Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №6 по дисциплине "Методы машинного обучения"

Тема Классификатор на базе многослойного персептрона

Студент Варламова Е. А.

Группа ИУ7-23М

Оценка (баллы) _____

Преподаватели Солодовников Владимир Игоревич

Москва — 2024 г.

СОДЕРЖАНИЕ

1 Теоретическая часть **3**
1.1 Постановка задачи 3
1.2 Основные понятия нейронной сети 4
1.3 Обучение нейросети 5
1.4 Функция активации ReLU 6
1.5 Сигмоидная функция активации 6

2 Практическая часть **8**
2.1 Выбор средств разработки 8
2.2 Исследование ПО 8

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **18**

1 | Теоретическая часть

Многослойный персептрон – это тип нейронной сети, состоящей из нескольких слоев нейронов, которые взаимодействуют друг с другом. Он является одним из наиболее распространенных видов искусственных нейронных сетей и широко применяется в области машинного обучения и глубокого обучения. Многослойный персептрон состоит из входного слоя, скрытых слоев и выходного слоя, причем каждый слой содержит нейроны, которые передают сигналы друг другу и выполняют сложные вычисления. Благодаря способности обучаться на основе данных и корректировать свои веса, многослойный персептрон способен решать разнообразные задачи, такие как классификация, регрессия и распознавание образов.

Целью данной лабораторной работы является применение многослойного персептрона для решения задачи классификации областей признакового пространства. Для этого необходимо решить следующие задачи:

— формализовать задачу;

— описать алгоритм работы многослойного персептрона;

— привести особенности реализации ПО, решающего поставленную задачу;

— оценить точность, полноту, F-меру классификатора; построить матрицу ошибок; построить графики изменения среднеквадратических ошибок на обучающей и тестовой выборках.

— провести исследование работы нейросети в зависимости от выбранной функции активации.

1.1 Постановка задачи

Осуществить генерацию исходных данных, которые представляют собой двумерное признаковое пространство, сгруппированное в 6 или более областей, отнесенных не менее чем к 4 классам. В каждой области содержится не менее 50 примеров, и данные распределены по нормальному закону распределения.

В ходе выполнения работы:

1. Визуализировать сгенерированные данные на плоскости.
2. Для сгенерированного датасета осуществить построение классификатор на базе многослойного персептрона.
3. Обосновать выбор числа слоев и нейронов в каждом слое.
4. Сравнить работу нейросети в зависимости от выбранной функции активации (сигмоида с разными значениями параметра, определяющего крутизну (b=1,b=100), ReLU).
5. В процессе обучения визуализировать разделяющие поверхности промежуточного слоя.
6. В процессе обучения построить графики изменения среднеквадратических ошибок на обучающей и тестовой выборках. Обосновать момент остановки процесса обучения. Оценить точность, полноту, F-меру. Построить матрицу ошибок.
7. Предусмотреть дополнительную возможность ввода пользователем новых, не входящих в сгенерированный датасет данных. Визуализировать их совместно с обучающей выборкой и разделяющими поверхностями, осуществить их классификацию.

1.2 Основные понятия нейронной сети

Перечислим основные понятия, используемые в нейросетях.

— Нейрон (или узел): Нейрон – это базовый элемент нейронной сети, имитирующий функцию биологического нейрона. Он принимает входные сигналы, обрабатывает их и генерирует выходной сигнал. Каждый нейрон обычно имеет несколько входов, каждый из которых соответствует весу (степени важности) исходного сигнала.

— Веса: Веса представляют собой параметры, которые определяют степень важности входной информации для каждого нейрона. Они отражают силу связей между нейронами и являются ключевыми для эффективного обучения нейронной сети.

— Функция активации: Функция активации определяет выходное значение нейрона на основе его взвешенного входа и возможно добавления смещения. Различные функции активации, такие как ReLU, сигмоида, tanh, используются для введения нелинейности в нейронные сети.

