

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** |

**О Т Ч Е Т**

о прохождении производственной практики.

(научно – исследовательская работа)

направление подготовки 01.03.02 «Прикладная математика и информатика»

профиль «Математические и компьютерные технологии»

|  |  |  |
| --- | --- | --- |
|  |  | Выполнил студент  гр. Б9121-01.03.02мкт  Домашев С.А. \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Отчет защищен:  с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | *(Ф.И.О.) (подпись)*  Руководитель практики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(должность, уч.звание)*  Кузнецов К.С.  *(Ф.И.О.) (подпись)*  «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025г. |
| Рег. № \_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |  | Практика пройдена в срок  с «23» апреля 2025 г.  по «05» мая 2025 г.  (2 недели, рассредоточенная)) |

г. Владивосток

2025

Оглавление

[Аннотация 3](#_Toc200046281)

[Введение 4](#_Toc200046282)

[Постановка задачи 5](#_Toc200046283)

[Требования к окружению 6](#_Toc200046284)

[Требования к аппаратному обеспечению 6](#_Toc200046285)

[Минимальное требования (Для прототипа/разработки) 6](#_Toc200046286)

[Рекомендуемые требования (для экспериментов с большими данными) 6](#_Toc200046287)

[Требования к программному обеспечению 6](#_Toc200046288)

[ОС: 6](#_Toc200046289)

[Язык программирования и библиотеки: 6](#_Toc200046290)

[Средства визуализации и документации: 7](#_Toc200046291)

[Требования к пользователям 7](#_Toc200046292)

[Базовые знания: 7](#_Toc200046293)

[Желательные навыки: 7](#_Toc200046294)

[Введение в предметную область 8](#_Toc200046295)

[Математическая теорема Колмагорова – Арнольда 8](#_Toc200046296)

[Универсальная аппроксимационная теорема 9](#_Toc200046297)

[Принцип работы MLP 10](#_Toc200046298)

[Принцип работы KAN 17](#_Toc200046299)

[Процесс обучения. 18](#_Toc200046300)

[Основная часть 19](#_Toc200046301)

[Общая математическая модель 19](#_Toc200046302)

[Замечание: Обобщение MLP до KAN 20](#_Toc200046303)

[Игрушечные датасеты 21](#_Toc200046304)

[Заключение 26](#_Toc200046305)

[Список литературы 27](#_Toc200046306)

Нейронные сети архитектуры KAN

# 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

# Введение

Нейронные сети играют ключевую роль в развитии искусственного интеллекта, обеспечивая прорывы в таких областях, как компьютерное зрение, обработка естественного языка и научные вычисления. Среди наиболее распространённых архитектур выделяются многослойные персептроны (MLP), которые, несмотря на свою эффективность, сталкиваются с ограничениями в интерпретируемости и вычислительной эффективности, и особенно при решении задач с малой размерностью данных.

В последние годы 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 унивариатные функции на рёбрах сети, полностью исключая линейные веса. Обычно эти функции параметризуются сплайнами, 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, решении уравнений с частными производными и анализе данных высокой размерности, таких как временные ряды, графовые данные и классификация гиперспектральных изображений ([A Comprehensive Survey on Kolmogorov Arnold Networks (KAN)](https://arxiv.org/html/2407.11075v1)). 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".

# 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 унивариатных функций для представления многомерных функций.
* **Архитектурный дизайн**: Подробное описание структуры KAN, включая использование сплайнов для параметризации функций, отсутствие линейных весов и сравнение с MLP. Например, исследования указывают, что KAN заменяют веса на функции, параметризованные сплайнами, что, вероятно, улучшает адаптивность модели ([KAN: Kolmogorov-Arnold Networks](https://arxiv.org/abs/2404.19756)).
* **Оценка производительности**: Анализ эффективности KAN в различных приложениях, таких как аппроксимация данных, решение УЧП и обработка данных высокой размерности, с сравнением с традиционными MLP и другими архитектурами. ([Kolmogorov-Arnold Network](https://www.geeksforgeeks.org/kolmogorov-arnold-network/" \t "_blank)).
* **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 ([A Beginner-friendly Introduction to Kolmogorov Arnold Networks (KAN)](https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/)).
* **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. Исследования указывают на возможные проблемы с переобучением и длительным временем обучения, что требует дальнейших исследований ([Kolmogorov-Arnold Networks: a Critique](https://medium.com/@rubenszimbres/kolmogorov-arnold-networks-a-critique-2b37fea2112e" \t "_blank)).

