МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)
ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ (ШКОЛА)
Департамент математического и компьютерного моделирования

Домашев Сергей Антонович

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

вид ВКР

Применение нейронной сети архитектуры Kolmagorov Arnold Networks для решения задач классификации и регрессии

по направлению подготовки 01.03.02 Прикладная математика и информатика
профиль «Математические и компьютерные технологии»

Владивосток
2025

В материалах данной выпускной квалификационной работы не содержатся сведения, составляющие государственную тайну, и сведения, подлежащие экспортному контролю

Уполномоченный по экспортному контролю

И.Л. Артемьева

подпись *и. о. фамилия*

«_____» _____ 20____ г.

Автор работы _____,

подпись

группа Б9121-01.03.02мкт

«_____» _____ 20___г.

Руководитель ВКР

должность, ученое звание

К.С. Кузнецов

подпись *и. о. фамилия*

«_____» _____20___г.

Нормоконтролер

М.В. Косенок

подпись *и. о. фамилия*

«_____» _____20___г.

Назначен рецензент* _____

ученое звание

Петров Дмитрий Владимирович

фамилия, имя, отчество

Защищена в ГЭК с оценкой

_____

Секретарь ГЭК

И.В. Жандармова

подпись *и. о. фамилия*

«_____» _____ 20____г.

«Допустить к защите»

Зав. кафедрой/ директор Департамента

_____

ученая степень, ученое звание

А.А. Сущенко

подпись *и. о. фамилия*

«_____» _____ 20____ г.

АННОТАЦИЯ

В рамках дипломной работы проводилось комплексное исследование новой архитектуры нейронных сетей на базе математической теоремы Колмагорова Арнольда. Исследование включает в себя изучение документации, научных статей и связанных модификаций, а также реализация архитектуры на различных задачах для анализа функциональности архитектуры и библиотеки. Особое внимание в ходе выполнения дипломной работы уделялось математической интерпретации, способности к аппроксимации, интерпретируемости, устойчивости к переобучению, а также применимости в задачах с ограниченными данными.

ABSTRACT

As part of the thesis, a comprehensive study of a new neural network architecture based on the Kolmogorov-Arnold theorem was conducted. The study includes the examination of documentation, scientific articles, and related modifications, as well as the implementation of the architecture on various tasks to analyze its functionality and library. During the thesis, special attention was paid to the mathematical interpretation, approximation capabilities, interpretability, resistance to overfitting, and applicability in tasks with limited data.

ОГЛАВЛЕНИЕ

ТЕРМИНЫ И СОКРАЩЕНИЯ

- **KAN** – Kolmagorov Arnold Networks
- **MLP** – Multy Layer Perceptron
- **Нейронная сеть** (Neural Network, NN) — математическая модель, вдохновленная работой мозга, состоящая из большого числа взаимосвязанных "нейронов", которые обрабатывают данные.
- **Градиентный спуск** (Gradient Descent) — метод оптимизации, который позволяет находить минимум функции потерь, постепенно корректируя параметры модели.
- **Переобучение** (Overfitting) — ситуация, когда модель слишком точно запоминает обучающие данные и плохо работает на новых данных.
- **Входной слой** (Input Layer) — принимает исходные данные.
- **Скрытые слои** (Hidden Layers) — внутренние слои, в которых происходит обработка информации.
- **Выходной слой** (Output Layer) — выдает финальный результат.
- **Нейрон** (перцептрон, узел) — базовый элемент сети, который получает вход, обрабатывает его и передает дальше.
- **Вес** (Weight) — коэффициент, который определяет важность входного сигнала для нейрона.
- **Смещение** (Bias) — дополнительное значение, которое позволяет гибко сдвигать функцию активации.
- **Аппроксимация** (Approximation) — это процесс приближения сложной функции или зависимой переменной с помощью более простой, обычно аналитически описываемой функции или семейства функций.
  В контексте машинного обучения и нейронных сетей, аппроксимация означает приближение неизвестной (или сложной) зависимости между входными и выходными данными посредством параметрической модели, такой как нейронная сеть.

- **Композиция функций** (Composition of functions) — это математическая операция, при которой результат одной функции передается на вход другой функции.
- **Сплайн** (Spline) — это кусочно-гладкая функция, которая состоит из нескольких полиномиальных отрезков, соединённых в определённых точках (узлах) таким образом, что обеспечивается гладкость (непрерывность функции и её производных до заданного порядка) на всём интервале определения.
- **B-сплайн** (Basis spline) — это обобщённый класс сплайнов, представленных как линейная комбинация базисных функций

ВВЕДЕНИЕ

В 2024 году в июне студентами из MIT была предложена новая архитектура — Kolmogorov-Arnold Networks (KAN), основанная на теореме Колмогорова-Арнольда, доказанной в середине XX века Андреем Колмогоровым и Владимиром Арнольдом. Эта теорема утверждает, что любая непрерывная многомерная функция может быть представлена как суперпозиция конечного числа непрерывных функций одной переменной. KAN используют этот принцип, заменяя фиксированные функции активации традиционных нейронных сетей на обучаемые функции одной переменной на рёбрах сети, полностью исключая линейные веса. Обычно эти функции параметризуются сплайнами (или любыми другими адаптивными функциями), что, как показывают исследования, позволяет KAN достигать высокой точности с меньшим количеством параметров по сравнению с MLP.

Преимущества KAN включают не только улучшенную точность, но и повышенную интерпретируемость. Их структура позволяет интуитивно визуализировать модель и взаимодействовать с ней, что делает KAN особенно ценными для научных приложений. Например, исследования показывают, что KAN эффективны в задачах аппроксимации данных и решении уравнений с частными производным. Более того, KAN могут выступать в роли "сотрудников" для учёных, помогая (пере)открывать математические и физические законы, что является неожиданным применением для нейронных сетей, обычно рассматриваемых как "чёрные ящики"1

1 ПОСТАНОВКА ЗАДАЧИ

Данная дипломная работа направлена на всестороннее исследование архитектуры KAN, охватывающее как теоретические, так и практические аспекты. Основной целью является понимание, как KAN могут решать текущие вызовы в области нейронных сетей, такие как недостаточная интерпретируемость и высокие вычислительные затраты традиционных архитектур.

Конкретные задачи включают:

- **Теоретические основы**: Изучение теоремы Колмогорова-Арнольда и её применения в проектировании нейронных сетей, включая анализ, как KAN используют суперпозицию одномерных функций для представления многомерных функций.
- **Архитектурный дизайн**: Подробное описание структуры KAN, включая использование сплайнов для параметризации функций, отсутствие линейных весов и сравнение с MLP.
- **Оценка производительности**: Анализ эффективности KAN в различных приложениях, таких как аппроксимация данных и обработка данных высокой размерности, в сравнении с традиционными MLP.
- **Интерпретируемость**: Исследование, как KAN улучшают прозрачность модели, и обсуждение их потенциального влияния на научные открытия. Например, KAN могут быть полезны для извлечения научных правил из данных, что особенно важно в физике и математике
- **Вызовы и будущие направления**: Обсуждение текущих ограничений, таких как повышенные требования к вычислительным ресурсам и сложности в настройке гиперпараметров, а также предложения по их преодолению.

2 ТРЕБОВАНИЯ К ОКРУЖЕНИЮ

2.1 Требования к аппаратному обеспечению

Проект предполагает обучение нейросетевых моделей (в т.ч. KAN и MLP), а также обработку высокоразмерных данных. Поэтому аппаратные требования зависят от объёма данных и сложности моделей:

2.1.1 Минимальное требования (Для прототипа/разработки)

- Процессор: 4-ядерный CPU (например, Intel Core i5 / AMD Ryzen 5)
- Оперативная память: от 8 ГБ
- Накопитель: SSD, минимум 20 ГБ свободного места
- Графический ускоритель: не обязателен, но желательно наличие хотя бы CUDA-совместимой видеокарты (например, GTX 1050 Ti) для ускоренного обучения

2.1.2 Рекомендуемые требования (для экспериментов с большими данными)

- Процессор: 6–8-ядерный CPU (например, Intel i7 / Ryzen 7 и выше)
- Оперативная память: от 16–32 ГБ
- Графический ускоритель: NVIDIA GPU с поддержкой CUDA (например, RTX 3060/3070/3090 или A100 для продвинутых задач)
- Накопитель: SSD от 100 ГБ (особенно если используется большой датасет)

2.2 Требования к программному обеспечению

2.2.1 ОС:

- Linux (Ubuntu, Arch, Debian и др.) — предпочтительно

- Windows / macOS — возможно, но может потребовать дополнительных настроек

2.2.2 Язык программирования:

- Python 3.9+

2.2.3 Основные библиотеки:

- PyTorch или TensorFlow (в зависимости от реализации KAN)
- torch-spline (если используется PyTorch и B-сплайны)
- NumPy, SciPy, Matplotlib — для анализа данных и визуализации
- scikit-learn — для базового сравнения с MLP/другими моделями
- Jupyter Notebook или VS Code — для разработки и отчётности

2.2.4 Средства визуализации и документации:

- TensorBoard, WandB, Matplotlib/Seaborn — для графиков и мониторинга
- LaTeX / Overleaf / Word / Markdown — для написания теоретической части и отчётов

2.3 Требования к пользователям

Этот проект ориентирован на пользователей, обладающих определёнными знаниями в математике и машинном обучении. Основные требования к компетенциям пользователя:

2.3.1 Базовые знания:

- Основы линейной алгебры, математического анализа и теории функций
- Понимание принципов машинного обучения и нейронных сетей (например, MLP, SGD, переобучение)

2.3.2 Желательные навыки:

- Знания в области функционального анализа (для понимания пространства функций, суперпозиции, Гильбертовых пространств)
- Навыки визуализации данных
- Навыки настройки гиперпараметров и проведения экспериментов
- Опыт чтения научных статей (для работы с оригинальными источниками по KAN)
- Навыки работы с Python и популярными ML-фреймворками (PyTorch/TensorFlow

3 ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ

3.1 Математическая теорема Колмагорова – Арнольда

На втором Международном конгрессе математиков в Париже в 1900 году математик Давид Гильберт представил список из 23 задач, охватывающие многие области математики: алгебру, теорию чисел, геометрию, топологию, алгебраическую геометрию, группы Ли, вещественный и комплексный анализ, дифференциальные уравнения, математическую физику, теорию вероятностей, а также вариационное исчисление. Проблемы должны были определить вектор развития математики в XX веке. Опубликованы задачи на английском были в 1902 годе и на тот момент все не были решены. Гильберт считал выдвинутые проблемы наиболее актуальными в математическом сообществе и по сей день решены не все.

В контекст всех задач погружаться не имеет никакого смысла, так как нас интересует только Тринадцатая проблема из списка задач Гильберта. Формулируется она следующим образом: можно ли решить общее уравнение седьмой степени с помощью функций, зависящих только от двух переменных?

В 1956 году советский математик Андрей Колмагоров доказал промежуточный результат, показывающий что любую непрерывную функцию $f: [0,1]^n \to R$ через суперпозицию функций трёх переменных, но это не опровергало гипотезу Гильберта, так как функции трёх переменных не сводились к функциям двух переменных.

В 1957 году ученик Колмогорова, Владимир Арнольд, в возрасте всего 19 лет, усовершенствовал результат своего учителя. Арнольд доказал, что непрерывные функции трёх переменных можно представить через суперпозицию функций двух переменных. Его работа показала, что для любой непрерывной функции $f(x, y, z)$ на компактном множестве существует представление вида (1):

$$f(x,y,z) = \sum_{q} \Phi_q\big(\Psi_q(x,y), z\big), \tag{1}$$

где Ψ_q - непрерывные функции двух переменных, а Φ_q - непрерывные функции двух переменных. Это доказательство опровергло предположение Гильберта, показав, что даже сложные функции трёх переменных (включая корни септического уравнения) могут быть выражены через суперпозицию функций меньшей размерности.

В том же 1957 году Колмогоров обобщил результаты, доказав свою знаменитую теорему, которая утверждает, что любая непрерывная функция $f: [0,1]^n \to R$ представима через суперпозицию функций одной переменной (2):

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \Psi_{q,p}(x_p) \right), \tag{2}$$

Этот результат стал ещё более сильным опровержением гипотезы Гильберта, так как он показал, что даже функции двух переменных не являются необходимыми — достаточно одномерных функций.

3.2 Универсальная аппроксимационная теорема

Идея нейронных сетей зародилась с работ Уоррена Мак-Каллока и Уолтера Питтса (1943), которые предложили модель искусственного нейрона. В 1960-х годах Фрэнк Розенблатт разработал перцептрон, но его ограничения, указанные Марвином Мински и Сеймуром Папертом в книге Perceptrons (1969), показали, что однослойные сети не могут решать задачи с нелинейно разделимыми данными (например, XOR). Это вызвало временный спад интереса к нейронным сетям.

В 1980-х годах начался ренессанс нейронных сетей благодаря разработке многослойных перцептронов (MLP) и алгоритма обратного

распространения ошибки (backpropagation), что позволило обучать сети с скрытыми слоями. Формулировка теоремы (1989): В 1989 году Джордж Цыбенко опубликовал статью [2] в журнале Mathematics of Control, Signals, and Systems. Он доказал, что нейронная сеть с одним скрытым слоем, использующая сигмоидальную функцию активации, может аппроксимировать любую непрерывную функцию на компактном подмножестве R^n с любой точностью, если количество нейронов в скрытом слое достаточно велико. Независимо от Цыбенко, Курт Хорник, Максвелл Стинчкомб и Хэлберт Уайт в 1989 году опубликовали похожий результат в статье [3] в журнале Neural Networks. Они расширили теорему, показав, что многослойные сети также обладают этой способностью, и уточнили условия на функции активации.

Универсальная аппроксимационная теорема (Universal Approximation Theorem, UAT), или теорема Цыбенко — это фундаментальный результат в теории искусственных нейронных сетей, который объясняет, почему нейронные сети способны моделировать широкий класс функций. Она утверждает, что нейронная сеть с одним скрытым слоем и достаточным количеством нейронов может аппроксимировать любую непрерывную функцию на компактном подмножестве R^n с любой заданной точностью, при условии, что функция активации является нелинейной и удовлетворяет определённым условиям (например, сигмоида или ReLU).

$$f(x) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(w_i \cdot x + b_i), \quad (3)$$

3.3 Принцип работы MLP

Многослойный перцептрон (MLP, Multilayer Perceptron) — это тип искусственной нейронной сети общий вид архитектуры которой представлен на рисунке 1. Он используется для решения задач машинного обучения, таких как классификация (например, определение того, является ли изображение кошкой или собакой) и регрессия (например, прогнозирование цены дома).

Это одна из самых простых, но мощных архитектур нейронных сетей, которая лежит в основе более сложных моделей глубокого обучения.

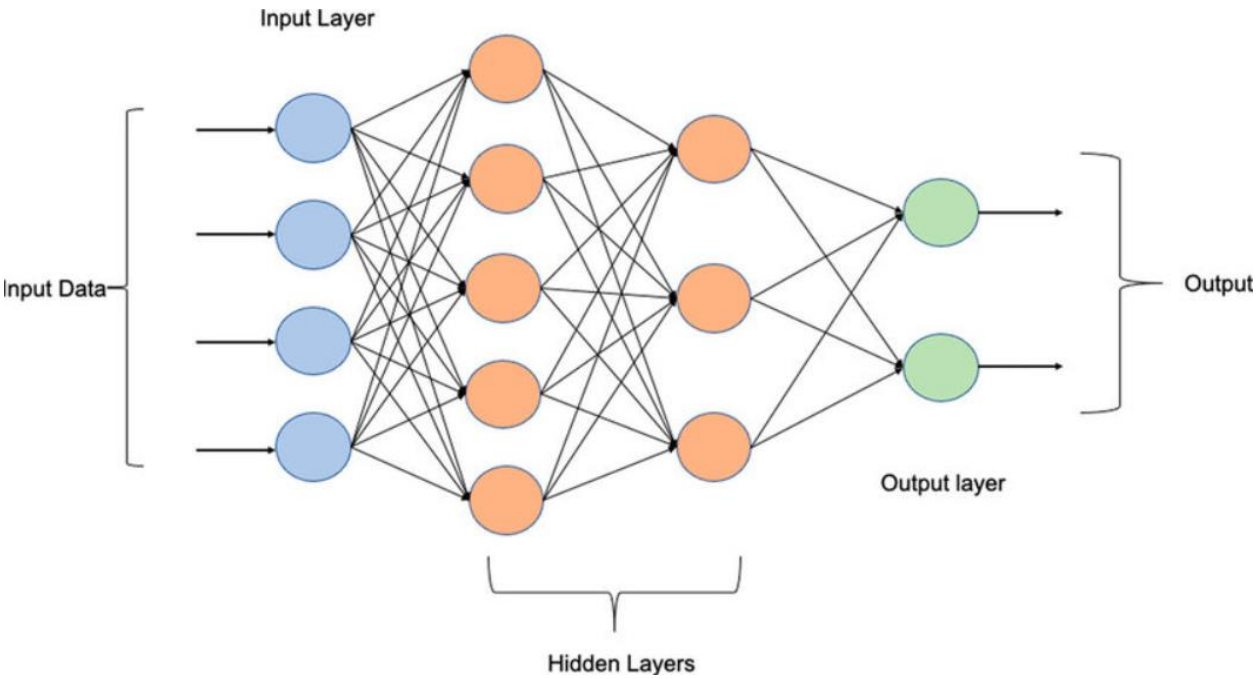

Рисунок 1 – визуализация MLP

MLP — это нейронная сеть, состоящая из нескольких слоёв нейронов, соединённых друг с другом. Каждый нейрон в сети выполняет простую задачу: принимает входные данные, обрабатывает их и передаёт результат дальше. Слои в MLP организованы следующим образом:

- Входной слой: принимает исходные данные (например, пиксели изображения или числовые характеристики).
- Скрытые слои: обрабатывают данные, извлекая из них сложные закономерности. Чем больше скрытых слоёв, тем более сложные задачи может решать сеть.
- Выходной слой: выдаёт окончательный результат, например, вероятность принадлежности к классу или числовое предсказание.

