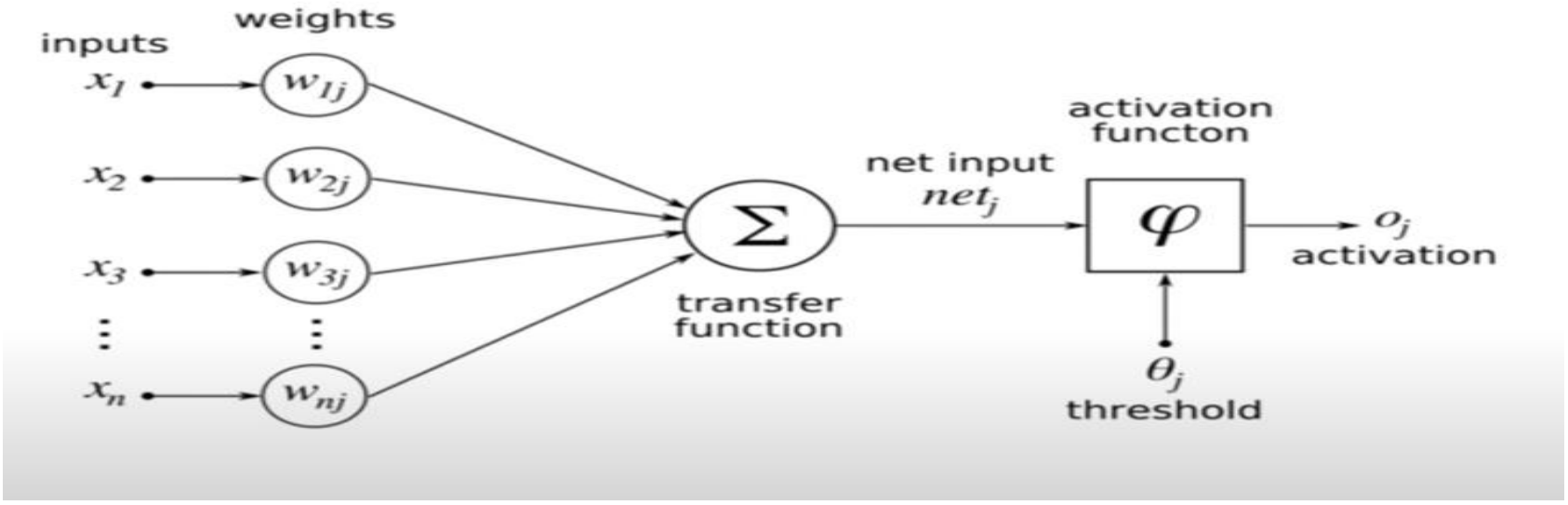
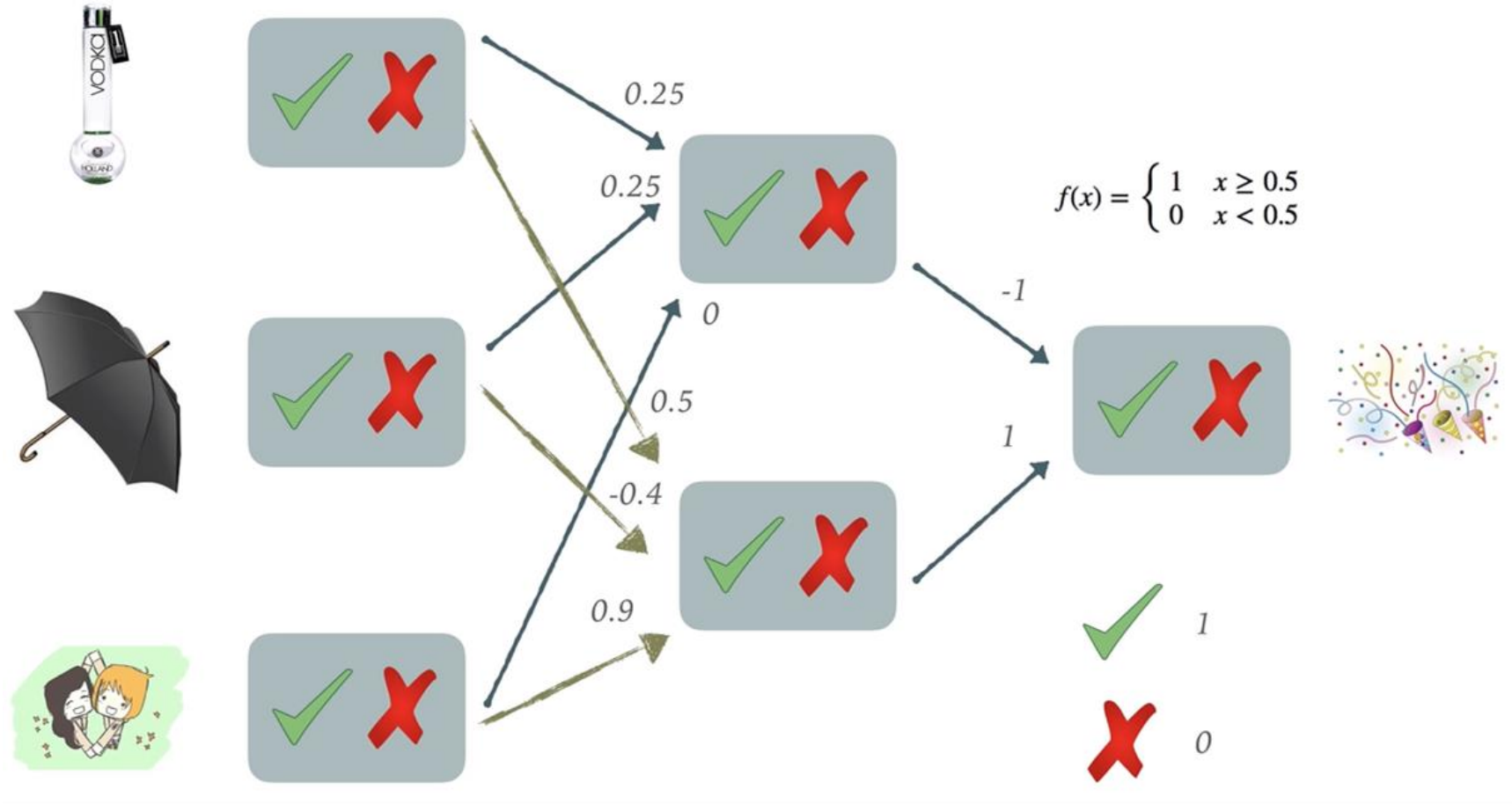
Семинар 17. Обучение нейросетей

