Министерство науки и высшего образования Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

Факультет информационных технологий и радиоэлектроники

Кафедра информационных систем

КУРСОВАЯ

РАБОТА

по курсу Моделирование информационных систем

на тему: Моделирование аффинных преобразований над признаковым пространством

Руководитель

к. т. н., доц. каф. ИС

(уч. степень, звание)

Еремеев С. В.

(оценка) (фамилия, инициалы)

(подпись) (дата)

Члены комиссии Студент ИС - 122 (группа)

Клинцов С. М.

(подпись) (Ф.И.О.) (фамилия, инициалы)

(подпись) (Ф.И.О.) (подпись) (дата)

Муром 2025

В ходе курсовой работе было разработано приложение для применения различных аффинных преобразований к нейронным сетям:

* обучены сети различной сложности;
* реализован алгоритм применения аффинных преобразований к весам обученной модели;
* разработан интерфейс взаимодействия с пользователем;
* проведено тестирование и отладка проекта.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc216626152)

[1. Анализ технического задания 8](#_Toc216626153)

[1.1 Языки программирования: Java, Python 8](#_Toc216626154)

[1.2 Нейронные сети: Eclipse Deeplearning4J 9](#_Toc216626155)

[1.3 Среда разработки: VScode 9](#_Toc216626156)

[2. Разработка модели 10](#_Toc216626157)

[2.1 Создание однослойной бинарной модели “Треугольник” 10](#_Toc216626158)

[2.2 Визуализация модели 10](#_Toc216626159)

[2.3 Знакомство с аффинными преобразованиями 14](#_Toc216626161)

[2.4 Аффинное преобразование поворота 15](#_Toc216626162)

[2.5 Аффинное преобразование растяжение/сжатие 19](#_Toc216626165)

[2.6 Аффинное преобразование расширения 23](#_Toc216626168)

[2.7 Создание многослойной бинарной модели “Два треугольника” 25](#_Toc216626170)

[2.7 Визуализация применения аффинных преобразований к модели “Два треугольника” 28](#_Toc216626172)

[2.8 Создание многослойной модели “MNIST” 31](#_Toc216626173)

[3. Исследование работы модели 35](#_Toc216626174)

[3.1 Анализ результатов экспериментов 36](#_Toc216626175)

[ЗАКЛЮЧЕНИЕ 37](#_Toc216626176)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_Toc216626177)

[Приложение A «Таблица экспериментов с поворотом на 256 градусов для модели Два треугольника» 39](#_Toc216626178)

[Приложение B «Таблица экспериментов с поворотом на 256 градусов для модели MNIST» 40](#_Toc216626179)

[Приложение C «Таблица экспериментов с растяжением в 2 раза для модели Два треугольника» 41](#_Toc216626180)

[Приложение D «Таблица экспериментов с растяжением в 2 раза для модели MNIST» 42](#_Toc216626181)

[Приложение E «Таблица экспериментов с расширением на коэффициент 0.2 для модели Два треугольника» 43](#_Toc216626182)

[Приложение F «Таблица экспериментов с расширением на коэффициент 0.2 для модели MNIST» 44](#_Toc216626183)

**ВВЕДЕНИЕ**

Современное машинное обучение, особенно в области глубоких нейронных сетей, достигло невероятных успехов в решении сложных задач, от компьютерного зрения до обработки естественного языка. Однако эта мощь сопряжена с существенными вызовами, среди которых — вычислительная дороговизна дообучения, а также растущая потребность в управлении и интерпретации уже развёрнутых моделей. Обученная нейросетевая модель представляет собой сложную иерархическую структуру, где веса каждого слоя кодируют определённые паттерны в данных. Прямое манипулирование этими весами без потери качества предсказаний задача не из простых, так как даже незначительные изменения могут привести к искажению выходов.

В этом контексте концепция аффинных преобразований над признаковым пространством приобретает ключевое значение. Признаковое пространство — это абстрактное многомерное пространство, в котором данные представлены векторами признаков, извлечённых слоями нейронной сети. Аффинные преобразования (линейные трансформации, такие как поворот, масштабирование, сдвиг и их комбинации) являются фундаментальным инструментом геометрии и линейной алгебры. Их применение к признаковым пространствам обученных моделей открывает принципиально новые возможности: целенаправленное изменение внутренних представлений, коррекцию смещений (bias), сжатие или адаптацию модели к смещённым распределениям данных — и всё это без риска разрушения уже усвоенных знаний.

Таким образом, актуальность данной работы обусловлена необходимостью разработки эффективных и безопасных методов постобработки обученных моделей, которые позволили бы вносить в них управляемые изменения, сохраняя при этом исходную точность предсказаний.

Исследования в области манипуляций с весами и представлениями нейронных сетей ведутся, но носят зачастую фрагментарный характер. Классический подход к модификации модели – её дообучение на новых данных или с новыми целями. Однако он требует вычислительных ресурсов, исходных данных и изменяет архитектуру процесса обучения, что не всегда допустимо.

Однако комплексный подход, который бы ставил своей целью целенаправленное, контролируемое и обратимое аффинное преобразование весового пространства модели с гарантией сохранения точности на исходном задании, в литературе представлен недостаточно.

Объект исследования: обученные нейросетевые модели.

Целью данной работы является исследование возможности применения аффинных преобразований к признаковым пространствам обученной нейронной сети и разработка на этой основе алгоритма, позволяющего адаптировать веса модели к изменённым данным без изменения её выходной точности.

Для достижения поставленной цели необходимо решить следующие задачи:

* провести теоретический анализ структуры признаковых пространств глубоких нейронных сетей и свойств аффинных преобразований в контексте линейной алгебры;
* сформулировать строгие математические условия, при которых аффинное преобразование, применённое к весам одного или нескольких слоёв сети, не изменяет итоговое предсказание для измененных данных;
* разработать и формализовать алгоритм поиска и применения аффинных преобразований к весам полносвязных слоёв предобученной модели;
* реализовать спроектированный алгоритм на языке программирования Java с использованием библиотеки для глубокого обучения deeplearning4j;
* экспериментально проверить корректность работы алгоритма.

## 1. Анализ технического задания

Исходным заданием работы является реализация алгоритма, который позволит менять веса уже обученной нейросетевой модели без изменения точности её предсказаний, не прибегая к дообучению. Основными сложностями в реализации стали:

* обработка многослойных и многомерных современных сетевых моделей;
* тонкости применения аффинных преобразований к весам этих моделей.

Для решения этих задач потребовался тщательное изучение устройства нейронных сетей и выбор технологий, обеспечивающих высокую производительность, удобство разработки и корректную работу алгоритмов машинного обучения.

## 1.1 Языки программирования: Java, Python

Основным языком разработки выбран Java, что обусловлено его надёжностью, производительностью и доминирующим положением в промышленной разработке корпоративных систем. В контексте реализации алгоритмов манипуляции с весами обученных моделей использование Java обеспечивает строгую типизацию и высокую предсказуемость выполнения, что критически важно для обеспечения корректности и воспроизводимости преобразований. Хотя Python традиционно доминирует в сфере прототипирования и исследований машинного обучения благодаря лаконичному синтаксису и обширной экосистеме специализированных библиотек, для данной задачи приоритетными стали производительность во время выполнения, многопоточность и возможности низкоуровневой оптимизации, которые в полной мере предоставляет Java.

Для графического отображения данных был использован Python в паре с Matplotlib - библиотекой для визуализации данных двумерной и трёхмерной графикой.

### 1.2 Нейронные сети: Eclipse Deeplearning4J

В качестве основы для обучения модели, её анализа, проведения аффинных преобразований и реализации дополнительных вычислительных компонентов был выбран DL4J в связке с библиотекой ND4J. Этот стек представляет собой наиболее эффективное решение для глубокого обучения на платформе Java/JVM.

### 1.3 Среда разработки: VScode

Средой разработки стал Visual Studio Code - легковесный, но мощный редактор, с широкими возможностями кастомизации и поддержкой огромного множества языков. Его универсальность и производительность делают его оптимальным выбором для подобных проектов по сравнению с более специализированными IDE.

**2. Разработка модели**

Для всех экспериментов использовались исключительно полносвязные нейронные сети, что позволило сфокусироваться на свойствах преобразований без усложнений, присущих свёрточным или рекуррентным архитектурам.

**2.1 Создание однослойной бинарной модели “Треугольник”**

Первоначальной задачей стало построение максимально простой и интерпретируемой модели для дальнейшей визуализации работы нейронной сети и её внутреннего представления.

Была разработана однослойная нейронная сеть с архитектурой 2 – 2 – 1 (точка x, y на вход, два нейрона в скрытом слое, один выходной нейрон). Для датасета программно генерировался сбалансированный набор точек так, чтобы данные формировали визуальный треугольник на графике, который потом должны были ограничивать линии нейронов скрытого слоя.

Модель решает бинарную задачу классификации: определение принадлежности точки с координатами (x, y) заданной геометрической области - треугольнику.

Обучение модели продолжалось до сходимости, что визуально выражалось в стабилизации значений функции потерь. Финальное значение accuracy было примерно 96%.

**2.2 Визуализация модели**

В общем случае получение значения активации нейрона на каждом слое полносвязной сети можно описать следующим уравнением:

, где (1)

σ – функция активации;

– вес n-го нейрона прошлого слоя;

– активация нейрона прошлого слоя;

– смещение;

– новое значение активации нейрона.

Вспомним как выглядит уравнение линии на двумерной плоскости:

Так становиться видно, что уравнение высчитывания новой активации нейрона имеет некоторую схожесть с геометрическим уравнением линии. Сходство становиться очевиднее для слоя с двумя нейронами. Отбросим применение функции активации, оставив только сумму произведений весов на активации прошлых нейронов:

Этот факт даёт нам возможность манипулировать этими данными, применяя к ним различные аффинные преобразования.

По этому уравнению мы и сможем построить линии сети, ограничивающие область решений, так как веса и активации – это константы, которые получаются после обучения.

