Был выполнен 4 вариант.

Архитектура нейросети:

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(3,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Функция для симуляции работы нейросети с использованием NumPy:

```
def numpy_f(layers, input):
    res = np.zeros((input.shape[0]))
    for i in range(len(res)):
        output1 = np.maximum(np.dot(np.transpose(layers[0].get_weights()[0]),
input[i]) + layers[0].get_weights()[1], 0)
        output2 = np.maximum(np.dot(np.transpose(layers[1].get_weights()[0]),
output1) + layers[1].get_weights()[1], 0)
        res[i] = sigmoid(np.dot(np.transpose(layers[2].get_weights()[0]),
output2) + layers[2].get_weights()[1])
    return np.reshape(res, (len(res), 1))
```

Функция для симуляции работы нейросети, в которой все операции реализованы как поэлементные операции над тензорами:

```
def naive_f(layers, input):
    res = np.zeros((input.shape[0]))
    for i in range(len(res)):
        output1 =
naive_relu(naive_matrix_vector_dot(np.transpose(layers[0].get_weights()[0]),
input[i])
                                + layers[0].get_weights()[1])
        output2 =
naive_relu(naive_matrix_vector_dot(np.transpose(layers[1].get_weights()[0]),
output1)
                                + layers[1].get_weights()[1])
        res[i] =
sigmoid(naive_matrix_vector_dot(np.transpose(layers[2].get_weights()[0]),
output2)
                           + layers[2].get_weights()[1])
    return np.reshape(res, (len(res), 1))
```

В обоих случаях входные данные прогоняются через все слои по следующей формуле:

output = relu(dot(W, input) + b),

где input это выход с прошлого слоя (либо то, что подано на входной слой), W – веса, b – bias (смещение)

Сигмоид:

```
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

RELU и умножение матрицы на вектор, без использования NumPy:

```
def naive_relu(x):
    assert len(x.shape) == 1
    x = x.copy()
    for i in range(x.shape[0]):
        x[i] = max(x[i], 0)
    return x


def naive_matrix_vector_dot(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 1
    assert x.shape[1] == y.shape[0]
    z = np.zeros(x.shape[0])
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            z[i] += x[i, j] * y[j]
    return z
```

Результаты работы:

[[0.5]

[0.56118166]

[0.5026692]

[0.5586256]

[0.5336479]

[0.5856387]

[0.51959217]

[0.5934181]]

NumPy:

[[0.5]

[0.56118171]

[0.5026692]

[0.55862557]

[0.53364789]

[0.5856387]

[0.51959219]

[0.59341814]]

Naive:

[[0.5]
[0.56118171]
[0.5026692]
[0.55862557]
[0.53364789]
[0.5856387]
[0.51959219]
[0.59341814]]

Обученная сеть:

[[9.119718e-09]
[9.364511e-11]
[6.544597e-10]
[2.431016e-08]
[8.395347e-08]
[1.000000e+00]
[1.000000e+00]
[1.000000e+00]]

NumPy:

[[9.11972919e-09]
[9.36450967e-11]
[6.54462009e-10]
[2.43101905e-08]
[8.39535104e-08]
[9.99999971e-01]
[9.99999971e-01]
[9.99999999e-01]]

Naive:

[[9.11972919e-09]

[9.36450967e-11]

[6.54462009e-10]

[2.43101905e-08]

[8.39535104e-08]

[9.99999971e-01]

[9.99999971e-01]

[9.99999999e-01]]

Как видно, результаты совпадают с точностью до округленных нейросетью знаков после запятой.