# Требования к окружению

## Требования к аппаратному обеспечению

Такой 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:

### 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

### 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)

## Требования к программному обеспечению

### ****ОС:****

* **Linux** (Ubuntu, Arch, Debian и др.) — предпочтительно
* **Windows / macOS** — возможно, но может потребовать дополнительных настроек

### ****Язык программирования и библиотеки:****

* **Python 3.9+**
* **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

### ****Средства визуализации и документации:****

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

## Требования к пользователям

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

### ****Базовые знания:****

* Основы линейной алгебры, математического анализа и теории функций
* Понимание принципов машинного обучения и нейронных сетей (например, MLP, SGD, переобучение)
* Навыки работы с Python и популярными ML-фреймворками (PyTorch/TensorFlow)

### ****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)

# Введение в предметную область

## 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](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0), [теорию чисел](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%87%D0%B8%D1%81%D0%B5%D0%BB), [геометрию](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F), [топологию](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D0%BF%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F), алгебраическую геометрию, [группы Ли](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%9B%D0%B8), вещественный и комплексный анализ, [дифференциальные уравнения](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5), математическую физику, [теорию вероятностей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9), а также [вариационное исчисление](https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5). 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 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 на компактном множестве существует представление вида:

где ​ - непрерывные функции двух переменных, а ​ - 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 представима через суперпозицию функций одной переменной:

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

## 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 [Approximation by Superpositions of a Sigmoidal Function](https://www.sci-hub.ru/10.1007/bf02551274) 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 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 [Multilayer Feedforward Networks are Universal Approximators](https://www.sci-hub.ru/10.1016/0893-6080(89)90020-8) 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 с любой заданной точностью, при условии, что функция активации является нелинейной и удовлетворяет определённым условиям (например, сигмоида или ReLU).

## Принцип работы MLP

Многослойный перцептрон (MLP, Multilayer Perceptron) — это тип искусственной нейронной сети, который 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.

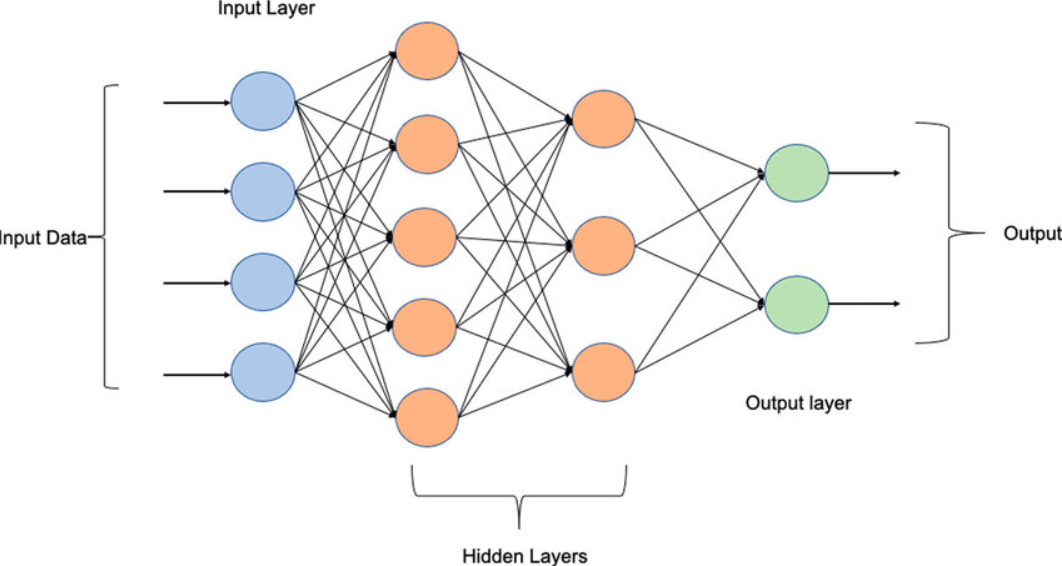


Рис. 1 – визуализация MLP

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 (Forward Propagation)**: сеть принимает входные данные, обрабатывает их через слои и выдаёт прогноз.
* **Обратное распространение (Backpropagation)**: сеть анализирует ошибку прогнозирования и корректирует свои параметры, чтобы улучшить результат в будущем.

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.

#### Шаг 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.

#### Шаг 2: Скрытые слои

Данные с 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)**: обнуляет отрицательные значения, оставляя положительные без изменений. Это ускоряет обучение и делает сеть устойчивой.

* **Сигмоида**: преобразует значение в диапазон от 0 до 1, что полезно для вероятностей.
* **Tanh**: преобразует значение в диапазон от -1 до 1, что хорошо для центрированных данных.

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

#### Шаг 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

#### Шаг 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)**: используется для регрессии, измеряет среднее квадратичное отклонение предсказания от истины.
* **Бинарная кросс-энтропия**: используется для бинарной классификации, оценивает, насколько предсказанные вероятности соответствуют истинным меткам.
* **Категориальная кросс-энтропия**: используется для многоклассовой классификации, оценивает качество вероятностного распределения.

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.