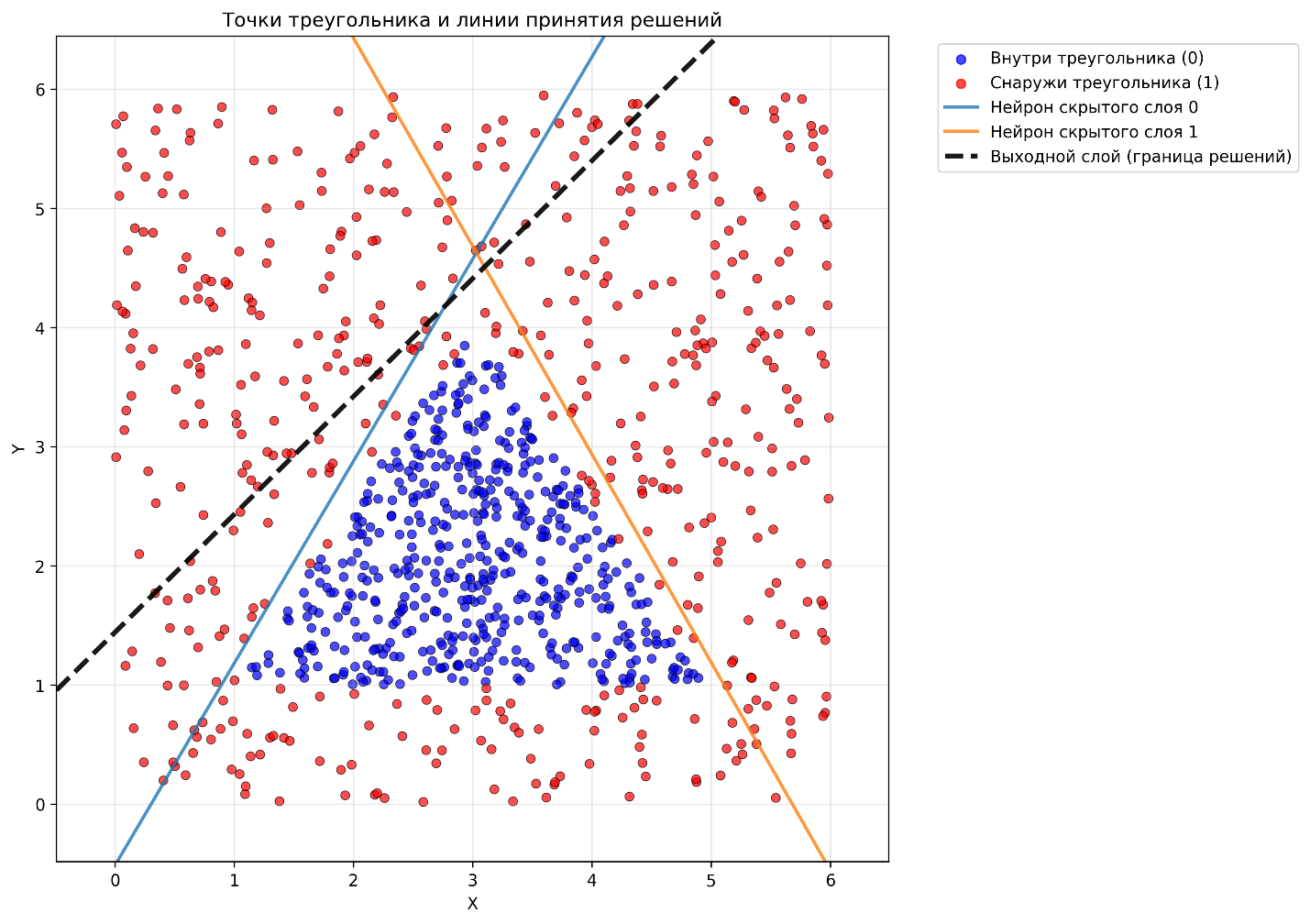


Рисунок 1 - Точки датасета "Треугольник" и ограничивающие линии нейронов

На рисунке 1 видно, как две линии скрытого слоя выделяют треугольник. Третья линия – это выходной слой, который оказывает решающее влияние в реальной работе модели. Можно заметь, что эта линия никак не ограничивает область треугольника. Это происходит, потому что её активация высчитывается по активации нейронов предыдущего слоя, что даёт неожиданный результат при визуализации. Зная это, в дальнейшем будем работать, только с весами и смещениями первого скрытого слоя, который работает непосредственно со входными данными.

**2.2.1 Отображение нормалей для линий нейронов**

Вспомним, что для каждого нейрона скрытого слоя с весами , и смещением уравнение разделяющей линии в пространстве признаков имеет вид:

(2)

Вектор n = [,] является нормалью (перпендикуляром) к данной линии, что следует из свойств скалярного произведения: если скалярное произведение двух ненулевых векторов равно нулю, то эти векторы перпендикулярны. Действительно, для любых двух точек (,) и (,​,​), лежащих на линии

Таким образом, вектор весов нейрона определяет не только положение разделяющей линии, но и её ориентацию в пространстве.

Зная направление нормали линии нейрона, мы сможем узнать, где находиться точка, передавая значения её координат.

Если результат уравнения , то нормаль указывает в сторону искомой области. Если , то область находиться в противоположном направлении от неё.

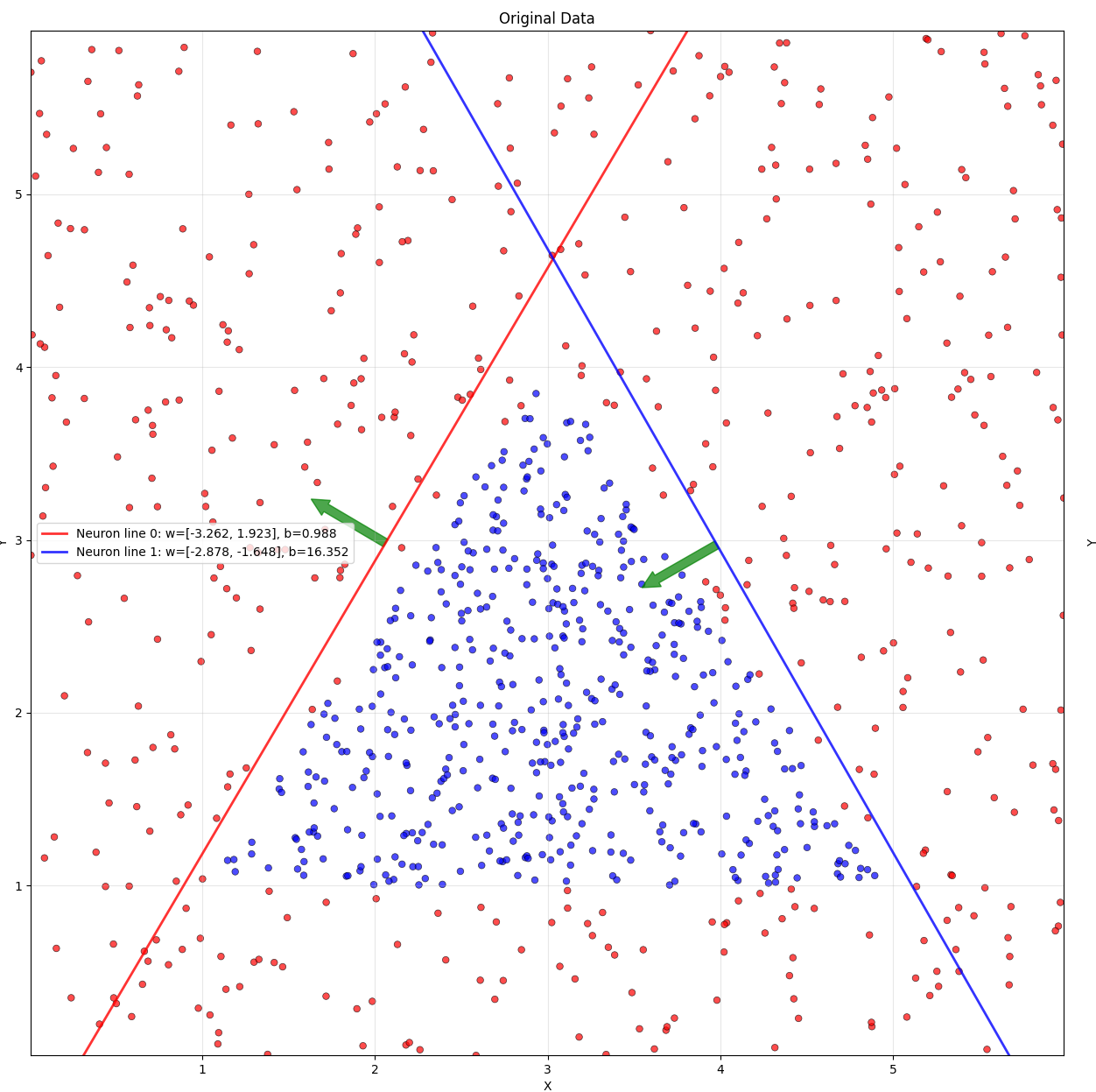


Рисунок 2 - Обновлённое отображение модели “Треугольник” с нормалями

Чтобы убедиться, что всё работает, согласно ожиданиям, проведём некоторые вычисления для конкретной точки.

Возьмём точку с *x* = 3, *y* = 3. Визуально можно оценить, что точка с данными координатами входит в область решений. Докажем это расчётами.

Подставляем в формулу 2 известные значения.

Для линии 0 (красная) это будет:

-3.26\*3.0 + 1.92\*3.0 + 0.98 = -3.04

Результат получился отрицательным. Точка находится противоположно направлению нормали.

Для линии 1 (синяя) это будет:

-2.87\*3.0 - 1.64\*3.0 + 16.35 = 2.82

Результат получился положительным. Точка находится по направлению нормали.

Комбинируя результаты, можно сказать, что точка действительно входит в область решений, так как находиться между линиями нейронов.

**2.3 Знакомство с аффинными преобразованиями**

Аффинное преобразование (от лат. affinis «соприкасающийся, близкий, смежный») — отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся — в пересекающиеся, скрещивающиеся — в скрещивающиеся.

«Отображение в себя» означает, что если мы находились в пространстве Rn, то после образования мы должны остаться в нем же. Например: если мы применили какое-то преобразование к прямоугольнику и получили параллелепипед, то мы вышли из R2 в R3. А вот если из прямоугольника у нас получился другой прямоугольник, то все хорошо, мы отобразили исходное пространство в себя.

Теперь запишем в общем виде, как выглядит преобразование координат в формульном виде.

Пусть у нас есть исходная система координат. Точка в этой системе характеризуется двумя числами - x и y. Совершить переход к новым координатам x' и y' мы можем с помощью следующей системы:

При этом, числа , , , должны образовывать невырожденную матрицу:

Можно записать и в более общем виде, где аффинное преобразование – это преобразование вида:

, где (3)

- аффинная матрица, а - n-мерный вектор.

Для дальнейшей реализации было выбрано 3 аффинных преобразования: поворот, растяжение/сжатие и сдвиг.

**2.4 Аффинное преобразование поворота**

Для первой экспериментальной проверки концепции модификации весов без дообучения было выбрано аффинное преобразование поворота в пространстве.

Матрица M для поворота против часовой стрелки примет вид:

(4)

И новые координаты будут выглядеть так:

Таким образом получаем повёрнутую систему координат на угол a.