— Структура и связи: Нейронные сети могут быть организованы в различные архитектуры, такие как многослойные перцептроны, сверточные нейронные сети, рекуррентные нейронные сети и другие. Связи между нейронами формируют слои и определяют поток информации в сети.

— Функция потерь: Функция потерь измеряет разницу между предсказанными значениями модели и фактическими значениями. Во время обучения нейронная сеть минимизирует эту функцию, чтобы улучшить качество своих прогнозов.

— Оптимизатор: Оптимизатор используется для коррекции весов нейронов в ходе обучения, с целью минимизации функции потерь. Примером может служить алгоритм градиентного спуска.

— Слои: Нейронные сети состоят из различных слоев, таких как входной, скрытый и выходной слои. Каждый слой выполняет определенные вычисления и обработку данных.

— Обучающие данные и цели: Нейронные сети обучаются на основе обучающих данных и их соответствующих целей (или меток). Эти данные используются для корректировки весов нейронов в процессе обучения.

1.3 Обучение нейросети

Обучение нейросети состоит из нескольких эпох (итераций). В течение одной эпохи обучения нейронной сети происходит несколько этапов, которые повторяются для каждого обучающего примера:

1. прямое распространение:
    — входные данные подаются на входной слой нейронов;
    — данные передаются через скрытые слои, взвешиваются с использованием соответствующих весов и агрегируются;
    — агрегированные значения проходят через функции активации каждого нейрона в скрытых слоях и выходном слое, что приводит к формированию выходов сети;

2. оценка ошибки : вычисляется ошибка между выходами сети и ожидаемыми значениями (целевыми метками/метками классов);

3. обратное распространение ошибки:

   — ошибка распространяется обратно через сеть, начиная с последнего слоя и двигаясь к входному слою;

   — для каждого слоя вычисляется градиент функции потерь по весам и смещениям сети;

4. обновление весов, чтобы уменьшить ошибку модели: используя градиент ошибки, веса сети обновляются с использованием метода оптимизации, такого как стохастический градиентный спуск или его модификации;

Эти этапы повторяются для каждой эпохи обучения с тем, чтобы постепенно корректировать веса сети и уменьшать ошибку прогноза. Процесс обучения заключается в том, чтобы минимизировать ошибку модели и достичь желаемой производительности в решении конкретной задачи.

1.4 Функция активации ReLU

Функция ReLU – это популярная функция активации в нейронных сетях, которая обладает нелинейными свойствами. Ее формула определяется следующим образом:

$$f(x) = \max(\alpha * x, x) \tag{1.1}$$

Таким образом, ReLU функция активации принимает следующие значения:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha * x, & \text{if } x \leq 0 \end{cases} \tag{1.2}$$

ReLU функция подходит для использования в нейронных сетях по нескольким причинам, включая то, что она обеспечивает нелинейность (что важно для изучения сложных функций) и простоту вычисления.

1.5 Сигмоидная функция активации

Сигмоидная функция активации используется для преобразования взвешенной суммы входов нейрона в диапазоне от 0 до 1. Это позволяет моделировать нелинейные зависимости и обеспечивает гладкое изменение выхода нейрона.

Формула сигмоидной функции активации выглядит следующим образом:

$$\sigma(x) = \frac{1}{1 + e^{-\alpha x}} \tag{1.3}$$

где

— x – входное значение нейрона

— $\sigma(x)$ – выходное значение после применения функции активации

— α – параметр функции активации.

Сигмоидная функция активации широко используется в задачах классификации, так как ее выход можно интерпретировать как вероятность принадлежности к определенному классу. Однако, она имеет недостатки, такие как проблема затухания градиента при обратном распространении ошибки. В связи с этим, сегодня более популярными являются другие функции активации, такие как ReLU и их модификации.

2 | Практическая часть

2.1 Выбор средств разработки

В качестве языка программирования был использован язык Python, поскольку этот язык кроссплатформенный и для него разработано огромное количество библиотек и модулей, решающих разнообразные задачи.

В частности, имеются библиотеки, включающие в себя алгоритм многослойного персептрона в библиотеке [1].

2.2 Исследование ПО

В листинге 2.1 представлен код классификации.