Каждый нейрон в одном слое связан со всеми нейронами в следующем слое, что делает MLP полносвязной сетью. Эти связи имеют веса —

параметры, которые определяют, насколько сильно сигнал от одного нейрона влияет на другой. Кроме того, у каждого нейрона есть смещение, которое помогает корректировать обработку данных.

Работа MLP делится на два основных этапа: Прямое распространение ошибки и обратно (backpropagation). Давайте разберём эти этапы подробно, а затем рассмотрим, как сеть инициализируется и обучается.

Прямое распространение — это процесс, при котором данные проходят через сеть от входного слоя к выходному, преобразуясь на каждом этапе. Представьте это как конвейер, где данные постепенно превращаются в прогноз.

3.3.1 Входной слой

Входной слой принимает данные. Например, если вы хотите классифицировать изображение, данные могут представлять собой яркость пикселей. Если это задача прогнозирования цены дома, данные могут включать площадь, количество комнат и т. д.

Каждый входной признак (например, значение пикселя) передаётся в сеть как отдельный нейрон входного слоя.

3.3.2 Скрытые слои

Данные со входного слоя передаются в первый скрытый слой. Каждый нейрон в скрытом слое выполняет следующие действия:

1. Собирает информацию: нейрон получает сигналы от всех нейронов предыдущего слоя. Каждый сигнал умножается на соответствующий вес (вес определяет, насколько важен этот сигнал).
2. Суммирует сигналы: нейрон складывает все взвешенные сигналы и добавляет к ним смещение (смещение помогает нейрону гибко настраивать результат).

3. Применяет функцию активации: сумма сигналов пропускается через функцию активации, которая добавляет нелинейность. Это важно, потому что без нелинейности сеть не смогла бы моделировать сложные зависимости в данных. Примеры функций активации:

- ReLU: (выпрямляющий линейный блок) (4): обнуляет отрицательные значения, оставляя положительные без изменений. Это ускоряет обучение и делает сеть устойчивой.

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}, \tag{4}$$

- Сигмоида (5): преобразует значение в диапазон от 0 до 1, что полезно для вероятностей.

$$f(x) = \frac{1}{1 + e^{x}}, \tag{5}$$

- Tanh (6): преобразует значение в диапазон от -1 до 1, что хорошо для центрированных данных.

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{6}$$

После применения функции активации нейрон передаёт свой результат (активацию) всем нейронам следующего слоя.

3.3.3 Выходной слой

Последний слой (выходной) получает данные от последнего скрытого слоя и выдаёт прогноз.

Тип функции активации в выходном слое зависит от задачи:

- Для бинарной классификации (например, «да» или «нет») используется сигмоида, которая выдаёт вероятность (число от 0 до 1).

- Для многоклассовой классификации (например, распознавания цифр от 0 до 9) используется функция softmax, которая распределяет вероятности между классами так, чтобы их сумма равнялась 1.
- Для регрессии (например, прогнозирования цены) может не использоваться функция активации, чтобы получить непрерывное число.

Результат выходного слоя — это предсказание сети

3.3.4 Оценка ошибки

После получения предсказания сеть сравнивает его с истинным значением (y), используя функцию потерь. Функция потерь измеряет, насколько сильно предсказание отличается от правильного ответа. Примеры:

- Среднеквадратичная ошибка (MSE) (7)**:** используется для регрессии, измеряет среднее квадратичное отклонение предсказания от истины.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2, \tag{7}$$

- Бинарная кросс-энтропия (8): используется для бинарной классификации, оценивает, насколько предсказанные вероятности соответствуют истинным меткам.

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\widehat{y_i}) + (1 - y_i)\log(1 - \widehat{y_i})], \tag{8}$$

- Категориальная кросс-энтропия (9): используется для многоклассовой классификации, оценивает качество вероятностного распределения.

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\widehat{y_{i,c}}), \tag{9}$$

Значение функции потерь показывает, насколько «плохо» сеть справилась с задачей. Цель обучения — минимизировать это значение.

Обратное распространение — это процесс, в ходе которого сеть «обучается», корректируя свои веса и смещения, чтобы уменьшить ошибку прогнозирования. Это похоже на то, как человек учится на своих ошибках: сеть анализирует, где она допустила ошибку, и настраивает свои параметры, чтобы в следующий раз быть точнее.

3.3.5 Оценка ошибки на выходе

После прямого распространения сеть знает, насколько её предсказание отличается от истины (это значение функции потерь). Обратное распространение начинается с выходного слоя, где ошибка наиболее очевидна.

Для каждого нейрона в выходном слое вычисляется, насколько он «виноват» в общей ошибке. Это зависит от:

- Разницы между предсказанием и истинным значением.
- Тип функции активации (например, сигмоида или softmax влияет на то, как распределяется ошибка).

3.3.6 Распространение ошибки назад

Ошибка от выходного слоя передаётся обратно через сеть, слой за слоем, до входного слоя. Для каждого нейрона в скрытых слоях определяется, насколько он повлиял на ошибку в следующем слое.

Этот процесс напоминает обратный конвейер: ошибка распределяется по сети, чтобы понять, какие нейроны и связи больше всего повлияли на неправильное предсказание.

При распределении ошибки учитывается:

- Вес связей: если связь между нейронами имеет большой вес, она сильнее влияет на ошибку.
- Функция активации: она определяет, как ошибка передаётся обратно (например, для ReLU ошибка не передаётся через нейроны с нулевой активацией).

3.3.7 Вычисление корректировок

Для каждого нейрона сеть вычисляет, как нужно изменить его веса и смещение, чтобы уменьшить ошибку. Это делается следующим образом:

Если нейрон сильно повлиял на ошибку, его веса и смещение корректируются сильнее.

Если нейрон мало повлиял, изменения будут минимальными.

Корректировки зависят от:

- Величины ошибки, приписанной нейрону.
- Активация нейронов предыдущего слоя (они показывают, какие входные данные повлияли на этот нейрон).
- Скорость обучения — параметр, определяющий, насколько сильно изменяются веса за один шаг. Низкая скорость обучения делает изменения осторожными, высокая — более агрессивными.

3.3.8 Обновление параметров

После вычисления корректировок веса и смещения каждого нейрона обновляются. Это похоже на настройку регуляторов: сеть подкручивает параметры, чтобы в следующий раз предсказание было ближе к истине.

Обновление происходит одновременно для всех параметров сети, чтобы учесть их совместное влияние на ошибку.

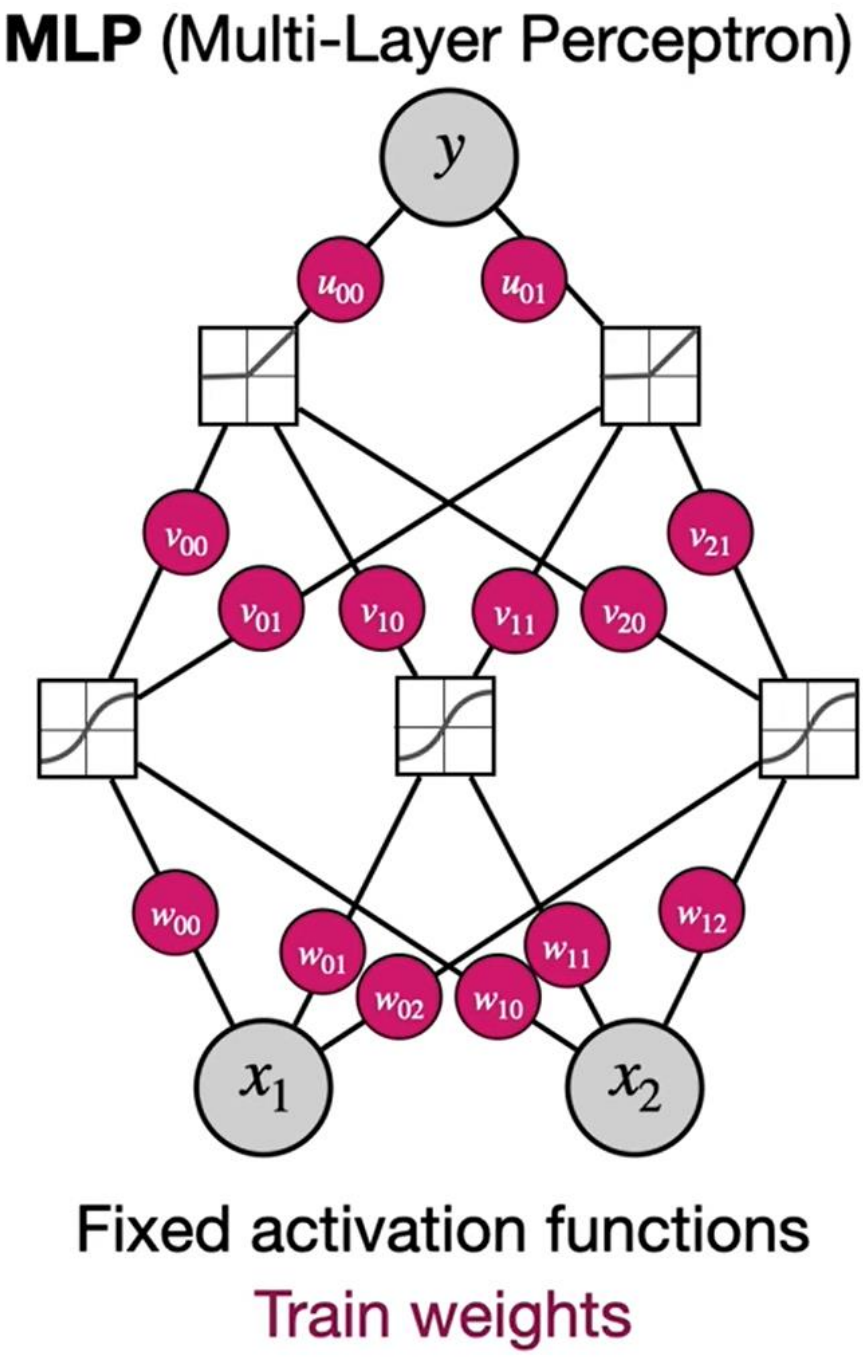

Рисунок 2 – Визуализация процесса обучения MLP

На Рисунке 2 можно увидеть визуализацию процесса обучение многослойного перцептрона, который имеет 2 скрытых слоя. На данном изображении хорошо видно, что у нас есть два входных значения, которые, поступая в первый скрытый слой, умножаются на веса, соответствующие их пути до конкретного нейрона, после чего подвергаются суммированию и

функции активации. Эти самые функции активации фиксированы и их мы выбираем заранее, при том для каждого скрытого слоя выбираем свою функцию. После прохода через скрытые слои, на выходе имеем одно значение y.

3.4 Принцип работы KAN

Kolmogorov-Arnold Networks (KAN) — это нейронная сеть, которая переосмысливает, как данные обрабатываются и преобразуются в сети. В отличие от традиционных MLP, которые используют фиксированные функции активации в нейронах и линейные преобразования (веса) для связей между ними. KAN применяет обучаемые функции непосредственно к связям (ребрам) между нейронами. Это ключевое различие делает KAN более гибкими и интерпретируемыми.

Основные отличия от MLP:

Где применяются функции активации:

- В MLP функции активации фиксированы и применяются к каждому нейрону после суммирования взвешенных входов.
- В KAN функции активации находятся на ребрах (связях между нейронами), а не в нейронах. Эти функции не фиксированы — они обучаются в процессе тренировки и могут быть уникальными для каждой связи.

Роль нейронов:

- В MLP нейроны выполняют сложную работу: суммируют входы, применяют веса и функцию активации.
- В KAN нейроны проще — они только суммируют сигналы, поступающие от предыдущего слоя, без применения нелинейных преобразований. Вся "магия" происходит на ребрах.

Интерпретируемость:

- MLP часто называют «чёрными ящиками», потому что их внутренние процессы трудно понять.
- KAN более прозрачны, так как обучаемые функции на рёбрах можно визуализировать и анализировать, чтобы понять, как сеть принимает решения.

Эффективность:

KAN часто требуют меньше нейронов и слоёв для достижения той же точности, что и MLP, что делает их более компактными и параметрически эффективными.

3.4.1 Процесс обучения.

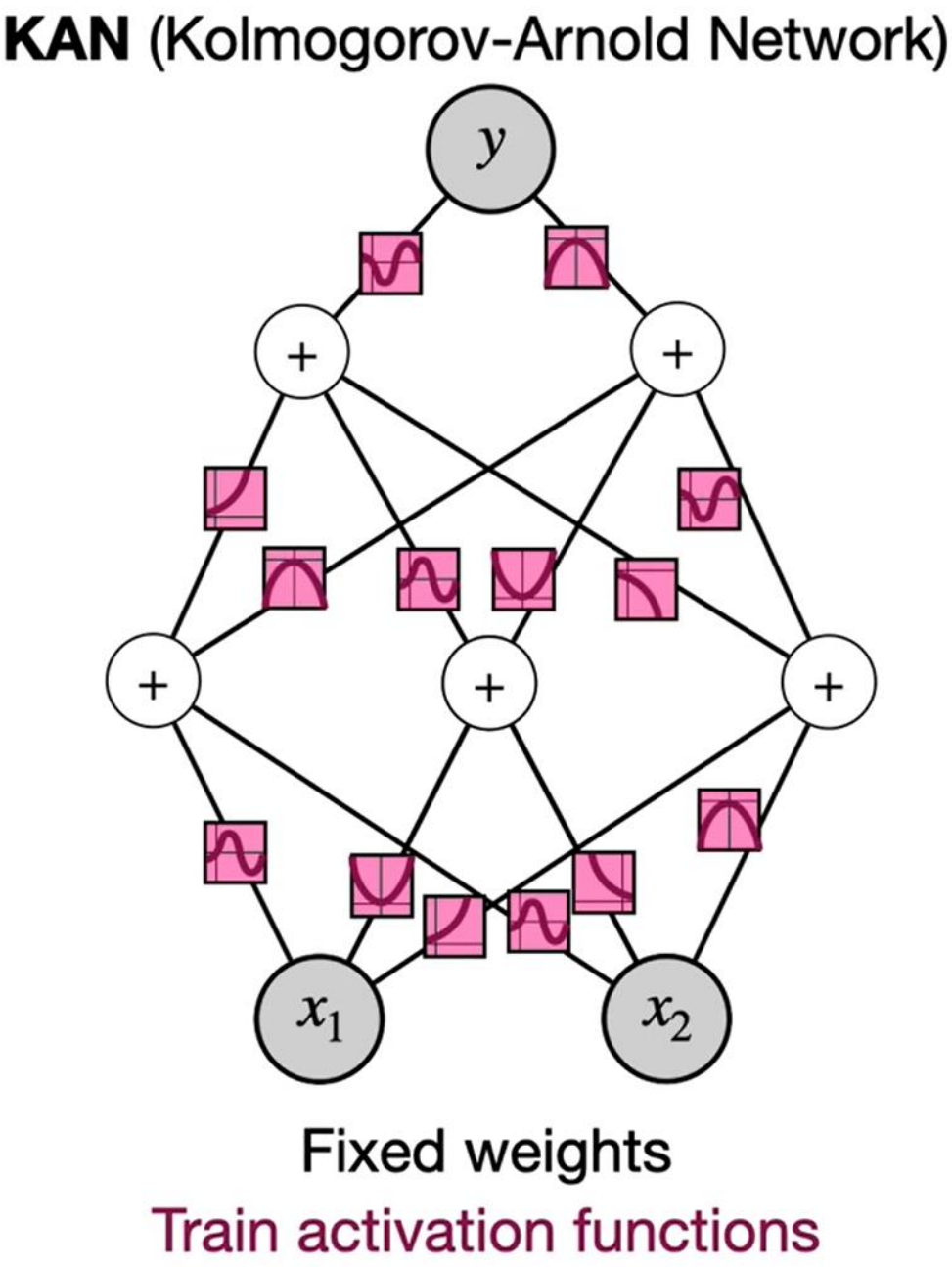

Рисунок 3 – визуализация процесса обучения KAN

На рисунке 3 можно лучше увидеть и сравнить две архитектуры. Основное сущностное различие в том, что веса изменились на единицу (Обычное сложение) и, соответственно, стали константными, а функции активации стали обучаемы и применяются на рёбрах. Таким образом, в

процессе обучения подбираться будут именно оптимальные функции для каждого ребра, а не веса нейронов.