Для поворота точки остаётся вычислить:

\*

**2.4.1 Обобщение алгоритма для n-мерной точки**

Для заключительного эксперимента будет обучена модель на базе датасета чисел MNIST, каждое из которых имеет размер 28\*28 пикселей, что в результате даёт 784. Нейроны на первом скрытом слое будут размерностью 784, так как должны будут обрабатывать входные 784 пикселя. Таким образом сложные многослойные сети могут состоять из нейронов, которые формируют гиперплоскости n-мерной размерности. Для обобщения матрицы поворота для n-мерного случая, нужно понять, как она формируется. Рассмотрим на примере 2-мерного случая.

Сначала создаётся пустая единичная квадратная матрица n\*n. Для двумерного случая мы имеем точку с координатами (осями) x, y. Для заполнения значениями матрицы поворота, мы будем брать комбинации осей x, y для определения плоскости вращения. xx = cos, xy = -sin, yx = sin, yy = cos. Получаем формулу 4.

Зная алгоритм для двумерной системы, можно применить его для того, чтобы повернуть точку в n-мерном пространстве вокруг гиперплоскости, заданной двумя осями axis1 и axis2. Выбираем 2 оси из n и вращаем в плоскости, которую они образуют.

Пример получения обобщенной матрицы преобразования на Java:

private double[][] createAffineMatrix(int dimensions, int axis1, int axis2) {

double[][] matrix = createIdentityMatrix(dimensions);

double cos = Math.cos(angleDegrees);

double sin = Math.sin(angleDegrees);

matrix[axis1][axis1] = cos;

matrix[axis1][axis2] = -sin;

matrix[axis2][axis1] = sin;

matrix[axis2][axis2] = cos;

return matrix;

}

**2.4.2 Визуализация модели сети с применением поворота**

Преобразования будем применять к скрытым слоям сети. В модели “Треугольник” один скрытый слой, который состоит из 2 нейронов. Вспомним, что каждый нейрон – это прямая, если говорить про двумерные случаи или гиперплоскости, если говорить о n-мерных случаях.

Так как смещение не зависит от входных координат, а веса — зависят, то при применении преобразования к линии нейрона, будем изменять только веса. Корректность данного вывода подтверждается эмпирически. При изменении не только весов, но и смещений, полностью нарушается архитектура модели, и итоговые результаты различны.

Воспользуемся формулой 3: для весов формулы 2 применим преобразование поворота, формула 4.

Для моделирования повёрнутых данных те же преобразования применим и к ним.

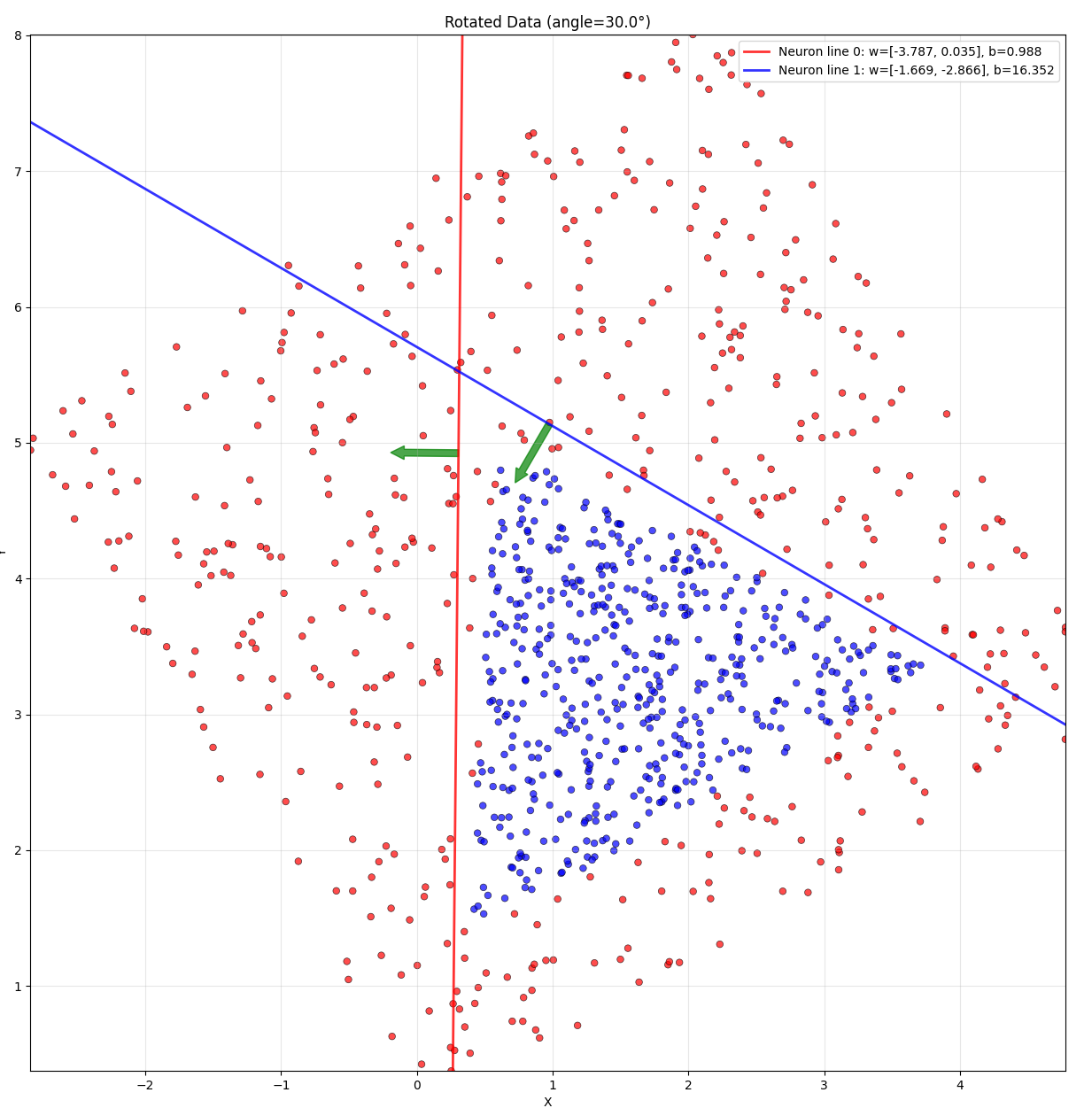


Рисунок 3 - Модель “Треугольник”, после применения преобразования поворота на 30 градусов

Веса сети поменялись, по сравнению с моделью на рисунке 2, в соответствии с изменёнными данными. Это позволяет модели предсказывать отношение точки к области решений с неизменной точностью.

Например, сonfidence для точки 3, 3 до манипуляций с весами и данными: 0.87179654, и сonfidence после поворота весов и данных также 0.87179654

**2.5 Аффинное преобразование растяжение/сжатие**

Для моделирования растяжения/сжатия сконструируем матрицу M из формулы 3 несколько иначе:

Новые координаты тогда принимают вид:

Из вида системы уравнений понятно, что мы просто растягиваем наши оси, если коэффициент меньше 1 и сжимаем, если больше 1.

Для n-мерного случая адаптировать матрицу также не составляет труда, просто ставим нужный коэффициент на тот элемент диагонали, ось которой хотим растянуть/сжать.

Пример получения обобщенной матрицы преобразования на Java:

public double[][] createAffineMatrix(int dimensions) {

double[][] matrix = createIdentityMatrix(dimensions);

for (int i = 0; i < dimensions; i++)

matrix[i][i] = scaleFactor;

return matrix;

}

**2.5.1 Ортогональные и не ортогональные матрицы аффинных преобразований**

Для корректного применения преобразований к весам модели, нужно учитывать один нюанс, а именно ортогональность полученной аффинной матрицы.

Ортогональность простыми словами означает перпендикулярность или независимость между объектами. Это значит, что они находятся под прямым углом друг к другу, как оси координат (X и Y) на графике.

Чтобы узнать, что перед нами ортогональная матрица нужно выяснить:

* является ли она квадратной;
* выполняется одно из условий:

либо A \* Aᵀ = E, где Aᵀ — транспонированная, а E — единичная матрица, либо обратная матрица A⁻¹ совпадает с транспонированной Aᵀ.

Знание того факта, что аффинная матрица ортогональна, подскажет нам, нужно ли компенсировать веса перед применением преобразования или нет.

Компенсировать – значит применять к матрице весов обратную матрицу аффинного преобразования, с целью корректного наложения изменённых весов на данные.

Простая аналогия, для лучшего понимания вышеописанного.

При повороте картинки на 30 градусов против часовой стрелки, требует от наблюдателя “повернуть” свои глаза также на 30 градусов против часовой, чтобы увидеть такую же картинку. Но, например, при растяжении изображения в 5 раз, если наблюдатель также “увеличит” глаза в 5 раз, то он просто ничего не увидит. В данном преобразовании, глаза наблюдателя нужно пропорционально “уменьшить” в 5 раз, чтобы увидеть ту же картинку.

На этой аналогии можно понять, что аффинная матрица поворота – ортогональна, а матрица расширения – нет.

Докажем это кодом на Python:

import numpy as np

angle = 30.3

angle\_rad = np.radians(angle)

rotate = np.array([

[np.cos(angle\_rad), -np.sin(angle\_rad)],

[np.sin(angle\_rad), np.cos(angle\_rad)]

])

print(rotate.T)

print()

print(np.linalg.inv(rotate))

print()

print(rotate @ rotate.T)

print()

print()

scale\_x = 5

scale\_y = 5

scale = np.array([

[scale\_x, 0],

[0, scale\_y]

])

print(scale.T)

print()

print(np.linalg.inv(scale))

print()

print(scale @ scale.T)

Результат:

[[ 0.86339555 0.50452762]

[-0.50452762 0.86339555]]

[[ 0.86339555 0.50452762]

[-0.50452762 0.86339555]]

# матрицы равны

[[ 1.00000000e+00 -2.23418993e-17]

[-2.23418993e-17 1.00000000e+00]]

# получили единичную матрицу

# значит матрица - ортогональна

[[5 0]

[0 5]]

[[0.2 0. ]

[0. 0.2]]

# матрицы не равны

[[25 0]

[ 0 25]]

# получили не единичную матрицу

# значит матрица – не ортогональна

Таким образом, в преобразовании растяжения/сжатия для изменения весов будем применять обратную аффинную матрицу.

**2.5.2 Визуализация модели сети с применением растяжения/сжатия**

Условия аналогичны случаю с поворотом, только к весам применяем обратную аффинную матрицу.