- Обучение – тренировка методом проб и ошибок.
- Обучение – коррекция подобранных весов, характеризующих связи между нейронами.



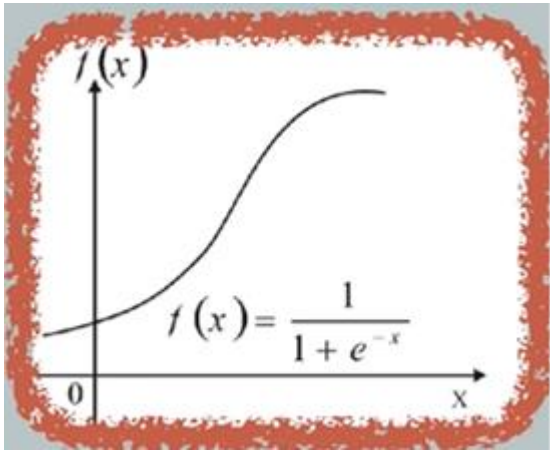
Пример с прошлого семинара



Реальная активационная функция

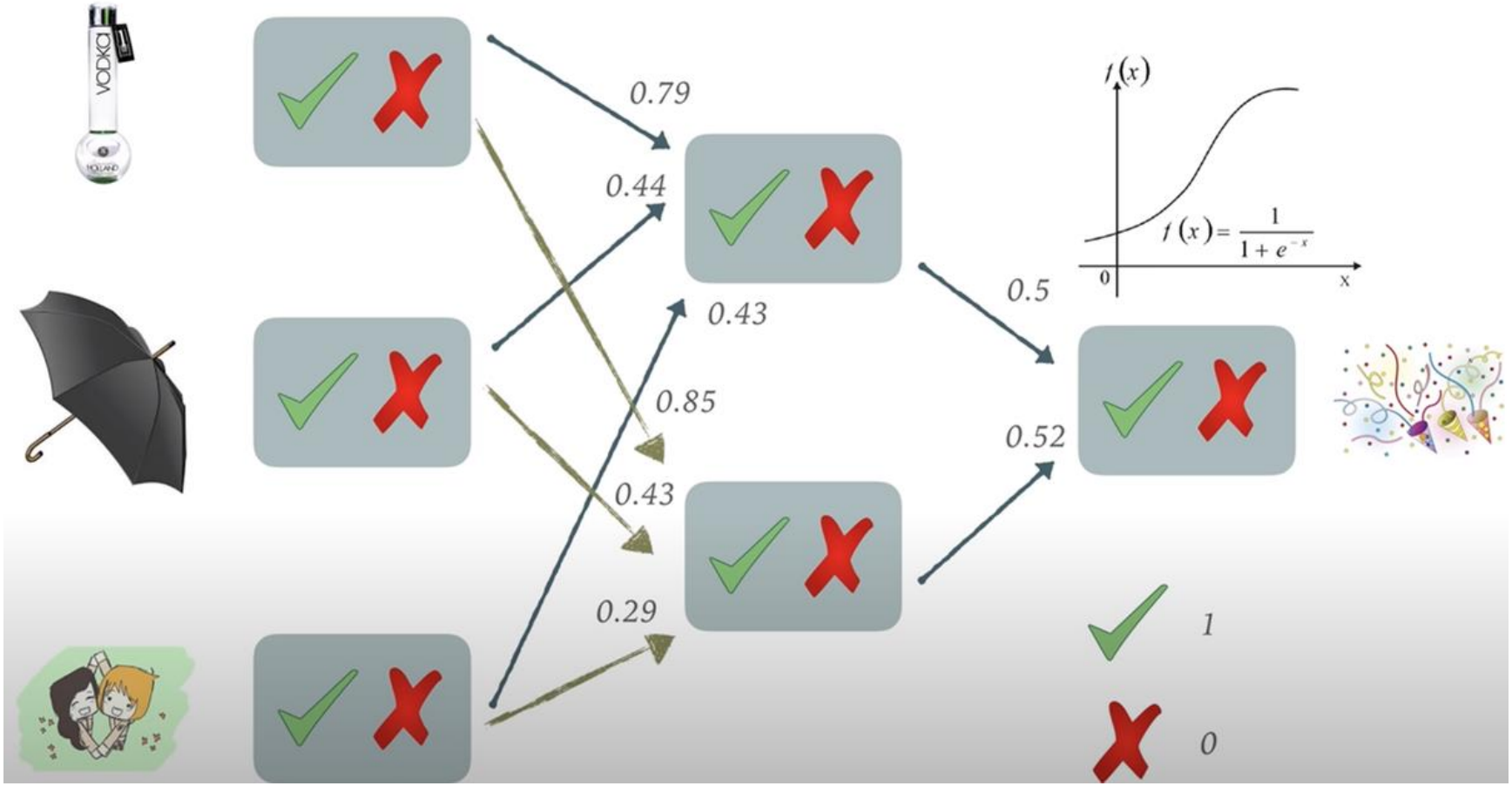
$$f(x) = \begin{cases} 1 & x \geq 0.5 \\ 0 & x < 0.5 \end{cases}$$

