PyKan 库目前的几个版本都仍然存在一些问题，尤其是 cuda 支持部分。当前教程针对 0.2.2 版本进行调整以修改问题，后续官方更新后可能可以安装后直接使用。

基本安装

虚拟环境
conda create -n #环境名称# python=3.10.14
官方要求 python 版本高于 3.9.7

安装库
pip install pykan==0.2.2
官方宣称的最新版本是 0.2.1，但库中有 bug 需要 0.2.2 甚至更高版本弥补。
在安装 0.2.2 版本后，将 Anaconda3\envs\kan\Lib\site-packages\kan 路径中的 MultKAN.py 文件内容替换为 github 上同名文件中的内容
https://github.com/KindXiaoming/pykan/blob/master/kan/MultKAN.py
在此版本下 0.2.2 才能顺利运行，同时也能够支持 CUDA。

如果想运行 0.2.1 版本，需要将 KANLayer.py 文件替换为 0.2.2 中的版本，并将 Anaconda3\envs\kan2\Lib\site-packages\kan 路径中的 assets 文件夹加入 Anaconda3\envs\kan\Lib\site-packages\kan 路径中，其中包含两张用于绘图的加法和乘法的图片。但 0.2.1 对 cuda 的支持依然存在问题。
Ps. envs\kan 和 envs\kan2 代表装有 0.2.1 和 0.2.2 pykan 版本的虚拟环境

安装依赖的库 requirements.txt
matplotlib==3.6.2
numpy==1.24.4
scikit_learn==1.1.3
setuptools==65.5.0
sympy==1.11.1
torch==2.2.2
tqdm==4.66.2
pyyaml
pandas
后两个是官方没有提到，但是实际库中有依赖的
pip install -r requirements.txt

安装 ipykernel
pip install ipykernel
pip install -n #环境名称# ipykernel
python -m ipykernel install --user --name #环境名称# --display-name “#显示名称#”

使用

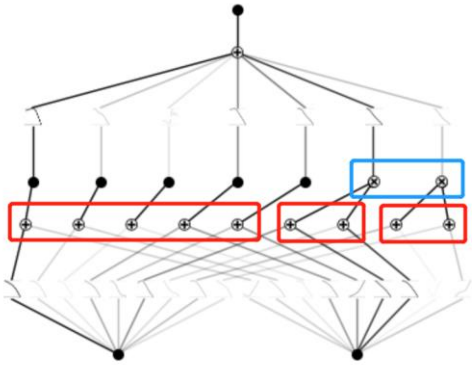
初始化

from kan import *
torch.set_default_dtype(torch.float64)

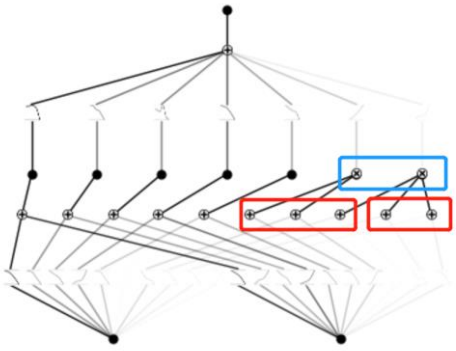
创建模型

创建基础 KAN 网络
model = KAN(width=[2,5,1])

创建包含乘法节点的 KAN 网络（MultKAN），两种网络除创建外其他的操作都相同
model = KAN(width=[2, [5,2], 1], mult_arity=2)
第一次有 2 个加法节点（小于等于输入变量数），第二层有五个加法节点，每个乘法节点则是“mult_arity”个额外加法节点的乘积



改变每个乘法节点下的加法节点
model = KAN(width=[2, [5,2], 1], mult_arity=[[], [2,3], []])
mult_arity 的每个[]对应的是每层的乘法节点，而[2,3]代表第二层的两个乘法节点分别的加法节点数