Для примера сделаем сжатие в 2 раза:

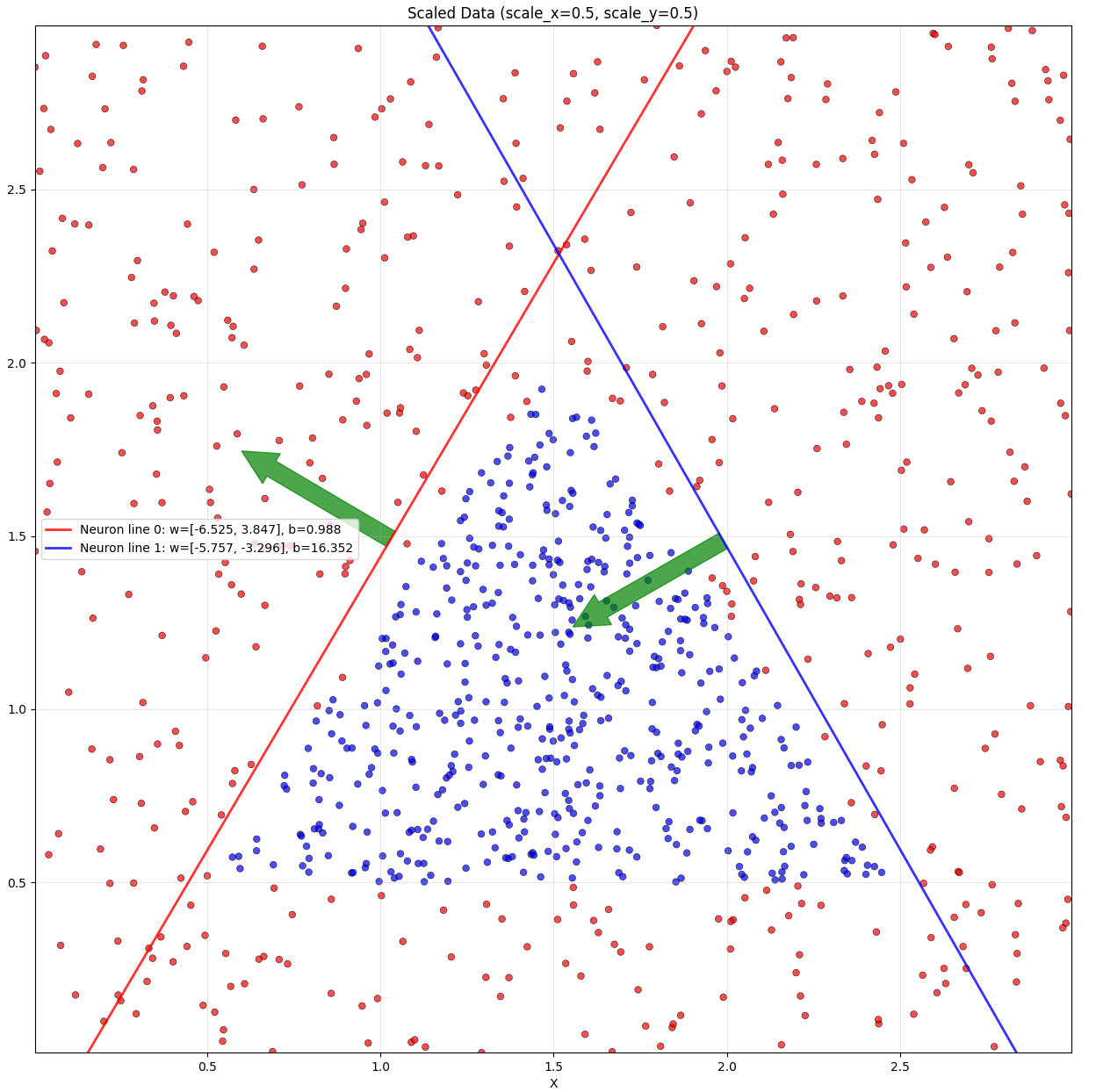


Рисунок 4 - Модель “Треугольник”, после применения преобразования сжатия в 2 раза

Веса сети также поменялись, в соответствии с изменёнными данными. Результат предсказаний не изменился.

**2.6 Аффинное преобразование расширения**

Для моделирования расширения сконструируем матрицу M из формулы 3 следующим образом:

Новые координаты принимают вид:

Из вида системы уравнений понятно, что мы растягиваем наши координаты вправо и вверх для и в противоположную сторону – вниз и влево для .

Для n-мерного случая это обобщается следующим образом: мы применяем расширение по координате относительно другой координаты. А значит мы делаем тоже самое что и для поворота: вставляем коэффициент расширения относительно требуемых координат, но оставляя диагональ нетронутой.

Пример получения обобщенной матрицы преобразования на Java:

private double[][] createAffineMatrix(int dimensions, int axis1, int axis2) {

double[][] matrix = createIdentityMatrix(dimensions);

matrix[axis1][axis2] = shear;

matrix[axis2][axis1] = shear;

return matrix;

}

Если сравнить все три преобразование, можно сделать интересный вывод: поворот осуществляется одновременным применением растяжения и сжатия, используя для этого тригонометрию. Это особенно хорошо видно, при сравнении кода для получения матриц каждого из преобразований.

**2.6.1 Визуализация модели сети с применением расширения**

Для корректной компенсации расширения, будем делать по аналогии с преобразованием растяжения/сжатия – к весам применяем обратную аффинную матрицу.

Для примера сделаем расширение на коэффициент 0.3:

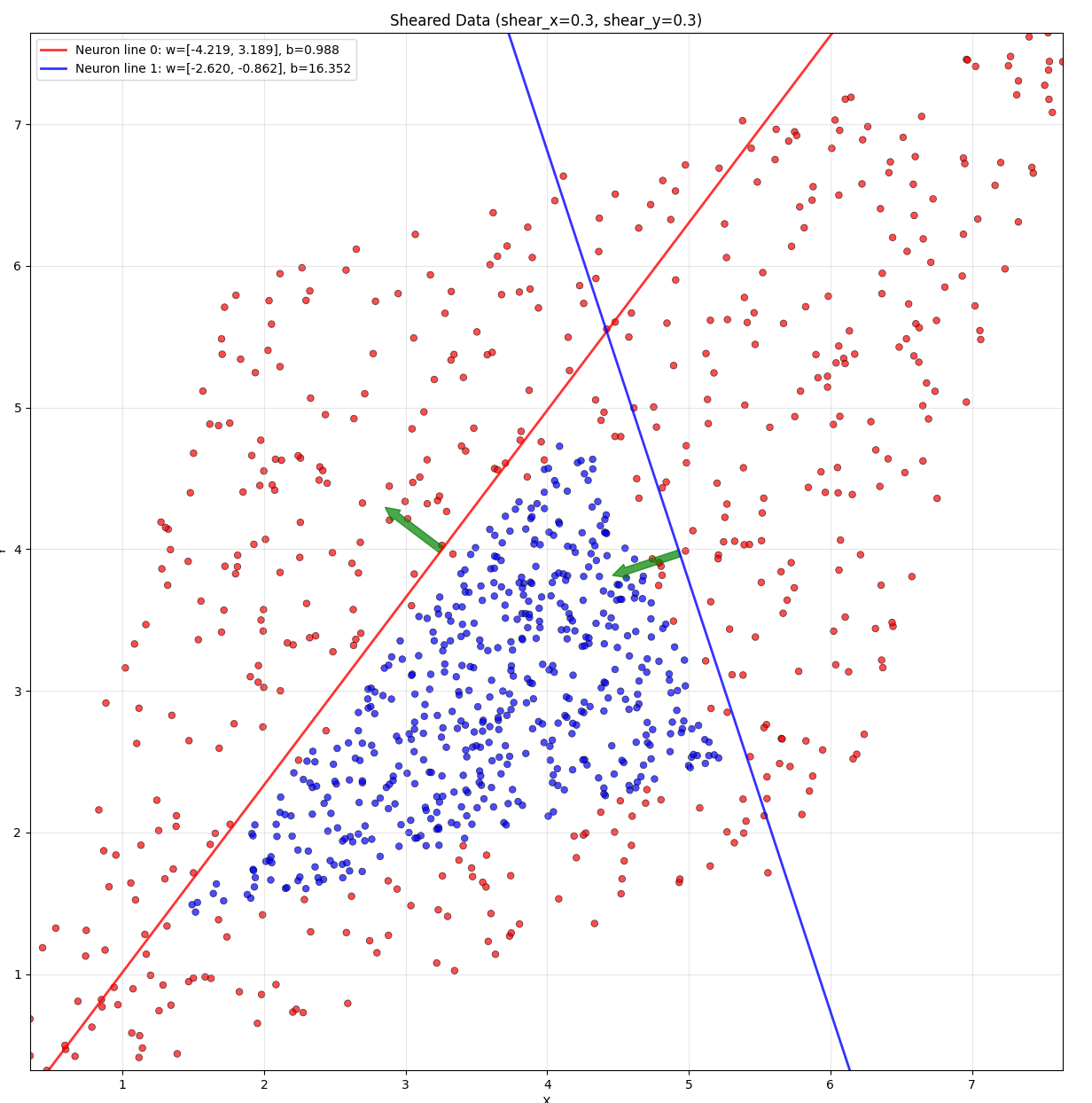


Рисунок 5 - Модель “Треугольник”, после применения преобразования расширение на коэффициент 0.3

**2.7 Создание многослойной бинарной модели “Два треугольника”**

После проведения всех экспериментов с преобразованиями на однослойной модели “Треугольник”, нужно выяснить, будет ли всё работать также хорошо и для многослойных моделей.

Для этого усложним текущую модель, добавив ещё один треугольник. Она всё ещё будет бинарной, то есть на выходе мы будем узнавать принадлежит ли точка одному из треугольников или нет. Для чёткого разделения этих треугольников нужно усложнить архитектуру модели: во-первых, увеличить количество нейронов на первом слое, по 3 на каждый треугольник и, во-вторых, добавить второй скрытый слой, который будет разделять области решений двух треугольников. Итоговая архитектура: 2 – 6 – 2 – 1.

**2.7.1 Обоснование выбора слоя для применения преобразований**

Аффинные преобразования будут применяться только к первому скрытому слою. Или можно сказать иначе, только к слою, который непосредственно работает с входными данными.

Чтобы понять, почему нужно делать, так, а не иначе, нужно вспомнить как вообще работают сети, основанные на полносвязных (dense) слоях.

Каждый полносвязный слой в нейронной сети выполняет преобразование по формуле 1. Из неё видно, что активация нейрона вычисляется по значению активации предыдущего нейрона, кроме первого слоя. Он работает непосредственно с исходными данными. Отсюда можно сделать вывод, что если при применении преобразования, первый скрытый слой выдаст тот же результат, как если бы этих преобразований не было, то все остальные слои автоматически сработают правильно и результат будет корректным.