Листинг 2.1: Код классификации

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
import imageio
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix
import seaborn as sns
from sklearn.metrics import mean_squared_error
from tensorflow.keras import utils
import pandas as pd
EPOCHS_COUNT = 50
def draw_line(
        x_data: np.ndarray,
        w_1: float,
        w_2: float,
        w_c: float):
    y_arr = -(w_1 * x_data + w_c) / w_2
    plt.plot(x_data, y_arr, linestyle='-')
    return
def show_lines(x_min: float, x_max: float, neurons: np.ndarray, bias: np.
    ndarray):
    line_data_x = np.arange(x_min, x_max, (x_min + x_max) / 4)
```

```
    # w_1*x + w_2*y + w_c = 0
    # y = -(w_1*x + w_c) / w_2
    for w_1, w_2, w_c in zip(neurons[0], neurons[1], bias):
        draw_line(line_data_x, w_1, w_2, w_c)
    return
def gen_data():
    n_samples = 300
    n_features = 2
    n_classes = 4
    X, y = make_blobs(n_samples=[300, 250, 200, 150], n_features=n_features,
        centers=None, cluster_std=1, random_state=42)
    X_additional, y_additional = make_blobs(n_samples=[150, 300], n_features
       =n_features, centers=None, cluster_std=1, random_state=45)
    X = np.vstack([X, X_additional])
    y = np.hstack([y, y_additional])
    return X, y
def draw_data(X, y):
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolors='k')
    plt.title('                                                    ')
    plt.xlabel('                1')
    plt.ylabel('                2')
    plt.savefig("data_viz.png")

def custom_activation(alpha=1.0):
    def activation(x):
        return 1 / (1 + tf.exp(-alpha * x))
    return activation
def plot_layer(model, X, y, name_graph):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min,
       y_max, 0.1))
    grid_points = np.c_[xx.ravel(), yy.ravel()]

    Z = model.predict(grid_points)
    Z = np.argmax(Z, axis=1).reshape(xx.shape)

    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolors='k')
    plt.title('
                                                ')
    plt.xlabel('                1')
    plt.ylabel('                2')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    show_lines(x_min, x_max, model.layers[0].get_weights()[0], model.layers
       [0].get_weights()[1])

```

```
68      plt.savefig(name_graph)
69      plt.clf()
70
71  X, y = gen_data()
72
73  draw_data(X, y)
74
75  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=42)
76  model_names = ['relu_1', "relu_100", "sigmoid_1", "sigmoid_5"]
77
78  models = [
79      tf.keras.models.Sequential([
80          tf.keras.layers.Dense(6, activation=tf.keras.layers.LeakyReLU(alpha
              =1), input_dim=2),
81          tf.keras.layers.Dense(4, activation='softmax')]), #]#,#]#,
82      tf.keras.models.Sequential([
83          tf.keras.layers.Dense(6, activation=tf.keras.layers.LeakyReLU(alpha
              =100), input_dim=2),
84          tf.keras.layers.Dense(4, activation='softmax')]),
85      tf.keras.models.Sequential([
86          tf.keras.layers.Dense(6, activation=custom_activation(alpha=1.0),
              input_dim=2),
87          tf.keras.layers.Dense(4, activation='softmax')]),
88      tf.keras.models.Sequential([
89          tf.keras.layers.Dense(6, activation=custom_activation(alpha=5),
              input_dim=2),
90          tf.keras.layers.Dense(4, activation='softmax')])     ]
91  model_num = 0
92  for model in models:
93      model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
94      name = model_names[model_num]
95      epochs_ctr = 0
96
97      images = []
98      errs_train = []
99      errs_test = []
100     layers = np.arange(1, EPOCHS_COUNT + 1)
101     def plot_intermediate_layer(model, X):
102         global epochs_ctr
103         epochs_ctr += 1
104         plot_layer(model, X, y_test, f"img_{name}_{epochs_ctr}.png")
105         y_train_pred = model.predict(X_train)
106         y_train_pred = np.argmax(y_train_pred, axis=1)
107         y_test_pred = model.predict(X_test)
108         y_test_pred = np.argmax(y_test_pred, axis=1)
109         errs_train.append(mean_squared_error(y_train, y_train_pred))
110         errs_test.append(mean_squared_error(y_test, y_test_pred))
111
```