3.5 Как обучить функцию?

3.5.1 MLP

Рассмотреть обучение функции можно на примере одной операции в нейронной сети, которая будет представлять из себя два входных нейрона и один выходной, и, соответственно будем настраивать две адаптивных функции

Для начала посмотрим, как работает обучение весов в MLP. Схема такого процесса представлена на рисунке 4

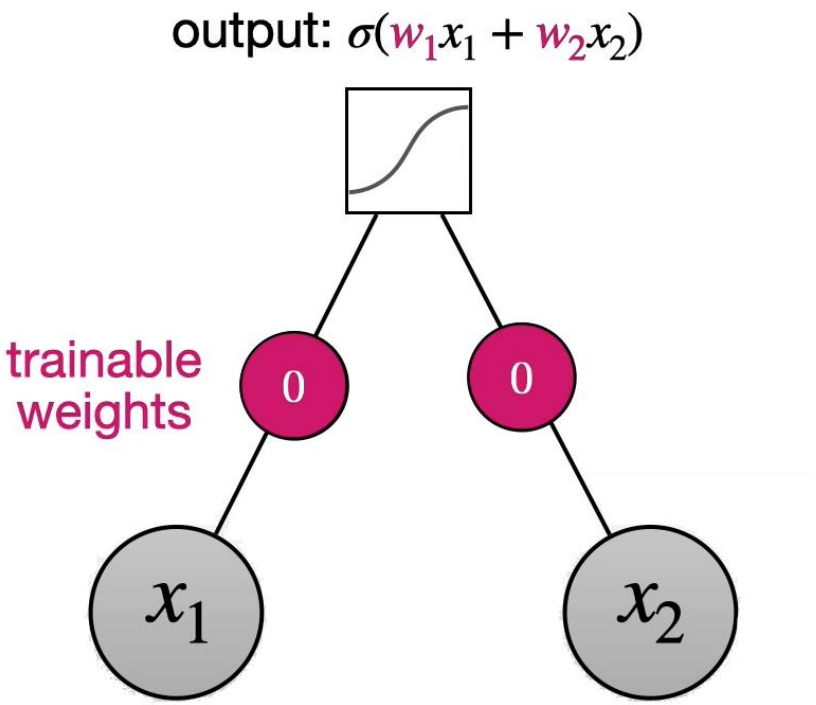

Рисунок 4 – простая структура MLP

Мы пытаемся настроить веса модели (числа на ребрах), которые инициализируются нулями, или любыми другими случайными значениями. Также на полученную сумму применяется фиксированная функция активации,

которая обеспечивает нелинейность. В контексте задачи машинного обучения мы всегда определяем функцию потерь (Loss function) – Это функция, которую надо минимизировать, чтобы получить наилучшее приближение. Путём минимизации этой функции оптимизационным методом градиентного спуска (рисунок 5) можно найти оптимальные веса для нашей модели

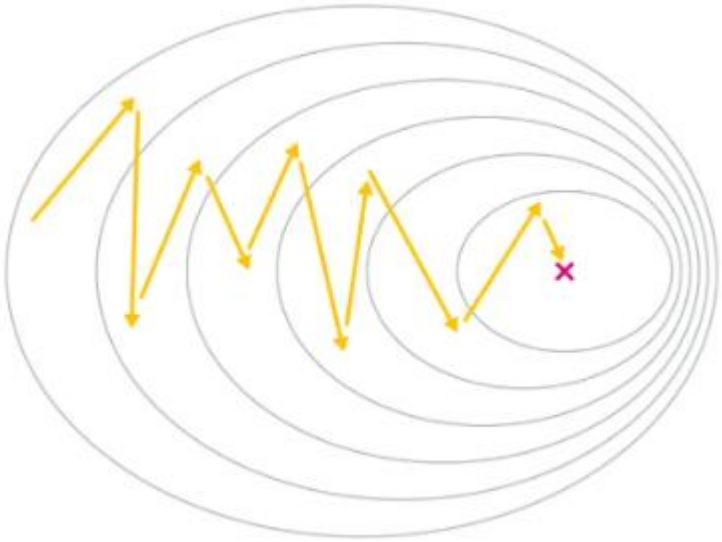

Рисунок 5 – Градиентный спуск

Если мы инициализируем веса случайными числами – мы, скорее всего, находимся не в оптимальной позиции весов. Поэтому мы начинаем делать шаги по весам, который уменьшают функционал качества. Таким образом обучаются коэффициенты, но как же обучить функцию?

3.5.2 KAN

В архитектуре KAN на замену обучаемым весам пришли обучаемые адаптивные функции. Из названия понятно, что, в отличие от MLP, функция активации в KAN будет не фиксированной

Адаптивная функция, в общем смысле, это – функция, форма которой изменяется в процессе обучения, оптимизации, или взаимодействия с

окружающей средой. У такой функции есть параметры, которые и подбираются в процессе обучения. Общий вид (10):

$$y = \sum_{j=1}^{G} \omega_j \varphi_j (x), \tag{10}$$

Такая функция представляет собой комбинацию базисов $\varphi_j(x)$, а веса ω_j обучаются. Общий вид одного слоя представлен на рисунке 6.

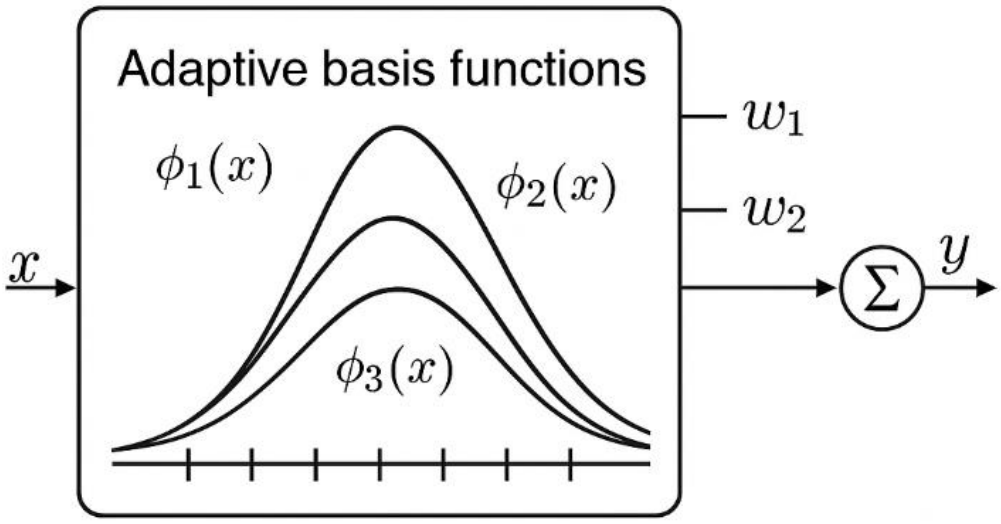

Рисунок 6 – модель нейрона KAN

Таких адаптивных функций может быть много. Архитектура KAN лишь подразумевает само обучение этих самых адаптивных функций, однако в официальной библиотеке kan для языка Python под названием pykan нет опции выбора базисной функции, там используются B-сплайны. Однако, помимо них могут использоваться, например, радиальные базисные функции.

На примере трех нейронов схема обучения представлена на рисунке 7

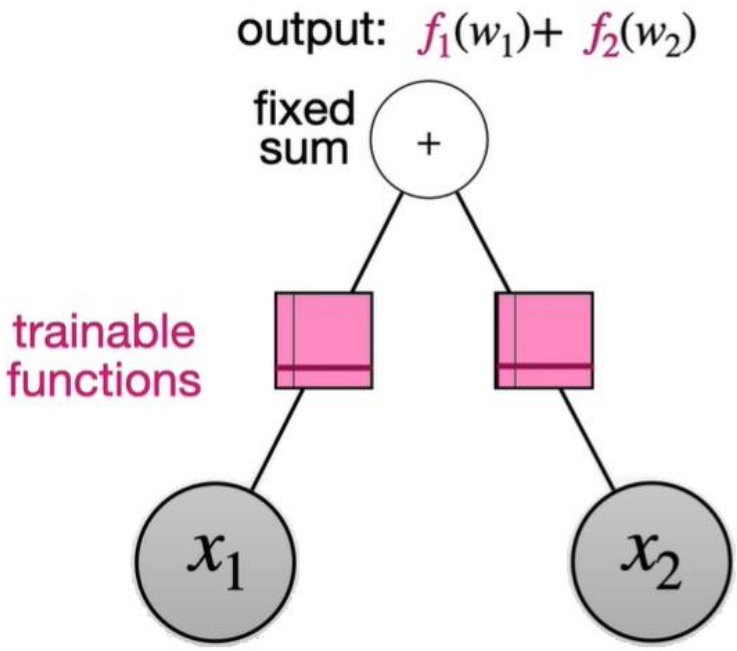

Рисунок 7 – Простая структура KAN

Как и в MLP функции инициализируются любым образом, например, нулём. Ввиду того что обучение просто функции – это сложная задача с огромной кучей параметров – мы и используем меньшее подмножество функций, которое можно обучить небольшим количеством параметров. Тогда, предположим, в простейшем случае мы хотим, например, обучить функцию как на рисунке 8

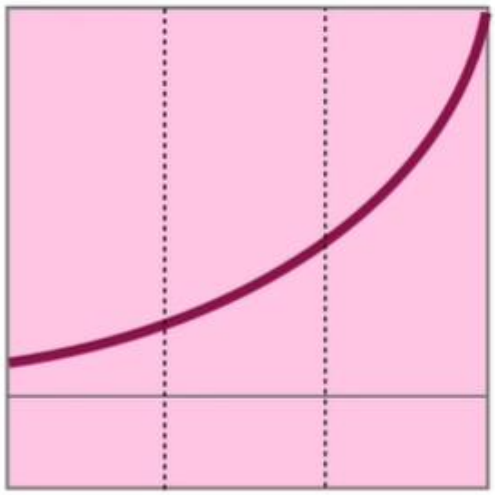

Рисунок 8 – Искомая функция

Тогда мы должны также обозначить базисные сплайны. В простейшем случае можно взять обычные константные прямые, как на рисунке 9

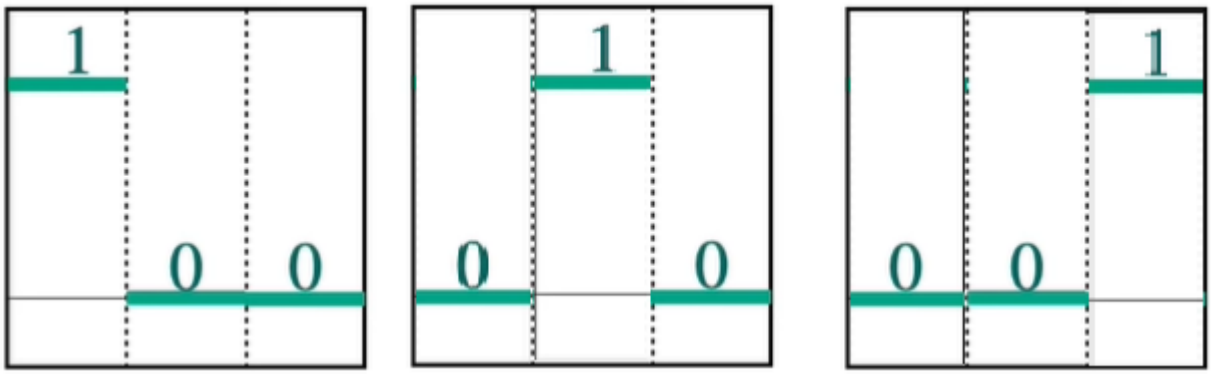

Рисунок 9 – базисные функции

Тогда, подбирая параметры для линейной комбинации данных базисных сплайнов, получим коэффициенты 0.1, 0.3, 0.7 соответственно и приближение получится как на рисунке 10:

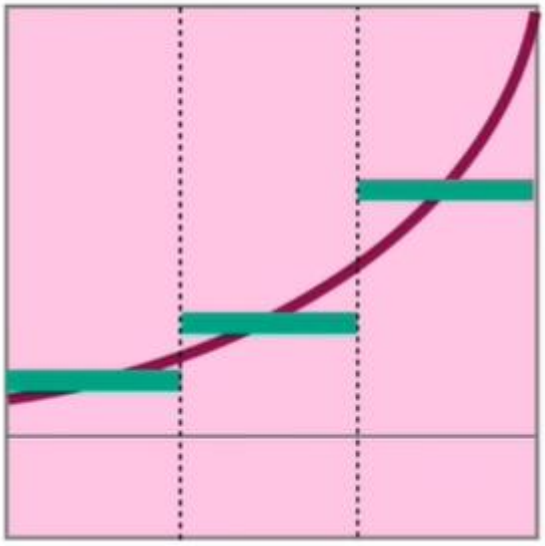

Рисунок 10 – Адаптивная функция

Само собой, вышло не точно, но в простейшем виде это – неплохой результат. Соответственно для его улучшения можно увеличивать сетку и брать более аккуратные базисные функции. Из такой архитектуры вытекают

достаточно серьезные проблемы с переобучаемостью и временем обучения, о которых я расскажу в основной части.

4 ИНТЕРПРЕТИРУЕМОСТЬ

Интерпретируемость модели определяется следующими критериями: как именно она применяет решения, на какие входы она реагирует и что конкретно делает каждый слой или блок. MLP – это, так называемый, black box: очень сложно понять, как конкретно они интерпретируют входы.

Как KAN интерпретирует решение. Они заменяют матричные умножение на сумму одноаргументных функций, за счёт чего можно визуализировать сеть, как на рисунке 11 и аналитически делать определенные выводы.

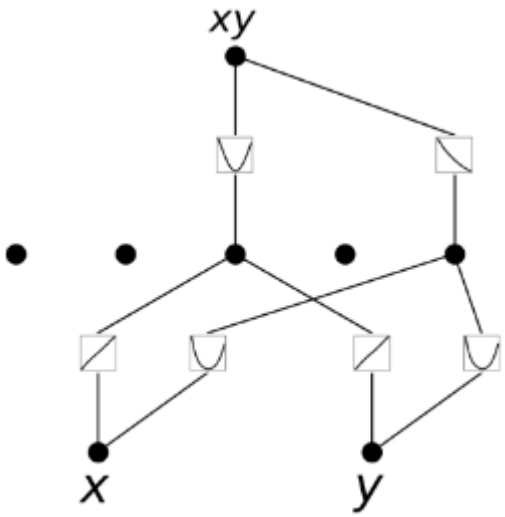

Рисунок 11 – график KAN

Такая визуализация становится доступна как раз ввиду использования адаптивных функций одной переменной. Она также встроена в библиотеку и вызывается командой model.plot()

Также за счёт обыкновенного суммирования функций появляется возможность извлечения символической формулы. Для этого можно либо использовать стандартную библиотеку функций (она достаточно мала), либо вручную аналитически подбирать каждому ребру свою функцию (У каждой функции есть свой порядковый номер, содержащий три цифры – номер слоя,

номер входа, номер ребра у входа) и, таким образом, на выходе можно получить приближенное описание поведения модели

Вся прочая интерпретируемость складывается из вышеперечисленных факторов.

