**УТВЕРЖДАЮ**
Профессор кафедры
ИАНИ ННГУ, д.т.н.

_____ Н.В. Старостин
«____»_____2021 г.

Научно-технический отчёт

**на опытно-конструкторскую работу**

# Разработка нейронной сети специального вида (автоэнкодера) для решения задачи редукции пространства многомерных функций

**(Шифр ПО «Enc»)**

2021 г.

## Содержание

# Введение

Рассматривается проблема разработки нейронной сети специального вида (автоэнкодера) для решения задачи редукции пространства многомерных функций.

В рамках данного проекта проведены следующие работы:

Разработка структуры автоэнкодера, разработка функции генерации данных для обучения автоэнкодера на базе рандомизированных средств с низкой расходимостью, тестирование автоэнкодера на четырех функциях, разработка функции полного перебора гиперпараметров автоэнкодера.

## 1. Содержательная постановка задачи

Для каждой предоставленной функции построить автоэнкодер, обладающий наилучшими характеристиками по сжатию пространства параметров функции и точности.

## 2. Входные данные

В качестве исходных данных выступает описание исходных функций. Для каждой функции должна быть выполнена программная реализация в рамках ПО «Enc». Программная реализация функций приведена на рисунке 1.

```
def func_1(self):
    def f(x):
        return tf.math.pow(x[0],2) + tf.math.pow(x[1],2) + tf.math.pow(x[2],2) + tf.math.pow(x[3],2) + tf.math.pow(x[4],2) + tf.math.pow(x[5],2) + tf.pow(x[6],2) + tf.pow(x[7],2)

    data_range = [(0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100),  (0, 100), (0, 100)]
    func = Function(f, 'func_1', 8, 0, data_range)
    return func

def func_2(self):
    def f(x):
        return tf.math.pow(x[0],4) + 4 * tf.math.pow(x[0],3) * x[1] + 6 * tf.math.pow(x[0],2) + tf.math.pow(x[1],2) + 4 * x[0] * tf.math.pow(x[1],3) + tf.math.pow(x[1],4)

    data_range = [(0, 25), (0, 25)]
    func = Function(f, 'func_2', 4, 2, data_range)
    return func

def func_3(self):
    def f(x):
        return tf.math.pow(x[0] - 100, 2) + tf.math.pow(x[1] + 3, 2) + 5 * tf.math.pow(x[2] + 10, 2)

    data_range = [(0, 100), (0, 100), (0, 100)]
    func = Function(f, 'func_3', 6, 3, data_range)
    return func

def func_4(self):
    def f(x):
        return tf.math.pow(x[0] - 1, 2) + tf.math.pow(x[1], 2) + x[2] + 2 * x[3] + tf.math.pow(x[4], 3) + x[5]

    data_range = [(0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100)]
    func = Function(f, 'func_4', 10, 4, data_range)
    return func
```

Рисунок 1 – реализация функций

Необходимо определить функцию $f(X)$, задать вектор $(data\_range)$ диапазонов допустимых значений для компонент вектора $X$. Параметрами $Function$ являются функция, имя функции, размерность пространства параметров, количество незначащих параметров, вектор диапазонов.

В качестве входных данных были даны следующие четыре функции:

1. $F(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2$

Размерность пространства параметров $= 8$

Количество незначащих переменных $= 0$

Не спектр, нелинейная

Область определения $x_i \in R, i = \overline{1,8}$

Область значений $\mathrm{F}(x) \in [0, +\infty]$

2. $F(x) = {x_1}^4 + 4 * {x_1}^3 * x_2 + 6 * {x_1}^2 + x_2^2 + 4 * x_1 * {x_2}^3 + x_2^4$