```
112     history = model.fit(X_train, y_train, batch_size=5, epochs=EPOCHS_COUNT,
            validation_data=(X_test, y_test), callbacks=[tf.keras.callbacks.
        LambdaCallback(on_epoch_end=lambda epoch, logs:
        plot_intermediate_layer(model, X_test))])
113
114     plt.plot(layers, errs_train, label='                                    ')
115     plt.plot(layers, errs_test, label='
        ')
116     plt.legend()
117     plt.title('      ')
118     plt.savefig(f"std_{name}.png")
119     plt.clf()
120
121     for i in range(1, EPOCHS_COUNT+1):
122         images.append(imageio.imread(f"img_{name}_{i}.png"))
123     imageio.mimsave(f'{name}_training_history.gif', images, duration=1000)
124
125     y_pred = model.predict(X_test)
126     y_pred = np.argmax(y_pred, axis=1)
127
128     accuracy = accuracy_score(y_test, y_pred)
129     precision = precision_score(y_test, y_pred, average='weighted')
130     recall = recall_score(y_test, y_pred, average='weighted')
131     f1 = f1_score(y_test, y_pred, average='weighted')
132     conf_matrix = confusion_matrix(y_test, y_pred)
133
134     sns.heatmap(conf_matrix, annot=True)
135     plt.title(f'acc={accuracy:.2f}, prec={precision:.2f}, Recall={recall:.2f
        }, f1={f1:.2f}')
136
137     plt.savefig(f"matrix_errors_{name}.png")
138     plt.clf()
139     model_num += 1
140
141     plot_layer(model, X_test, y_test, f"before_{name}.png")
142     want = "0"
143     u = 0
144     while want == "1":
145         print("                                    ? (1 —     )")
146         want = input()
147         if want == "1":
148             x1 = float(input("x1: "))
149             x2 = float(input("x2: "))
150             cluster = int(input("cluster: (0, 1, 2, 3): "))
151
152
153             plot_layer(model, np.vstack([X_test, [x1, x2]]), np.append(
                y_test, cluster), f"after_{name}_{u}.png")
154             u += 1
```

Были сгенерированы входные данные, показанные на рисунке 2.1.

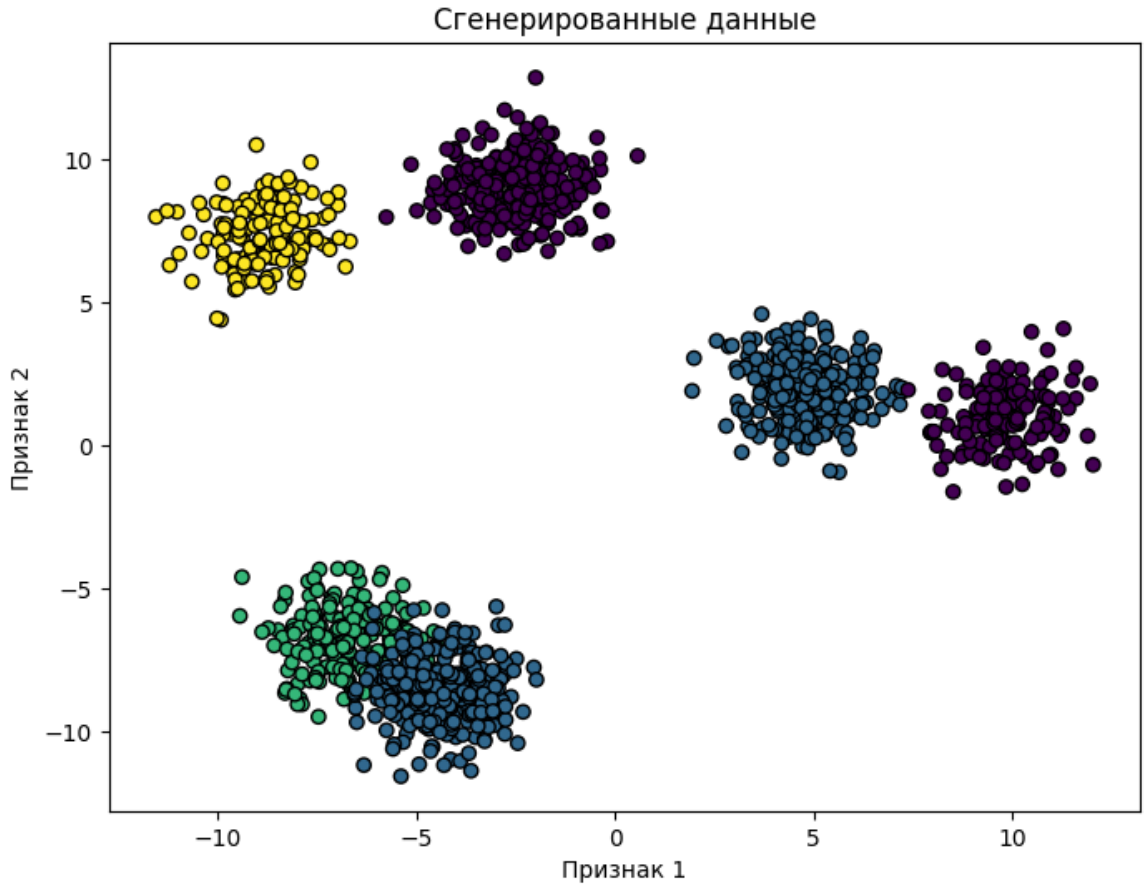

Рис. 2.1: Входные данные

Были использованы 4 функции активации для обучения 4 классификаторов на основе многослойного персептрона:

1. ReLU со значением $\alpha = 1$;