->

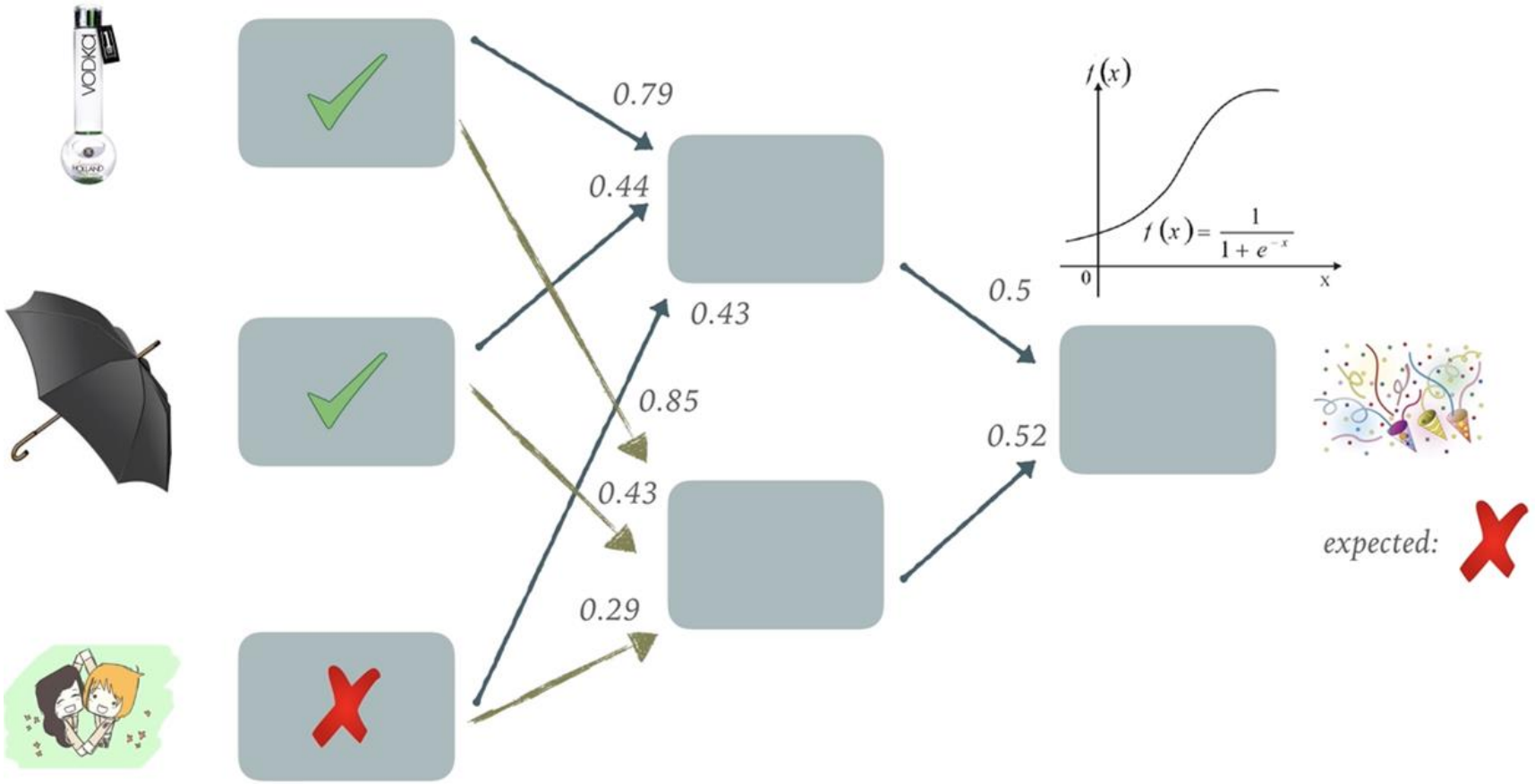


Плавный переход из состояния I в состояние j
(гладкая, а значит