Что бы это доказать, нужно сравнить активации нейронов первых скрытых слоёв до всех преобразований и после изменения весов и входных данных. Применим преобразование поворота на 30 градусов к модели “Два треугольника”, со входной точкой 3, 3:

====== Before weight and data transformation ======

------ First hiden layer weights ------

-0.2565350830554962 0.9576932787895203

0.27009227871894836 0.1462019383907318

-0.019547274336218834 -0.04933701455593109

1.410359501838684 -1.0748653411865234

0.05246511474251747 -2.7907469272613525

-0.7326761484146118 -0.7785971164703369

------ Neuron activations ------

3.0 3.0

-0.3422616422176361 -0.9828152060508728 -0.9952481389045715 0.775757372379303 -0.9999905228614807 0.49449196457862854

-0.9999547600746155 -0.7523653507232666

0.985088050365448

------ Prediction ------

Confidence: 0.985088050365448

====== After weight and data transformation ======

------ First hiden layer weights ------

0.7476058476903883 0.6512082321163086

0.24999965462541984 -0.1783956796000739

-0.05290834426082423 -0.004116314716547248

-0.3459362576915701 -1.7391887567843332

-2.4839285710086605 -1.2732320504960983

-1.0213258509573662 0.3161033286724542

------ Neuron activations ------

4.012990772626067 -1.3769186827180608

-0.3422614336013794 -0.9828152060508728 -0.9952481389045715 0.7757572531700134 -0.9999905228614807 0.4944923520088196

-0.9999547600746155 -0.75236576795578

0.9850881695747375

------ Prediction ------

Confidence: 0.9850881695747375

На полученных данных, можно увидеть, как активации нейронов первого скрытого слоя до и после преобразований равны до 5-6 знаков после запятой. А это значит, что следующие слои, используя эти данные, не нарушат результат предсказания.

**2.7 Визуализация применения аффинных преобразований к модели “Два треугольника”**

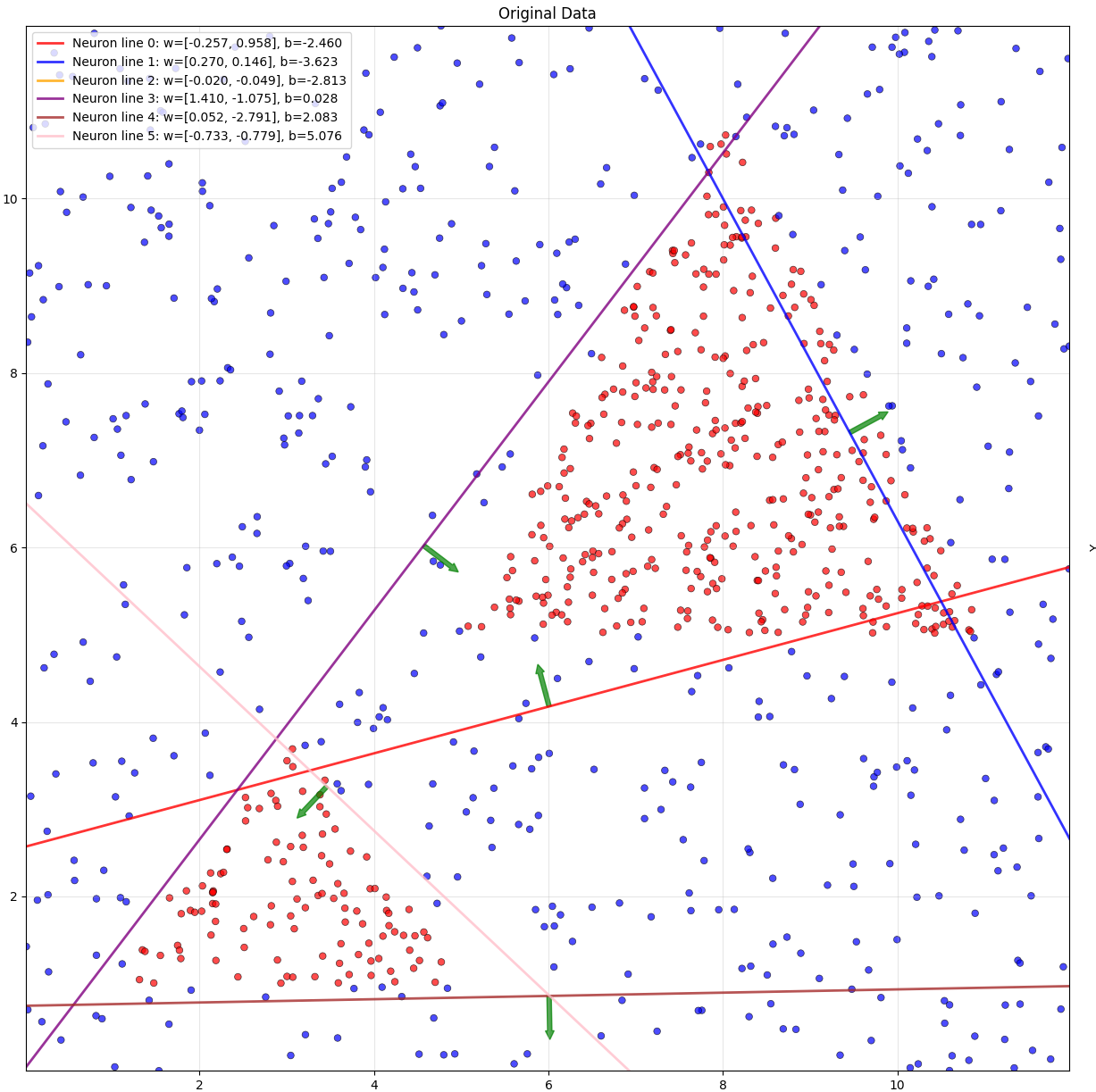


Рисунок 6 - Модель “Два треугольника”, до применения каких-либо преобразований

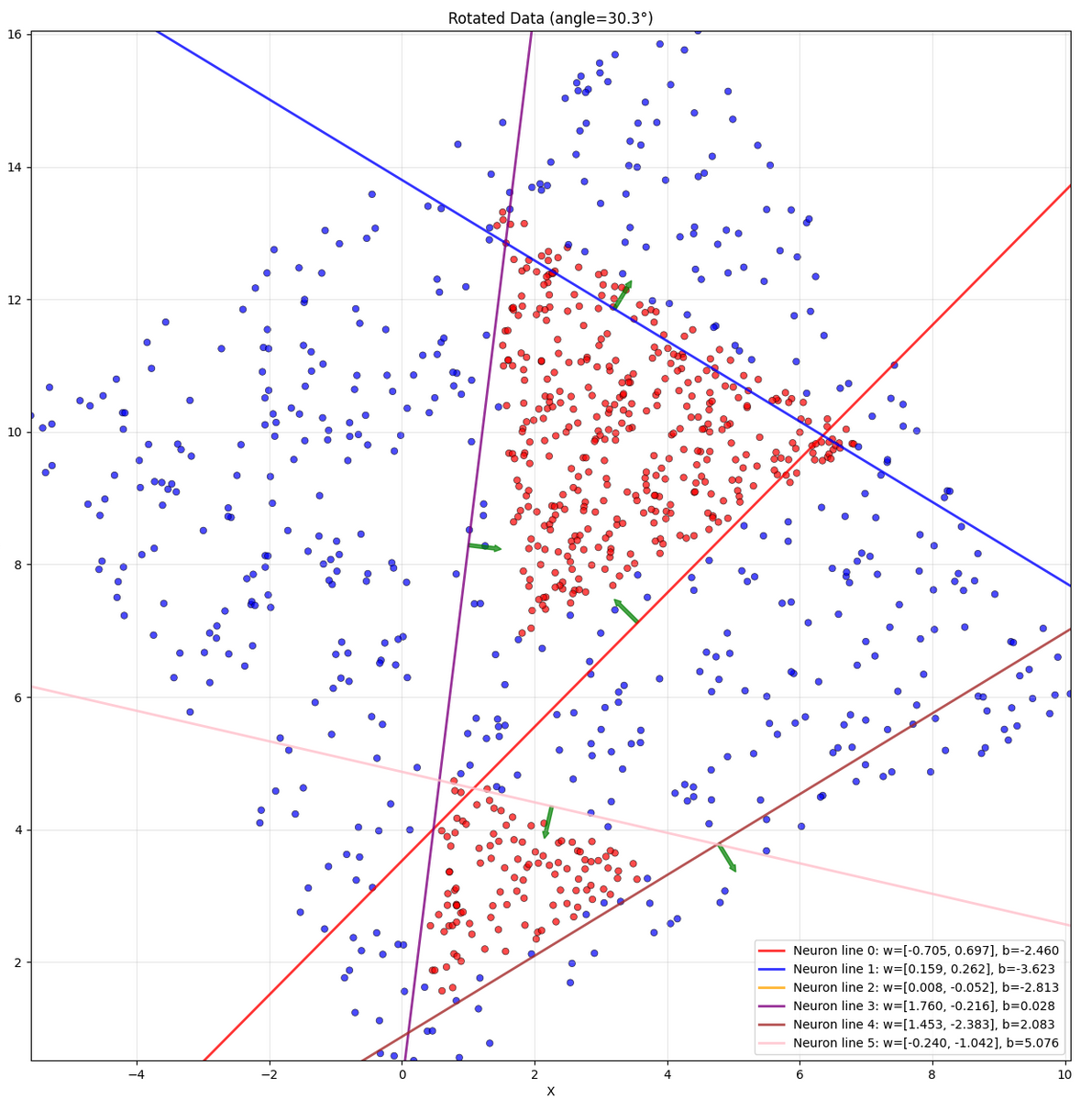


Рисунок 7 - Модель “Два треугольника”, после применения преобразования поворота на 30 градусов

