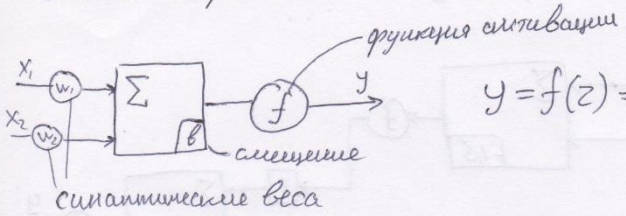
Создание простой нейронной сети в ручном режиме.

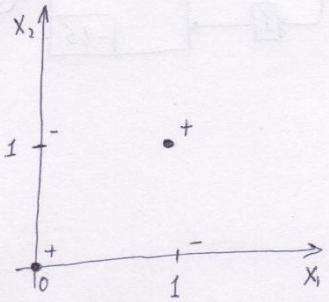
Схема нейрона.



$$y = f(z) = f(w_1 x_1 + w_2 x_1 + \dots + b)$$

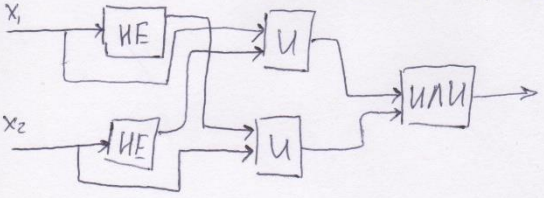
Задача: решить задачу „исключающего или" с помощью нейронной сети.

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0



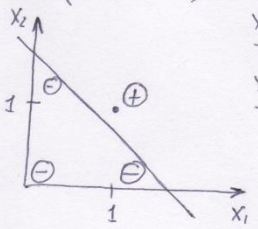
С помощью одного нейрона нельзя решить данную задачу, так как один нейрон имеет линейную разделяющую поверхность.

Как известно $x_1 \oplus x_2 = (\bar{x}_1 \& x_2) \vee (x_1 \& \bar{x}_2)$

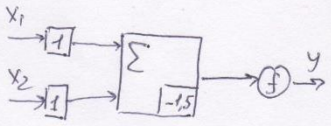


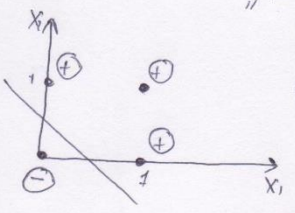
$$f(x) = \begin{cases} 1 \text{ при } x > 0 \\ 0 \text{ при } x \le 0 \end{cases}$$

будем использовать пороговую ф-цию активации.

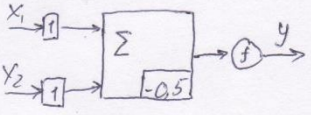


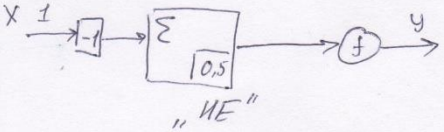
„И"



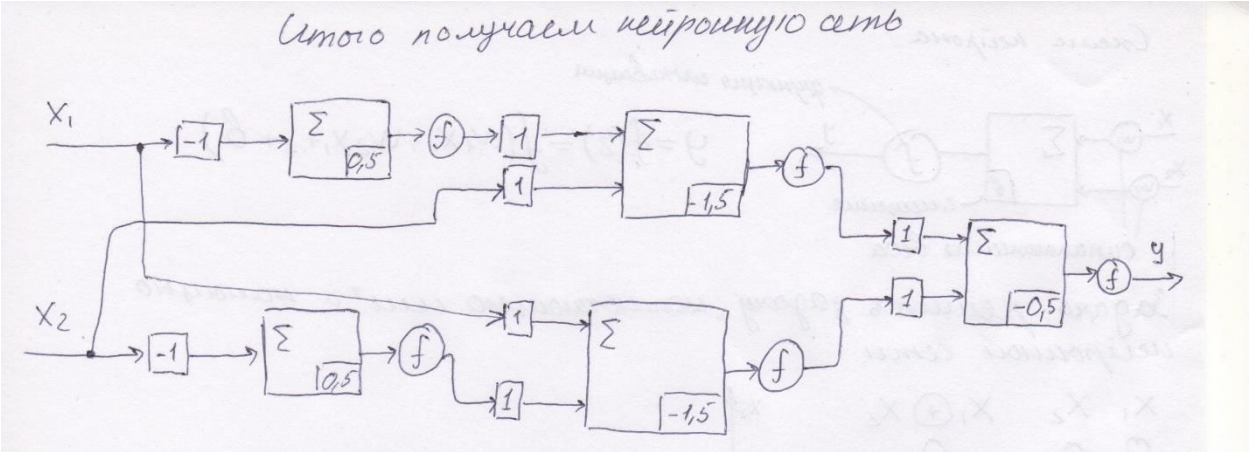


„ИЛИ"





„НЕ"



Обучение нейронной сети.

Параметры разработанной сети

Функция активации: сигмоида

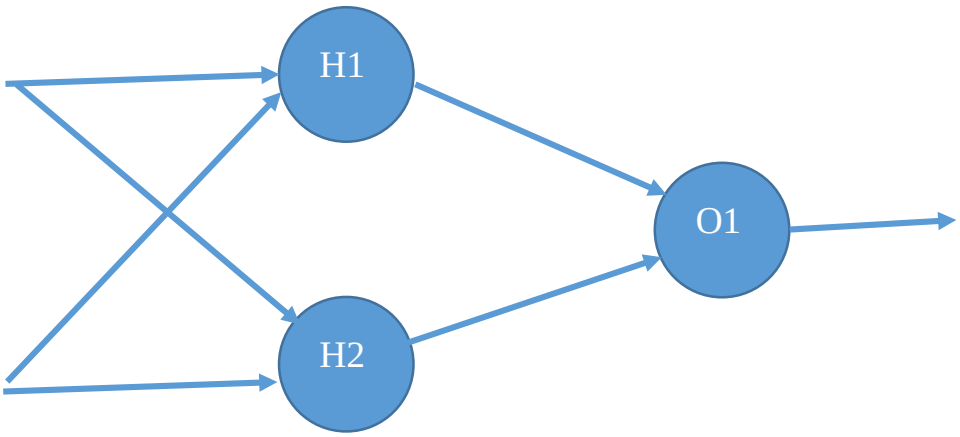
Функция потерь: средний квадрат ошибки

Метод оптимизации: стохастический градиентный спуск

Шаг обучения: фиксированный 0.1

Количество эпох обучения: 10000

Архитектура сети:



Код программы, реализующий нейронную сеть на Python.

```
import numpy as np


def sigmoid(x):
    # Функция активации sigmoid:: f(x) = 1 / (1 + e^(-x))
```

```
    return 1 / (1 + np.exp(-x))


def deriv_sigmoid(x):
    # Производная от sigmoid: f'(x) = f(x) * (1 - f(x))
fx = sigmoid(x)     return fx * (1 - fx)


def mse_loss(y_true, y_pred):
    # y_true и y_pred являются массивами numpy с одинаковой длиной
return ((y_true - y_pred) ** 2).mean()


class SimpleNeuralNetwork:
    """
    Нейронная сеть, у которой:
- 2 входа
- скрытый слой с двумя нейронами (h1, h2)
- слой вывода с одним нейроном (o1)
    """     def
__init__(self):
        # Вес         self.w1 =
np.random.normal()        self.w2 =
np.random.normal()        self.w3 =
np.random.normal()        self.w4 =
np.random.normal()        self.w5 =
np.random.normal()        self.w6 =
np.random.normal()        #
Смещения         self.b1 =
np.random.normal()        self.b2 =
```

```
np.random.normal()        self.b3 =
np.random.normal()

    def feedforward(self, x):
        # x является массивом numpy с двумя элементами
h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
return o1

    def train(self, X, y, learn_rate=0.1, epochs=10000):
        """
- X is a (n x 2) numpy array, n = # of samples in the dataset.
- y is a numpy array with n elements.
        """

        for epoch in range(epochs):
for x, y_true in zip(X, y):
            # --- Выполняем обратную связь (нам понадобятся эти значения в
дальнейшем)
            sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
h1 = sigmoid(sum_h1)

            sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
h2 = sigmoid(sum_h2)

            sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
o1 = sigmoid(sum_o1)
            y_pred = o1
```

```
            # --- Подсчет частных производных
            # --- Наименование: d_L_d_w1 представляет " dL / dw1"
d_L_d_ypred = -2 * (y_true - y_pred)

            # Нейрон o1             d_ypred_d_w5 = h1 *
deriv_sigmoid(sum_o1)             d_ypred_d_w6 = h2 *
deriv_sigmoid(sum_o1)             d_ypred_d_b3 =
deriv_sigmoid(sum_o1)

            d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

            # Нейрон h1             d_h1_d_w1 = x[0] *
deriv_sigmoid(sum_h1)             d_h1_d_w2 = x[1] *
deriv_sigmoid(sum_h1)             d_h1_d_b1 =
deriv_sigmoid(sum_h1)

            # Нейрон h2             d_h2_d_w3 = x[0] *
deriv_sigmoid(sum_h2)             d_h2_d_w4 = x[1] *
deriv_sigmoid(sum_h2)             d_h2_d_b2 =
deriv_sigmoid(sum_h2)

            # --- Обновляем вес и смещения
            # Нейрон h1             self.w1 -= learn_rate * d_L_d_ypred *
d_ypred_d_h1 * d_h1_d_w1                  self.w2 -= learn_rate * d_L_d_ypred *
d_ypred_d_h1 * d_h1_d_w2                  self.b1 -= learn_rate * d_L_d_ypred *
d_ypred_d_h1 * d_h1_d_b1                  # Нейрон h2                 self.w3 -=
learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3                  self.w4 -=
learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4                  self.b2 -=
learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2
```

```
            # Нейрон o1            self.w5 -= learn_rate *
d_L_d_ypred * d_ypred_d_w5              self.w6 -= learn_rate *
d_L_d_ypred * d_ypred_d_w6              self.b3 -= learn_rate *
d_L_d_ypred * d_ypred_d_b3

        # --- Подсчитываем общую потерю в конце каждой фазы
if epoch % 1000 == 0:
            y_preds = np.apply_along_axis(self.feedforward, 1, X)
loss = mse_loss(y, y_preds)             print("Epoch %d loss:
%.3f" % (epoch, loss))



# Определение набора данных
X = np.array([
  [0, 0],
  [0, 1],
  [1, 0],
  [1, 1],
])

y = np.array([
  0,
  1,
  1,
  0,
])
# Тренируем нейронную сеть network
= SimpleNeuralNetwork()
network.train(X, y)
```

#Тестируем сеть

test1 = np.array([0, 0]) test2 = np.array([1, 0]) test3 = np.array([3, 2]) # посмотрим на поведение нейронной сети на других значениях

print("Test1: %.3f" % network.feedforward(test1)) print("Test2: %.3f" % network.feedforward(test2)) print("Test3: %.3f" % network.feedforward(test3))

Результат работы написанной программы.

Как можно заметить, на данных, которые отличаются от обучающей выборки нейронная сеть делает предсказания более неточно. Увеличение числа эпох обучения улучшает результат модели на тренировочной выборке. Но не стоит забывать про эффект переобучения модели. Его можно отследить только если тестовые данные будут отличаться от тех данных, на которых модель обучалась. В данной задаче из-за очень маленькой выборки данных это реализовать не получилось.

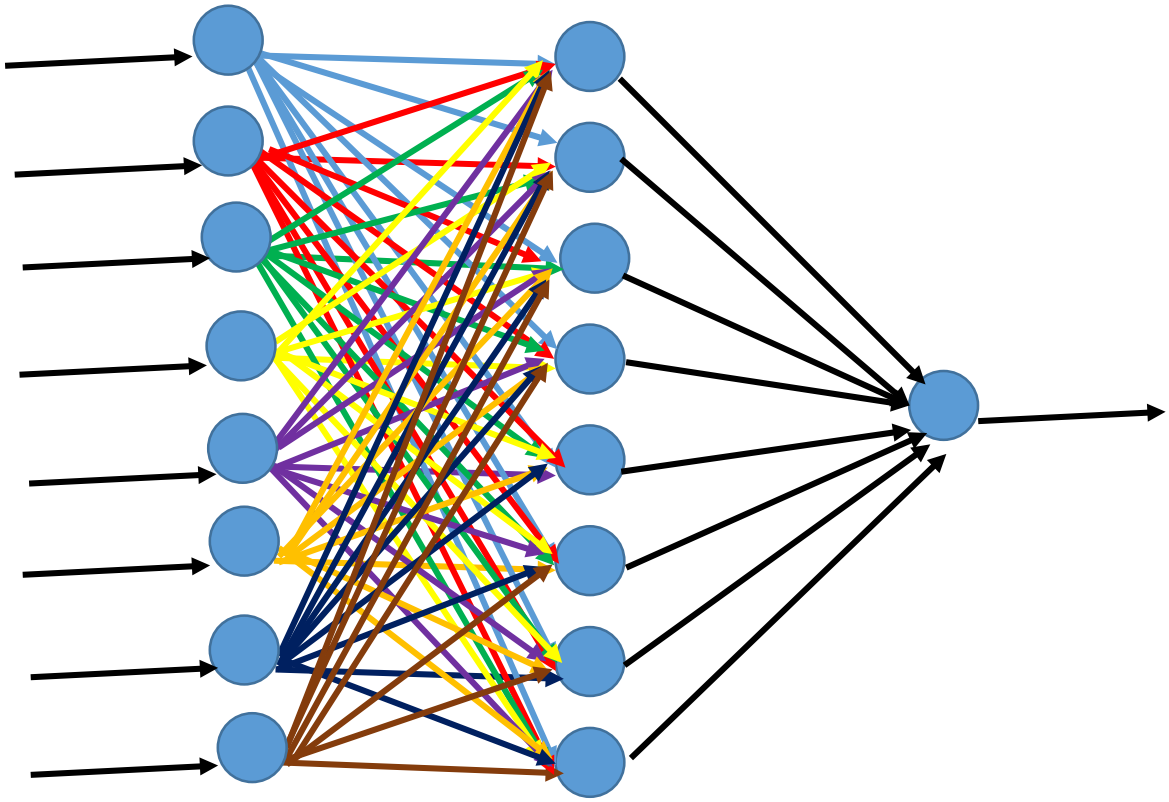
Таблица данных для решения задачи.