Размерность пространства параметров $= 4$

Количество незначащих переменных $= 2$

Спектр, нелинейная

Область определения $x_i \in R, i = \overline{1,4}$

Область значений $\mathrm{F}(x) \in [0, +\infty]$

3. $F(x) = (x_1 - 100)^2 + (x_2 + 3)^2 + 5 * (x_3 + 10)^2$

Размерность пространства параметров $= 6$

Количество незначащих переменных $= 3$

Спектр. нелинейная

Область определения $x_i \in R, i = \overline{1,6}$

Область значений $\mathrm{F}(x) \in [0, +\infty]$

4. $F(x) = (x_1 - 1)^2 + {x_2}^2 + x_3 + 2 * x_4 + {x_5}^3 + x_6$

Размерность пространства параметров $= 10$

Количество незначащих переменных $= 4$

Спектр. нелинейная

Область определения $x_i \in R, i = \overline{1,10}$

Область значений $\mathrm{F}(x) \in [-\infty, +\infty]$

## 3. Выходные данные

К выходным данным ПО «Enc» относятся:

- описание структуры и параметров обученной нейронной сети, обеспечивающей сжатие параметров исходной функции

Сохранение весов происходит в соответствии со стандартной функцией сохранения весов API Keras. Файл с весами будет сохранен в папке encoderProject-master\Saved models\Weights.

Файл с параметрами нейронной сети будет сохранен в папке encoderProject-master\Saved models\Params. В файле будут записаны выбранные гиперпараметры нейронной сети. Тип автоэнкодера указан в названии файла. Пример файла с параметрами приведен на рисунке 2.

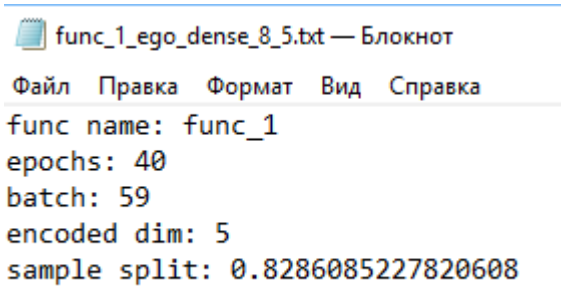


Рисунок 2 – файл, содержащий параметры автоэнкодера.

- показатели нейронной сети по степени сжатия и точности аппроксимации

Для обученной сети ПО «Enc» строит графики отклонений по всем параметрам и указывает среднее отклонение по $Y$. Диапазон каждого параметра разбит на 10 равных частей для выявления областей с наибольшей ошибкой. Пример графиков указан на рисунке 3.

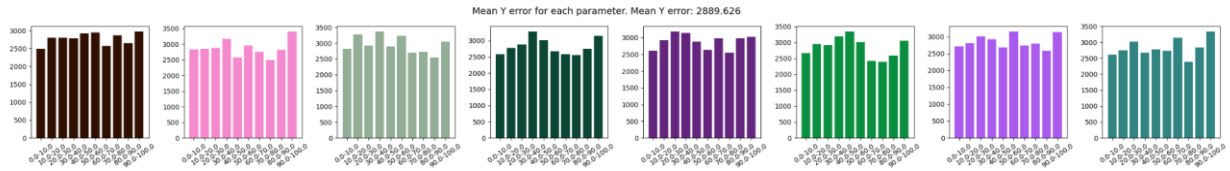


Рисунок 3 – Графики отклонений по параметрам функции

## 4. Разработка автоэнкодера

### 4.1. Создание обучающей выборки. Задание исходной функции. Функция потерь. Структура автоэнкодера

Для каждой компоненты вектора $X$ задается диапазон допустимых значений. Создается $n$ векторов $X$, являющих последовательностью Соболя – квазислучайной последовательностью с низкой расходимостью. Данная функция реализована с помощью библиотеки sobol_seq [https://github.com/naught101/sobol_seq]. На рисунке 4 продемонстрировано расположение 1000 точек последовательности Соболя на двумерной плоскости, а также рассчитана их дисперсия $= 0.0832$, при увеличении количества точек дисперсия будет уменьшаться и стремиться к константе. Доказательство, что последовательность Соболя является последовательностью с низкой расходимостью, приведено в *[Соболь, И.М. и Левитан, Ю.Л. (1976). «Производство точек, равномерно распределенных в многомерном кубе» Тех. Доп. 40, Институт прикладной математики АН СССР]*.

```
import matplotlib.pyplot as plt
d = sobol_seq.i4_sobol_generate(2, 1000)
x, y = np.array_split(d, 2, axis = 1)
plt.plot(x,y,'o', color = 'orange')
print(f'Variance: {np.var((x, y)):.4f}')
```

```
Variance: 0.0832
```