дифференцируема)

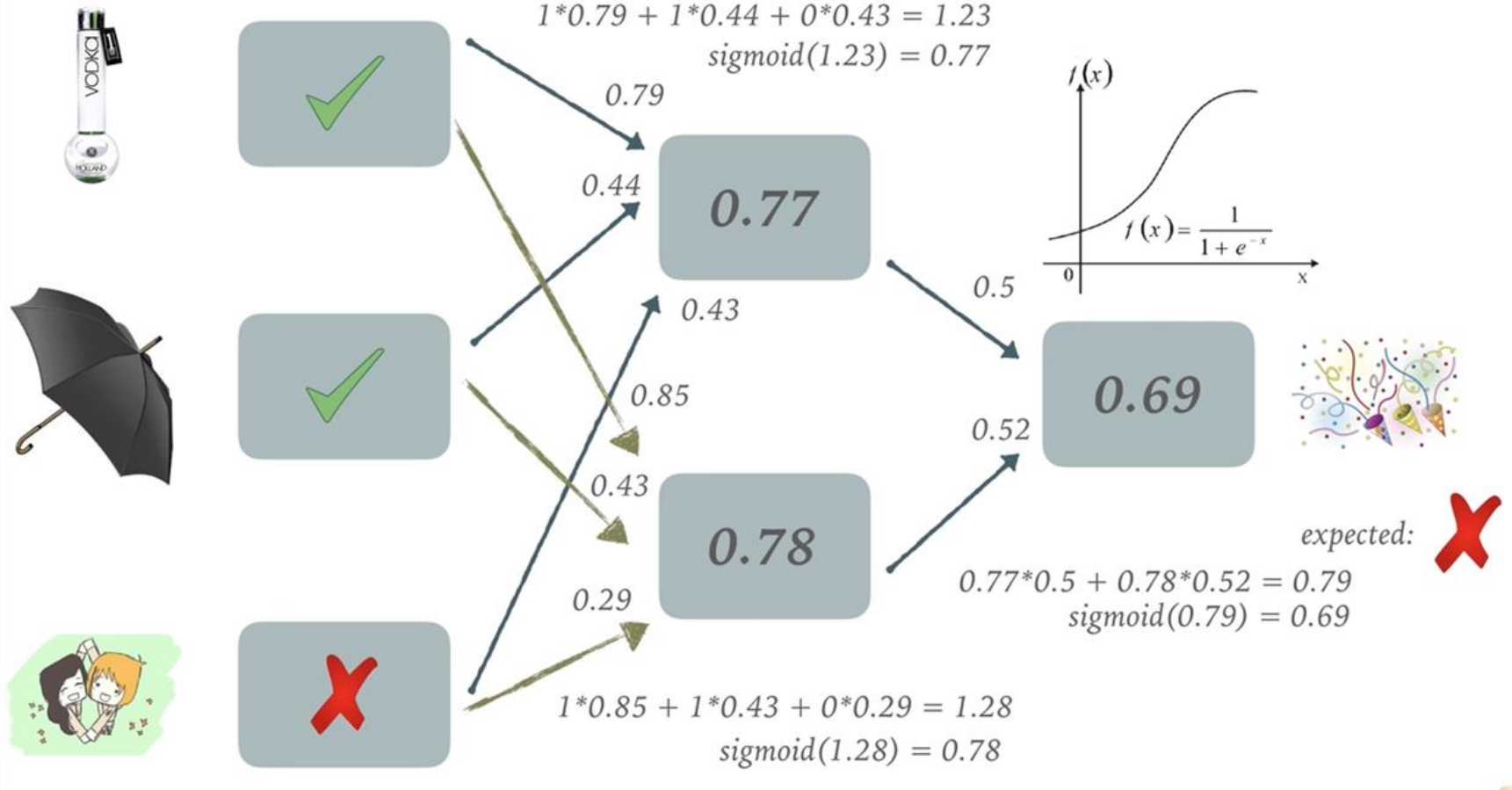
Подготовка. Шаг 1 Произвольные коэффициенты



