МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет информационных технологий и программирования

ПРИКЛАДНАЯ МАТЕМАТИКА

ИНДИВИДУАЛЬНЫЙ ПРОЕКТ

Моделирование искусственных нейронных сетей

Выполнила студент группы М3306:
Шакирова Владислава Эдуардовна

Проверила:
Москаленко Мария Александровна

Санкт-Петербург
2019

Задача 1

$$f(x,y) = xy \sim y = \overline{xy} * \bar{y} + xy * y = (\bar{x} + \bar{y}) * \bar{y} + xy = \overline{x + y} + \bar{y} + xy$$

x	*y*	*f*
0	0	1
0	1	0
1	0	1
1	1	1

Таблица 1. Данные для 1 задачи

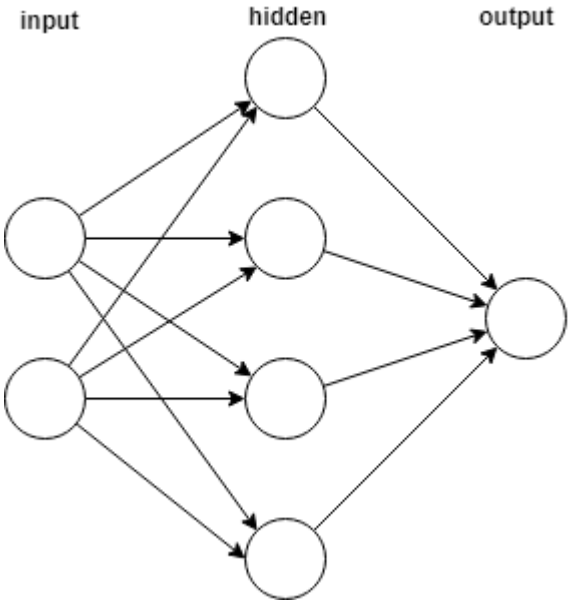

Рисунок 1. Архитектура 1 сети

Задача 2

$$f(x,y,z) = (x \to \bar{y}) * (\bar{x} + z) = (\bar{x} + \bar{y}) * (\bar{x} + z)$$

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Таблица 2. Данные для 2 задачи