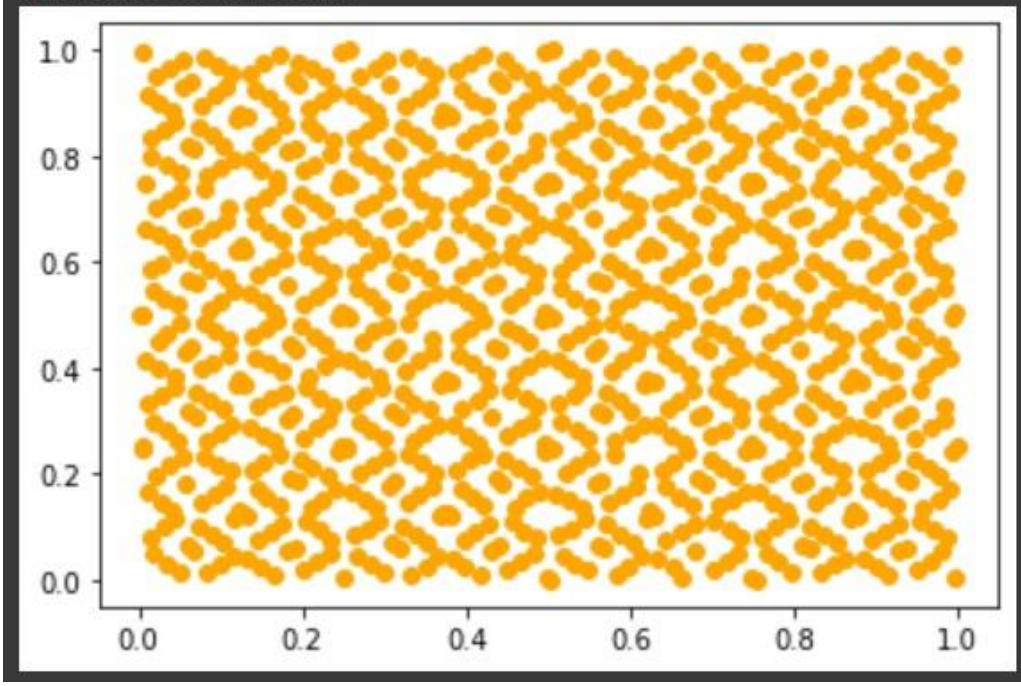


Рисунок 4 – расположение 1000 точек последовательности Соболя.

$d\%$ векторов выбирается для тренировочной выборки, оставшиеся выбираются для валидационной выборки.

### 4.1.1. Первая модель автоэнкодера

Для каждого вектора $X$ высчитывается соответствующий вектор $Y$ и конкатенируется с ним. Получившийся вектор подается на вход автоэнкодера. Функцией потерь выбирается средняя разность квадратов между поданным на вход вектором и выходным вектором автоэнкодера.

### 4.1.2. Вторая модель автоэнкодера

Вектор $X$ подается на вход автоэнкодера. Функцией потерь выбирается средняя абсолютная разность между вектором $Y$, рассчитанном по входному вектору $X$ и вектором $Y'$, рассчитанном по выходному вектору автоэнкодера.

## 4.2.Описание структуры автоэнкодера

Общая структура автоэнкодеров. Входной слой, слои кодировщика, внутренний слой – последний слой кодировщика, входной слой декодировщика, слои декодировщика, выходной слой – последний слой декодировщика.

Входной слой, слои кодировщика и внутренний слой образуют структуру кодировщика (Encoder).