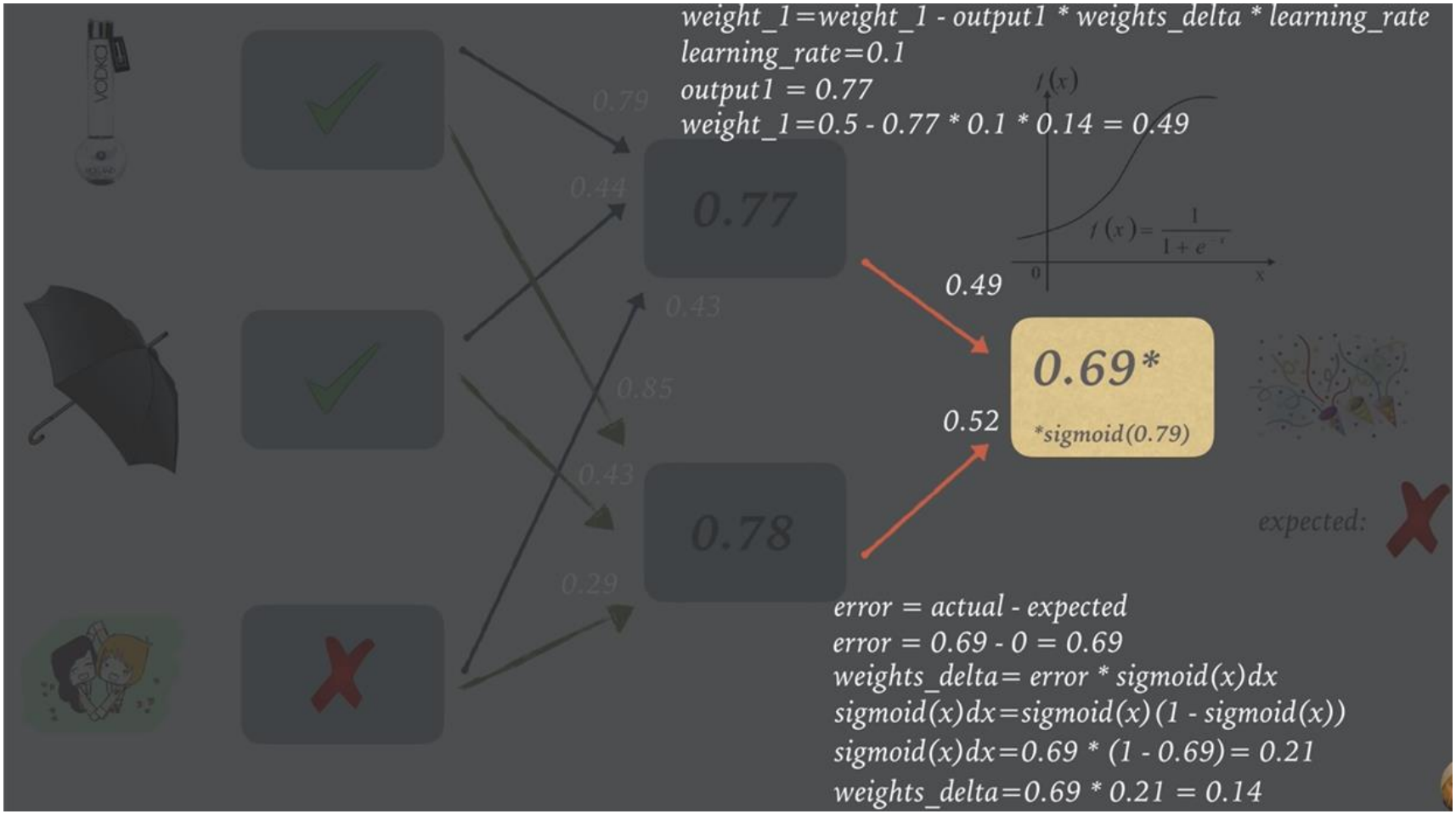
Подготовка. Шаг 2 Моделируем результат ожидания