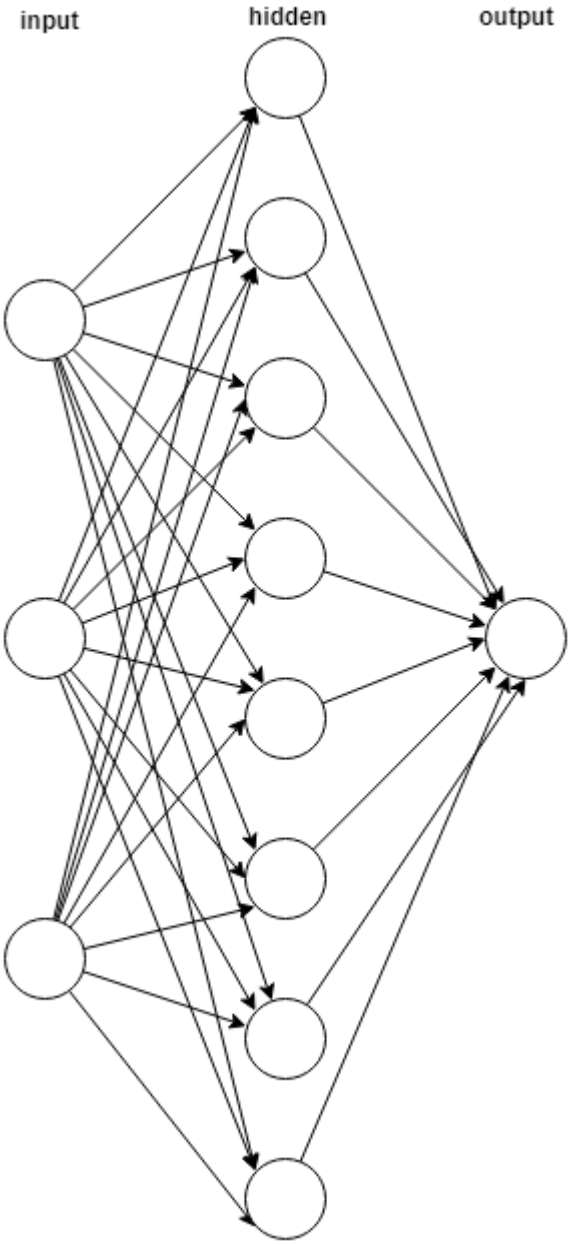

Рисунок 2. Архитектура 2 сети

Ход решения

Генерация данных

Данные состоят из строк таблиц 1 и 2, перемешанных в случайном порядке.

```
def generate_data_for_first_func():
	x = [0, 0, 1, 1]
	y = [0, 1, 0, 1]

	for i in range(5):
		x.extend(x)
		y.extend(y)

	output = [int(not(x[i] or y[i]) or not(y[i]) or (x[i] and y[i])) for i in range(len(x))]

	df = pd.DataFrame({'x':x, 'y':y, 'F':output})
	np.random.shuffle(df.values)
	df.to_csv('../data/data1.csv', index=False)

def generate_data_for_second_func():
	x = [0, 0, 0, 0, 1, 1, 1, 1]
	y = [0, 0, 1, 1, 0, 0, 1, 1]
	z = [0, 1, 0, 1, 0, 1, 0, 1]

	for i in range(5):
		x.extend(x)
		y.extend(y)
		z.extend(z)

	output = [int((not x[i] or not y[i]) and (not x[i] or z[i])) for i in range(len(x))]

	df = pd.DataFrame({'x':x, 'y':y, 'z':z, 'F':output})
	np.random.shuffle(df.values)

	    df.to_csv('../data/data2.csv', index=False)
```

Функция активации

Так как обе задачи являются задачами классификации с выбором 0, 1, то в качестве функции активации была выбрана сигмоида. Программная реализация также включает подсчет производной для этой функции.

```
def sigmoid(x, derivative=False):
    sigm = 1. / (1. + np.exp(-x))
    if derivative:
        return sigm * (1. - sigm)

    return sigm
```

Обучение сети

Проход вперед

Первоначально веса инициализированы случайными величинами, в среднем около 0. Далее реализуется проход вперед: начиная с 1 (и единственного в нашем случае) скрытого слоя на вход активационной функции каждого нейрона приходит величина, равная значениям на входных нейронах, перемноженным на соответствующие веса. И так далее, пока не дойдем до последнего нейрона на выходном слое.

Обратное распространение ошибки

Далее происходит обратный проход, который и реализует ***метод обратного распространения ошибки***.

Сначала вычисляется ошибка на выходном слое, как разница между текущим и ожидаемым:

```
l2_error = self.output - self.y
```

Далее рассчитывается дельта весов, как ошибка на выходном слое, умноженная на производную функции активации:

```
d_weights2 = l2_error * sigmoid((self.output), derivative=True)
```

Ошибка на нейронах скрытого слоя рассчитывается как величина, показывающая насколько сильно ошибка на выходном слое влияет на ошибку нейронов скрытого слоя:

```
l1_error = d_weights2.dot(self.weights2.T)
```

Также рассчитываем дельту для скрытого слоя:

```
d_weights1 = l1_error * sigmoid((self.layer1), derivative=True)
```

Вычисляем новые веса:

```
self.weights2 -= self.layer1.T.dot(d_weights2 * learning_rate)
self.weights1 -= self.input.T.dot(d_weights1 * learning_rate)
```

Производим проход вперед и обратное распространение определенное количество итераций, или пока не достигнем определенной точности вычислений.

Полный код для обучения сети:

```
def __init__(self, x, y, num_on_hidden=2):
        self.input = x
        self.y = y
        self.output = np.zeros(self.y.shape)
        self.weights1 = 2 * np.random.rand(self.input.shape[1], num_on_hidden) - 1
        self.weights2 = 2 * np.random.rand(num_on_hidden, 1) - 1

def feedforward(self, X):
        self.layer1 = sigmoid(np.dot(X, self.weights1))
        self.output = sigmoid(np.dot(self.layer1, self.weights2))

def backprop(self, learning_rate):
        # как сильно мы ошиблись относительно нужной величины?
        l2_error = self.output - self.y

        # в какую сторону нужно двигаться?
        # если мы были уверены в предсказании, то сильно менять его не надо
        d_weights2 =  l2_error * sigmoid((self.output), derivative=True)

        # как сильно значения l1 влияют на ошибки в l2?
```

```
            l1_error = d_weights2.dot(self.weights2.T)

            # в каком направлении нужно двигаться, чтобы прийти к l1?
            # если мы были уверены в предсказании, то сильно менять его не надо
            d_weights1 = l1_error * sigmoid((self.layer1), derivative=True)

            self.weights2 -= self.layer1.T.dot(d_weights2 * learning_rate)
            self.weights1 -= self.input.T.dot(d_weights1 * learning_rate)

      def train(self, num_epochs, learning_rate, display_loss):
            for i in range(num_epochs):
                  self.feedforward(self.input)
                  if display_loss:
                        print("Epoch " + str(i) + ': ' + '; MSE: ' +
str(mean_squared_error(self.y, self.output)))
                  self.backprop(learning_rate)


          return mean_squared_error(self.y, self.output)
```