Входной слой декодировщика, слои декодировщика и выходной слой образуют структуру декодировщика (Decoder).

### 4.2.1. Однослойный автоэнкодер

Слои кодировщика и декодировщика имеют по одному полносвязному слою.

### 4.2.2. Двухслойный автоэнкодер

Слои кодировщика и декодировщика имеют по два полносвязных слоя.

### 4.2.3. Вариационный автоэнкодер

В слое декодировщика входной вектор сжимается до размера внутреннего слоя полносвязным слоем, получившийся вектор принимается за математическое ожидание и логарифм дисперсии. Создаётся вектор случайных величин с нормальным распределением, математическое ожидание и дисперсию которых получили на предыдущем слое. Слой декодировщика имеет два полносвязных слоя.

## 5. Разработка алгоритма выбора гиперпараметров для построения автоэнкодера с наилучшими показателями точности и сжатия

Для тестирования были выбраны следующие гиперпараметры:

$m$ – размер внутреннего слоя

$d$ – процент разделения тестовых данных на обучающую выборку и валидационную

$enc\_type$ – тип автоэнкодера

$epoch$ – количество эпох обучения

$batch$ – количество пакетов, на которое разобьётся выборка

Для построения автоэнкодера с наилучшими показателями точности и сжатия был реализован алгоритм полного перебора гиперпараметров. Данный алгоритм работает очень долго и для ускорения был применен алгоритм эффективной глобальной оптимизации. (smt.applications.EGO)

Алгоритм эффективной глобальной оптимизации основан на Байесовской оптимизации. Более подробно с данным алгоритмом можно ознакомиться в статье [*Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. Journal of Global optimization, 13(4), 455-492.*] или в документации к используемой библиотеке [https://smt.readthedocs.io/en/latest/_src_docs/applications/ego.html]. В проекте используется алгоритм EGO с количеством начальных точек n + 1, где n – размерность исходного пространства.

Алгоритм полного перебора обучает нейросеть для каждого набора гиперпараметров. Гиперпараметры выбираются следующим образом:

$enc\_type$ - будут рассмотрены все разработанные типы автоэнкодеров: однослойный, двухслойный и вариационный

$epoch$ перебирается с 5 по 55 с шагом 5

$batch$ перебирается как степени двойки с $2^4$ по $2^8$

$m$ перебирается с половины размерности пространства параметров по размерность пространства параметров минус один с шагом один

$d$ перебирается с 0.5 по 0.9 с шагом 0.1

Для нахождения решения генерировались наборы данных размерностью 60000.

## 6. Тестирование

Результаты тестирования автоэнкодеров приведены на рисунках 5-16. Тестирование проводилось алгоритмом эффективной глобальной оптимизации при количестве эпох = 25 для всех типов автоэнкодера.