2. ReLU со значением $\alpha = 100$;
3. сигмоидная со значением $\alpha = 1$;
4. сигмоидная со значением $\alpha = 5$.

Для каждого классификатора было необходимо оценить точность, полноту, F-меру классификатора; построить матрицу ошибок; построить графики изменения среднеквадратических ошибок на обучающей и тестовой выборках. Соответствующие значения приведены на рисунках 2.2-2.9.

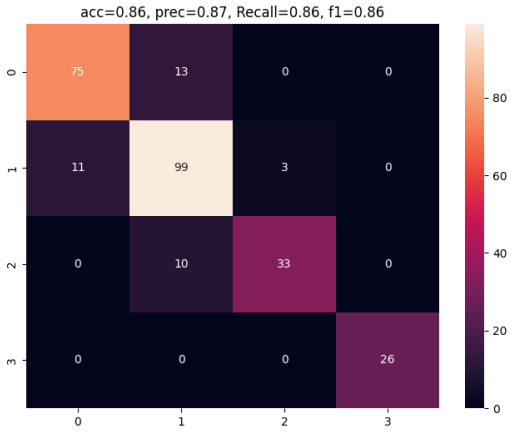

Рис. 2.2: матрица ошибок и оценки точности для ReLU, alpha = 1

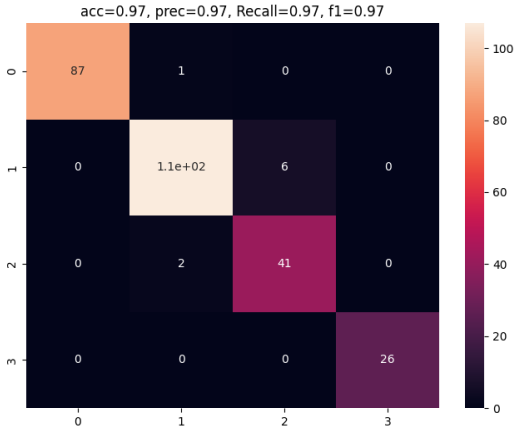

Рис. 2.3: матрица ошибок и оценки точности для ReLU, alpha = 100

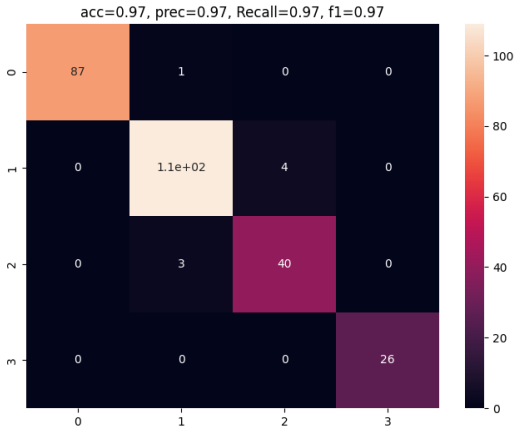

Рис. 2.4: матрица ошибок и оценки точности для сигмоидной, alpha = 1

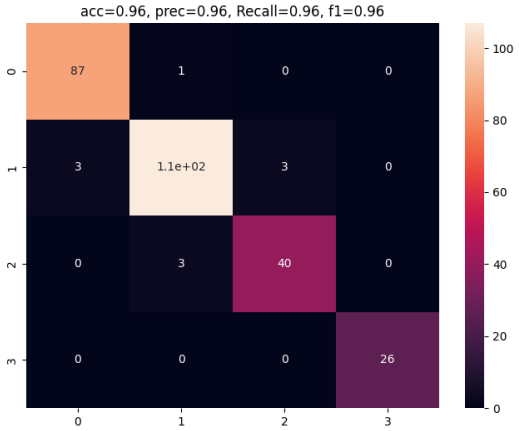

Рис. 2.5: матрица ошибок и оценки точности для сигмоидной, alpha = 5

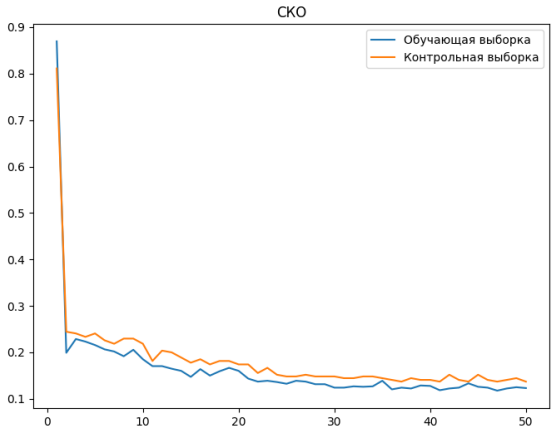

Рис. 2.6: среднеквадратические ошибки для ReLU, alpha = 1

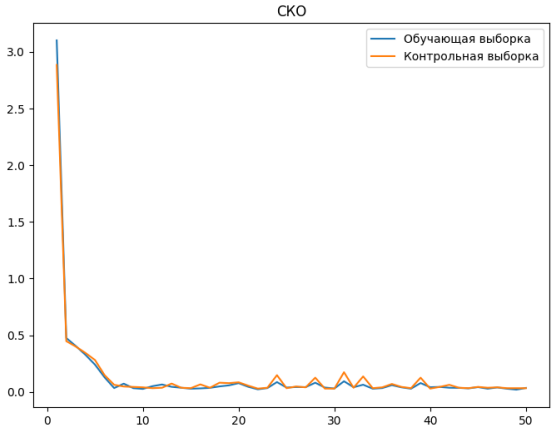

Рис. 2.7: среднеквадратические ошибки для ReLU, alpha = 100

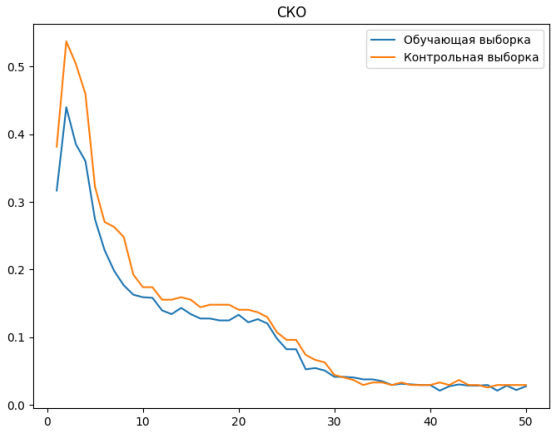