#### Шаг 1: 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).

#### Шаг 2: 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).

#### Шаг 3: 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.

#### Шаг 4: 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.

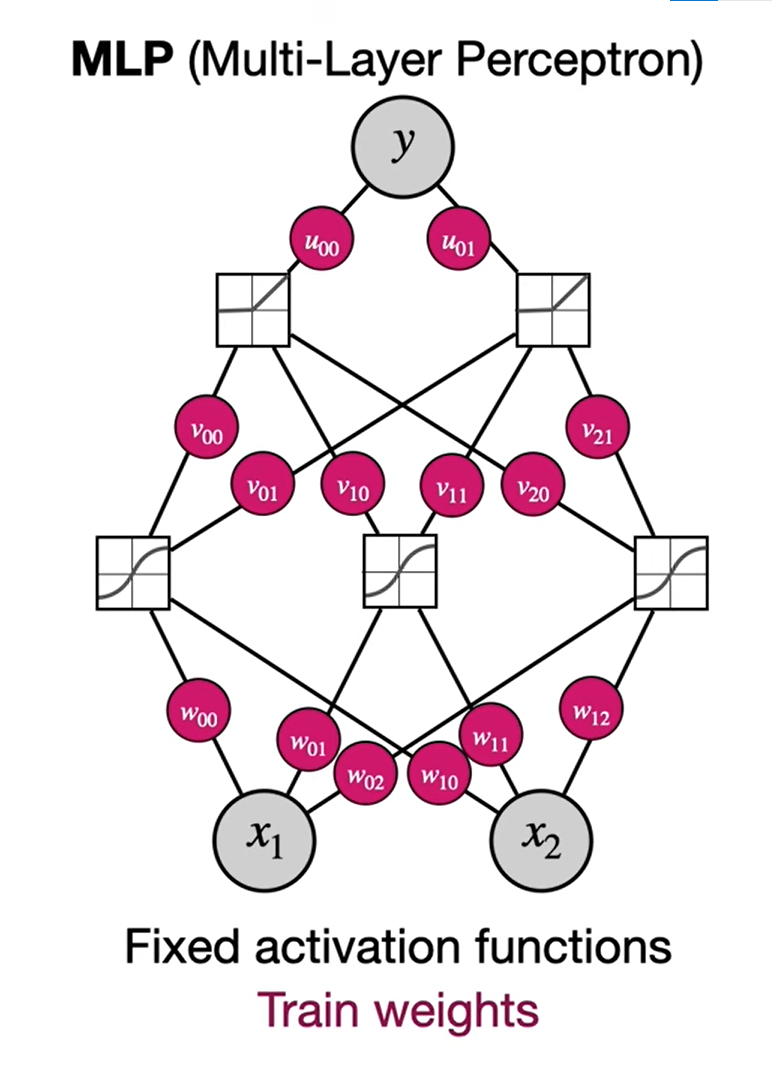


Рис. 2 – Визуализация процесса обучения MLP

На рис. 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.