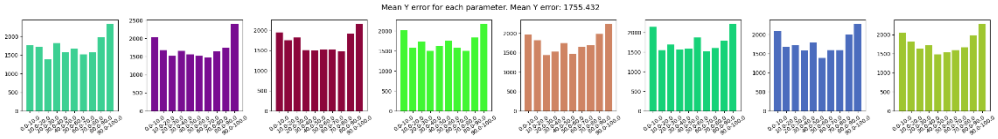


Рисунок 5 – Функция 1, однослойный автоэнкодер

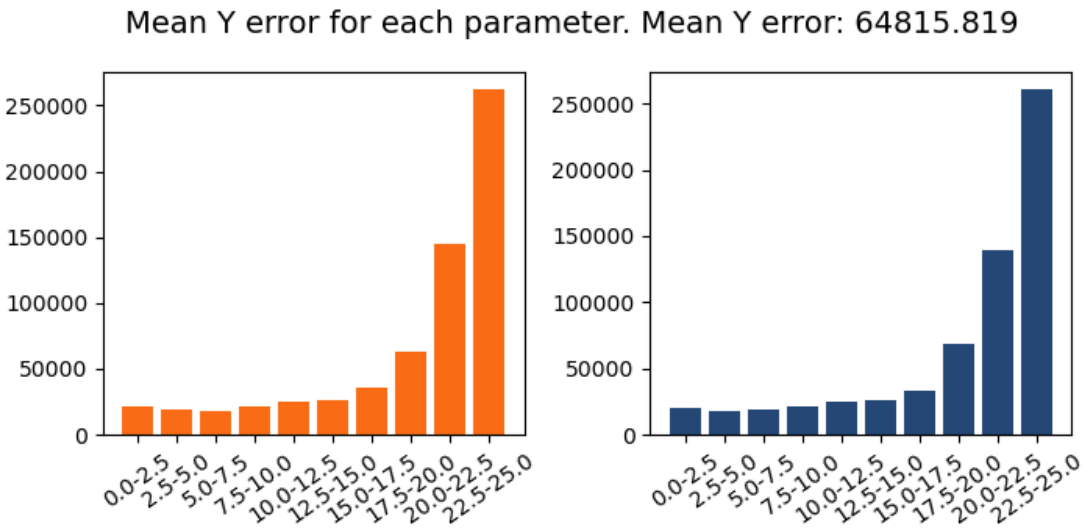


Рисунок 6 – Функция 2, однослойный автоэнкодер

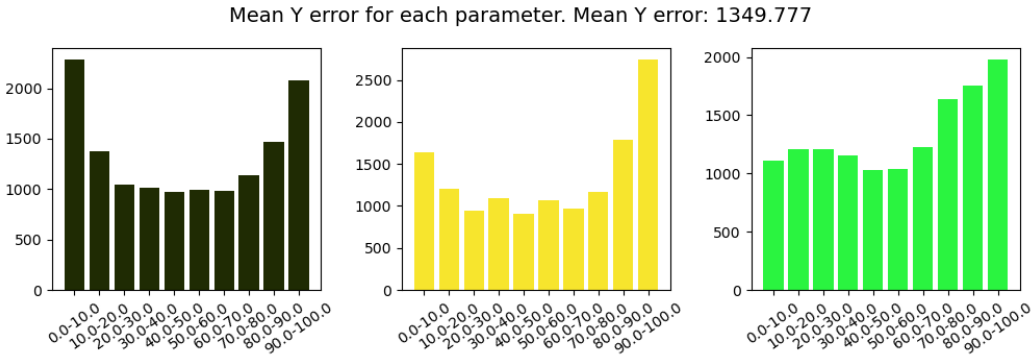


Рисунок 7 – Функция 3, однослойный автоэнкодер


Рисунок 8 – Функция 4, однослойный автоэнкодер

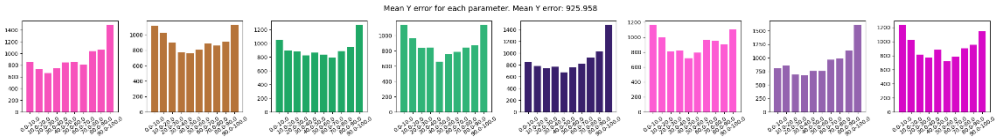


Рисунок 9 – Функция 1, двухслойный автоэнкодер

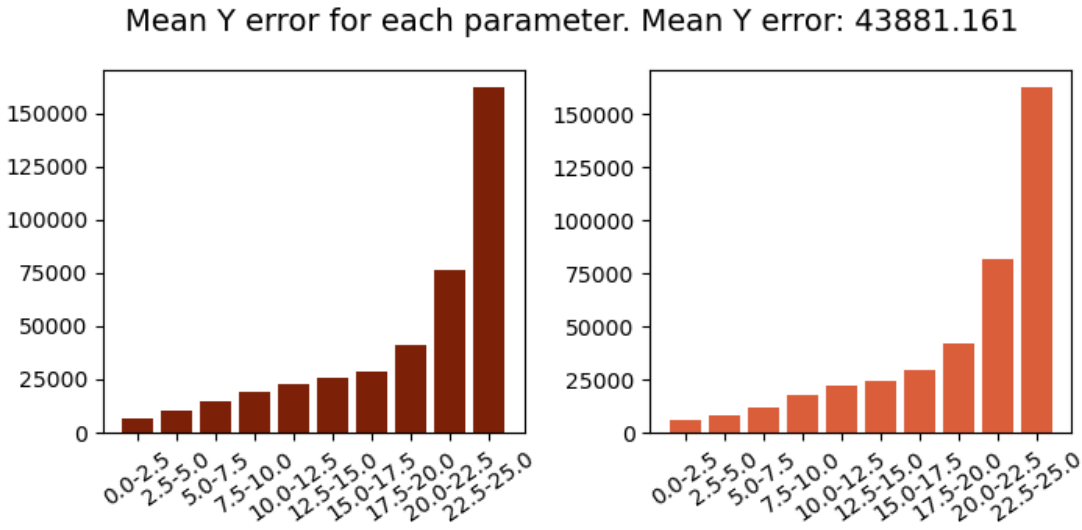


Рисунок 10– Функция 2, двухслойный автоэнкодер