Рис. 2.8: среднеквадратические ошибки для сигмоидной, alpha = 1

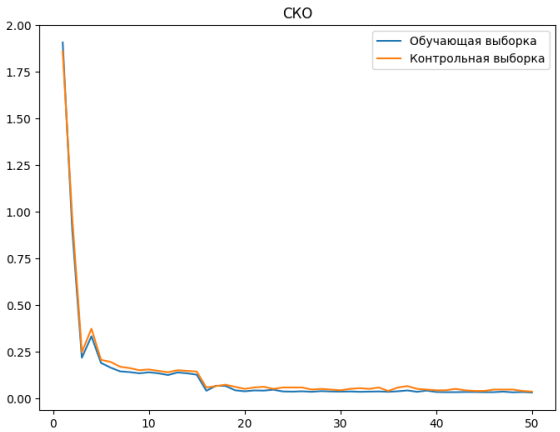

Рис. 2.9: среднеквадратические ошибки для сигмоидной, alpha = 5

Для обучения классифкатора был выбран один промежуточный слой нейронов и 6 нейронов в нем. Определение оптимального числа нейронов было осуществлено из геометрических соображений после оценки сгенерированных данных.

Правильность выбора 6 нейронов промежуточного слоя может быть проилюстрирована следующим образом. Построим прямые с помощью весов и смещений промежуточного слоя.

На рисунках 2.10-2.11 видим, что прямые проходят по границам областей на моделях с сигмоидной функцией активации.

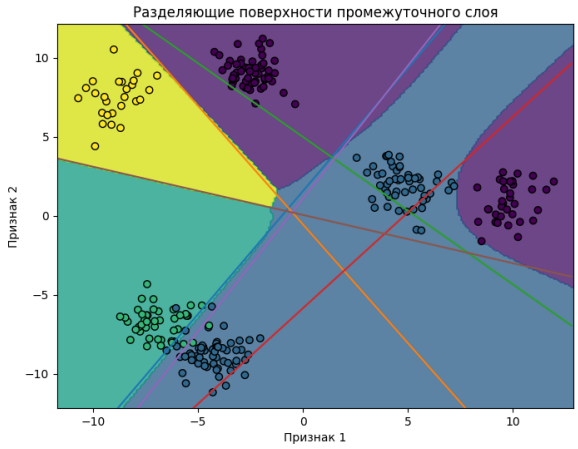

Рис. 2.10: разделяющие поверхности промежуточного слоя для сигмоидной, alpha = 1

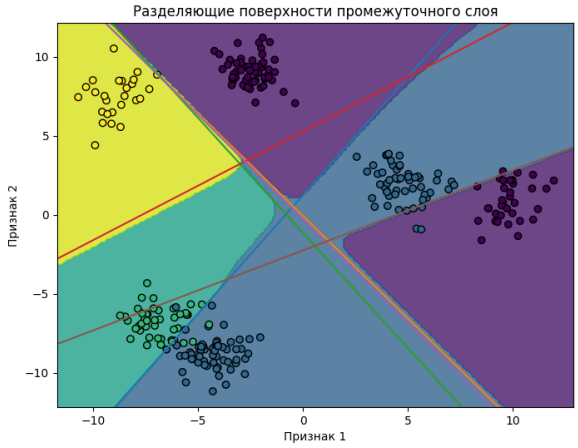

Рис. 2.11: разделяющие поверхности промежуточного слоя для сигмоидной, alpha = 5

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Virtanen P.*, *Gommers R.*, *Oliphant T. E.* SciPy: Fundamental Algorithms for Scientific Computing in Python. — 2020. — DOI: 10.1038/s41592-019-0686-2.