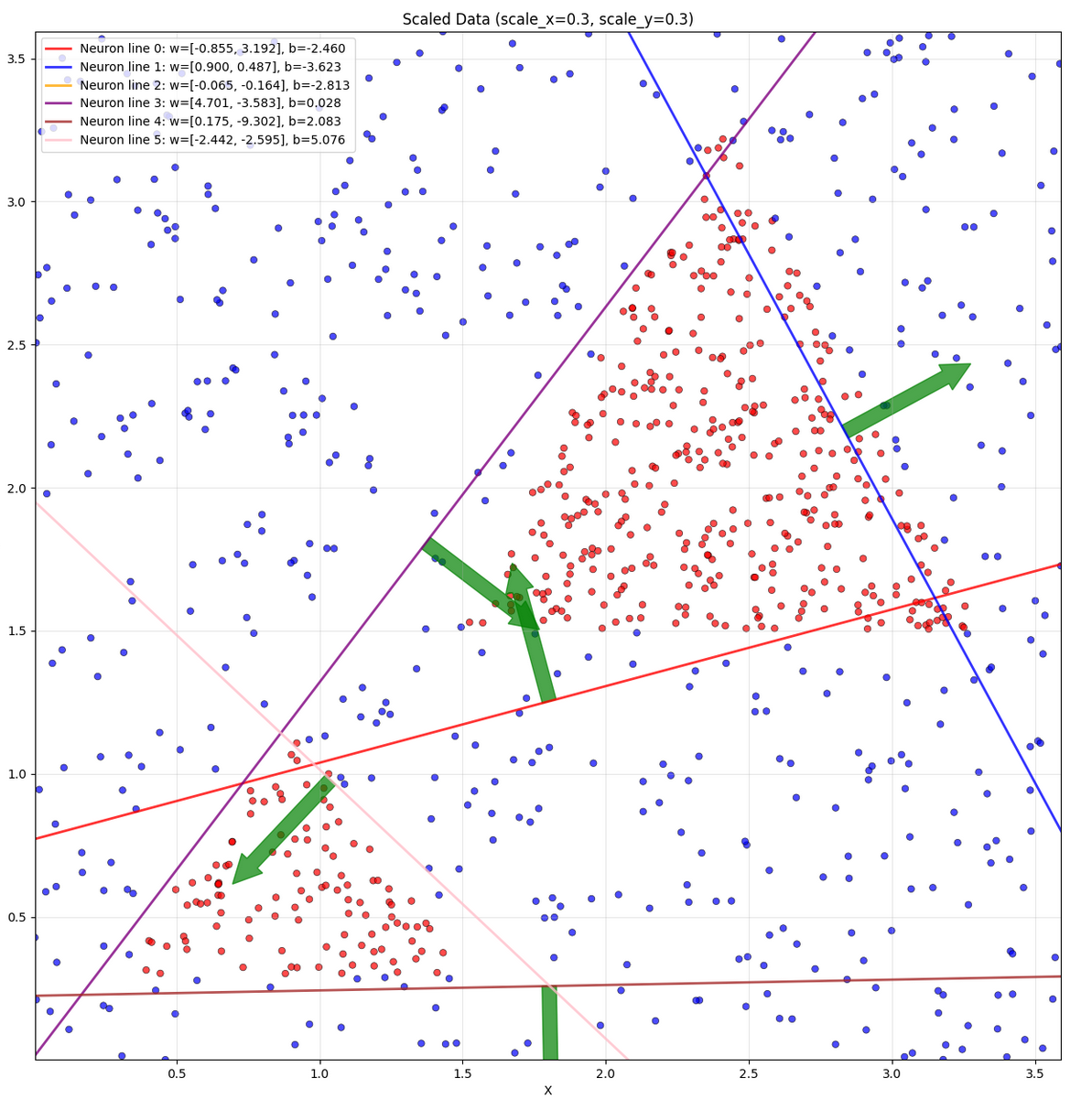
****

Рисунок 8 - Модель “Два треугольника”, после применения преобразования сжатия на коэффициент 0.3

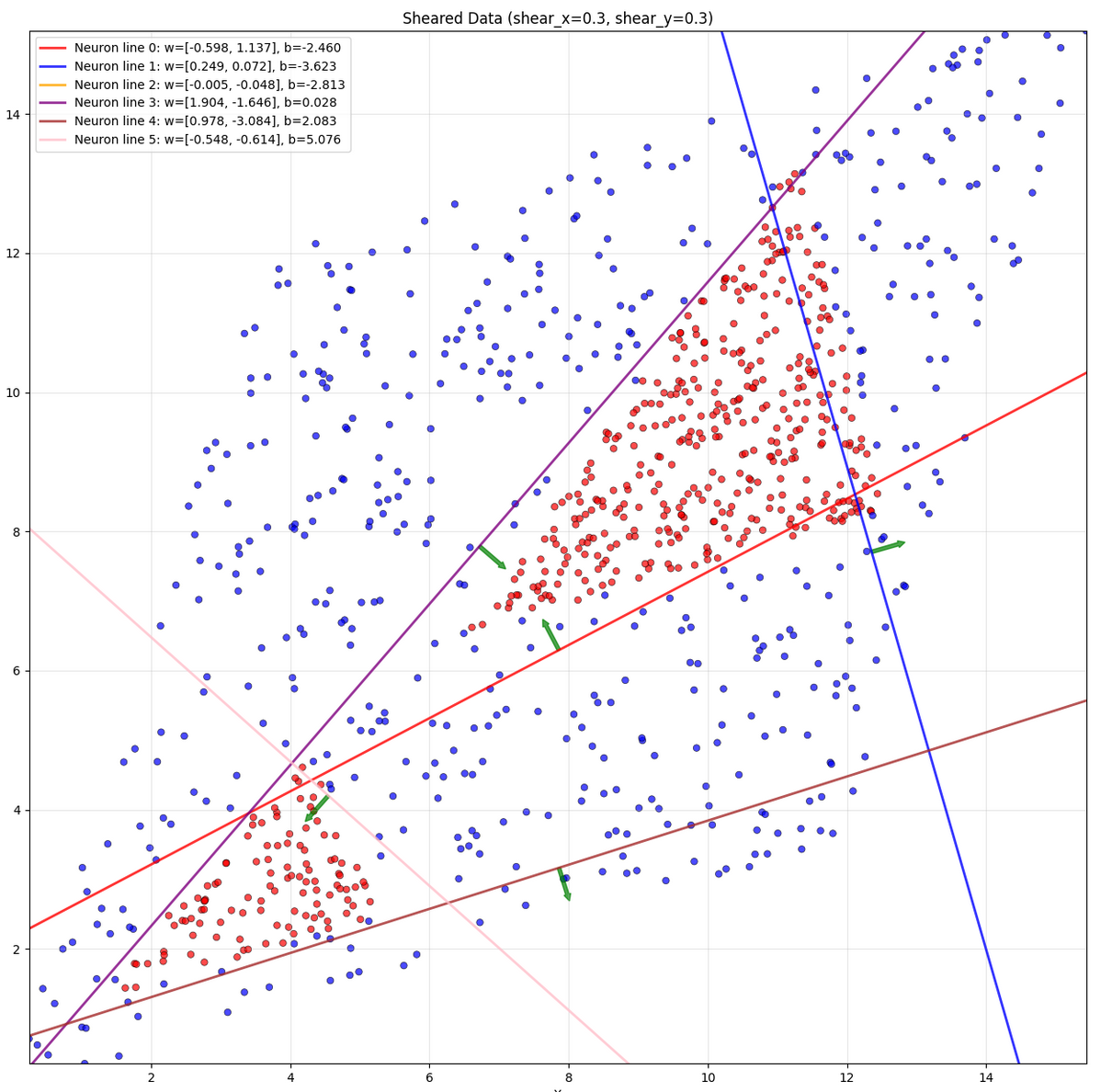
****

Рисунок 9 - Модель “Два треугольника”, после применения преобразования расширение на коэффициент 0.3

**2.8 Создание многослойной модели “MNIST”**

Прежде все модели получали на вход двумерную точку, с координатами x, y. Это позволило нам визуализировать результаты. Но теперь, имея достаточно информации и экспериментальных данных, можно расширить применение преобразований до n-мерных данных.

Для этого заключительной экспериментальной моделью послужит нейронная сеть на базе датасета чисел “MNIST”, где каждая цифра от 0-9 представлена изображением 28\*28 пикселей. Её архитектура: 784 – 64 – 32 – 16 – 10.

Каждый нейрон на входном слое будет обрабатывать 784 пикселя входного изображения, или в геометрическом смысле он будет представлять собой гиперплоскость размерностью 784.

Имея обобщённые схемы построения матриц преобразований, не составит труда применить эти преобразования к модели и данным.

Для лучшей наглядности, будем применять преобразования ко всем осям последовательно. Так для матриц поворот между axis1 и axis2, результирующей будет все последовательно перемноженные матрицы поворотов. Похожим образом поступим и с другими преобразованиями.

Результаты после поворота изображения цифры 3 и весов первого скрытого слоя:

====== Before weight and data transformation ======

------ First hiden layer weights ------

5.7705078397325956E-39 4.703835244793152E-39 -8.87386489726088E-38 -6.339193992912607E-41 5.253545014169277E-39 1.5848125112127951E-40 3.264153814232014E-39 ...

1.6218250075510065E-39 5.218488730487263E-39 5.335082367912945E-39 -1.9093742550273876E-39 3.201489148205872E-39 6.238678854066588E-38 -6.376859494335194E-39 ...

-1.4172003992979751E-39 -4.3820831043995306E-39 -6.025940447445423E-38 -6.51858822231547E-39 6.97999544922186E-38 2.940001249569004E-39 -3.0647378323708777E-40 ...

...

------ Neuron activations ------

... 241.0 255.0 255.0 255.0 255.0 255.0 244.0 237.0 128.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 50.0 235.0 255.0 245.0 254.0 246.0 245.0 245.0 245.0 248.0 ...

232.97018432617188 627.9869995117188 0.0 0.0 1112.656982421875 629.7344970703125 0.0 0.0 261.0110168457031 879.9109497070312 278.156005859375 209.40101623535156 ...

1049.260009765625 951.4986572265625 836.6201171875 0.0 1646.404296875 20.05770492553711 0.0 1008.3165283203125 94.4949722290039 604.4305419921875 176.78518676757812 ...

109.54348754882812 1440.6514892578125 915.6256713867188 0.0 1968.618896484375 3928.3076171875 152.3181610107422 375.6053771972656 1093.341552734375 ...

0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0

------ Prediction ------

Predicted digit: 3

Confidence: 1.0

====== After weight and data transformation ======

------ First hiden layer weights ------

1.839428899626901E-4 8.9961941361909E-5 1.0014484522300589E-4 1.1148036461767837E-4 1.240989655295359E-4 1.381458815488908E-4 1.537827854364992E-4 ...

2.1830041127386877E-4 1.067653596302841E-4 1.1885026327233648E-4 1.3230307216514982E-4 1.4727862120277814E-4 1.6394927123321212E-4 1.825068928428718E-4 ...

-4.259201622248689E-4 -2.0830707110614785E-4 -2.3188560716871792E-4 -2.581330270089811E-4 -2.873514248960607E-4 -3.198770895245563E-4 -3.5608437452402394E-4 ...

...

