**实验环境：** Linux系统。

**结果与分析：** **实验代码**

**空间复杂度**

import Mathlib.Data.Real.Basic

-- 定义输入层、隐藏层和输出层的计算结果存储需求

def storage\_computation\_results (n : ℕ) : ℕ :=

  n + n \* (2 \* n + 1) + (2 \* n + 1) + (2 \* n + 1) + 1

-- 定义函数参数存储需求

def storage\_function\_parameters (n k m : ℕ) : ℕ :=

  n \* (2 \* n + 1) \* k + (2 \* n + 1) \* m

-- 定义总空间复杂度

def total\_space\_complexity (n k m : ℕ) : ℕ :=

  storage\_computation\_results n + storage\_function\_parameters n k m

-- 定义大O表示法的空间复杂度

def big\_O\_space\_complexity (n k m : ℕ) : ℕ :=

  n^2 \* k + n \* m

-- 示例：计算给定 n, k, m 的总空间复杂度

def example\_1 : ℕ :=

  total\_space\_complexity 3 2 4

-- 示例：计算给定 n, k, m 的大O表示法的空间复杂度

def example\_big\_O : ℕ :=

  big\_O\_space\_complexity 3 2 4

#eval example\_1

#eval example\_big\_O

**时间复杂度**

-- 定义输入层的时间复杂度

def input\_complexity (n : Nat) : Nat :=

  n

-- 定义隐藏层的时间复杂度

def hidden\_complexity (n : Nat) : Nat :=

  (2 \* n + 1) \* n

-- 定义输出层的时间复杂度

def output\_complexity (n : Nat) : Nat :=

  2 \* n + 1

-- 定义总时间复杂度

def total\_time\_complexity (n : Nat) : Nat :=

  input\_complexity n + hidden\_complexity n + output\_complexity n

-- 定义大O表示法的时间复杂度

def big\_O\_time\_complexity (n : Nat) : Nat :=

  n^2

-- 示例：计算给定 n 的总时间复杂度

def example\_1 : Nat :=

  total\_time\_complexity 3

-- 示例：计算给定 n 的大O表示法的时间复杂度

def example\_big\_O : Nat :=

  big\_O\_time\_complexity 3

#eval example\_1

#eval example\_big\_O

实验结果表明，随着输入数据规模的增加，KAN 网络的时间复杂度呈现二次增长，与 Lean4 代码计算的结果一致。同时，空间复杂度的变化也符合预测，表明本文的方法能够准确反映 KAN 网络的存储需求。

文本

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成