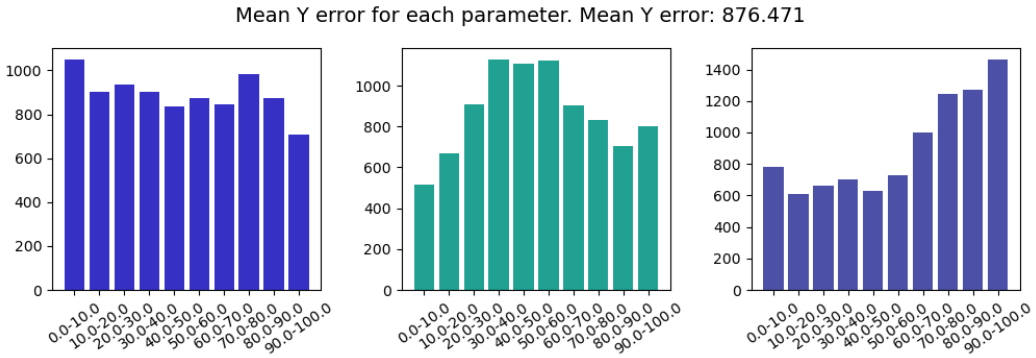


Рисунок 11 – Функция 3, двухслойный автоэнкодер

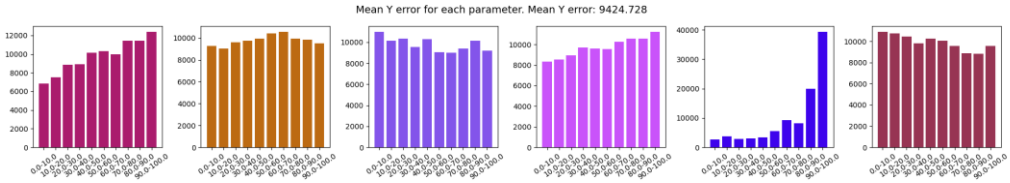


Рисунок 12 – Функция 4, двухслойный автоэнкодер

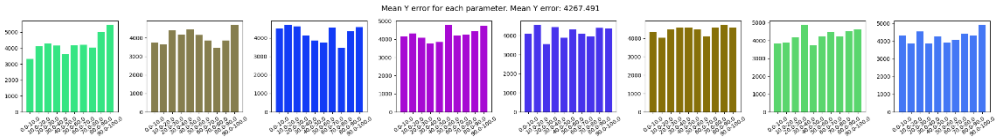


Рисунок 13 – Функция 1, вариационный автоэнкодер

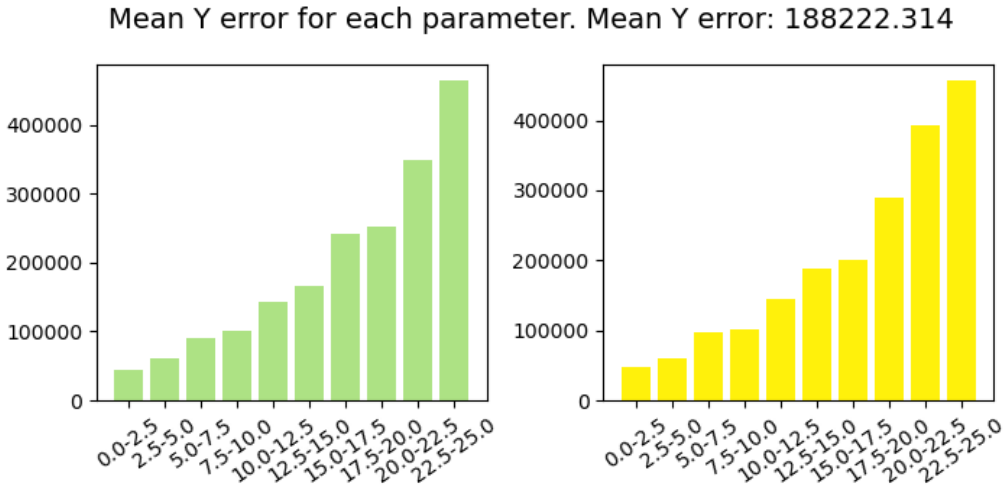


Рисунок 14 – Функция 2, вариационный автоэнкодер

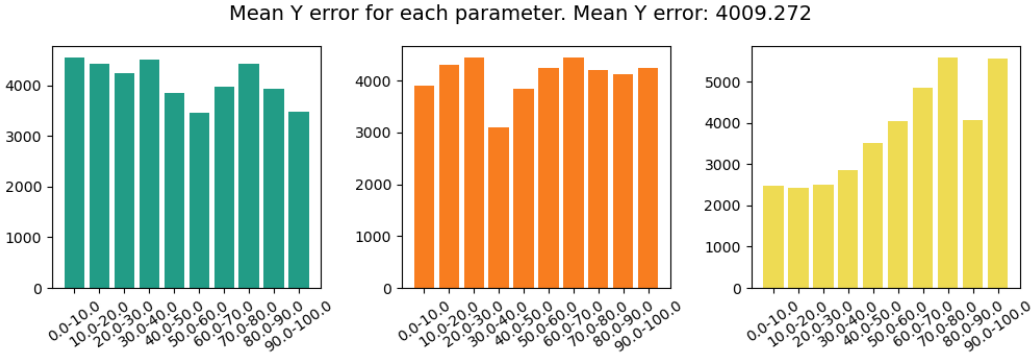


Рисунок 15 – Функция 3, вариационный автоэнкодер

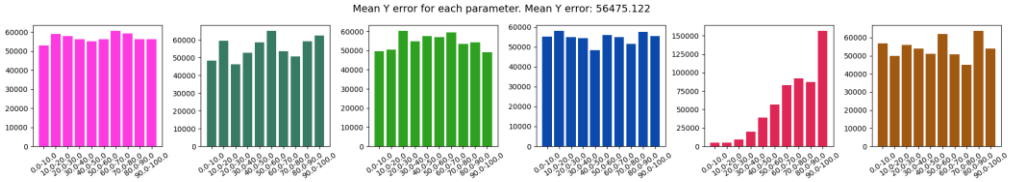


Рисунок 16 – Функция 4, вариационный автоэнкодер

## Заключение

Была поставлена задача разработки нейронной сети специального вида (автоэнкодера) для решения задачи редукции пространства многомерных функций. В ходе работы все требования к функциональным характеристикам разрабатываемого ПО были выполнены. Из результатов видно, что двухслойная сеть в 1.5 раза лучше по результатам, чем однослойная. Также заметно, что вариационный в среднем лучше сжимает пространство, но по отклонению в текущей реализации не так хорош, как хотелось бы.