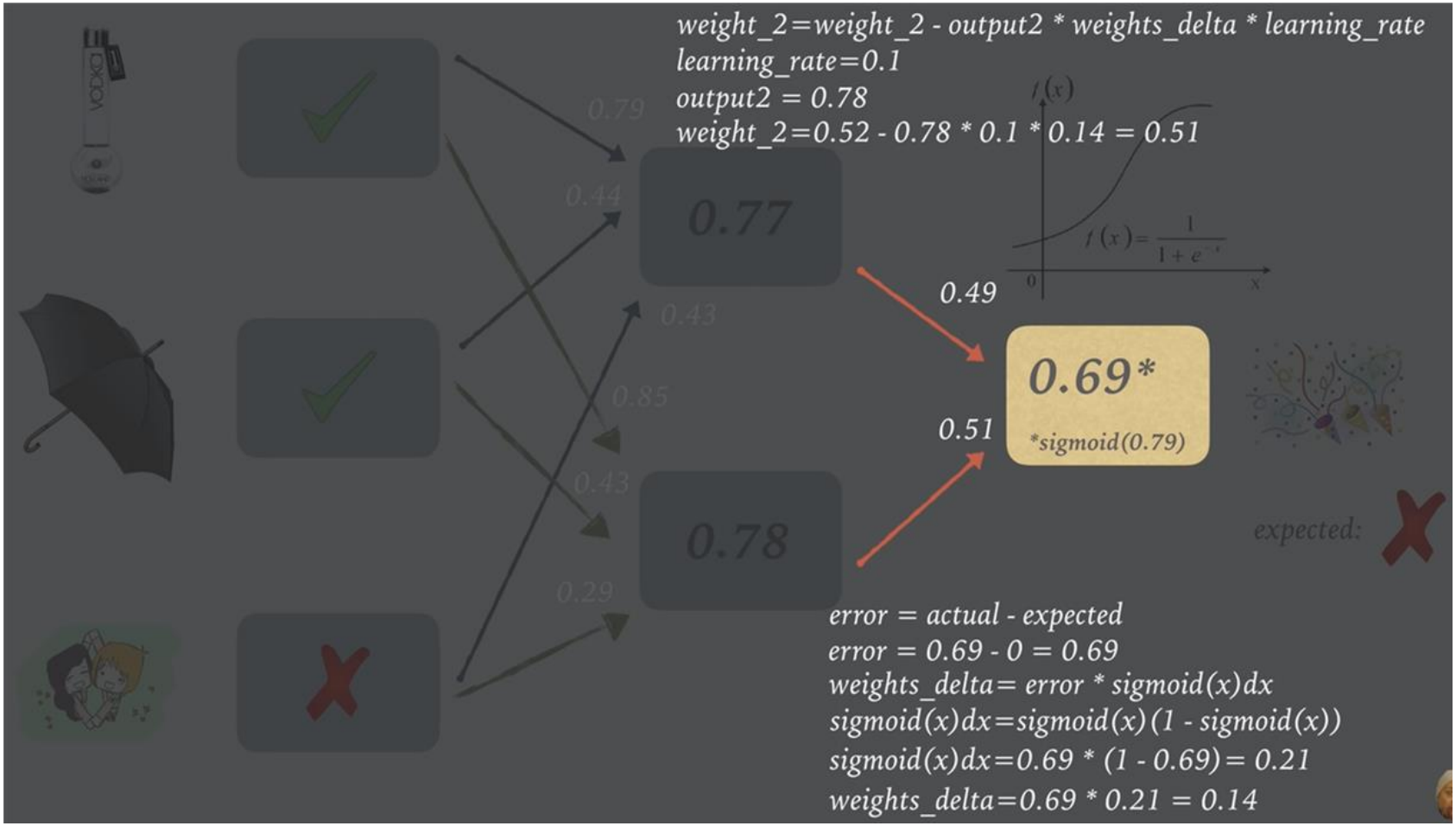
Подготовка. Шаг 2 Подсчет результата активационной функции по выбранным коэффициентам