Тестирование сети

Тестирование от обучения отличается тем, что там нет пересчета весов, используется только проход вперед

```
def test(self, X):
      self.feedforward(X)
      return self.output[:len(X)]
```

Результаты

Обе сети показывают точность в 100% на тестовой выборке, однако 2 сеть способна предсказывать со 100% точностью на текущих тестовых данных за меньшее количество итераций обучения:

First FFN
Number of neurons: 7
Count of epochs: 80
MSE at the last epoch: 0.01282
Learning rate: 0.1
Accuracy score on test data: 100.0%

Second FFN
Number of neurons: 12
Count of epochs: 50
MSE at the last epoch: 0.03635
Learning rate: 0.1
Accuracy score on test data: 100.0%

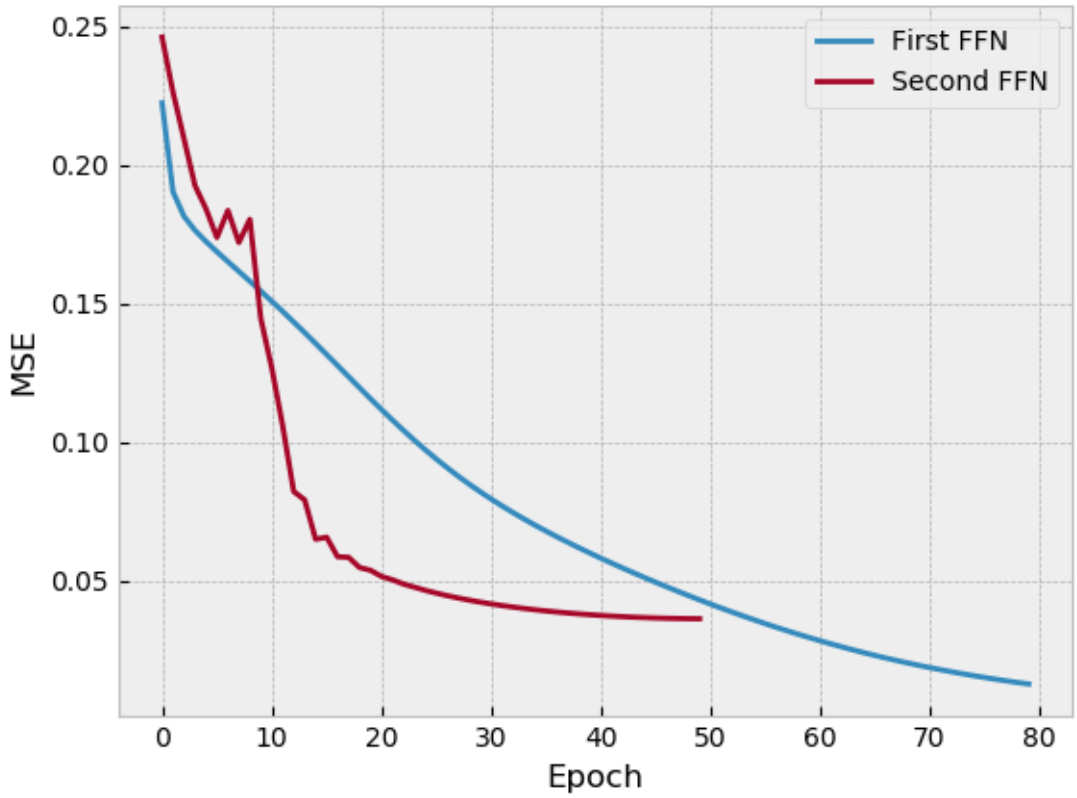

Рисунок 3. Зависимость MSE от количества эпох