------ Neuron activations ------

... 6.764818237218361E-8 3.308506143967021E-8 3.6830000630381116E-8 4.099883413870403E-8 4.56395430888585E-8 5.0805539648099246E-8 5.655628177322188E-8 6.295795754098908E-8 ...

232.97021484375 627.9871826171875 0.0 0.0 1112.6568603515625 629.734619140625 0.0 0.0 261.0109558105469 879.9109497070312 278.1557312011719 209.4010467529297 ...

1049.260498046875 951.4990234375 836.6199951171875 0.0 1646.404296875 20.057880401611328 0.0 1008.316650390625 94.494873046875 604.4302978515625 176.78517150878906 ...

109.54325103759766 1440.6514892578125 915.62548828125 0.0 1968.619384765625 3928.30810546875 152.3181610107422 375.60498046875 1093.342041015625 ...

0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0

------ Prediction ------

Predicted digit: 3

Confidence: 1.0

Этот эксперимент показал подобные результаты с экспериментом с моделью “Два треугольника”. Активации нейронов до преобразований также примерно равны активациям нейронов после преобразований, с погрешностью в несколько знаков после запятой.

**3. Исследование работы модели**

Для исследования того, что моделирование аффинных преобразований над признаковым пространством выполняется корректно и соответствует поставленной задаче: модифицирование весов модели без изменения её выходной точности на валидационном множестве исходной задачи, выполним ряд экспериментов. За основу возьмём две последние многослойные модели “Два треугольника” и “MNIST”.

Для модели “Два треугольника” – 50 экспериментов для различных точек x, y. Каждый эксперимент будет состоять из:

* значений принадлежности точки к одному из треугольников (prediction) и уверенности (confidence) до преобразований;
* значений принадлежности точки к одному из треугольников (prediction) и уверенности (confidence) после преобразований только точки;
* значений принадлежности точки к одному из треугольников (prediction) и уверенности (confidence) после преобразований только весов;
* значений принадлежности точки к одному из треугольников (prediction) и уверенности (confidence) после преобразований точки и весов одновременно.
* для значений до и после вычисляется соответствие результатов и возможные изменения между ними.

Подробные результаты см. в приложениях A, C, E.

Для MNIST-a это будет по 5 штук на каждую цифру от 0 – 9. Каждый эксперимент будет состоять из:

* значений предсказанной цифры (prediction) и уверенности (confidence) до преобразований;
* значений предсказанной цифры (prediction) и уверенности (confidence) после преобразований только изображения;
* значений предсказанной цифры (prediction) и уверенности (confidence) после преобразований только весов;
* значений предсказанной цифры (prediction) и уверенности (confidence) после преобразований изображения и весов одновременно;
* для значений до и после вычисляется соответствие результатов и возможные изменения между ними.

Подробные результаты см. в приложениях B, D, F.

* 1. **Анализ результатов экспериментов**

Из результатов, представленных в таблицах Приложений A-F, следует, что все проведённые эксперименты подтверждают теоретические ожидания.

Полное совпадение точности при согласованных преобразованиях. Когда аффинное преобразование применяется одновременно и к входным данным, и к весам первого слоя модели, точность классификации остаётся неизменной. Предсказания модели до и после таких преобразований полностью идентичны.

Снижение точности при частичных преобразованиях. Если преобразование применяется только к данным или только к весам, точность модели существенно падает. Это особенно заметно на более простой архитектуре "Два треугольника", где малое количество нейронов делает сеть чувствительной даже к небольшим рассогласованиям между данными и весами.

Влияние сложности модели. На сложной модели "MNIST" нарушения точности при частичных преобразованиях также наблюдаются, но наиболее выражены они при преобразовании поворота. Это связано с тем, что поворот сильнее нарушает пространственные соотношения признаков, важных для распознавания рукописных цифр.

Таким образом, эксперименты доказывают, что сохранение точности нейронной сети при аффинных преобразованиях входных данных возможно при согласованном преобразовании её весовых параметров.

**ЗАКЛЮЧЕНИЕ**

В рамках данной работы было проведено исследование возможности применения аффинных преобразований к весам обученных нейронных сетей с сохранением их исходной точности предсказаний. Была достигнута поставленная цель – разработан и реализован алгоритм модификации весов модели без её дообучения, экспериментально подтверждена его корректность и эффективность.

Строго доказано, что аффинное преобразование, применённое к входным данным модели, может быть скомпенсировано соответствующим преобразованием весов первого полносвязного слоя. Для сохранения функционального отображения модели необходимо применять к весам обратное преобразование относительно преобразования данных, но только если матрица этого преобразования не ортогональна.

Был разработан универсальный алгоритм для трёх базовых аффинных преобразований: поворота, масштабирования и расширения. И было экспериментально доказано его работоспособность.

Таким образом, работа продемонстрировала принципиальную возможность и практическую реализуемость безопасного преобразования весов обученных нейронных сетей через аффинные преобразования. Полученные результаты подтверждают, что нейронные сети, несмотря на свою сложность, обладают определённой алгебраической структурой, позволяющей осуществлять контролируемые манипуляции с их параметрами при сохранении функциональной целостности.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

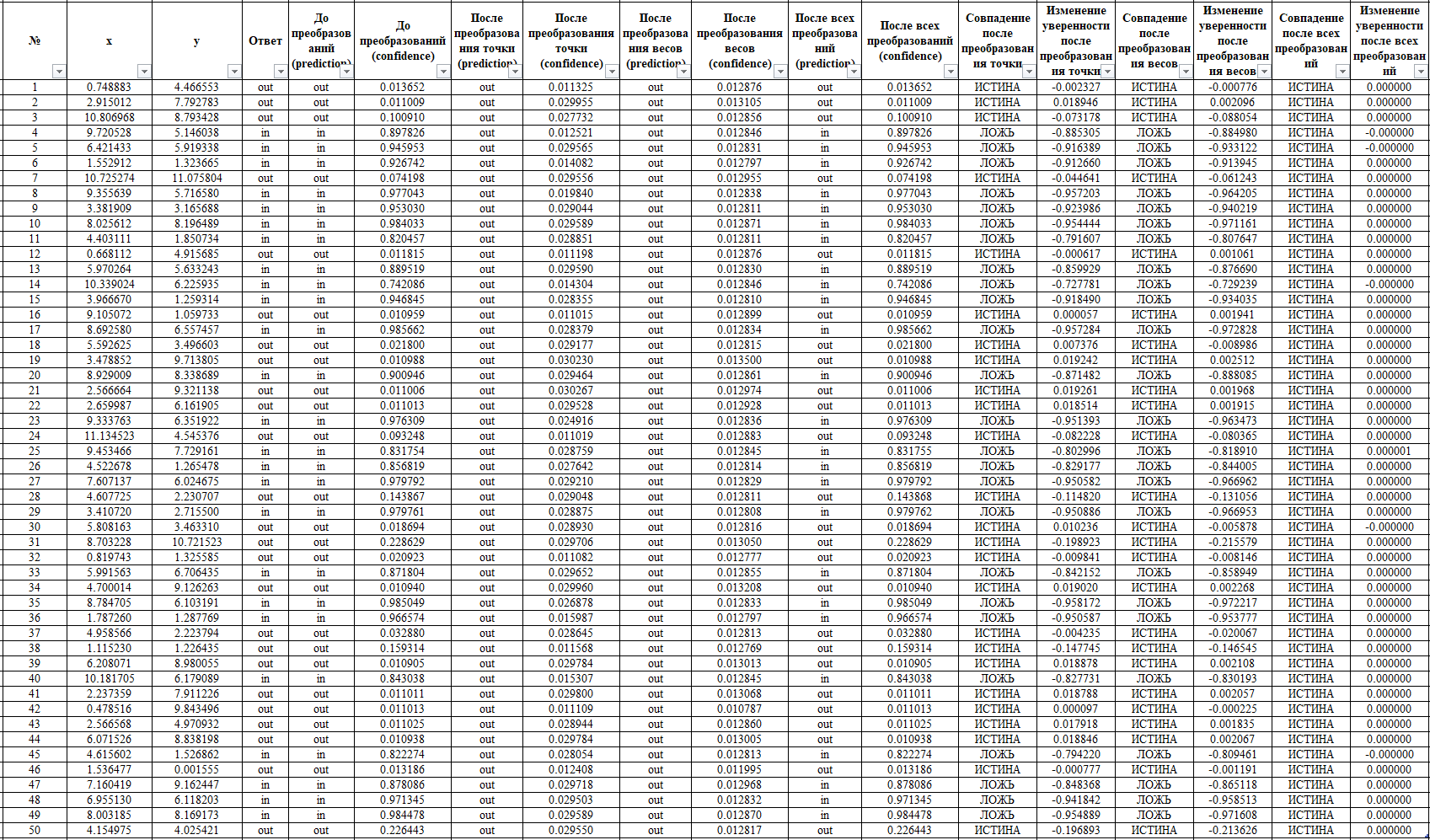
1. Шилдт, Герберт Java. Полное руководство: 12-е изд. : Пер. с англ. – СПб. : ООО “Диалектика”, 2023 – 1638 с.

2. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с анг. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.