5 ЭКСПЕРЕМЕНТАЛЬНАЯ ЧАСТЬ И ОЦЕНКА РЕЗУЛЬТАТОВ

5.1 Общая математическая модель

KAN семейство нейронных сетей (11) выглядит следующим образом:

$$f_{KAN}(x) = \sum_{i_L=1}^{n_L} \varphi_{L-1,i_L,i_{L-1}} \left(\sum_{i_{L-1}=1}^{n_{L-1}} \varphi_{L-2,i_{L-1},i_{L-2}} \left(\cdots \left(\sum_{i_1=1}^{n_1} \varphi_{1,i_2,i_1} \left(\sum_{i_0=1}^{n_0} \varphi_{0,i_1,i_0} \right) \right) \right) \right), \quad (11)$$

Согласно нашей аппроксимационной теореме, искомую функцию можно разложить на суперпозиции суммы функций одной переменной $\varphi_{L,i_{L+1},i_L}(x_{i_L})$. Суперпозиция получается сколь угодно глубокой за счёт того, что репрезентативная теорема описывает лишь один слой архитектуры KAN. В последнем слое i_L означает, что функция может быть векторозначной (12)

$$\sum_{i_L=1}^{n_L} \varphi_{L,i_{L+1},i_L}(x_{i_L}), \quad (12)$$

Где каждая сумма представляет собой преобразование подобного рода

$$x_{l+1} = \sum_{i_l=1}^{n_l} \varphi_{l,i_{l+1},i_l}(x_{i_l})$$

$$= \begin{bmatrix} \varphi_{l,1,1}(x_1) & + & \varphi_{l,1,2}(x_2) & + \cdots + & \varphi_{l,1,n_l}(x_{n_l}) \\ \varphi_{l,2,1}(x_1) & + & \varphi_{l,2,2}(x_2) & + \cdots + & \varphi_{l,1,n_l}(x_{n_l}) \\ \cdots & & \cdots & \cdots & \cdots \\ \varphi_{l,n_{l+1},1}(x_1) & + & \varphi_{l,n_{l+2},2}(x_2) & + \cdots + & \varphi_{l,n_{l+1},n_l}(x_{n_l}) \end{bmatrix}$$

B-spline – это всего лишь частный случай KAN, который будет использоваться почти для всех реализуемых задач в контексте дипломной

работы. Соответственно сама архитектура концептуально представляет собой нечто большее, но в контексте реализации конкретных задач было принято решение воспользоваться наиболее оптимальным вариантом (Также на примере MNIST рассмотреть вариант решения, используя полиномы Чебышева). В свою очередь, BSKAN представляется следующим образом:

$$\varphi_{l,i_{l+1},i_l}(x_{i_l}) = \varphi BSKAN_{l,i_{l+1},i_l}(x_{i_l}),$$

$$\varphi BSKAN(x) = \omega_b f_{base}(x) + \omega_s f_{spline}(x),$$

$$f_{base}(x) = SiLU(x) = \frac{1}{1+e^{-x}},$$

$$f_{spline}(x) = \sum_i \omega_i B_i^k(x),$$

5.2 Замечание: Обобщение MLP до KAN

В контексте выполнения дипломной работы было тяжело не заметить, что MLP является частным случаем архитектуры KAN. Так как в открытом доступе и в оригинальной статье я не нашёл упоминаний и доказательств данного вопроса – я опишу его самостоятельно

Можно заметить, что слой KAN представляет собой применение функции активации и последующее аффинное преобразование. Первый слой будет состоять из линейных функций. $\varphi_{0,i_1,i_0}(x_{l,i}) = \omega_{i_1,i_0} x_{i_0} + b_{i_1,i_0}$.

В контексте такой интерпретации слоя нейронной сети в случае KAN каждая функция $\varphi_{l,i_{l+1},i_l}(x_{i_l})$ будет обучаемой, хотя и не могут аппроксимировать в большинстве своем функции одной переменной. Это связано с тем, что они, по сути, являются масштабированными функциями активации, которые также в большинстве своем не обладают данной возможностью.

Из этого небольшого доказательства банально следует что KAN обобщает MLP на случай произвольных функций одной переменной

Для пущего понимания можно рассмотреть теорему Цыбенко в контексте KAN (13). Тогда будем иметь один скрытый слой, произвольное число нейронов, и сигмоиду в качестве функции активации:

$$f_{Cybenko}(x) = \sum_{i_1=1}^{n_1} \varphi_{1,i_1} \left(\sum_{i_0=1}^{n_0} \varphi_{0,i_1,i_0}(x_{,i_0}) \right), \tag{13}$$

Тогда первый слой будет иметь следующий вид:

$$x_1 = \sum_{i_0=1}^{n_0} \omega_{i_1,i_0} x_{i_0} + b_{i_1}$$

$$= \begin{bmatrix} \omega_{i_1,i_0} x_{i_0} & + & \omega_{i_1,i_0} x_{i_0} & + \cdots + & \omega_{i_1,i_0} x_{i_0} \\ \omega_{i_1,i_0} x_{i_0} & + & \omega_{i_1,i_0} x_{i_0} & + \cdots + & \omega_{i_1,i_0} x_{i_0} \\ \cdots & & \cdots & \cdots & \cdots \\ \omega_{i_1,i_0} x_{i_0} & + & \omega_{i_1,i_0} x_{i_0} & + \cdots + & \omega_{i_1,i_0} x_{i_0} \end{bmatrix}$$

$$= \sum_{i_0=1}^{n_0} \varphi_{i_1,i_0}(x_{,i_0})$$

$$= \begin{bmatrix} \varphi_{1,1}(x_1) & + & \varphi_{1,2}(x_2) & + \cdots + & \varphi_{1,n_0}(x_{n_0}) \\ \varphi_{2,1}(x_1) & + & \varphi_{2,2}(x_2) & + \cdots + & \varphi_{1,n_0}(x_{n_0}) \\ \cdots & & \cdots & \cdots & \cdots \\ \varphi_{n_1,1}(x_1) & + & \varphi_{n_1,2}(x_2) & + \cdots + & \varphi_{n_1,n_0}(x_{n_0}) \end{bmatrix}$$

А второй слой будет иметь следующий вид:

$$f_{Cybenko}(x_1) = \sum_{i_1=1}^{n_0} \omega_{i_1} \sigma(x_{i_1}) + b$$

$$= \omega_1 \sigma(x_1) + \omega_2 \sigma(x_2) + \cdots + \omega_{n_1} \sigma(x_{n_1}) + b$$

$$= \sum_{i_1=1}^{n_1} \varphi_{1,i_1} \left(\sum_{i_0=1}^{n_0} \varphi_{0,i_1,i_0}(x_{,i_0}) \right),$$

Соответственно получаем модель, которая сначала выполняет аффинное преобразование, затем применяется сигмоидная функция активации (σ). После чего выполняется еще одно аффинное преобразование. Затем на выходе имеем скаляр (предсказание)

5.3 Игрушечные датасеты

Для сравнения архитектур MLP (многослойный перцептрон) и KAN (сети Колмогорова-Арнольда) на игрушечных наборах данных я представлю результаты исследования, в котором мы тестируем обе модели на двух типах задач: регрессии (предсказание значений заранее заданных функций) и классификации (на классическом наборе данных Moons). Я опишу, как проводилось тестирование, какие метрики использовались и какие результаты были получены, объясняя их в контексте особенностей каждой архитектуры.

5.3.1 Простая одномерная функция

Цель эксперимента — сравнить эффективность двух архитектур нейронных сетей, KAN (Kolmogorov-Arnold Network) и MLP (Multilayer Perceptron), при аппроксимации одномерной функции (14).

$$f(x) = \sin(\pi x) * \exp(-x^2) + \cos(5x) \qquad (14)$$

Оценка проводилась по метрикам ошибки (MSE) на обучающей и тестовой выборках, а также с учетом числа параметров моделей. В эксперименте использовались различные конфигурации KAN и MLP, чтобы изучить их способность к обобщению и эффективность.

5.3.2 Реализация

Для реализации такого эксперимента использовался python. Сперва я использовал для задач лишь библиотеки pykan и PyTorch (библиотеки для языка python, в которых реализованы актуальные версии библиотек). Также для визуализации классический matplotlib. Сама реализация представляет из себя инициализацию искомой функции, обучающей выборки и архитектур. Для каждой из нейросетей я сразу написал три различных архитектуры для более удобного сравнения: простую (малая ширина, большой lr, малое количество эпох, малая глубина), среднюю (средняя ширина, средний lr, среднее количество эпох, средняя глубина), и, соответственно, большую (глубокая, широкая, с низким lr, с большим количеством эпох).

5.3.3 MLP: предварительный анализ

Реализация MLP стандартная библиотечная как крепко устоявшаяся архитектура крайне проста в написании и даёт эффективную точность по мере увеличения самой модели. Настройка гиперпараметров для задачи восстановления функциональной довольно линейна.

Нет смысла рассматривать архитектуры в условиях одинакового количества нейронов и слоёв, так как MLP в этом плане определенно уступит, хотя обучаться будет в десятки раз быстрее. Ниже представлена реализация довольно простых сетей для аппроксимации простой нелинейной функции (15)

$$f(x) = \sin(\pi x) * \exp(-x^2) + 0.1 * \cos(5x) \qquad (15)$$

В качестве решения ниже представлены архитектуры следующего формата:

{"layers": [1, 2, 3, 1], "epochs": 1000, "lr": 0.01, "name": "MLP Simple"},

{"layers": [1, 5, 5, 1], "epochs": 2000, "lr": 0.005, "name": "MLP Medium"},

{"layers": [1, 100, 100, 100, 1], "epochs": 3000, "lr": 0.001, "name": "MLP Deep"}

Здесь соответственно наглядно видно почему такие архитектуры рассматривать нет смысла. В случае функции активации RELU приближение становится кусочно – линейным (представлено на рисунке 12) и нам не годится (третья архитектура в данном примере – хорошая аппроксимация для сравнения)

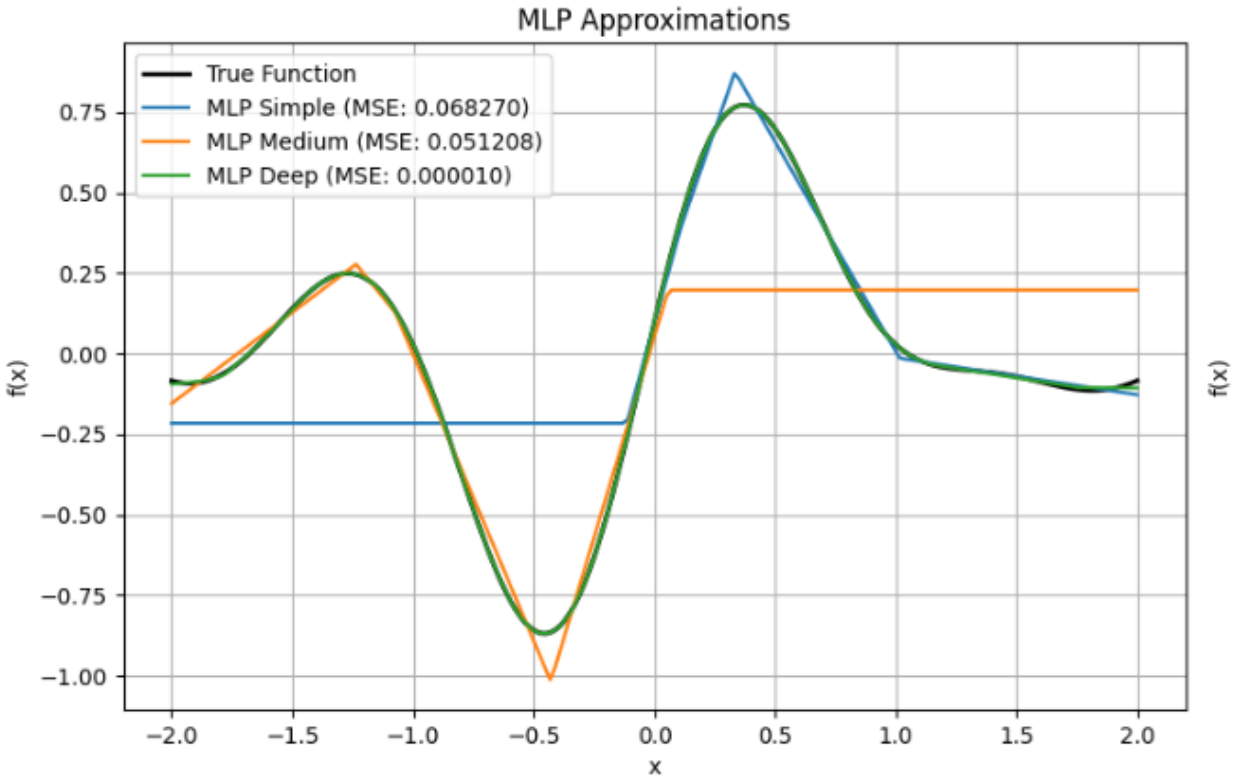

Рисунок 12 – $f(x) = \sin(\pi x) * \exp(-x^2) + 0.1 * \cos(5x)$ (MLP)

5.3.4 KAN: Предварительный анализ

Ввиду того что на данном этапе выполнения работы я лишь изучал настройку гиперпараметров для библиотеки pykan – я проводил соответствующий анализ функций, которые аппроксимировал

5.3.5 Структура функции:

Функция $\sin(\pi x) * \exp(-x^2) + 0.1 * \cos(5x)$ представляет собою сумму двух компонент: произведения периодической функции $\sin(\pi x)$ с умеренной частотой и гауссовской функции $\exp(-x^2)$, которая обеспечивает затухание и второй компоненты $0.1 * \cos(5x)$, которая представляет собою высокую осцилляцию с малой амплитудой.

Функция гладкая и аддитивная и, соответственно, чтобы выделить все особенности функции будет достаточно одного слоя для выделения обеих компонент в отдельные сплайны. Второй слой KAN введёт дополнительный уровень композиции, который станет избыточным для аппроксимации такой функции

5.3.6 Избыточная архитектура

При исследовании функции мною была совершена ошибка и проведено множество тестов с двухслойной архитектурой KAN, которые демонстрирует склонность к переобучению модели при неправильной настройке гиперпараметров. Ниже в данном подразделе я представлю результаты исследования при использовании двух скрытых слоёв вместо одного

Архитектуры финальной модели были следующие
{"width": [1, 6, 4, 1], "grid": 16, "k": 4, "lamb": 0.1, "name": "KAN simple"},

{"width": [1, 8, 6, 1], "grid": 16, "k": 4, "lamb": 0.1, "name": "KAN medium"},

{"width": [1, 10, 8, 1], "grid": 16, "k": 4, "lamb": 0.1, "name": "KAN deep"}

200 шагов обучения при размере шага = 0.01

Визуализации полученных архитектур представлены на рисунках 13, 14 и 15:

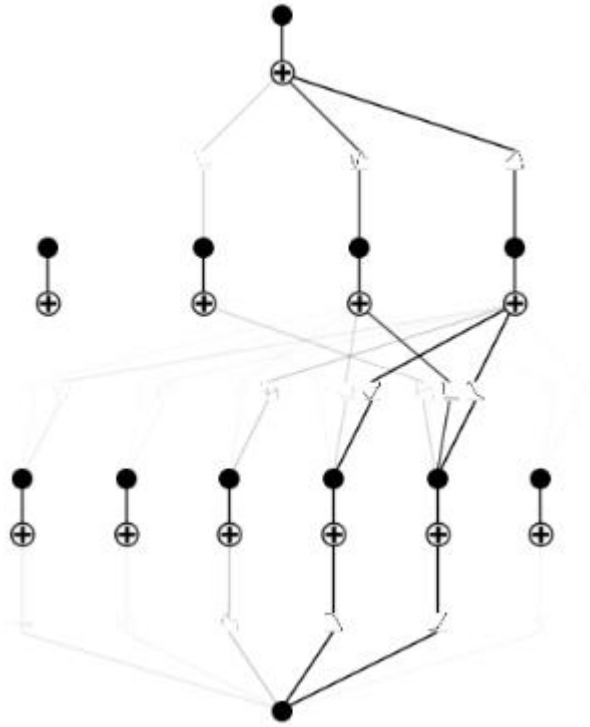

Рисунок 13 – Архитектура [1, 6, 4, 1] KAN

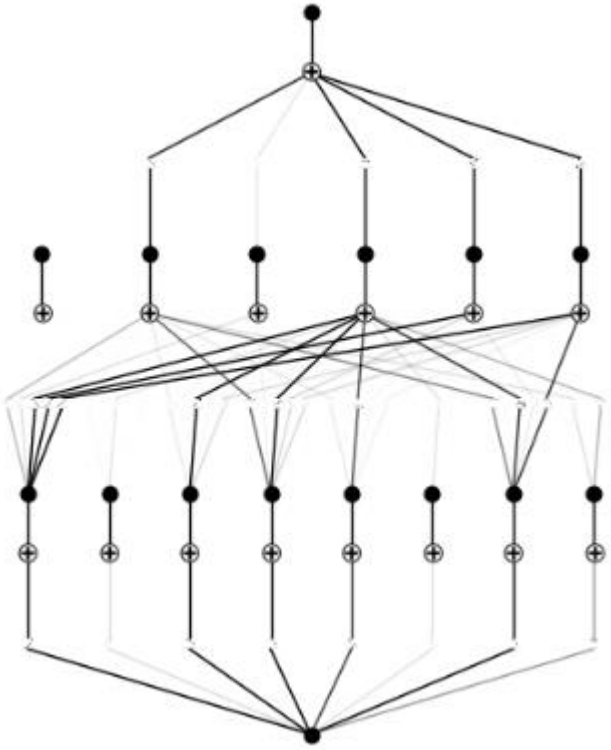

Рисунок 14 – Архитектура [1, 8, 6, 1] KAN

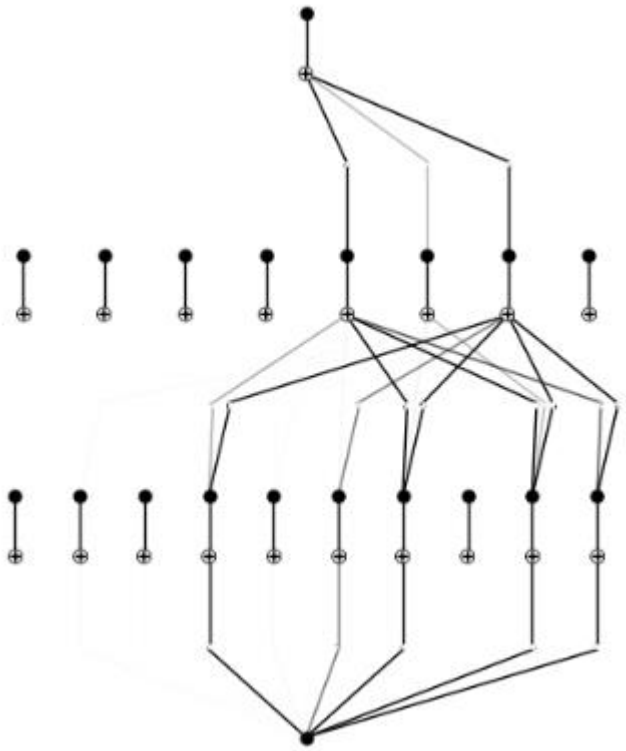

Рисунок 15 – Архитектура [1, 10, 8, 1] KAN

Из данных визуализаций уже можно заметить, что архитектура для нашей зависимости избыточна. Аппроксимация показана на рисунке 16.