X		Y
0001	1000	1
0001	0010	0
0010	0001	1
0010	0100	0
0011	1001	1
0011	0110	0
0100	0010	1
0100	1000	0
0110	0011	1
0110	1100	0
0111	1011	1
0111	1110	0
1000	0100	1
1000	0001	0
1001	1100	1
1001	0011	0
1011	1101	1
1011	0111	0
1100	0110	1
1100	1001	0
1101	1110	1
1101	1011	0
1110	0111	1
1110	1101	0

Разделим выборку на тестовую и тренировочную в соотношении 1/3. Зелёным отмечены тестовые значения.

Создание и обучение нейронной сети

Параметры нейронной сети остаются такие же, как в 5 практической работе. Меняется только архитектура. Количество весов стало больше, поэтому они обрабатываются в циклах, а не в ручную как в прошлой практической работе.



Код программы на Python, реализующий данную нейросеть.

```
import numpy as np
import matplotlib.pyplot as plt
  def
sigmoid(x):
    # Функция активации sigmoid:: f(x) = 1 / (1 + e^(-x))
return 1 / (1 + np.exp(-x))
  def
deriv_sigmoid(x):
    # Производная от sigmoid: f'(x) = f(x) * (1 - f(x))
fx = sigmoid(x)     return fx * (1 - fx)
  def mse_loss(y_true,
y_pred):
    # y_true и y_pred являются массивами numpy с одинаковой длиной
```

```
    return ((y_true - y_pred) ** 2).mean()
  class
SmartNeuralNetwork:
    """
    Нейронная сеть, у которой:
- n входов
- один скрытый слой с двумя нейронами с n нейронами
- слой вывода с одним нейроном
    """      def
__init__(self, n):
self.n = n          #
Веса
        self.weights = [np.random.normal() for i in range(n * n
+ n)]
        # Смещения
        self.offsets = [np.random.normal() for i in range(n
+ 1)]
        # Нейроны
self.h = [0] * n
     def feedforward(self,
x):          # x является
массивом numpy
         k = 0          for i in
range(self.n):              s = 0
for j in range(self.n):
s += self.weights[k] * x[j]
k += 1
            self.h[i] = sigmoid(s +
self.offsets[i])          s = 0          for i
in range(self.n):              s +=
self.weights[k] * self.h[i]             k += 1
        return sigmoid(s + self.offsets[-1])
     def  train(self,  X,  y,  learn_rate=0.1,  epochs=10000,  with_test=False,
X_test=None, y_test=None, plot=False):
        """
- X is a (m x n) numpy array, m = # of samples in the dataset.
- y is a numpy array with n elements.
        """
        train_loss = []
test_loss = []
         for epoch in
range(epochs):
            for x, y_true in zip(X, y):
                # --- Выполняем обратную связь (нам понадобятся эти значения в
дальнейшем)
                y_pred = self.feedforward(x)

                # ---Подсчет частных производных
d_L_d_ypred = -2 * (y_true - y_pred)
d_ypred_d_h = []

                #        Выходной
нейрнон                  for i in
range(self.n):
                    d_ypred_d_h.append(self.h[i] * deriv_sigmoid(y_pred))
                d_ypred_d_b =
deriv_sigmoid(y_pred)
```

```
                # Нейроны основного слоя                k = 0
dh_dw = []                  dh_db = []                for i in
range(self.n):                      for j in range(self.n):
dh_dw.append(x[j] * deriv_sigmoid(self.h[i]))
dh_db.append(deriv_sigmoid(self.h[i]))
                # --- Обновляем вес и смещения                  for i in
range(self.n):                      for j in range(self.n):
self.weights[i * self.n + j] -= learn_rate * d_L_d_ypred * dh_dw[i * self.n + j] *
d_ypred_d_h[                                i]
                    self.offsets[i] -= learn_rate * d_L_d_ypred * d_ypred_d_h[i] *
dh_db[i]
                for i in range(self.n):                      self.weights[self.n *
self.n + i] -= learn_rate * d_L_d_ypred * d_ypred_d_h[i]
                self.offsets[-1] -= learn_rate * d_L_d_ypred * d_ypred_d_b
            # --- Подсчитываем общую потерю в конце каждой фазы
y_preds = np.apply_along_axis(self.feedforward, 1, X)
train_loss.append(mse_loss(y, y_preds))             if with_test:
y_preds2 = np.apply_along_axis(self.feedforward, 1, X_test)
                test_loss.append(mse_loss(y_test, y_preds2))             if
epoch % 1000 == 0:                  print("Epoch %d loss: %.3f" % (epoch,
mse_loss(y, y_preds)))          if plot:             plt.plot(train_loss,
label='MSE тренировочных данных')             plt.plot(test_loss, label='MSE
тестовой выборки')              plt.show()



# Определение набора данных
X_train = np.array([
    [0, 0, 0, 1, 1, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 1],
    [0, 0, 1, 0, 0, 1, 0, 0],
    [0, 0, 1, 1, 0, 1, 1, 0],
    [0, 1, 0, 0, 0, 0, 1, 0],
    [0, 1, 0, 0, 1, 0, 0, 0],
    [0, 1, 1, 0, 1, 1, 0, 0],
    [0, 1, 1, 1, 1, 0, 1, 1],
    [0, 1, 1, 1, 1, 1, 1, 0],
    [1, 0, 0, 0, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 1, 0, 0, 1, 1],
    [1, 0, 1, 1, 1, 1, 0, 1],
    [1, 0, 1, 1, 0, 1, 1, 1],
    [1, 1, 0, 0, 0, 1, 1, 0],
    [1, 1, 0, 1, 1, 1, 1, 0],
    [1, 1, 0, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 0, 1, 1, 1]
])
y_train = np.array([1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1])
```

```
X_test = np.array([
    [0, 0, 0, 1, 0, 0, 1, 0],
    [0, 0, 1, 1, 1, 0, 0, 1],
    [0, 1, 1, 0, 0, 0, 1, 1],
    [1, 0, 0, 1, 1, 1, 0, 0],
    [1, 1, 0, 0, 1, 0, 0, 1],
    [1, 1, 1, 0, 1, 1, 0, 1]])

y_test = np.array([0, 1, 1, 1, 0, 0])

# Тренируем нейронную сеть network
= SmartNeuralNetwork(8)
network.train(X_train, y_train, with_test=True, X_test=X_test, y_test=y_test,
plot=True)

print('0 0 0 1 -> 0 0 1 0')
print('Предсказание:', network.feedforward(np.array([0, 0, 0, 1, 0, 0, 1, 0])))

print('0 1 1 0 -> 0 0 1 1')
print('Предсказание:', network.feedforward(np.array([0, 1, 1, 0, 0, 0, 1, 1])))
```

Результат работы:

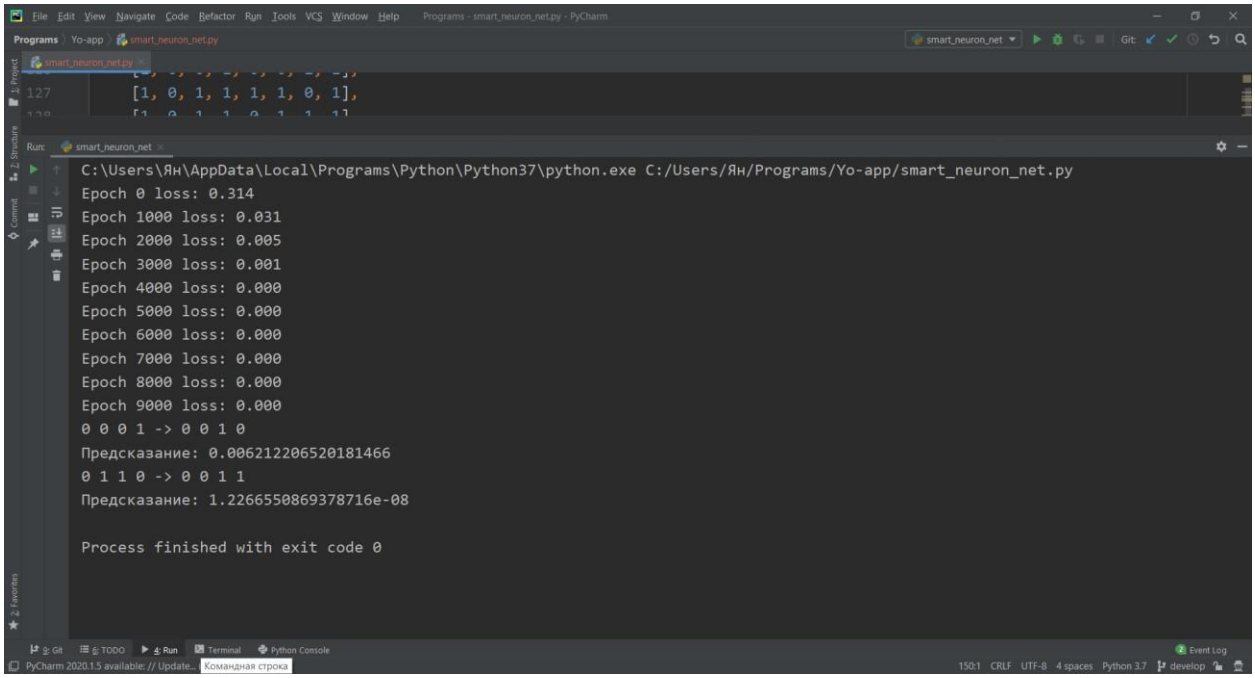
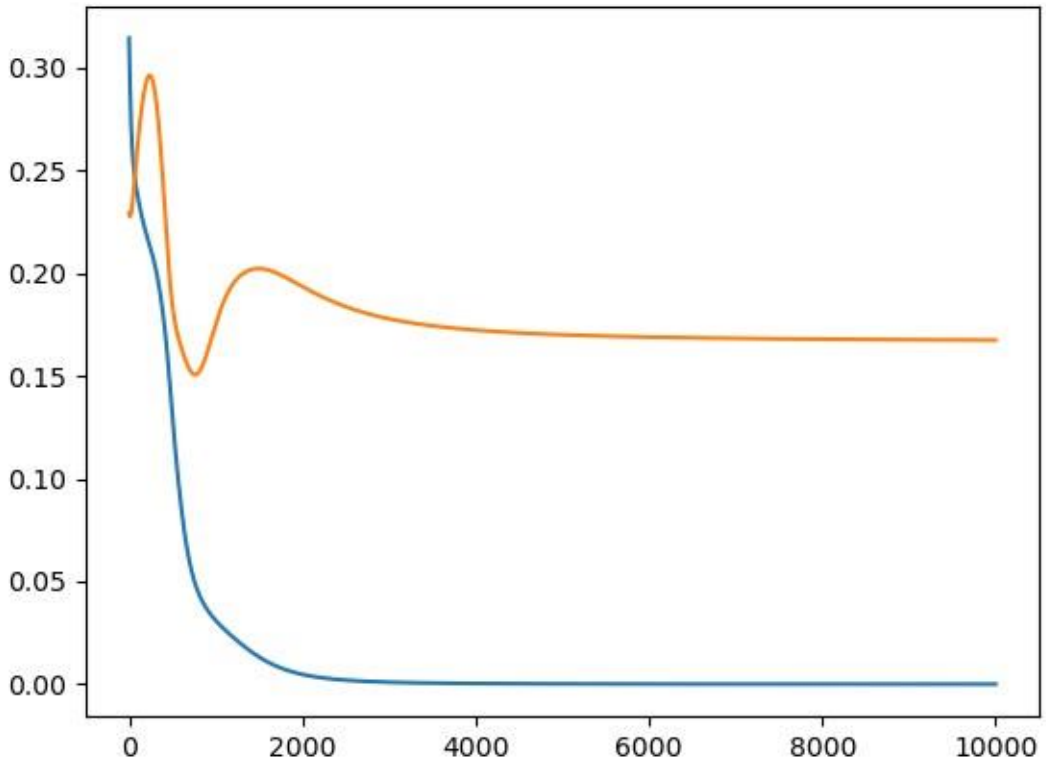


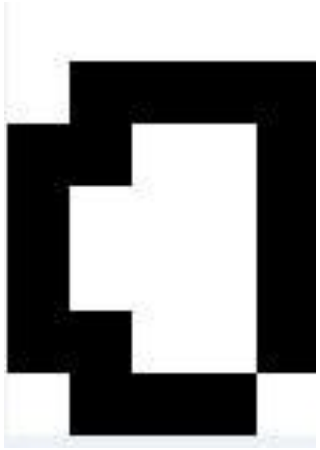
График функции потерь на тестовой (оранжевый) и на тренировочной (синий) выборке.

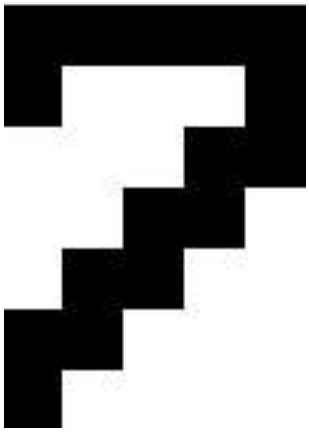


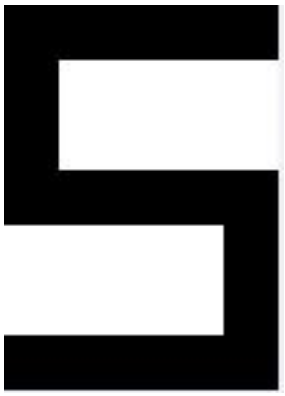
Из данной практической работы можно сделать вывод о том, как важно отслеживать переобучение нейронной сети для того, чтобы эффективность предсказаний не снижалась.

Вариант 1.

Для решения задачи распознавания образов с помощью нейронной сети я создал выборку из изображений цифр. Например:







Всего я создал 50 таких картинок. 10 из них для тестов, а 40 для тренировки сети. Так как на выходе должен быть 4-ёх битный двоичный код, я создал 4 нейронные сети, подобные сетям из прошлых практических работ. Каждая в отдельности предсказывает свой бит.