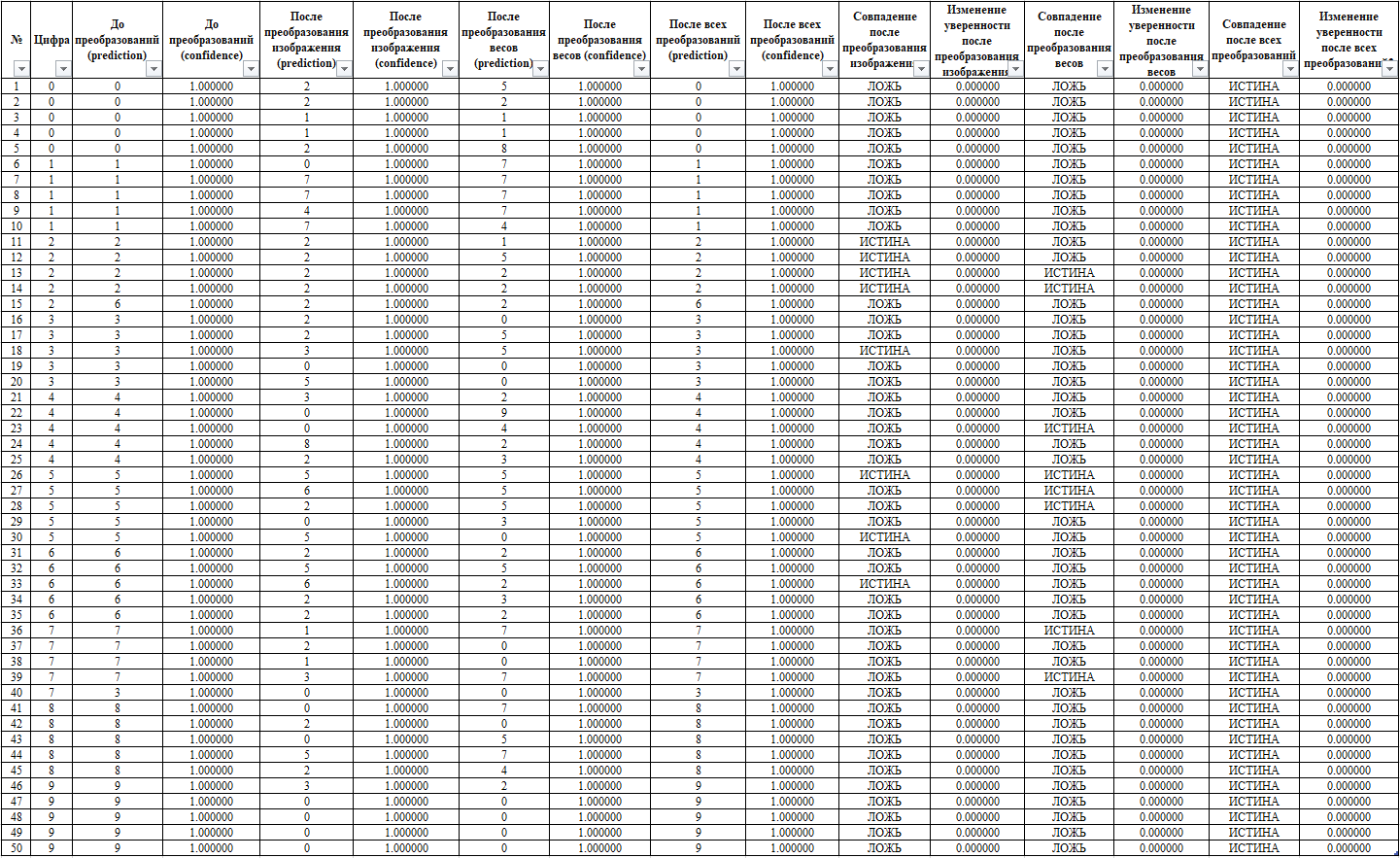
3. Документация по Deeplearning4j [электронный ресурс]: Режим доступа: <https://deeplearning4j.konduit.ai/>.

4. Аффинные преобразования [электронный ресурс]: Режим доступа: <https://en.wikipedia.org/wiki/Affine_transformation>.

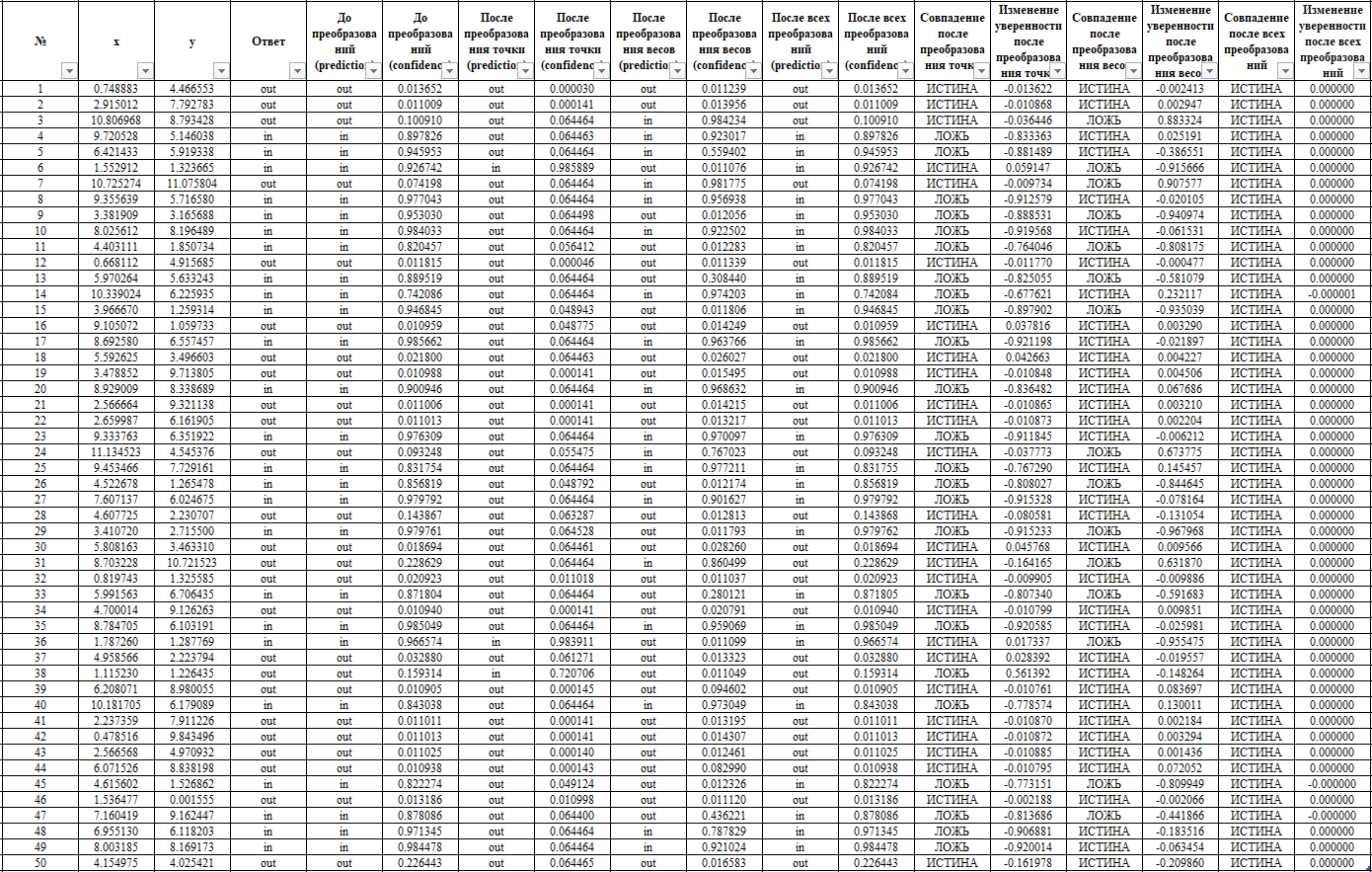
**Приложение A «Таблица экспериментов с поворотом на 256 градусов для модели Два треугольника»**



**Приложение B «Таблица экспериментов с поворотом на 256 градусов для модели MNIST»**



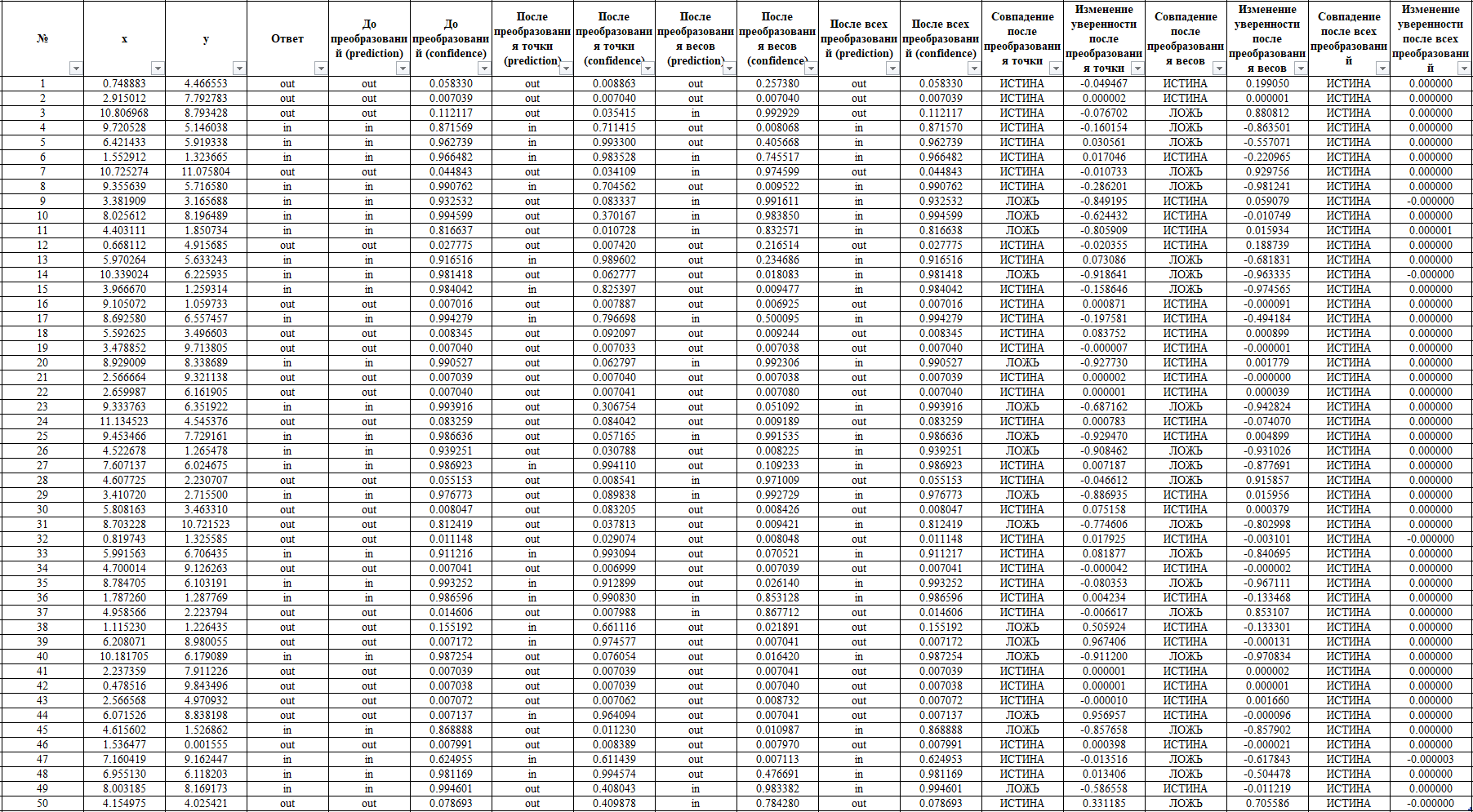
**Приложение C «Таблица экспериментов с растяжением в 2 раза для модели Два треугольника»**



**Приложение D «Таблица экспериментов с растяжением в 2 раза для модели MNIST»**



**Приложение E «Таблица экспериментов с расширением на коэффициент 0.2 для модели Два треугольника»**



**Приложение F «Таблица экспериментов с расширением на коэффициент 0.2 для модели MNIST»**