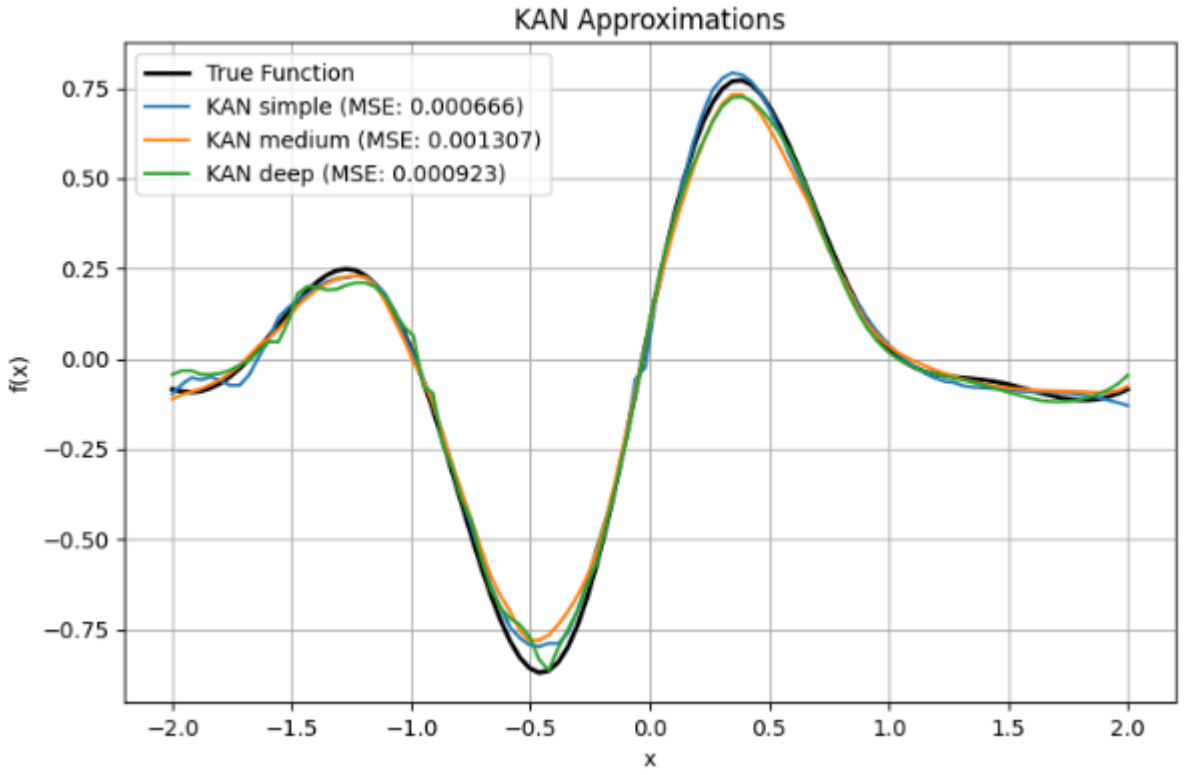

Рисунок 16 – Визуализация аппроксимации переобученных моделей

Данную аппроксимацию нельзя назвать шибко неудачной ввиду того, что ошибка на тестовых данных достаточно мала, однако для нейронной сети и непрерывной функции это – плохой результат. Но ввиду того, что при построении архитектуры мы, само собой, целевую функцию не знаем, а пытаемся найти, на данном примере я пробовал построить KAN самых разных размеров. Несколько таких архитектур представлены на рисунках 17, 18 и 19:

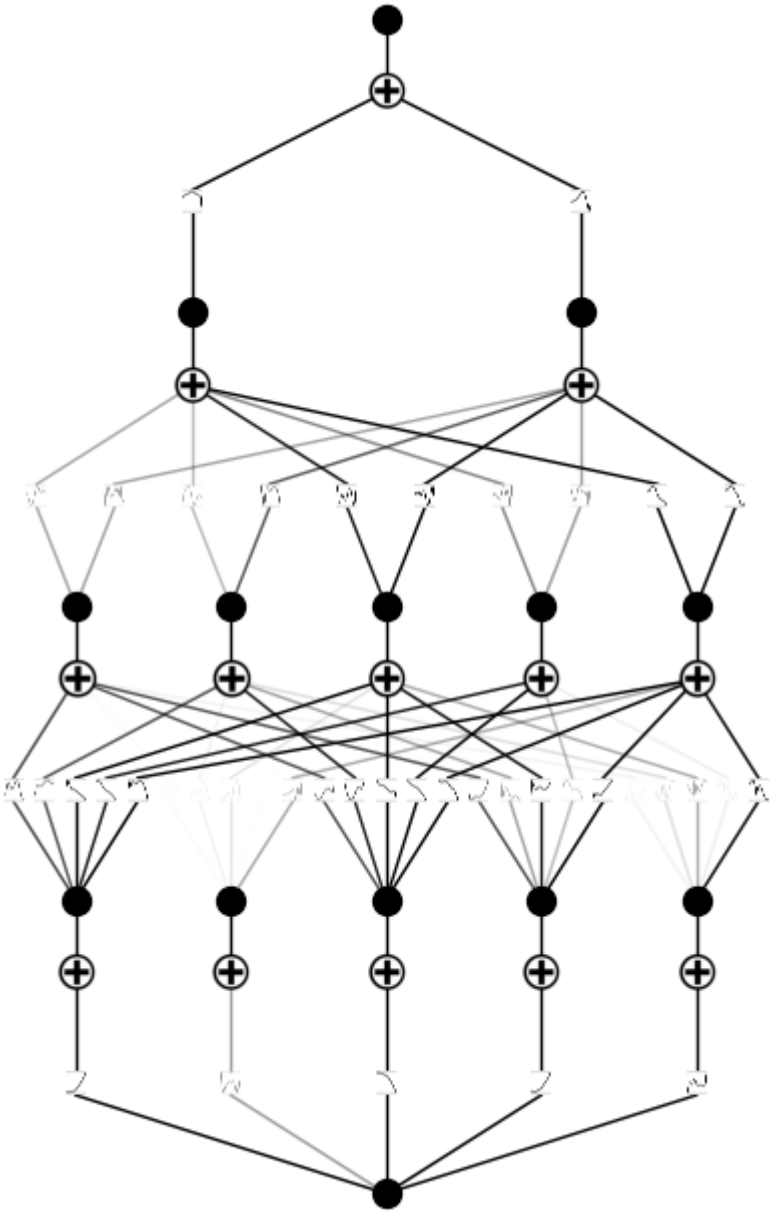

Рисунок 17 - Архитектура [1, 5, 5, 2, 1] KAN

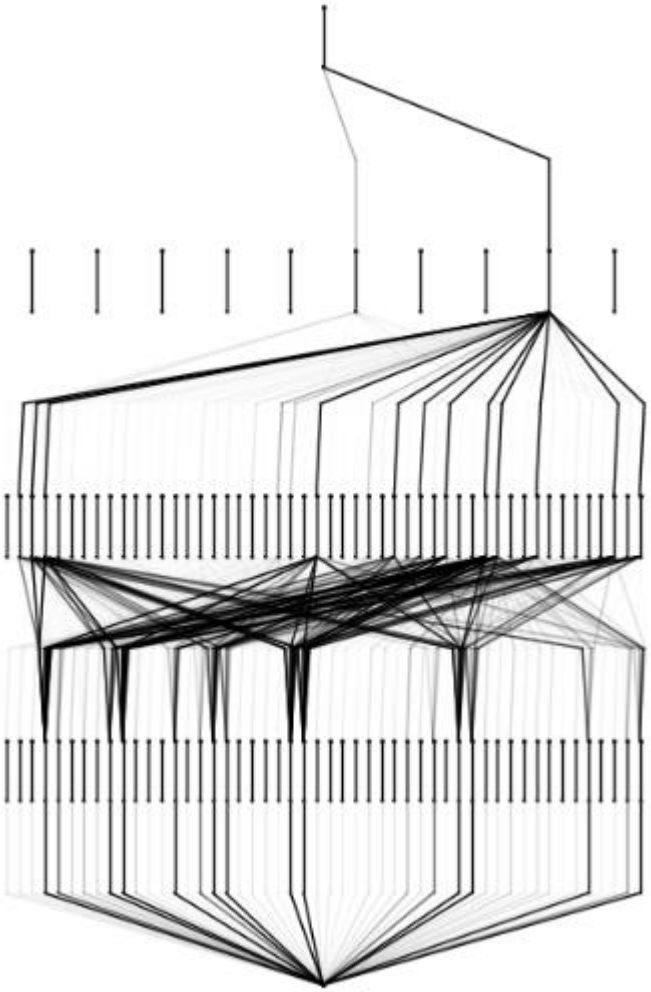

Рисунок 18 – Архитектура [1, 50, 50, 10, 1] KAN

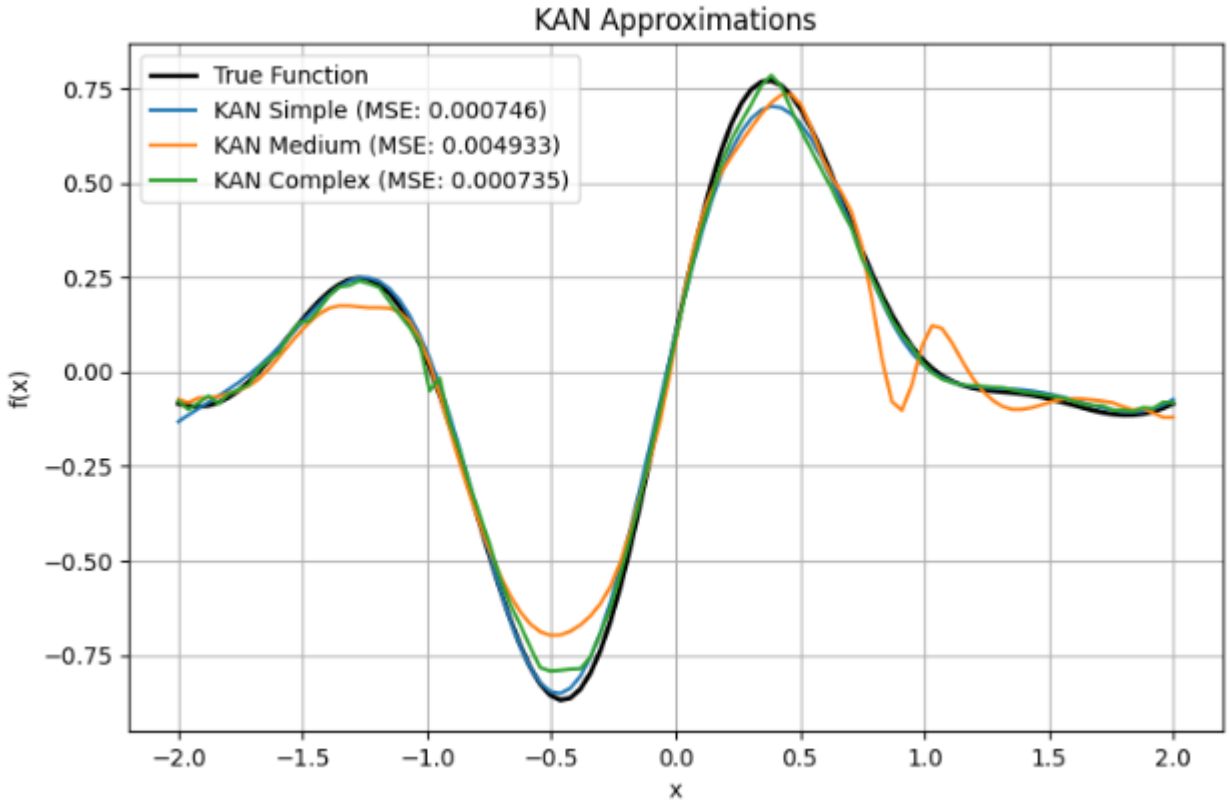

Рисунок 19 – Результаты аппроксимации для избыточных архитектур

Здесь, как и в случае с лучшими результатами, будут приемлемые Результаты, но определённо не сравнятся с однослойной KAN:

Для оптимального решения задачи достаточно одного нейрона, но можно, ради интереса, прописать больше нейронов. Для финального обучения использовались конфиги:

{"width": [1, 3, 1], "grid": 8, "k": 3, "name": "KAN Optimal"},

{"width": [1, 5, 1], "grid": 10, "k": 3, "name": "KAN Medium"},

{"width": [1, 7, 1], "grid": 15, "k": 3, "name": "KAN Complex"},

{"width": [1, 1, 1], "grid": 15, "k": 3, "name": "KAN Complex"},

5.3.7 Результаты сравнения: Простая аппроксимация

Результаты сравнения представлены на рисунке 20

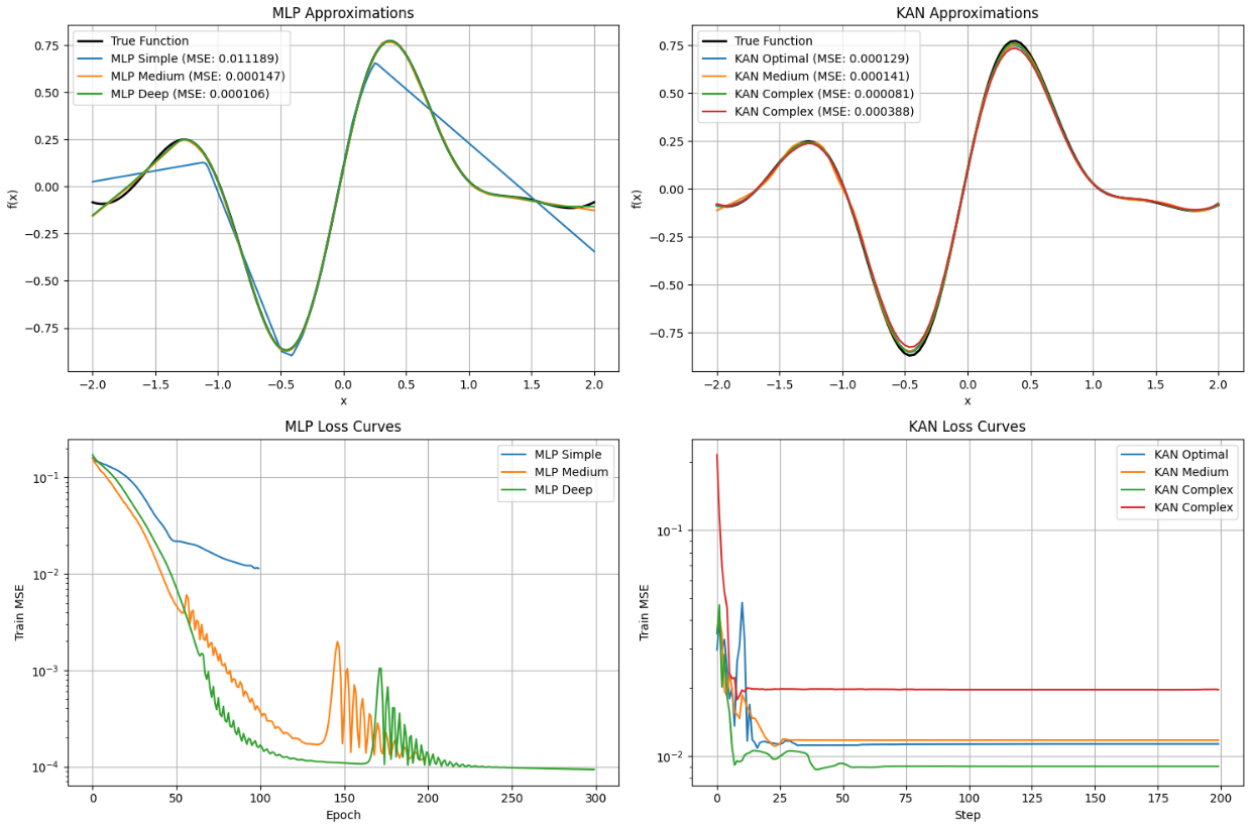

Рисунок 20 – Результаты первого эксперимента

Выводы на базе первого эксперимента:

- При удачном выборе архитектуры KAN, она показывает значительно лучшие результаты при меньшем количестве входных данных и параметрах модели
- Даже при выставлении очень больших параметров (3000 эпох, три скрытых слоя на 100 нейронов, шаг обучения = 0.001) MLP отказывается улавливать поведение функции на конце (А в случае с настройками выше обе границы)
- KAN обучалась значительно дольше, чем MLP (На платформе Kaggle с использованием акселератора GPU P 100). При обучении MLP в районе 10 секунд, KAN обучает три модели примерно 50 секунд

- Огромная разница в количестве параметров. 48 – для KAN, 20501 – для KAN (При том, что аппроксимация вышла хуже)
- KAN полностью обучается за 60 шагов
- KAN лучше справилась с поставленной задачей
- Для задач восстановления зависимости не получилось достигнуть желаемой точности на train выборке

5.3.8 Комплексная одномерная функция

Рассмотрим также в качестве примера более сложную функцию, для которой уже будет проще подобрать оптимальную архитектуру KAN.