## 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.

### Процесс обучения.

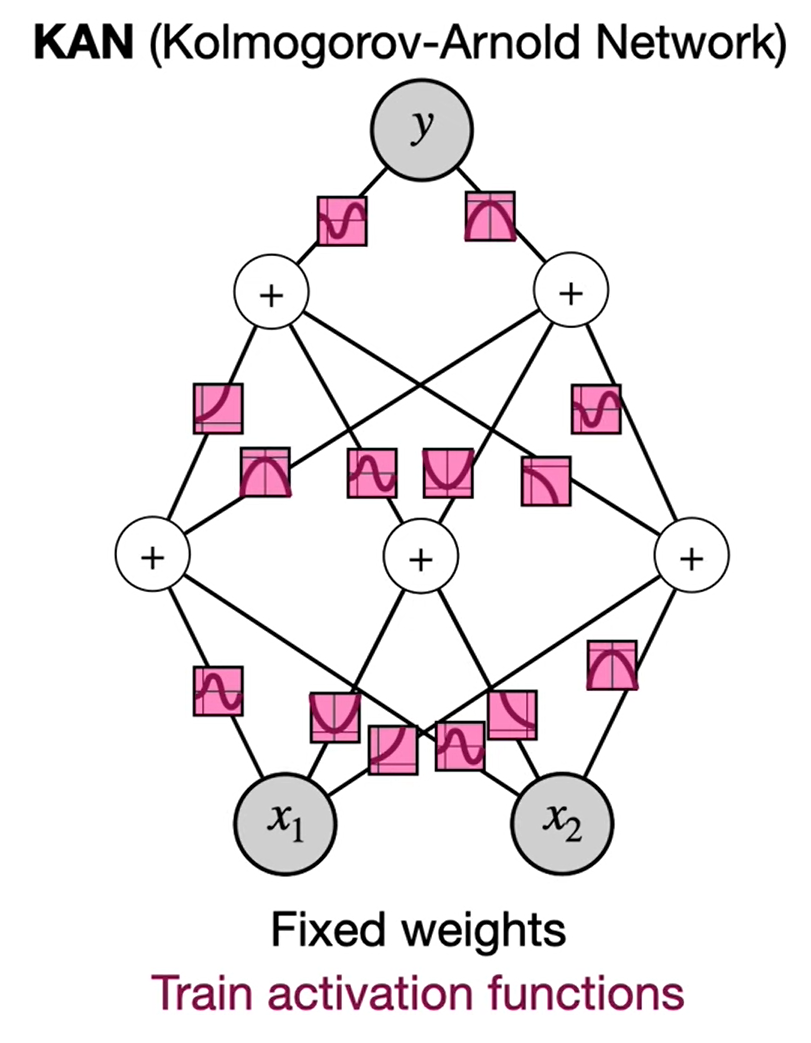


Рис. 3 – визуализация процесса обучения KAN

На данной визуализации 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

## Как обучить функцию?

### 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

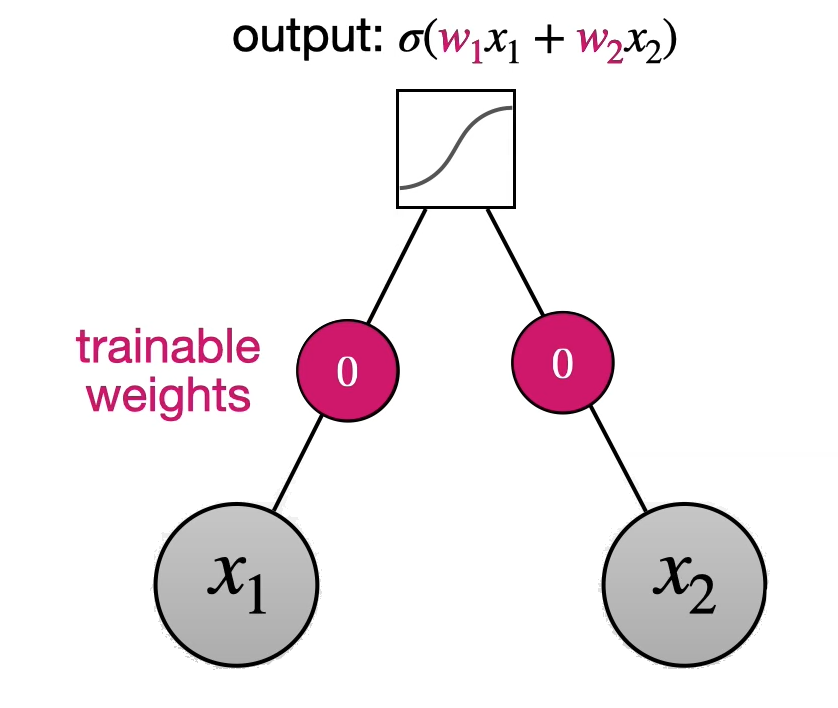


Рис. 4 – простая структура MLP

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** можно найти оптимальные веса для нашей модели

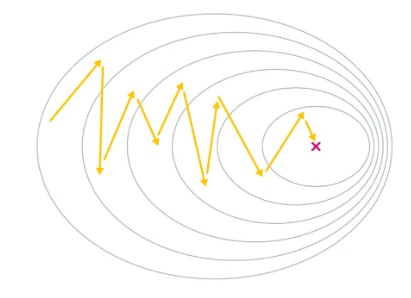


Рис. 5 - Градиентный спуск

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?

### 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:

Такая функция представляет собой комбинацию базисов , а веса обучаются.

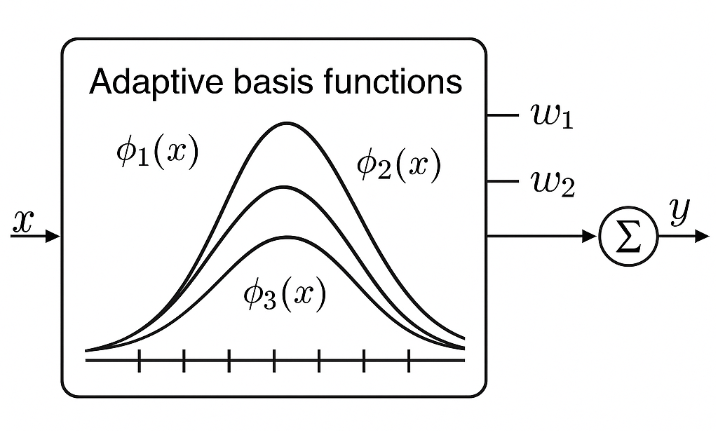
**

Рис. 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 это выглядит следующим образом:

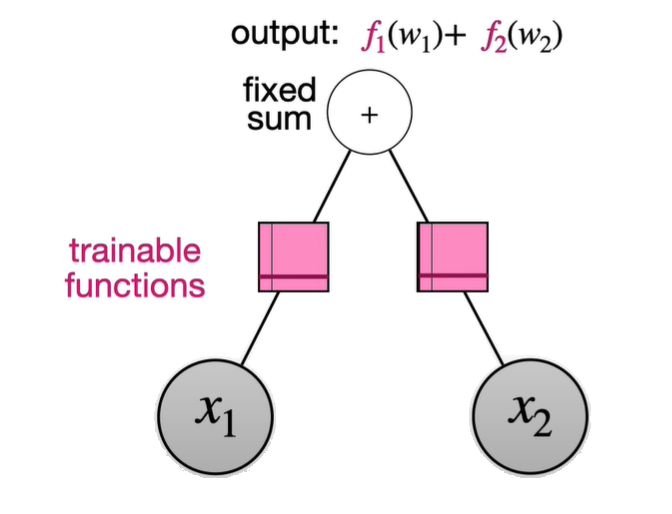


Рис. 7 – Простая структура KAN

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

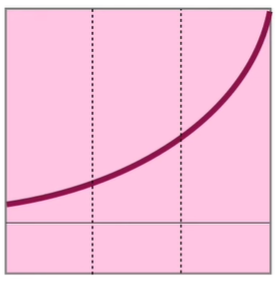


Рис. 8 – Искомая функция

Тогда мы должны также обозначить базисные сплайны. В простейшем случае можно взять обычные константные прямые (Рисунок 9)