其他的初始化参数如下

```
class MultKAN(torch.nn.modules.module.Module)
    MultKAN(width=None, grid=3, k=3, mult_arity=2, noise_scale=1.0, scale_base_mu=0.0, scale_base_sigma=1.0, base_fun='silu', symbolic_enabled=True, affine_trainable=False, grid_eps=1.0, grid_range=[-1, 1], sp_trainable=True, sb_trainable=True, device='cpu', seed=0, save_plot_data=True, sparse_init=False, auto_save=False, first_init=True, ckpt_path='./model', state_id=0)
```

创建数据集

```
f = lambda x: x[:,0] * x[:,1]
dataset = create_dataset(f, n_var=2)
```
为模型加载数据集(可选,在未训练时 plot 模型需要先加载数据集,直接训练则不需要)
```
model.get_act(dataset)
```
画出如上模型结构及函数图像
```
model.plot()
```

训练模型
```
loss = model.fit(dataset, steps=n_step)
```
loss 中存储着 loss 的变化，可以绘图观察
```
steps = np.arange(1, n_step+1)
train_loss = np.array(loss['train_loss'])
test_loss = np.array(loss['test_loss'])
plt.plot(steps, train_loss, label="train")
plt.plot(steps, test_loss, color='orange', label="test")
plt.legend()
```

fit 函数其他参数如下

```
fit(self, dataset, opt='LBFGS', steps=100, log=1, lamb=0.0, lamb_l1=1.0, lamb_entropy=2.0, lamb_coef=0.0, lamb_coefdiff=0.0, update_grid=True, grid_update_num=10, loss_fn=None, lr=1.0, start_grid_update_step=-1, stop_grid_update_step=50, batch=-1, small_mag_threshold=1e-16, small_reg_factor=1.0, metrics=None, save_fig=False, in_vars=None, out_vars=None, beta=3, save_fig_freq=1, img_folder='./video', device='cpu', singularity_avoiding=False, y_th=1000.0, reg_metric='fa')
```

其中 opt="Adam"可以有效提高训练速度，但拟合效果有待商榷；lamb_l1, lamb_entropy 为正则化中的两项系数；lamb, lamb_coef, lamb_coefdiff 中有一个为正则化整体系数，其余两个未知。

模型在训练后会更替为新的版本，所有模型的版本和版本结构图会保存在 ipynb 文件同路径下的 model 和 figures 文件夹内

裁剪模型
裁剪对计算结果影响低于某一阈值的边、节点或输入变量（无关变量），默认为 0.01
```
model.prune(threshold=0.01) 裁剪边和节点
model.prune_node(threshold=0.01) 只裁剪节点
model.prune_input(threshold=0.01) 只裁剪输入，见后文
```

模型符号化

Functionality	Descriptions
`model.train(dataset)`	training model on dataset
`model.plot()`	plotting
`model.prune()`	pruning
`model.fix_symbolic(l,i,j,fun)`	fix the activation function $\phi_{l,i,j}$ to be the symbolic function `fun`
`model.suggest_symbolic(l,i,j)`	suggest symbolic functions that match the numerical value of $\phi_{l,i,j}$
`model.auto_symbolic()`	use top 1 symbolic suggestions from `suggest_symbolic` to replace all activation functions
`model.symbolic_formula()`	return the symbolic formula

Table 7: KAN functionalities

一般使用 model.auto_symbolic()来符号化，在符号化后继续训练将微调确定函数的参数。

输出符号表达式
sf = model.symbolic_formula()[0][0]
nsimplify(ex_round(ex_round(sf, 3),3))

额外功能

用公式初始化模型 KAN Compiler
通过 sympy 库创建变量和表达式，再由表达式加载模型。
from sympy import *
input_variables = x,y = symbols('x y')
expr = exp(sin(pi*x)+y**2)
model = kanpiler(input_variables, expr)
训练时默认从微调符号函数参数开始。深入训练函数类型（接触固定符号函数）
model.perturb(mode="minimal") 只训练仍然有效的函数
model.perturb(mode="all") 训练所有的函数（完整的模型）
在文中的安装版本里，kanpiler 名称并未被使用，而是使用 sf2kan