Листинг программы

```
import os from PIL import Image import numpy as np import matplotlib.pyplot as plt
 y = [] X = [] for file in os.listdir('digits'):
    y.append(int(file[file.index('.') + 1]))
img = Image.open('digits/' + file)
pixels = list(img.getdata())     x = []
for pix in pixels:         if pix[0] == 255:
            x.append(1)
else:
            x.append(0)
    X.append(x)
new_y = [[i // 8, i % 8 // 4, i // 2 % 2, i % 2] for i in y]
y1, y2, y3, y4 = [], [], [], [] for i in new_y:
    y1.append(i[0])
y2.append(i[1])     y3.append(i[2])
y4.append(i[3])
Y = [y1, y2, y3, y4]

def sigmoid(x):
    # Функция активации sigmoid:: f(x) = 1 / (1 + e^(-x))
return 1 / (1 + np.exp(-x))
    def deriv_sigmoid(x):
    # Производная от sigmoid: f'(x) = f(x) * (1 - f(x))
fx = sigmoid(x)     return fx * (1 - fx)
    def mse_loss(y_true, y_pred):
```

```
    # y_true и y_pred являются массивами numpy с одинаковой
длиной
    return ((y_true - y_pred) ** 2).mean()
 class
SmartNeuralNetwork:
    """
    Нейронная сеть, у которой:
- n входов
- один скрытый слой с двумя нейронами с n нейронами
- слой вывода с одним нейроном
    """      def
__init__(self, n):
        self.n = n
# Веса
        self.weights = [np.random.normal() for i in range(n
* n + n)]

        # Смещения
        self.offsets = [np.random.normal() for i in range(n
+
1)]

        # Нейроны
self.h = [0] * n
     def feedforward(self,
x):
        # x является массивом numpy
         k = 0         for i in
range(self.n):
```

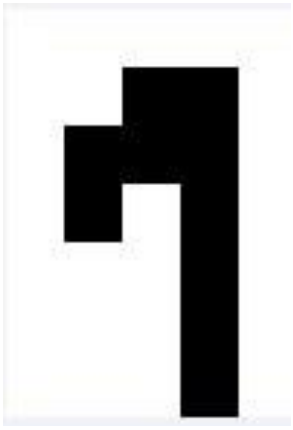
```
            s = 0              for j
in range(self.n):
                s += self.weights[k] * x[j]
k += 1
            self.h[i] = sigmoid(s + self.offsets[i])
s = 0          for i in range(self.n):
            s += self.weights[k] * self.h[i]
k += 1
        return sigmoid(s + self.offsets[-1])

    def train(self, X, y, learn_rate=0.1, epochs=150,
with_test=False, X_test=None, y_test=None, plot=False):
        """
- X is a (m x n) numpy array, m = # of samples in the
  dataset.
- y is a numpy array with n elements.
        """
train_loss = []
test_loss = []
         for epoch in range(epochs):
for x, y_true in zip(X, y):
                # --- Выполняем обратную связь (нам
понадобятся эти значения в дальнейшем)
y_pred = self.feedforward(x)

                # ---Подсчет частных производных
d_L_d_ypred = -2 * (y_true - y_pred)
d_ypred_d_h = []
```

```
                # Выходной нейрнон
for i in range(self.n):
                    d_ypred_d_h.append(self.h[i]            *
deriv_sigmoid(y_pred))
                d_ypred_d_b = deriv_sigmoid(y_pred)


                # Нейроны основного слоя
k = 0                 dh_dw = []
dh_db = []                 for i in
range(self.n):                      for j
in range(self.n):



                        dh_dw.append(x[j]                   *
deriv_sigmoid(self.h[i]))
dh_db.append(deriv_sigmoid(self.h[i]))


                # --- Обновляем вес и смещения
for i in range(self.n):
for j in range(self.n):
                        self.weights[i * self.n + j] -=
learn_rate * d_L_d_ypred * dh_dw[i * self.n + j] *
d_ypred_d_h[                                i]
                    self.offsets[i]  -=  learn_rate    *
d_L_d_ypred * d_ypred_d_h[i] * dh_db[i]
                 for        i        in        range(self.n):
self.weights[self.n * self.n + i] -= learn_rate * d_L_d_ypred
* d_ypred_d_h[i]
                self.offsets[-1] -= learn_rate * d_L_d_ypred
* d_ypred_d_b
```

```
            # --- Подсчитываем общую потерю в конце каждой
фазы
            y_preds = np.apply_along_axis(self.feedforward,
1, X)
            train_loss.append(mse_loss(y, y_preds))
if with_test:
                y_preds2                                    =
np.apply_along_axis(self.feedforward, 1, X_test)
test_loss.append(mse_loss(y_test, y_preds2))             if
epoch % 10 == 0:
                print("Epoch %d loss: %.3f" % (epoch,
mse_loss(y, y_preds)))
        if plot:
            plt.plot(train_loss, label='MSE тренировочных
данных')
            plt.plot(test_loss, label='MSE тестовой
выборки')             plt.show()

X_test = np.array(X[0:10])
X_train = np.array(X[10:])
Y_test = np.array([i[0:10] for i in Y]) Y_train
= np.array([i[10:] for i in Y])
 networks = [] for
i in range(4):
    print('Сеть №', i + 1)
net = SmartNeuralNetwork(35)
    net.train(X_train, Y_train[i], with_test=True, plot=True,
X_test=X_test,                               y_test=Y_test[i])
print(net.feedforward(X_test[0]))
```

Результат работы программы

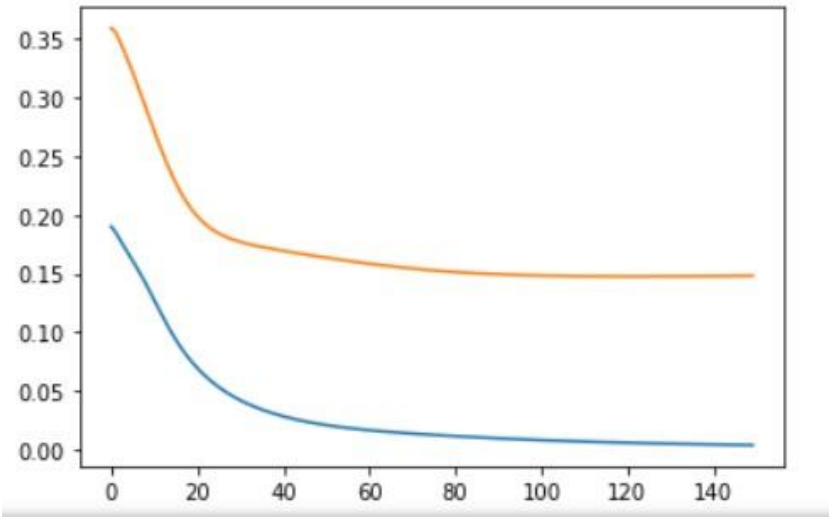
Модели требовалось предсказать следующее изображение для представления работы:

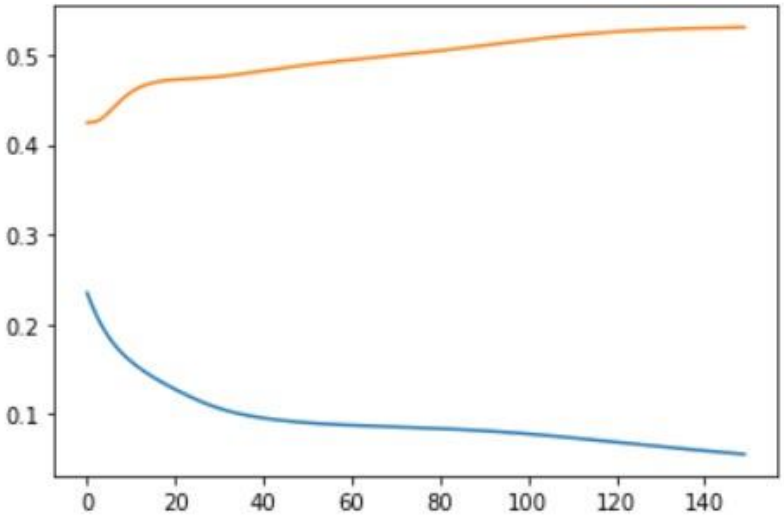


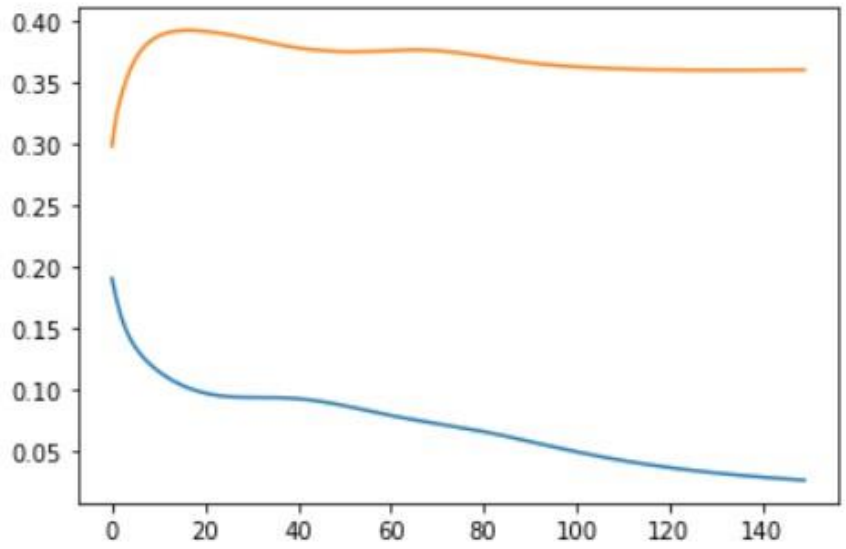
```
Сеть № 1
Epoch 0 loss: 0.190
Epoch 10 loss: 0.128
Epoch 20 loss: 0.070
Epoch 30 loss: 0.042
Epoch 40 loss: 0.028
Epoch 50 loss: 0.021
Epoch 60 loss: 0.017
Epoch 70 loss: 0.014
Epoch 80 loss: 0.011
Epoch 90 loss: 0.010
Epoch 100 loss: 0.008
Epoch 110 loss: 0.007
Epoch 120 loss: 0.006
Epoch 130 loss: 0.005
Epoch 140 loss: 0.004
0.027744254713099662

Сеть № 2
Epoch 0 loss: 0.235
Epoch 10 loss: 0.158
Epoch 20 loss: 0.127
Epoch 30 loss: 0.106
Epoch 40 loss: 0.096
Epoch 50 loss: 0.090
Epoch 60 loss: 0.088
Epoch 70 loss: 0.086
Epoch 80 loss: 0.084
Epoch 90 loss: 0.082
Epoch 100 loss: 0.078
Epoch 110 loss: 0.074
Epoch 120 loss: 0.069
Epoch 130 loss: 0.064
Epoch 140 loss: 0.059
0.2385395411525337

Сеть № 3
Epoch 0 loss: 0.191
```

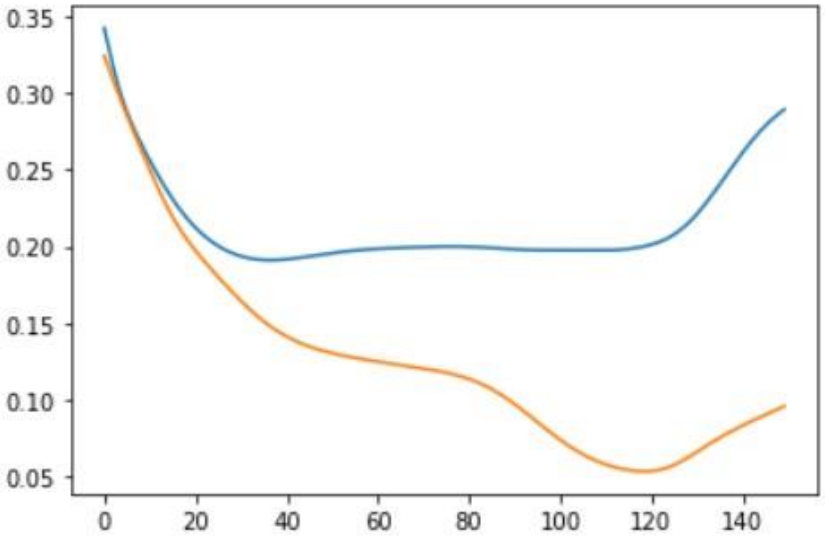
```
Epoch 10 loss: 0.115
Epoch 20 loss: 0.097
Epoch 30 loss: 0.094
Epoch 40 loss: 0.093
Epoch 50 loss: 0.087
Epoch 60 loss: 0.079
Epoch 70 loss: 0.072
Epoch 80 loss: 0.066
Epoch 90 loss: 0.058
Epoch 100 loss: 0.049
Epoch 110 loss: 0.042
Epoch 120 loss: 0.037
Epoch 130 loss: 0.032
Epoch 140 loss: 0.029
0.013404214921159199
Сеть № 4
Epoch 0 loss: 0.343
Epoch 10 loss: 0.256
Epoch 20 loss: 0.212
Epoch 30 loss: 0.194
Epoch 40 loss: 0.192
Epoch 50 loss: 0.196
Epoch 60 loss: 0.198
Epoch 70 loss: 0.200
Epoch 80 loss: 0.200
Epoch 90 loss: 0.198
Epoch 100 loss: 0.198
Epoch 110 loss: 0.197
Epoch 120 loss: 0.201
Epoch 130 loss: 0.221
Epoch 140 loss: 0.262
0.9231473737585756
```

На выходе должно быть 0 0 0 1. Модель предсказала 0.028 0.239 0.013 0.923. Из графиков, отражающих результат работы модели, мы видим, что модель с большим количеством входных данных (а следовательно и весов в нейронной сети) быстрее переобучается.


0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
140


0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
120
140





Дополнительные тесты также показали, что модель, которая классифицирует не каждый бит, а 10 классов цифр показывает более высокую точность предсказания. Также точность повысила бы кросс-энтропия в качестве функции потерь.

Задача искусственного носа.

Параметры нейронной сети для данной задачи:

Скрытых слоёв: 3

Функция активации сети: Softmax

Функция потерь: Кросс-энтропия

Метод обучения (оптимизатор весов): стохастический градиентный спуск

```
X_train = [[1, 0.05, 0.1, 0.3, 0.07, 0.08, 0.2, 0.05, 0.2, 0.6, 0.8],
           [0.8, 0.4, 0.7, 0.6, 0.1, 0.5, 1, 0.75, 0.5, 0.7, 0.8],
           [0.9, 0.2, 0.4, 0.5, 0.1, 0.7, 0.6, 0.5, 0.5, 0.7, 0.8],
           [0.85, 0.7, 0.8, 0.65, 0.1, 0.4, 1, 0.7, 0.4, 0.6, 0.7],
           [0.9, 0.3, 0.3, 0.4, 0.04, 0.1, 0.5, 0.3, 0.2, 0.7, 0.8],
[0.95, 0.18, 0.21, 0.3, 0.05, 0.1, 0.3, 0.2, 0.2, 0.5, 0.7]] y_train =
[1, 2, 3, 4, 5, 6]


import matplotlib.pyplot as plt import
numpy as np

import torch
```

```
X_train_tensor = torch.FloatTensor(X_train)
y_train_tensor = torch.LongTensor(y_train.astype(np.int64))
```

```
length = y_train_tensor.shape[0]
num_classes = 6  # количество классов, в нашем случае 6 видов веществ
# закодированные OneHot-ом метки классов
y_onehot = torch.FloatTensor(length, num_classes)

y_onehot.zero_()
y_onehot.scatter_(1, y_train_tensor.view(-1, 1), 1)



# N - размер батча (batch_size, нужно для метода оптимизации)
# D_in - размерность входа (количество признаков у объекта)
# H - размерность скрытых слоёв;
# D_out - размерность выходного слоя (суть - количество классов) D_in,
H, D_out = 11, 66, 6

# определим нейросеть:
net = torch.nn.Sequential(
torch.nn.Linear(D_in, H),
torch.nn.ReLU(),     torch.nn.Linear(H,
D_out),     torch.nn.Softmax()
)  def generate_batches(X, y, batch_size=64):
for i in range(0, X.shape[0], batch_size):
        X_batch, y_batch = X[i:i+batch_size], y[i:i+batch_size]
```