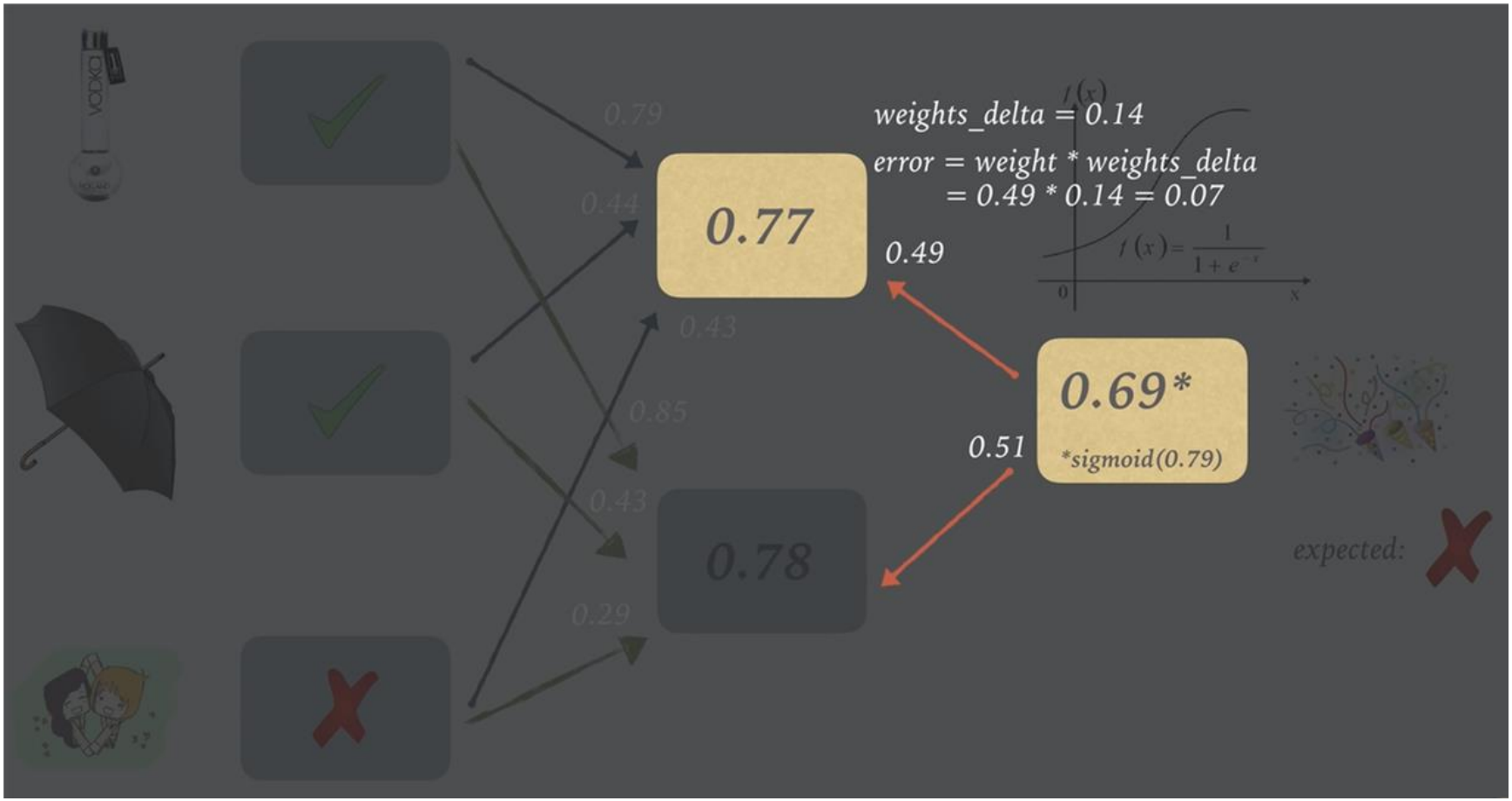
Обучение. Обратное распространение ошибки. Формулы