Функцию (График представлен на рисунке 21) рассмотрим следующую (16):

$$f = \sin(2\pi * \exp(-x^2)) * \cos(3 * \pi * x^2) \quad (16)$$

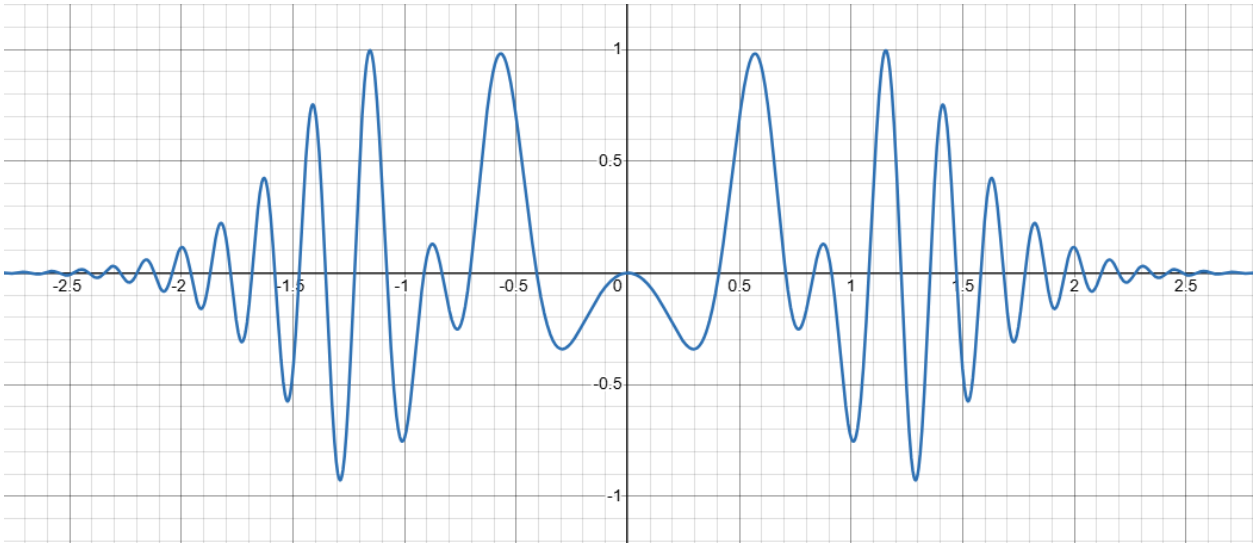

Рисунок 21 – $f = \sin(2\pi * \exp(-x^2)) * \cos(3 * \pi * x^2)$

Предположительно, данная функция будет более презентативна, так как имеет высокую частоту осцилляций. В данном случае оптимальной окажется также стандартная сеть KAN (с одним скрытыми слоем)

5.3.9 Архитектура

В случае MLP с предварительной настройкой рассматривать особо нечего, хотя я усилил архитектуру на фоне предыдущего раза. Если применять ту же архитектуру, получим результаты, представленные на рисунках 22 и 23)

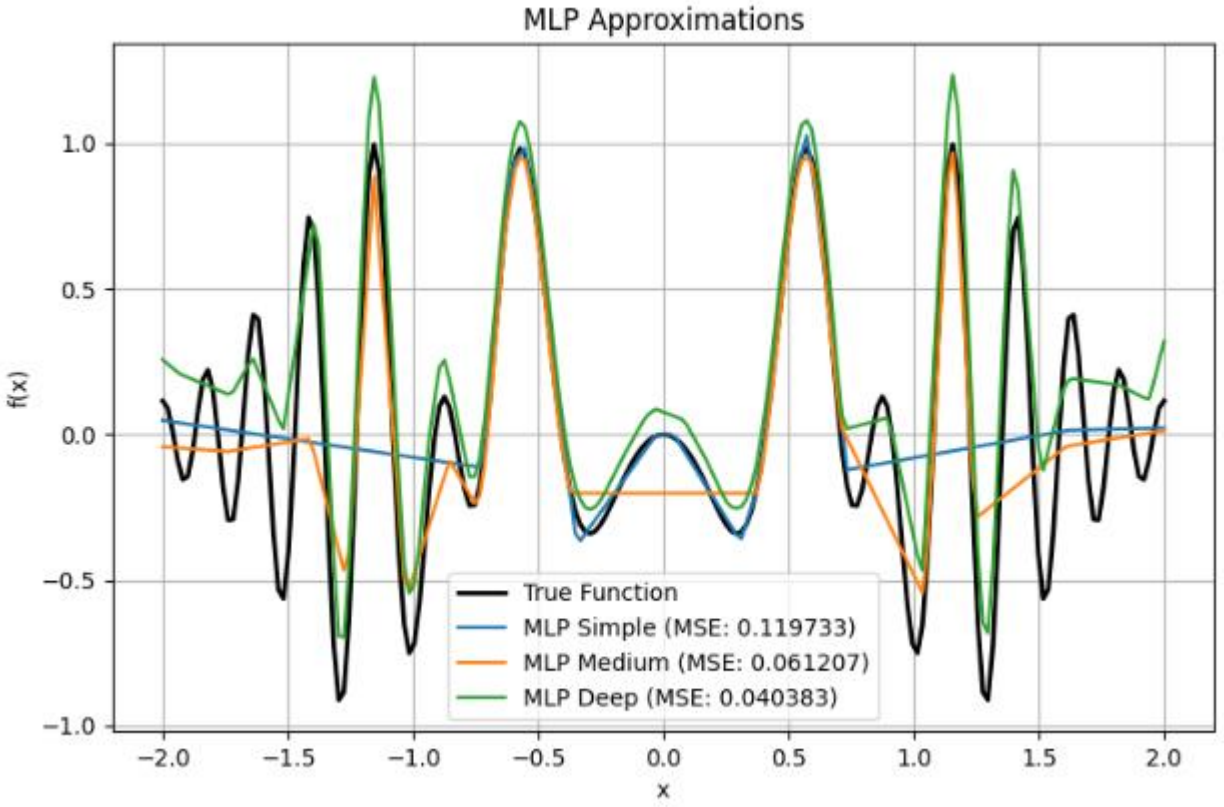

Рисунок 22 – Слабая MLP

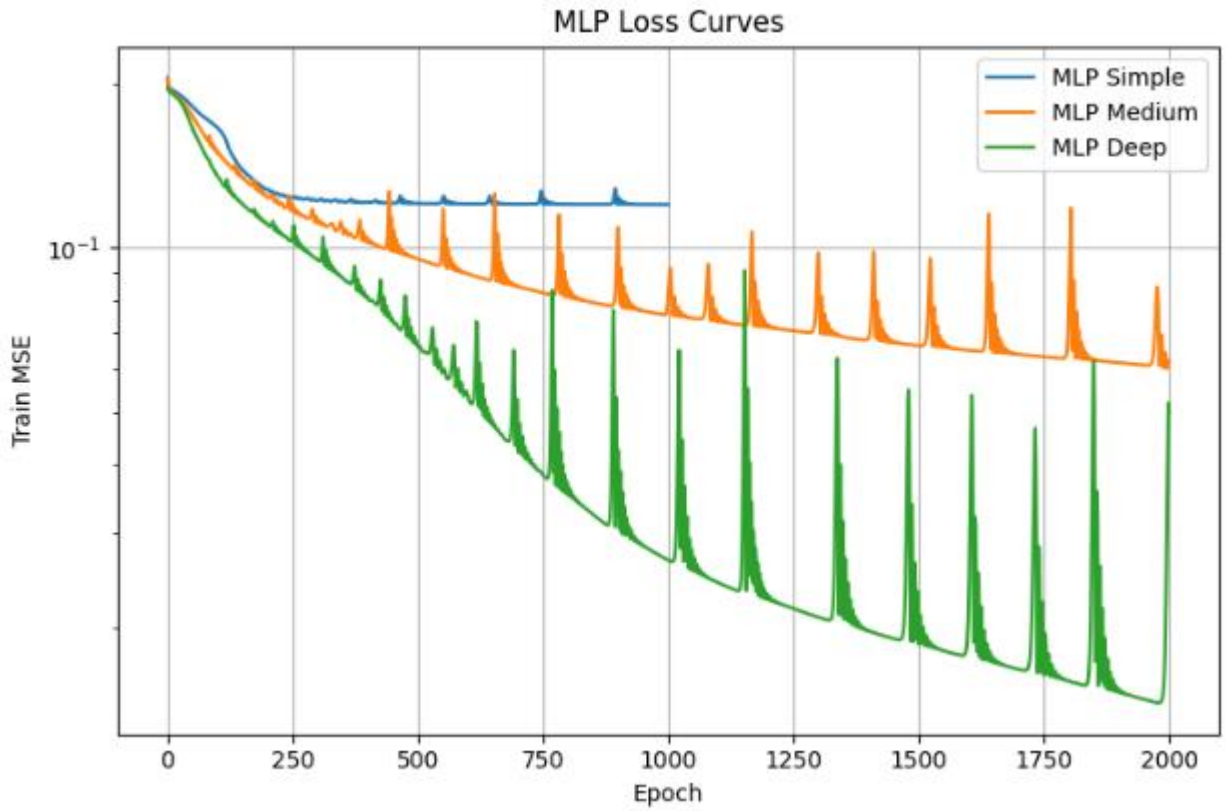

Рисунок 23 – Падение метрики слабой MLP для комплексной функции

Такая модель, очевидно, никуда не годится. Точность аппроксимации очень маленькая, поведение функции не улавливается, и видно, как метрика продолжает убывать. Архитектуры MLP для данной функции выбраны следующие:

{"layers": [1, 10, 10, 1], "epochs": 1000, "lr": 0.01, "name": "MLP Simple"}, {"layers": [1, 50, 50, 1], "epochs": 5000, "lr": 0.005, "name": "MLP Medium"}, {"layers": [1, 200, 200, 200, 1], "epochs": 10000, "lr": 0.001, "name": "MLP Deep"},

Увеличено количество эпох в 5 раз и количество нейронов в 2 раза в Deep архитектуре

В случае архитектуры KAN оставляем 1 слой и добавляем 2 нейрон для минимальной достаточной архитектуры. Также в целях наблюдения создадим ещё две архитектуры с большим количеством нейронов

{"width": [1, 2, 1], "grid": 15, "k": 3, "name": "KAN 2"},

{"width": [1, 5, 1], "grid": 15, "k": 3, "name": "KAN 5"},

{"width": [1, 7, 1], "grid": 15, "k": 3, "name": "KAN 7"},

После обучения обученные функции показаны на рисунках 24, 25 и 26:

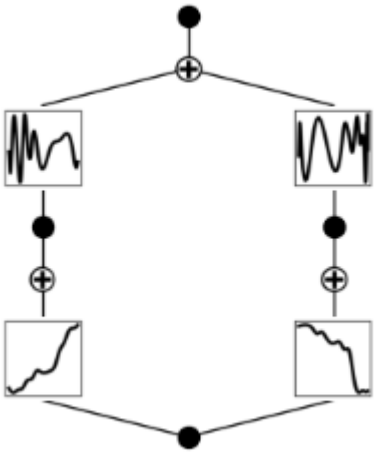

Рисунок 24 – Архитектура [1, 2, 1] KAN

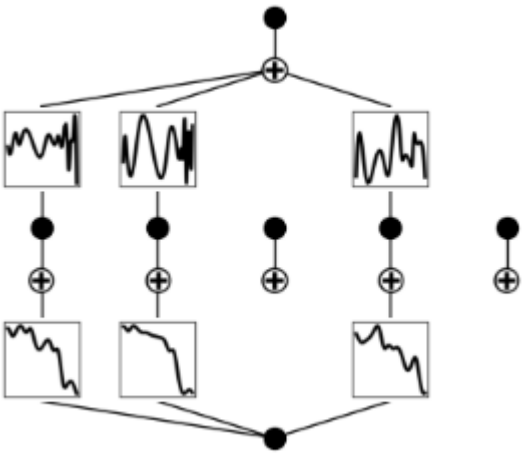

Рисунок 25 – Архитектура [1, 5, 1] KAN

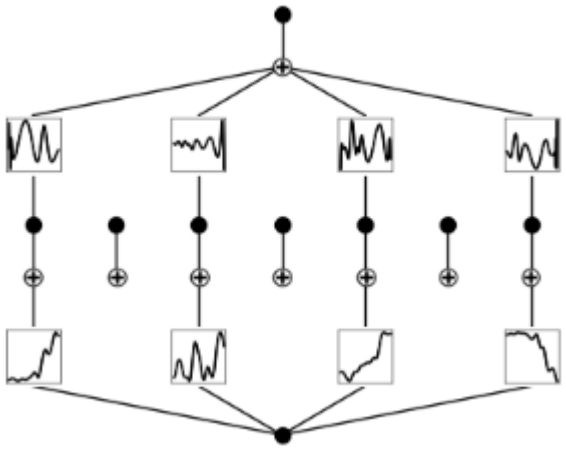

Рисунок 26 – Архитектура [1, 7, 1] KAN

5.3.10 Результаты сравнения: Комплексная одномерная функция

Результаты численных экспериментов представлены на рисунке 27

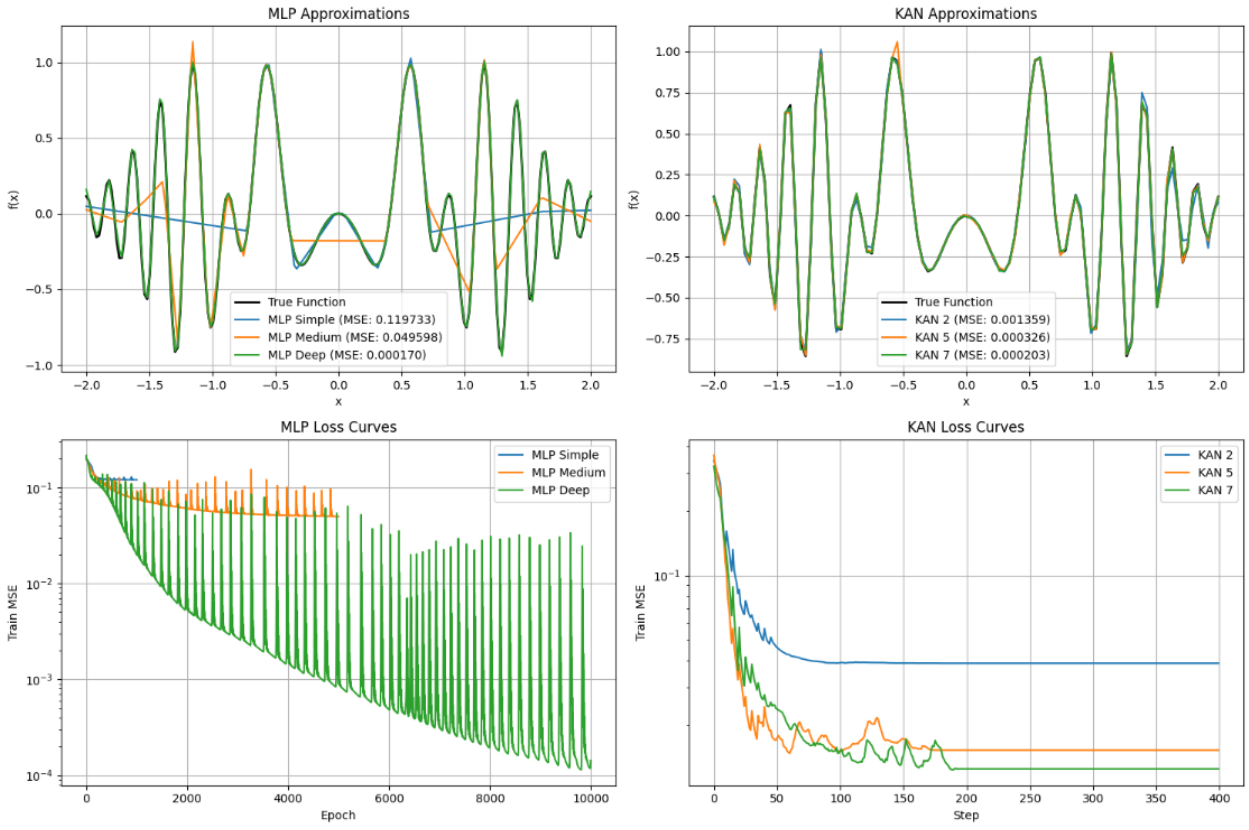

Рисунок 27 – Результаты второго эксперимента

Предварительные выводы:

- KAN отлично улавливает столь тяжёлую нелинейную зависимость, имея в запасе всего два нейрона
- KAN понадобилось для обучения в 50 раз меньше итераций
- Огромная разница в количестве параметров (110 у KAN против 81000 у MLP)
- MLP со столь большой архитектурой как и в первом случае плохо аппроксимирует концы отрезка
- Для обоих тестов явно заметна тяжёлая настройка гиперпараметров KAN, что сказывается сильнее при долгом обучении

5.4 Moons классификация

В качестве простой задачи для изучения способности модели к классификации я выбрал два классических датасета: Moons и WineQuality.