进一步扩展网络的方式
在末尾加一层有一个节点的线性函数
model.expand_depth()
在第 i 层增加 j 个加法节点
model.expand_depth(i, j)
在第 i 层增加 j 个乘法节点，mult_arity 分别为 p1, ⋯, pj
model.expand_width(i, j, sum_bool=False, mult_arity=[p1, ⋯, pj])
最后用 model.perturb(mode="all")激活训练

输入变量裁剪
各输入变量对结果的影响权重
model.feature_score

在模型结构图中标出各变量名称
input_vars = [r'$x_'+str(i)+'$' for i in range(4)]
model.plot(in_vars=input_vars)
裁剪
model = model.prune_input(threshold=0.01)
获取仍有效的变量 id
model.input_id
手动裁剪
model = model.prune_input(active_inputs=[0,1,2,3,4])

GPU 支持训练模型

安装 CUDA Toolkit 与 torch 相关库
检查电脑 NVIDIA 所支持的最高 CUDA 版本，在终端输入 nvidia-smi，在 CUDA Version 中；如果已安装 CUDA Toolkit，可以通过 nvcc --version 得到当前安装的版本
谷歌搜索 cuda toolkit #版本号# download，根据 win10/win11 选择本地安装包，跟随安装引导完成安装。

在 PyTorch 官网找到对应 CUDA Toolkit 版本的 torchvision 和 torchaudio 版本
https://pytorch.org/get-started/locally/
更早版本：https://pytorch.org/get-started/previous-versions/
安装对应版本后，在 jupyter notebook 中 import torch; torch.cuda.is_available()应当为 True

使用 CUDA 训练模型
需要对数据集和模型加载有一些小调整，默认为支持 CPU，有的方法仅支持 CPU。
cuda = torch.device("cuda")
cpu = torch.device("cpu")
支持 cuda 的数据集
dataset = create_dataset(f, n_var=2, device=cuda)
如下设置环境变量，否则会报错
os.environ["CUBLAS_WORKSPACE_CONFIG"]=":4096:8"
加载支持 cuda 的模型，在创建模型后将模式转为 cuda
model = KAN(width=[2,[5,2],1], mult_arity=2).to(cuda)
训练 cuda 支持的模型，**需要把 update_grid 设置为 False**
loss = model.fit(dataset, steps=n_step, lamb=0.01, **update_grid=False**)

prune 和 symbolic 相关方法仅支持 CPU，需要先将模型转为 CPU，再用 CPU 数据集训练一步后才能使用
支持 cpu 的数据集：dataset_cpu = create_dataset(f, n_var=2)
model.to(cpu)
model.get_act(dataset_cpu)

model.prune()/model.auto_symbolic()
需要转回 CUDA 支持则再使用 model.to(cuda)并用 CUDA 支持的数据集进行训练

参考

论文
https://arxiv.org/html/2404.19756v1
PyKan GitHub
https://github.com/KindXiaoming/pykan
Anaconda, Jupyter 和 ipykernel
https://blog.csdn.net/qq_44184049/article/details/121911689?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216975080661680018418676 9%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=169750806616800184186769&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-2-121911689-null-null.142^v96^pc_search_result_base4&utm_term=anaconda%E6%96%B0%E8%99%9A%E6%8B%9F%E7%8E%AF%E5%A2%83%E8%A3%85jupyter&spm=1018.2226.3001.4187
CUDA Toolkit
https://www.geeksforgeeks.org/how-to-set-up-and-run-cuda-operations-in-pytorch/
https://blog.csdn.net/qq_46390120/article/details/123582479?ops_request_misc=&request_id=&biz_id=102&utm_term=python%20cuda%E7%8E%AF%E5%A2%83%E9%85%8D%E7%BD%AE&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-123582479.142^v100^pc_search_result_base9&spm=1018.2226.3001.4187
sympy
https://scipy-lectures.org/packages/sympy.html
https://docs.sympy.org/latest/tutorials/intro-tutorial/basic_operations.html