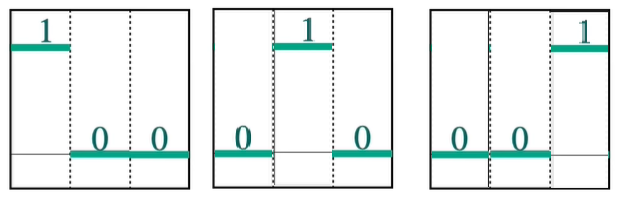


Рис. 9 – базисные функции

Тогда, подбирая параметры для **линейной комбинации** данных базисных сплайнов, получим коэффициенты 0.1, 0.3, 0.7 соответственно и приближение получится следующим:

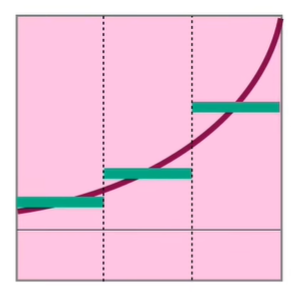


Рис. 10 – Адаптивная функция

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.

# Основная часть

## Общая математическая модель

KAN семейство нейронных сетей выглядит следующим образом:

Согласно нашей аппроксимационной теореме, искомую функцию можно разложить на суперпозиции суммы функций одной переменной . В последнем слое означает, что функция может быть векторозначной

Где каждая сумма представляет собой преобразование подобного рода

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 очеред BSKAN представляется следующим образом:

## 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. .

В контексте такой интерпретации слоя нейронной сети в случае KAN каждая функция 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. Тогда будем иметь один скрытый слой, произвольное число нейронов, и сигмоиду в качестве функции активации:

Тогда первый слой будет иметь следующий вид:

А второй слой будет иметь следующий вид:

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

## 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.

### Введение: Простая одномерная функция

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

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.

### 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).

### 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

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 и нам не годится (третья архитектура в данном примере – хорошая аппроксимация для сравнения)

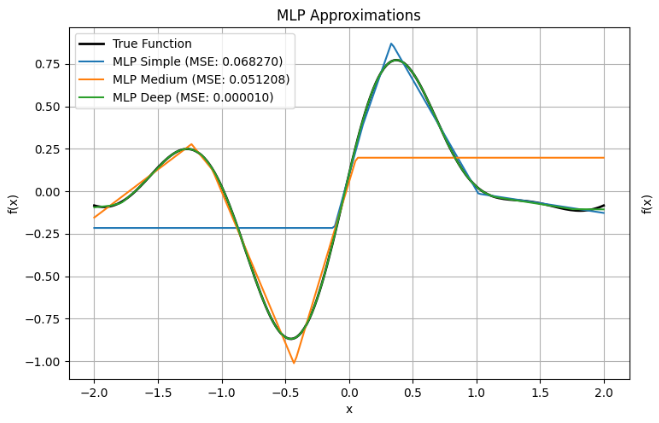


Рис. 11 - (MLP)

### 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

**Структура функции:**

Функция представляет собою сумму двух компонент: произведения периодической функции с умеренной частотой и гауссовской функции , которая обеспечивает затухание и второй компоненты , 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

### 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

**Результаты** были следующие:

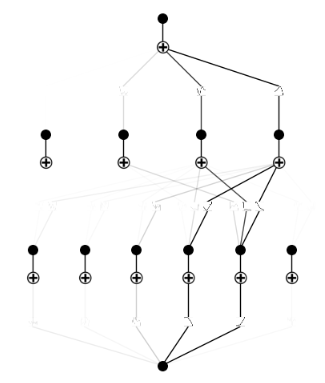


Рис. 12 – Архитектура [1, 6, 4, 1] KAN

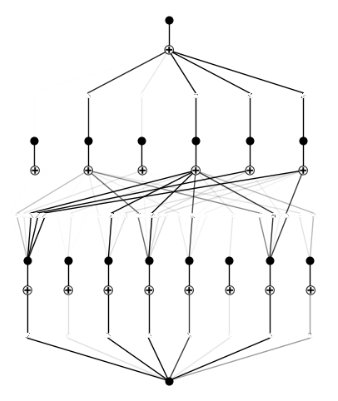


Рис. 13 – Архитектура [1, 8, 6, 1] KAN

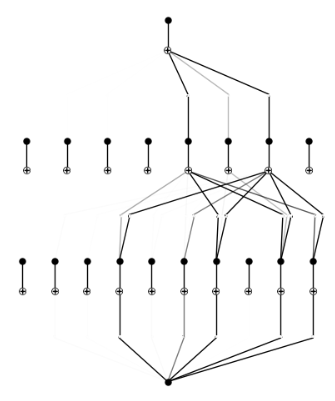


Рис. 14 – Архитектура [1, 10, 8, 1] KAN

Из данных визуализаций уже можно заметить, что архитектура для нашей зависимости избыточна. Аппроксимация выглядит следующим образом:

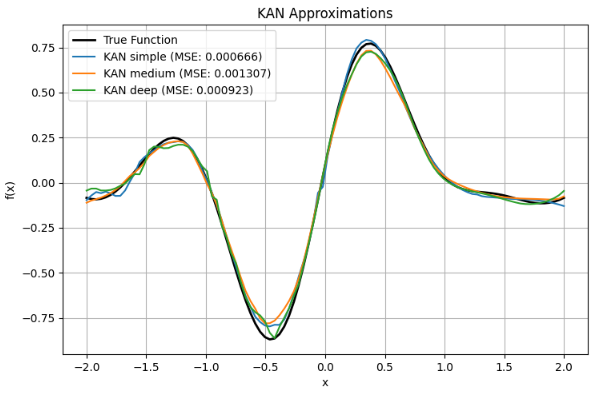


Рис. 15 – Визуализация аппроксимации переобученных моделей

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. Вот пара примеров:

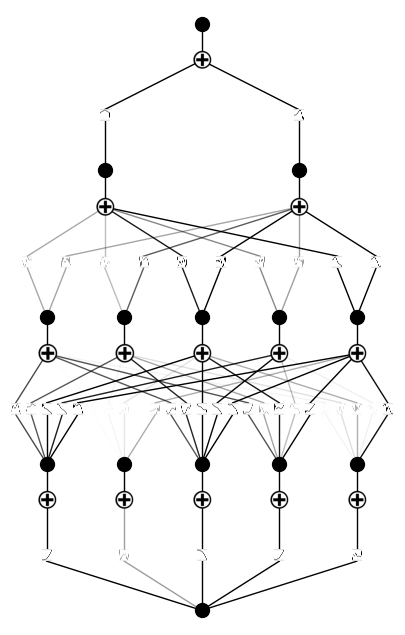


Рис. 16 - Архитектура [1, 5, 5, 2, 1] KAN

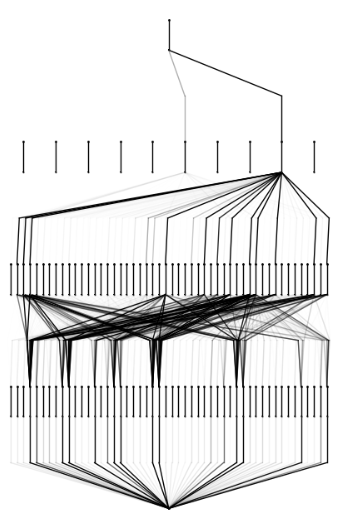


Рис. 17 - Архитектура [1, 50, 50, 10, 1] KAN

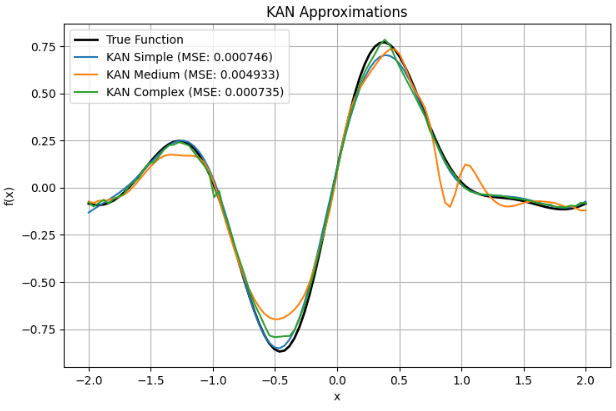


Рис. 18 – Результаты аппроксимации для избыточных архитектур

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"},

### Результаты сравнения: Простая аппроксимация

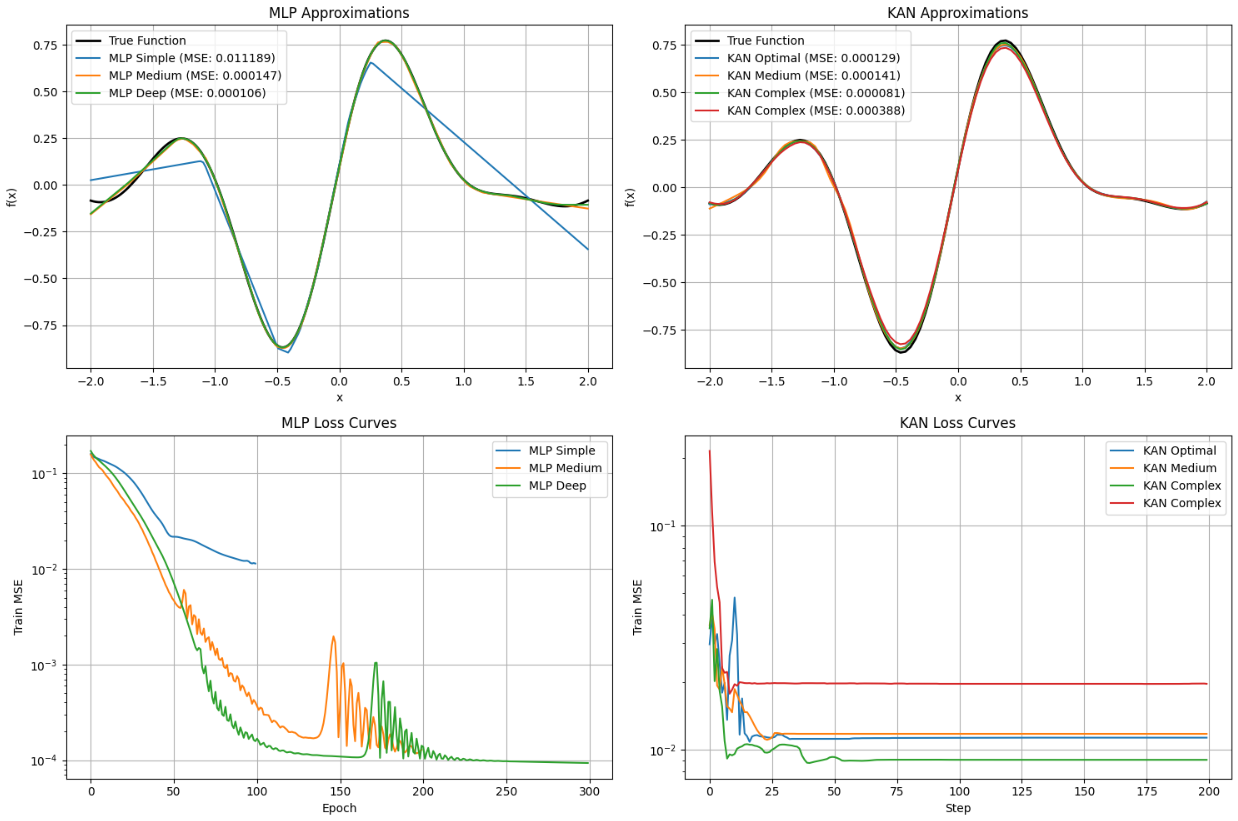


Рис. 19 – Результаты первого эксперимента