Moons (график зависимости представлен на рисунке 28) представляет собой набор двумерных точек, которые делятся на два пересекающихся множества (соответственно линейно неразделимы)

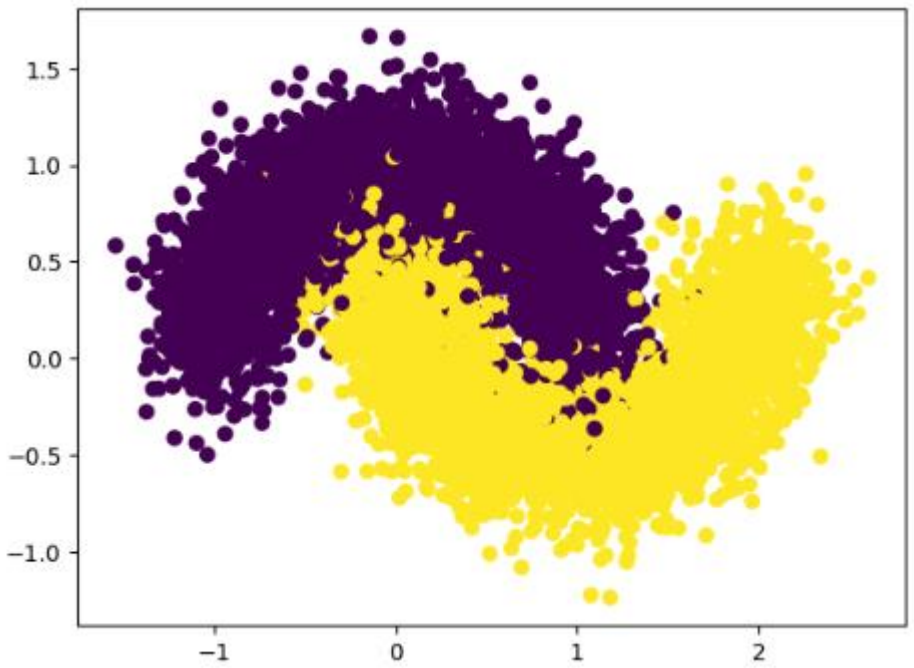

Рисунок 28 – Moons

Обе архитектуры должны хорошо справиться с задачей, однако полностью разделить классы не получится, поэтому будет интересно посмотреть, как обе сети справятся с задачей.

5.4.1 KAN

Для KAN подходящей архитектурой станет однослойная сеть с одним скрытым слоем с двумя нейронами, соответственно два входа (двумерное пространство) и один выход

model = KAN(width=[2,2,1], grid=3, k=3)

Модель показала следующую точность по метрикам: 0.9674285650253296 – На тренировочной выборке

0.9693333506584167 – На тестовой выборке

В данном параграфе обсудим лишь интерпретируемость лишь для нашей задачи. Удалось вывести символическую формулу, и она выглядит следующим образом (17):

$$0.51\tanh\left(64.23\sin(0.17x_1 - 7.57) + 60.6 - 5.68e^{-1.43(0.53x_2)^2}\right) + 0.49 + 1.05e^{-27.84\,(0.31\sin(1.92x_1+7.73)-1+0.75e^{-0.61(-x_2-0.18)^2})^2} \quad (17)$$

Точность данной формулы можно проверить независимо от самой модели, хотя, в теории, конечно, вывод данной формулы не должен отличаться от самой модели (Только с погрешностью на округление)

Точность на тестовой выборке = 0.9844

5.4.2 MLP

Для MLP, в отличие от восстановления зависимостей не понадобится огромных архитектур и 3000 эпох. Обучать будем модель следующего вида на 300 эпохах, с обучающим шагом = 0.0002

model = Sequential([Dense(8, activation='relu', input_shape=(2,)), Dense(4, activation='relu'), Dense(1, activation='sigmoid')])

Точность модели = 0.9679999947547913

5.4.3 Результаты сравнения: Классификация

- Модели получили одинаковые показатели метрик (KAN чуть точнее)
- KAN дольше обучалась
- Удалось получить символическую формулу для разделения классов

5.5 MNIST (hw)

В качестве более приближённой к реальности задачи я решил выбрать задачу MNIST, потому что это – по сути, классическая задача в машинном обучении и является многоклассовой классификацией, примеров которой выше не было

5.5.1 Формулировка задачи

На вход нам подаются изображения с написанными от руки числами, представленными на рисунке 29

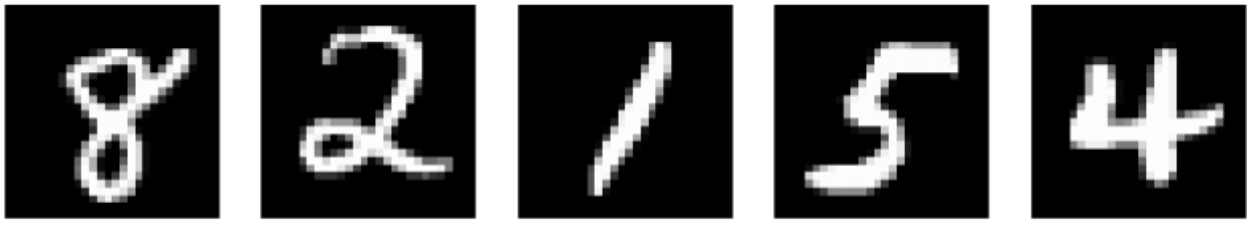

Рисунок 29 – Датасет MNIST

Надо обучить модель, которая для любого входа будет возвращать метку класса (число) на полученном изображении

Также я решил для задачи MNIST реализовать собственный алгоритм, который будет дублировать математическую модель оригинальной нейронной сети, то есть в качестве библиотек я импортирую только библиотеки **numpy** (Библиотека для работы с числовыми данными и многомерными массивами), **pandas** (Библиотека для работы с табличными и временными данными), **matplotlib** (Библиотека для визуализации в python), но отсюда рождается небольшая проблема

5.5.2 Проблема реализации архитектуры KAN

При реализации MLP ввиду фиксированных функций активации можно явно указать. Для всех компонентов (умножение, сложение, стандартные σ) уже известны аналитические формулы для производных. Все они заранее прописаны в библиотеке (PyTorch, TensorFlow). Соответственно можно те же аналитические градиенты прописать явно. Важно: Функция активации в MLP фиксированная. Её не нужно обучать. Ты обучаешь только веса W и смещения b, а градиенты по ним легко вычисляются.

В KAN обучаются параметры одномерных функций (обычно сплайны, piecewise linear, radial basis functions и т.д.). Во время обучения нужны градиенты функции потерь по этим параметрам. Это значит — нужно уметь делать автоматическое дифференцирование (autodiff), либо пользоваться приближёнными методами (например, численное дифференцирование), но тут возникает ещё большее количество проблем, таких как еще более медленное обучение, подбор дополнительного гиперпараметра ϵ, ошибки округления и проблемы с тяжёлыми структурами.

Ввиду всего вышеперечисленного было принято решение достать из библиотеки PyTorch алгоритм автодифференцирования, а остальную реализацию выполнить согласно бумаге KAN: Kolmagorov Arnold Networks. Однако я сохранил версию, которую назвал KAN_frozen_parametres, в которой можно дописать любой алгоритм.

5.5.3 Результаты сравнения: MNIST

- В случае сравнения полноценных библиотечных решений MLP выигрывает у KAN как в точности, так и во времени обучения
- Метрики для ручных реализаций получились почти одинаковые (0.8680 для KAN и 0.844 для MLP)

5.6 WineQuality

Данная задача также является классической задачей ML. Постановка задачи следующая: Нам на вход дают набор параметров для каждой бутылки (распределение по двум признакам представлено на рисунке 30)) вина и нам нужно определить метку класса (Их всего 3). По сути, задача проще MNIST, но для решения данной задачи я применял только библиотечные реализации KAN и MLP.

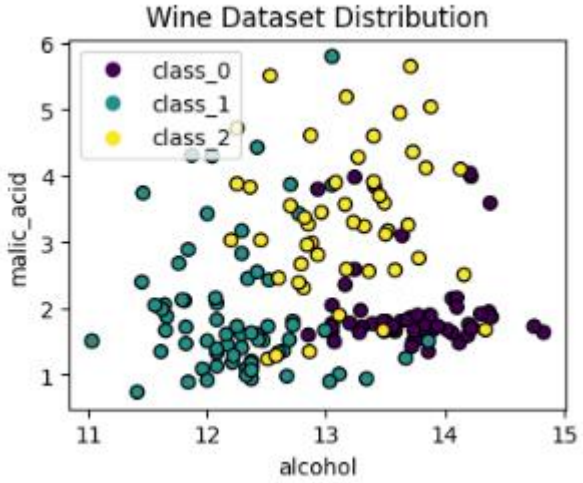

Рисунок 30 – Разделение на классы вин по первым двум фичам

5.6.1 MLP

Для данной задачи многоклассовой классификации производится минимальная предобработка данных и можно сразу строить модель. Для такой задачи можно использовать сетку следующей архитектуры

model = Sequential([

Dense(32, activation='relu', input_shape=(13,)),

Dense(8, activation='relu'),

Dense(3, activation='softmax')])

5.6.2 KAN

По аналогии с MLP требуется минимальная предобработка данных и дальше строится модель. Для такой задачи достаточно будет следующей сетки

model = KAN(width=[13,3,3], grid=3, k=4, device=device)

Также получилось по аналогии с Moons получить символическую формулу, которая, в отличие от задачи Moons выглядит уже гораздо сложнее и куда менее интерпретируемо, как можно видеть по рисунку 30.

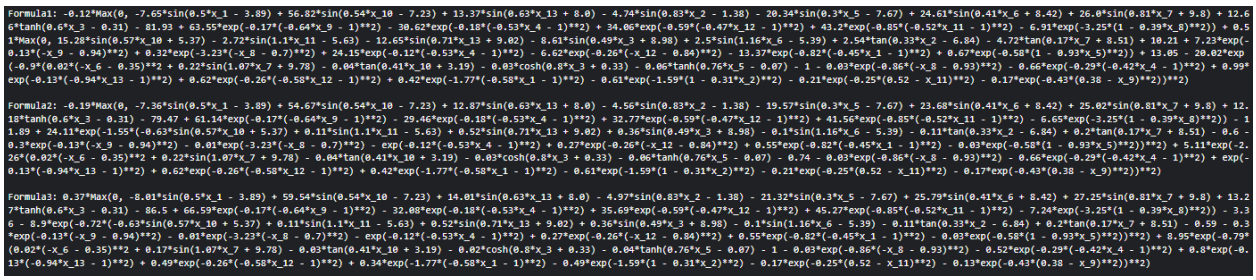

Рисунок 31 – Символическая формула WineQuality

5.6.3 Результаты сравнения: WineQuality

- Интерпретируемость при решении данной задачи не достигается символической формулой. Проще будет достичь её построив, например, дерево решений
- Обе архитектуры отлично справились с задачей (0.9844 – для KAN и для MLP)
- KAN обучается значительно дольше и не имеет смысла для подобных задач

5.7 Diffusion learning model

В процессе выполнения данной дипломной работы была поставлена задача реализации восстановления исходной зависимости с помощью диффузионной модели ввиду текущей популярности такого подхода. За основу был взят проект, в котором автор реализовал аппроксимацию датасета make_swiss_roll, который показан на рисунке 32:

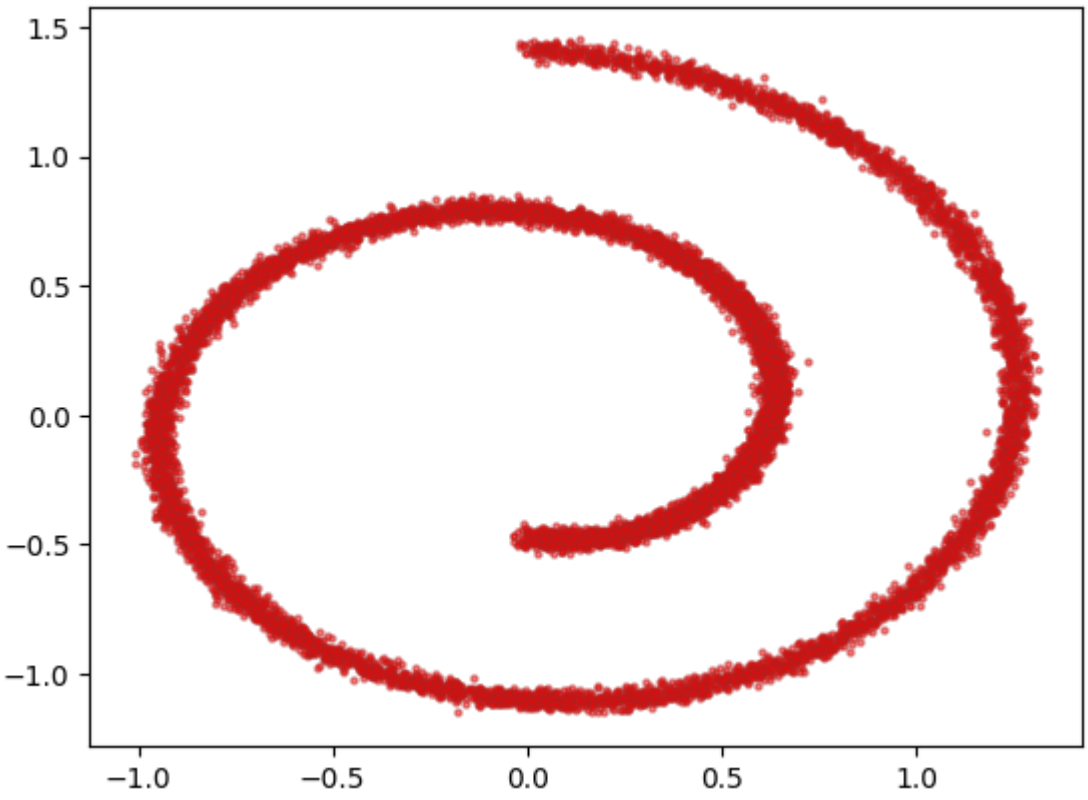

Рисунок 32 – Make_swiss_roll

Стандартно в первую очередь данные приближаются к Гауссовскому шуму, что носит название прямого процесса и показано на рисунке 33

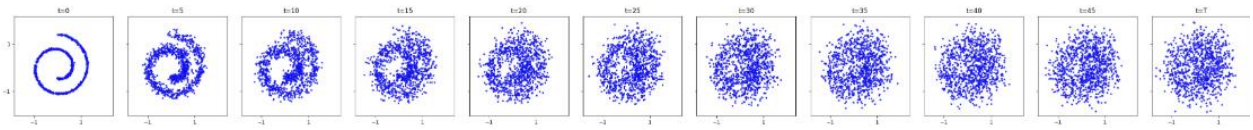

Рисунок 33 – Прямой процесс диффузии

Полное описание задачи и соответствующую реализацию с помощью MLP можно прочитать в статье Marzio на Github[5]. Я же реализовал данную задачу, используя KAN.