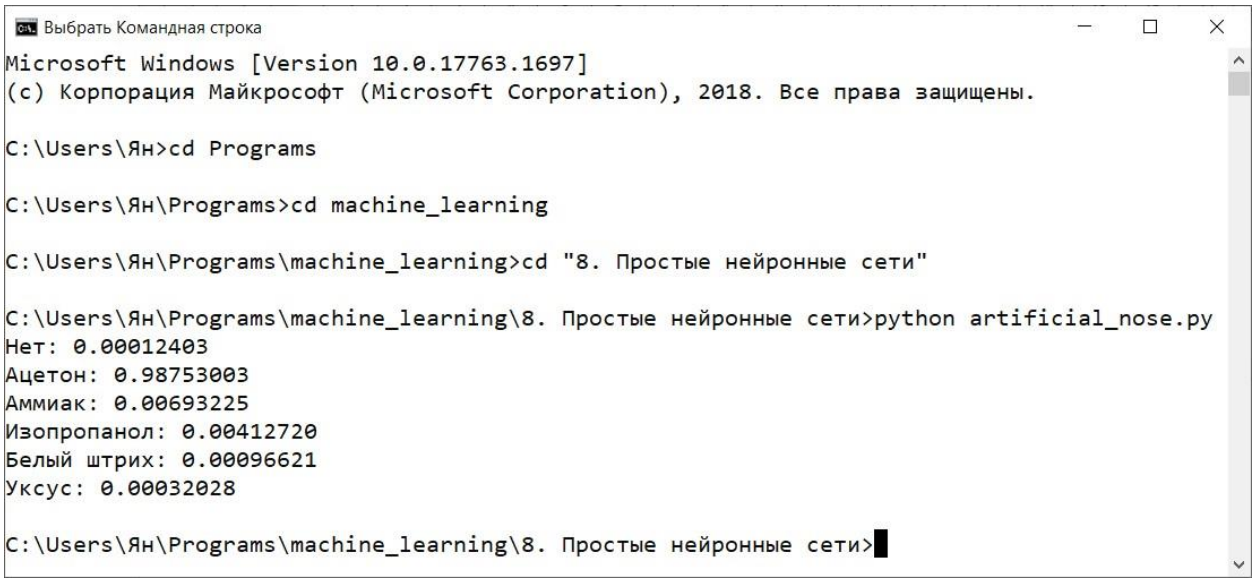
```
        yield X_batch, y_batch


BATCH_SIZE = 64
NUM_EPOCHS = 100

loss_fn = torch.nn.CrossEntropyLoss(size_average=False)

learning_rate = 1e-4
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)
 for epoch_num  in
range(NUM_EPOCHS):
    iter_num = 0
running_loss = 0.0
    for X_batch, y_batch in generate_batches(X_train_tensor, y_train_tenso
r, BATCH_SIZE):
        # forward (подсчёт ответа с текущими весами)
y_pred = net(X_batch)

        # вычисляем loss'ы
        loss = loss_fn(y_pred, y_batch)

        running_loss += loss.item()

        # выводем качество каждые 2000 батчей
                    if iter_num
% 100 == 99:
            print('[{}, {}] current loss: {}'.format(epoch_num, iter_num +
1, running_loss / 2000))
running_loss = 0.0

        # зануляем градиенты
optimizer.zero_grad()

        # backward (подсчёт новых градиентов)
loss.backward()

        # обновляем веса
optimizer.step()

        iter_num += 1
 class_correct = list(0. for i in
range(6)) class_total = list(0. for i in
range(6))

classes = ['Нет', 'Ацетон', 'Аммиак', 'Изопропанол', 'Белый штрих',
'Уксус']
 with
torch.no_grad():
    for X_batch, y_batch in generate_batches(X_train_tensor, y_train_tenso
r, BATCH_SIZE):
        y_pred = net(X_batch)
        _, predicted = torch.max(y_pred, 1)
c = (predicted == y_batch).squeeze()
for i in range(len(y_pred)):
            label = y_batch[i]
```

```
            class_correct[label] += c[i].item()
class_total[label] += 1

test = [0.75, 0.4, 0.7, 0.65, 0.15, 0.5, 1.1, 0.75, 0.5, 0.7, 0.8]
print(net.predict(test))
```

Замечу, что разделение выборки по батчам было необязательным. Тем более если у нас небольшой обучающий паттерн, но я решил прописать эту возможность для того, чтобы программа подходила для большинства случаев и для демонстрации.

Для теста я выбрал немного изменённые данные показывающие запах ацетона.

Результат работы



Годы	*Исходные данные*					
	y	***x1***	***x2***	***x3***	***x4***	***x5***
1990	**475,3**	**0,986**	**0,978**	**0,970**	**1,060**	**0,880**
1991	**413,5**	**0,876**	**0,858**	**0,870**	**1,082**	**0,840**
1992	**401,7**	**0,699**	**0,690**	**0,764**	**1,104**	**0,480**
1993	**400,9**	**0,605**	**0,619**	**0,685**	**1,126**	**0,475**
1994	**401,3**	**0,514**	**0,559**	**0,566**	**1,148**	**0,525**
1995	**402,5**	**0,483**	**0,492**	**0,534**	**1,170**	**0,575**
1996	**401,1**	**0,459**	**0,501**	**0,570**	**1,230**	**0,582**
1997	**404,4**	**0,464**	**0,506**	**0,593**	**1,304**	**0,539**
1998	**406,2**	**0,478**	**0,430**	**0,640**	**1,336**	**0,512**
1999	**412,2**	**0,507**	**0,587**	**0,695**	**1,370**	**0,519**
2000	**416,5**	**0,671**	**0,777**	**0,730**	**1,400**	**0,617**
2001	**425,8**	**0,801**	**1,109**	**0,758**	**1,430**	**0,624**
2002	**435,4**	**0,981**	**1,267**	**0,794**	**1,528**	**0,634**
2003	**445,4**	**1,117**	**1,425**	**0,830**	**1,626**	**0,656**
2004	**454,6**	**1,254**	**1,583**	**0,866**	**1,724**	**0,682**
2005	**466,1**	**1,411**	**1,741**	**0,904**	**1,824**	**0,729**
2006	**475,2**	**1,568**	**1,899**	**1,075**	**1,887**	**0,780**

Оценка по методу группового учёта аргументов.

