data words read on line 7 of file "C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_0.txt". (Current address [6], address range [0:5])    : C:/intelFPGA_lite/18.1/KAN2/KANLinear.v(18)
#    Time: 0 ps  Iteration: 0  Instance: /TestBench/kan/layer1
# ** Warning: (vsim-PLI-3407) Too many data words read on line 10 of file "C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_1.txt". (Current address [9], address range [0:8])    : C:/intelFPGA_lite/18.1/KAN2/KANLinear.v(21)
#    Time: 0 ps  Iteration: 0  Instance: /TestBench/kan/layer2
# ** Warning: (vsim-PLI-3407) Too many data words read on line 4 of file "C:/intelFPGA_lite/18.1/KAN2/spline_weight_layer_2.txt". (Current address [3], address range [0:2])    : C:/intelFPGA_lite/18.1/KAN2/KANLinear.v(24)
#    Time: 0 ps  Iteration: 0  Instance: /TestBench/kan/layer3
# Output (Test 1):     0
# Output (Test 2):     0
# Output (Test 3): 29440
# Output (Test 4): 16128
# Output (Test 5): 48384
```

Now: 60,910,180 ps Delta: 0   sim:/TestBench