Нейронная сеть должна научиться предсказывать шум, который был добавден на шаге t. Обучение – это реконструкция шума, чтобы inference восстановить x_0 из x_T. По ходу реализации возникало множество проблем. Например, потеря информации через t/T, неустойчивость генерации (обратного процесса) и переобучение на неудачном количестве данных, а также тяжёлая настройка гиперпараметров и долгое обучение.

Для реализации такой задачи были выбраны следующие параметры: T = 250 (число шагов диффузии), β_all = 0.025**2 * torch.ones(T + 1) (шум по времени), batch_size = 8192, lr = 0.002, steps = 4000, KAN([3, 32]), KAN([32, 32]), KAN([32, 32]), KAN([32, 2]), loss function = huber_loss, num_samples = 10000.

В результате выполнения программы получаем аппроксимацию на рисунке 35. На рисунке 34 показан процесс обучения:

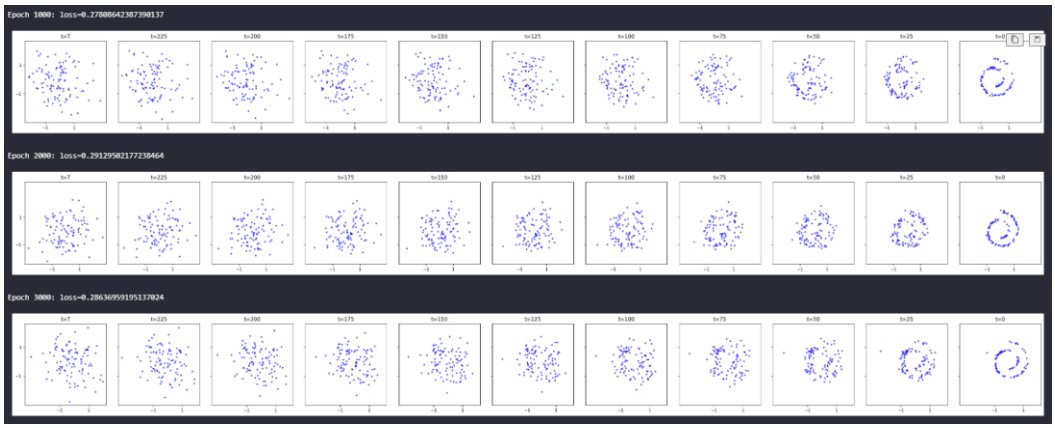

Рисунок 34 – процесс обучения

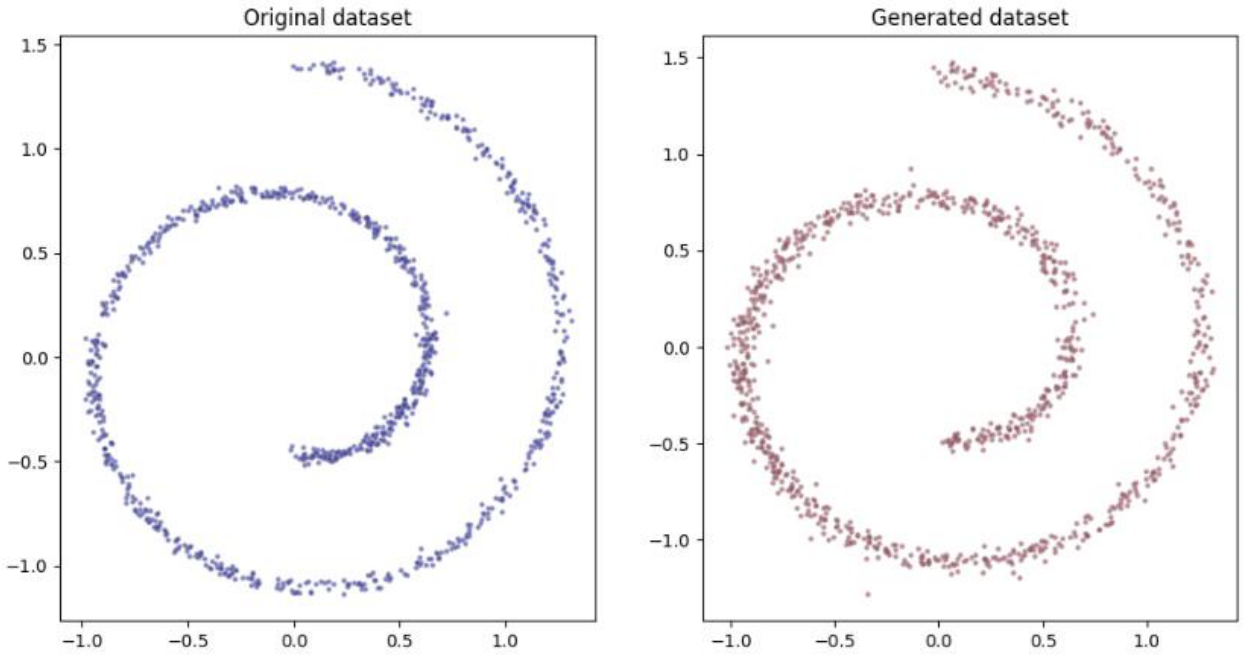

Рисунок 35 – результат обучения

Получилось неплохо, но идёт ли это в сравнение с MLP? Для ответа на этот вопрос я перенес код для обучения MLP из оригинальной статьи и провел сравнение двух архитектур на рисунках 36 и 37

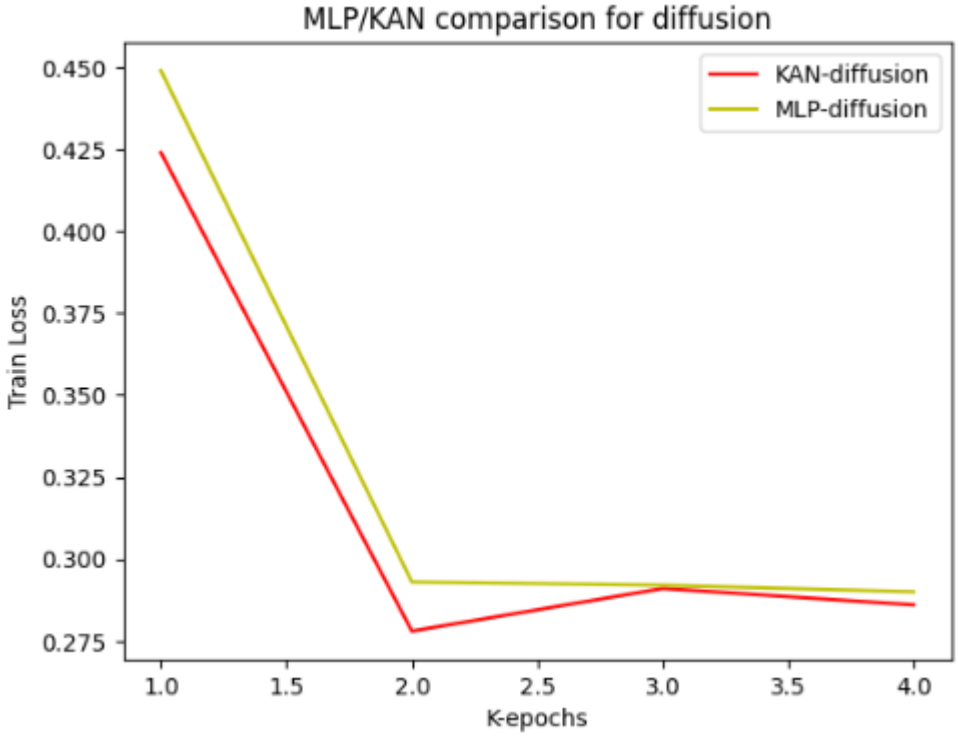

Рисунок 36 – сравнение kan, mlp diffusion

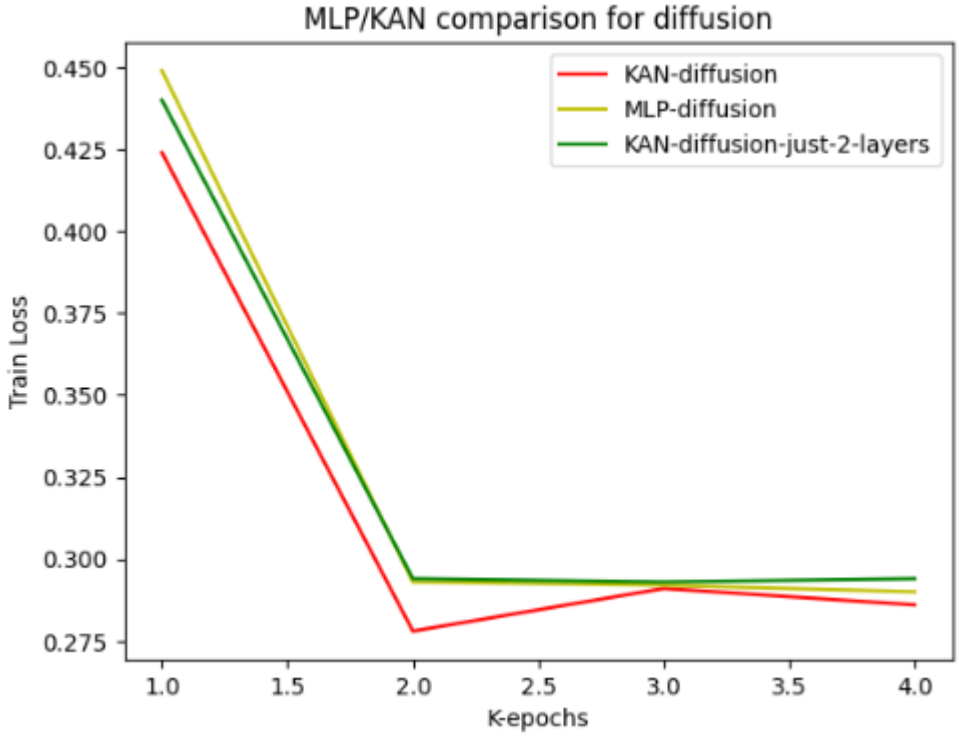

Рисунок 37 – 2 - слойная kan for diffusion

Как можно заметить, KAN для данной задачи получает лучшую точность, но при этом её очень тяжело настраивать и обучается в 10 – 20 раз дольше.

6 ВЫВОДЫ

В данной работе проведён сравнительный анализ архитектур нейронных сетей MLP (Multi-Layer Perceptron) и KAN (Kolmogorov-Arnold Network), как с теоретической, так и с практической точек зрения. Исследование включало в себя математическое обоснование особенностей обеих архитектур, а также серию численных экспериментов на синтетических и реальных данных. На основе проделанной работы можно сделать следующие выводы:

1. Качество аппроксимации сложных функций: В задачах аппроксимации сложных нелинейных зависимостей архитектура KAN продемонстрировала сопоставимую или лучшую точность по сравнению с MLP. Это связано с тем, что KAN более естественно строит сложные функции за счёт явного разложения их на суперпозиции одномерных функций, что соответствует теореме Колмогорова.
2. Обобщающая способность: KAN в ряде экспериментов демонстрировал лучшую способность к обобщению на новых данных по сравнению с MLP. Особенно это проявляется в случаях, когда исходная функция содержит высокую степень нелинейности, а объём обучающей выборки ограничен.
3. Скорость обучения и сложность настройки: MLP, как более изученная и распространённая архитектура, проще в настройке и имеет большое количество проверенных на практике рекомендаций по подбору гиперпараметров. В то же время KAN требует более тщательного подбора параметров, в том числе выбора форм одномерных функций и методов их аппроксимации.
4. Объём необходимых вычислительных ресурсов: В ряде случаев KAN может требовать меньшего количества параметров для достижения аналогичной точности, что положительно сказывается на объёме памяти и вычислительных ресурсах. Однако сложность реализации KAN несколько выше.

5. Интерпретируемость: Основная проблема MLP заключается в том, что архитектура крайне неинтерпретируема, даже получила свое популярное название “Черный ящик”. Архитектура KAN для определенного набора задач крайне эффективно может показывать вклад каждой отдельной переменной и даже выводить символическую формулу
6. Практическая применимость: MLP остаётся универсальным инструментом, подходящим для широкого спектра задач, в том числе в области компьютерного зрения, обработки текста и временных рядов. KAN, благодаря своей способности эффективно аппроксимировать сложные функции при ограниченных данных, может найти применение в задачах научного моделирования, численного анализа и инженерных расчётах.

Таким образом, обе архитектуры имеют свои сильные и слабые стороны. Выбор между ними зависит от конкретной задачи, объёма доступных данных, требований к точности и интерпретируемости, и доступных вычислительных ресурсов. Перспективным направлением для дальнейших исследований видится развитие гибридных моделей, сочетающих сильные стороны обеих архитектур.

7 ЗАКЛЮЧЕНИЕ

После исследования данной темы можно выделить несколько направлений для будущего развития. Несмотря на бурные обсуждения и относительную популярность самой статьи – есть куда менять существующую реализацию (это в том числе подтверждают существующие реализации улучшенной KAN)

Модификация базовой конструкции разложения Колмагорова.

В классической теореме многомерная функция (18) представляется в следующем виде:

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \Psi_{q,p}(x_p) \right), \quad (18)$$

Однако, данное представление не является единственным возможным. Например, существует более перспективное разложение Лацковича (19), созданное на базе теоремы Колмагорова – Арнольда

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{m} \Phi_q \left(\sum_{p=1}^{n} \gamma_{q,p} \Psi_p(x_p) \right), \quad (19)$$

Преимущество такого разложения в том, что количество композиций в сумме увеличивается линейно с ростом параметров, в отличие от квадратичного роста оригинального разложения.

Увеличение набора функций для лучшей интерпретируемости

В библиотеке, реализованной студентами из MIT есть крайне ограниченный набор базисных функций для аппроксимации одномерных компонент. Это ограничивает способность сети быть интерпретируемой, что является основным плюсом нейросети. Например, гиперболические функции.

Оптимизация вычислительных алгоритмов библиотеки

С практической точки зрения, официальная библиотека KAN может быть улучшена за счёт:

- Поддержки более эффективного пакетного обучения на GPU.
- Поддержки автоматического выбора порядка разложения в зависимости от сложности данных.
- Более тесной интеграции с современными фреймворками (PyTorch, JAX, TensorFlow).
- Введения гибридных моделей: совместного использования MLP и KAN в единой архитектуре.

Регуляризация взаимодействий между компонентами

Для уменьшения переобучения и повышения устойчивости к шуму можно ввести дополнительные регуляризаторы, которые ограничивают сложность каждой одномерной компоненты, а также контролируют силу взаимодействия между ними. Это позволит сохранять высокую точность при меньшем числе параметров.

8 СПИСОК ЛИТЕРАТУРЫ

1. Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, Max Tegmark; KAN: Kolmagorov Arnold Networks, 2024
2. G. Cybenko: Approximation by Superpositions of a Sigmoidal Function, 1989
3. Kurt Hornik, Maxwell Stinchcobe, Halbert White: Multilayer Feedforward Networks are Universal Approximators, 1989
4. Rubens Zimbres: Kolmagorov Arnold Networks a critique, 2025
5. M. Marzio: An introduction to diffusion models, 2025
6. Avi Chawla: A Beginner-friendly Introduction to Kolmagorov Arnold Networks (KAN)
7. Leonardo Ferreira Guilhoto, Paris Perdikaris: Deep Learning Alternatives of the Kolmagorov Superpositions Theorem, 2025
8. Zhangchi Zhao, Jun Shu, Deyu Meng, Zongben Xu: Improving Memory Efficiency for Training KANs via Meta Learning
9. Yuntian Hou, Di Zhang, Jinheng Wu, Xiaohang Feng: A Comprehensive Survey on Kolmagorov Arnold Networks (KAN), 2024
10. Simon Haykin. Neural networks: a comprehensive foundation. Prentice Hall PTR, 1994.
11. A.N. Kolmogorov. On the representation of continuous functions of several variables as superpositions of continuous functions of a smaller number of variables. Dokl. Akad. Nauk, 108(2), 1956.
12. Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In Doklady Akademii Nauk, volume 114, pages 953–956. Russian Academy of Sciences, 1957.
13. Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. arXiv preprint arXiv:1807.03973, 2018.

14. Federico Girosi and Tomaso Poggio. Representation properties of networks: Kolmogorov's theorem is irrelevant. Neural Computation, 1(4):465–469, 1989.
15. Carl De Boor. A practical guide to splines, volume 27. springer-verlag New York, 1978.
16. Ming-Jun Lai and Zhaiming Shen. The kolmogorov superposition theorem can break the curse of dimensionality when approximating high dimensional functions. arXiv preprint arXiv:2112.09963, 2021
17. Yongji Wang and Ching-Yao Lai. Multi-stage neural networks: Function approximator of machine precision. Journal of Computational Physics, page 112865, 2024.