**Предварительные выводы:**

* На обучающей выборке MLP показывает лучшие результаты, чем KAN, однако на тестовой выборке метрика получается лучше именно у KAN
* 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

### Введение: Комплексная одномерная функция

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

Функцию рассмотрим следующую:

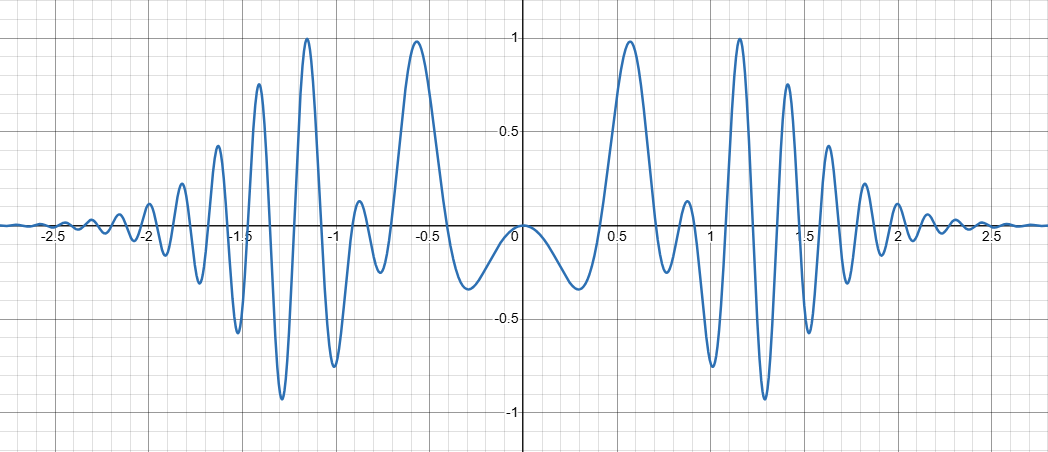


Рис. 20 -

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

### Архитектура

В случае **MLP** с предварительной настройкой рассматривать особо нечего, хотя я усилил архитектуру на фоне предыдущего раза. Если применять ту же архитектуру, получим следующие результаты:

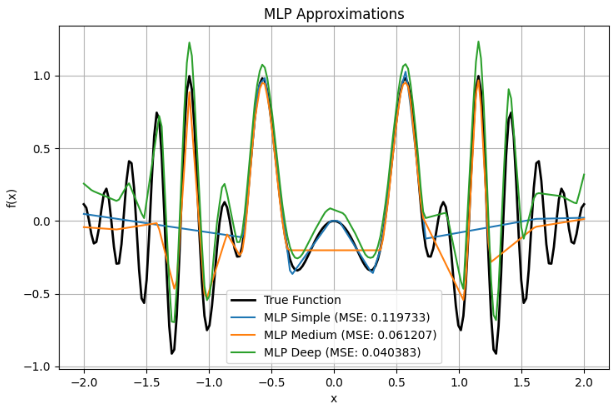


Рис. 21 – Слабая MLP

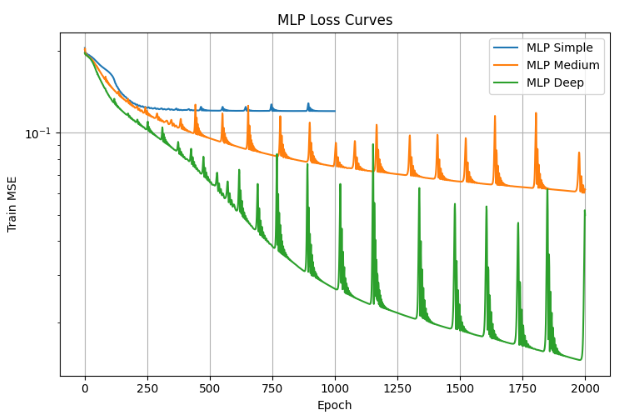


Рис. 22 – Падение метрики слабой MLP для комплексной функции

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 выглядят следующим образом:

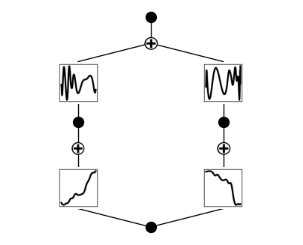


Рис. 23 - Архитектура [1, 2, 1] KAN

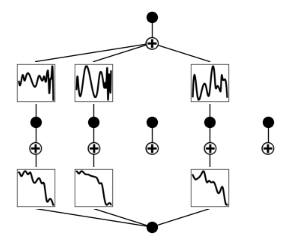


Рис. 24 - Архитектура [1, 5, 1] KAN

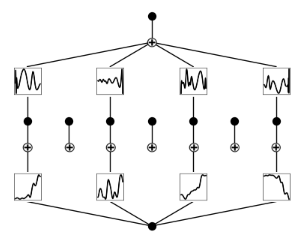


Рис. 25 - Архитектура [1, 7, 1] KAN

### Результаты сравнения: Комплексная одномерная функция

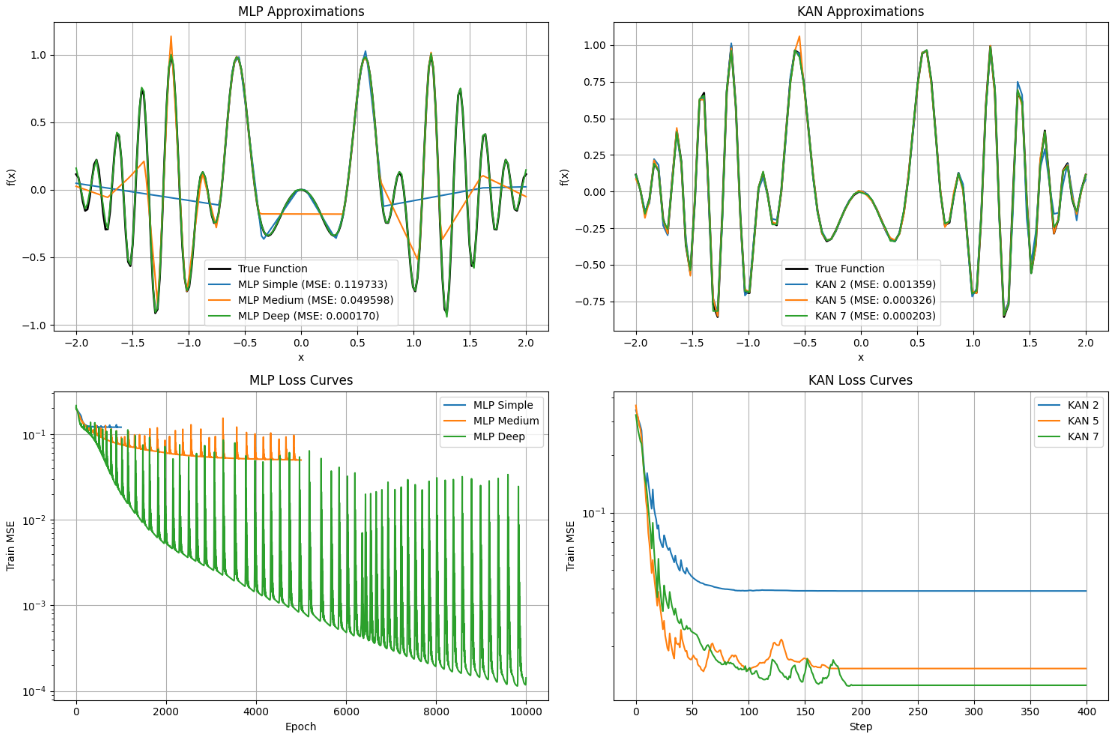


Рис. 26 – Результаты второго эксперимента

**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

# 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