Оценка моделей первой итерации

y	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10
425,8	458,744	431,606	465,333	443,229	479,427	427,535	423,120	437,460	418,963	417,922
435,4	472,568	438,597	484,126	451,819	496,493	432,817	432,174	445,877	422,195	420,054
445,4	486,441	445,616	503,027	460,409	513,559	438,100	440,047	455,163	426,254	423,675
454,6	501,318	453,376	524,171	469,161	530,653	443,650	448,252	464,738	430,590	427,791
466,1	516,196	472,725	543,760	488,626	547,232	465,642	458,885	475,833	436,516	434,530

475,2	353,395	334,619	295,275	333,636	332,247	315,900	469,804	487,218	452,109	441,428
2702,5	2788,662	2576,538	2815,693	2646,880	2899,612	2523,644	2672,282	2766,288	2586,627	2565,400

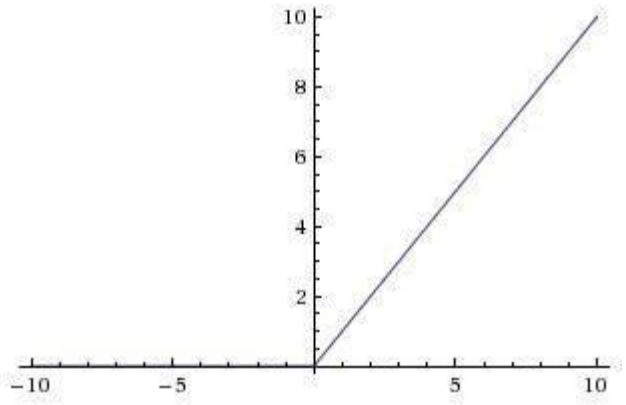
E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
1085,338453	33,709164	1562,833944	303,764013	2875,897263	3,008946	7,180817	135,951072	46,749966	62,070147
1381,443068	10,218355	2374,234383	269,585502	3732,395202	6,669839	10,409236	109,766608	174,382189	235,499655
1684,383886	0,046487	3320,923130	225,278820	4645,686216	53,287632	28,658371	95,309313	366,553634	471,994055
2182,607392	1,499195	4840,135833	212,012258	5784,065743	119,905105	40,298070	102,776122	576,473757	718,712474
2509,561284	43,895521	6031,119610	507,434555	6582,379006	0,209356	52,055911	94,731533	875,230195	996,652073
14836,499895	19763,020363	32372,880512	20040,312897	20435,588107	25376,519479	29,115878	144,424714	533,185311	1140,524937
23679,834	19852,389	50502,127	21558,388	44056,012	25559,600	913,149176	4068,904165	13426,629371	18796,451708
3946,638996	3308,731514	8417,021235	3593,064674	7342,668590	4259,933393	152,191529	678,150694	2237,771562	3132,741951

На втором этапе селекции удалось найти лучшую модель МГУА. Средний квадрат ошибки составил: 62,83.

Теперь посмотрим, как с данной задачей справится нейросеть.

Нейронная сеть с одним скрытым слоем, которую я с составил в предыдущих работах с данной задачей справилась плохо. Для увеличения эффективности прогнозирования я воспользовался библиотекой keras. С её помощью я разработал нейронную сеть с 3-мя слоями. Использовал функцию активации RELU вместо сигмоиды.

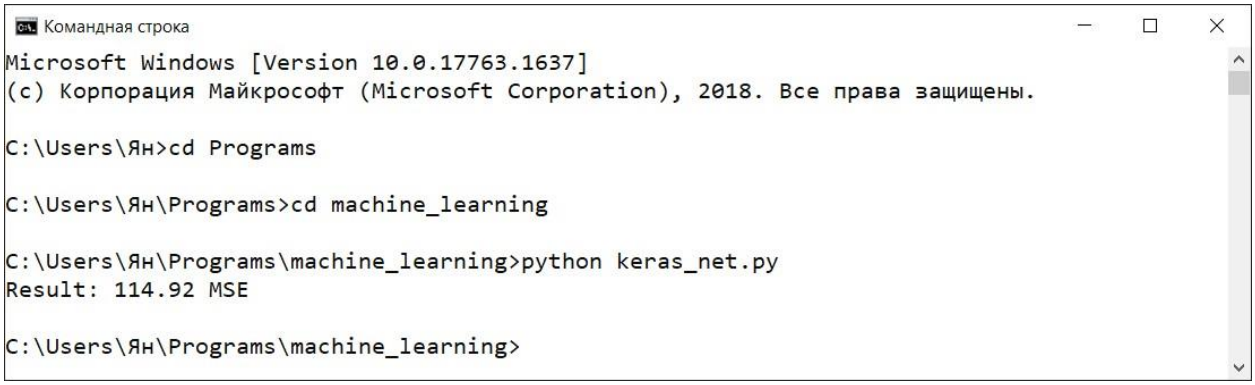
График функции Relu.



Листинг программы import numpy as np from keras.models import Sequential from keras.layers import Dense from keras.wrappers.scikit_learn import KerasRegressor from sklearn.model_selection import cross_val_score from sklearn.model_selection import KFold from sklearn.preprocessing import StandardScaler from sklearn.pipeline import Pipeline

```
seed = 1
Y = [475.3, 413.5, 401.7, 400.9, 401.3, 402.5, 401.1, 404.4, 406.2, 412.2, 416.5,
425.8, 435.4, 445.4, 454.6, 466.1, 475.2]
X = [[0.986, 0.978, 0.970, 1.060, 0.880],
    [0.876, 0.858, 0.870, 1.082, 0.840],
    [0.699, 0.690, 0.764, 1.104, 0.480],
    [0.605, 0.619, 0.685, 1.126, 0.475],
    [0.514, 0.559, 0.566, 1.148, 0.525],
    [0.483, 0.492, 0.534, 1.170, 0.575],
    [0.459, 0.501, 0.570, 1.230, 0.582],
    [0.464, 0.506, 0.593, 1.304, 0.539],
    [0.478, 0.430, 0.640, 1.336, 0.512],
    [0.507, 0.587, 0.695, 1.370, 0.519],
    [0.671, 0.777, 0.730, 1.400, 0.617],
    [0.801, 1.109, 0.758, 1.430, 0.624],
    [0.981, 1.267, 0.794, 1.528, 0.634],
    [1.117, 1.425, 0.830, 1.626, 0.656],
    [1.254, 1.583, 0.866, 1.724, 0.682],
    [1.411, 1.741, 0.904, 1.824, 0.729],
    [1.568, 1.899, 1.075, 1.887, 0.780]]
Y = np.array(Y) X
= np.array(X) def
larger_model():
```

```
# create model model = Sequential() model.add(Dense(5, input_dim=5, kernel_initializer='normal', activation='relu')) model.add(Dense(3, kernel_initializer='normal', activation='relu')) model.add(Dense(1, kernel_initializer='normal'))
        # Compile model
 model.compile(loss='mean_squared_error', optimizer='adam')  return model

np.random.seed(seed) estimators = [] estimators.append(('standardize', StandardScaler())) estimators.append(('mlp', KerasRegressor(build_fn=larger_model, epochs=50, batch_size=2, verbose=0))) pipeline = Pipeline(estimators) kfold = KFold(n_splits=2) results = cross_val_score(pipeline, X, Y, cv=kfold) print("Result: %.2f MSE" % (results.mean()))
```



Вывод:

МГУА сильный алгоритм и в некоторых случаях очень хорошо подходит для прогнозирования данных и регрессионного анализа данных. Однако нейронные сети более универсальный инструмент, который подходит для решения задач разного класса.