Обучение. Шаг 1 Формулы



Обучение. Шаг 2 Ход назад. Формулы2

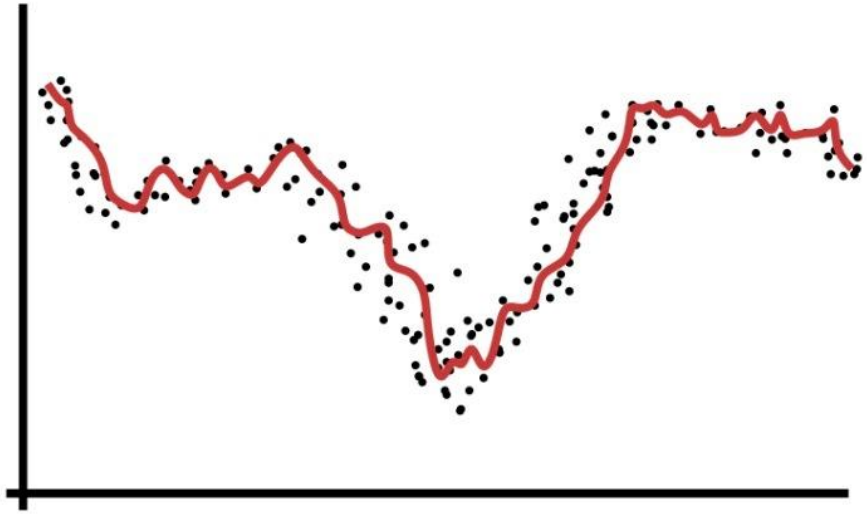


Обучение. Шаг 3

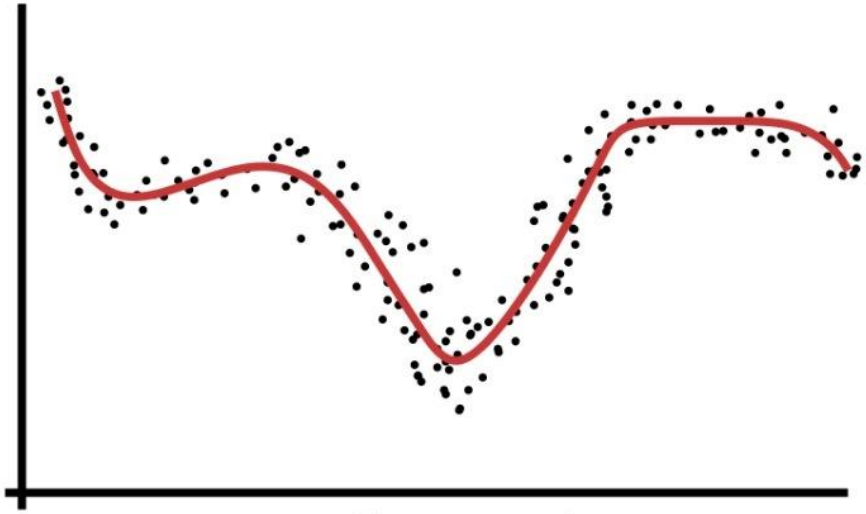
- Подборка эпох (итераций обучения)
- Подборка learning rate
- Сети нужно давать разные данные для успешного обучения

Пример переобучения НС

Переобученная НС



Обученная НС



Пример

- https://colab.research.google.com/drive/1jNwjk8_Zvz6vV9-WGmv4NYsnq7KjoMPt?usp=sharing

Пример обучения сети

In [1]:
```
import numpy as np
import sys
```

In [28]:
```
class PartyNN(object):

    def __init__(self, learning_rate=0.1):
        self.weights_0_1 = np.random.normal(0.0, 2 ** -0.5,(2, 3))
        self.weights_1_2 = np.random.normal(0.0, 1, (1, 2))
        self.sigmoid_mapper = np.vectorize(self.sigmoid)
        self.learning_rate = np.array([learning_rate])

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def predict(self, inputs):
        inputs_1 = np.dot(self.weights_0_1, inputs)
        outputs_1 = self.sigmoid_mapper(inputs_1)

        inputs_2 = np.dot(self.weights_1_2, outputs_1)
        outputs_2 = self.sigmoid_mapper(inputs_2)
        return outputs_2

    def train(self, inputs, expected_predict):
        inputs_1 = np.dot(self.weights_0_1, inputs)
        outputs_1 = self.sigmoid_mapper(inputs_1)

        inputs_2 = np.dot(self.weights_1_2, outputs_1)
        outputs_2 = self.sigmoid_mapper(inputs_2)
        actual_predict = outputs_2[0]

        error_layer_2 = np.array([actual_predict - expected_predict])
        gradient_layer_2 = actual_predict * (1 - actual_predict)
        weights_delta_layer_2 = error_layer_2 * gradient_layer_2
        self.weights_1_2 -= (np.dot(weights_delta_layer_2, outputs_1.reshape(1, len(outputs_1)))) * self.learning_rate

        error_layer_1 = weights_delta_layer_2 * self.weights_1_2
        gradient_layer_1 = outputs_1 * (1 - outputs_1)
        weights_delta_layer_1 = error_layer_1 * gradient_layer_1
        self.weights_0_1 -= np.dot(inputs.reshape(len(inputs),1), weights_delta_layer_1).T * self.learning_rate
```

Пример обучения сети

In [29]:
```
def MSE(y,Y):
    return np.mean((y-Y)**2)
```

In [30]:
```
train = [
    ([0,0,0],0),
    ([0,0,1],1),
    ([0,1,0],0),
    ([0,1,1],0),
    ([1,0,0],1),
    ([1,0,1],1),
    ([1,1,0],0),
    ([1,1,1],1),
]
```

In [31]:
```
epochs = 5000
learning_rate = 0.05

network = PartyNN(learning_rate=learning_rate)

#losses = {'train':[], 'validation':[]}
for e in range(epochs):
    inputs_ = []
    correct_predictions = []
    for input_stat, correct_predict in train:
        network.train(np.array(input_stat), correct_predict)
        inputs_.append(np.array(input_stat))
        correct_predictions.append(np.array(correct_predict))

    train_loss = MSE(network.predict(np.array(inputs_).T), np.array(correct_predictions))
    sys.stdout.write("\rProgress: {}, Training loss: {}".format(str(100 * e/float(epochs))[:4], str(train_loss)[:5]))
```

```
Progress: 99.9, Training loss: 0.003
```

Пример обучения сети

In [32]:
```
for input_stat, correct_predict in train:
    print("For input: {} the predictions is: {}, expected: {}".format(
        str(input_stat),
        str(network.predict(np.array(input_stat))> .5),
        str(correct_predict == 1)))
```

```
For input: [0, 0, 0] the predictions is: [False], expected: False
For input: [0, 0, 1] the predictions is: [ True], expected: True
For input: [0, 1, 0] the predictions is: [False], expected: False
For input: [0, 1, 1] the predictions is: [False], expected: False
For input: [1, 0, 0] the predictions is: [ True], expected: True
For input: [1, 0, 1] the predictions is: [ True], expected: True
For input: [1, 1, 0] the predictions is: [False], expected: False
For input: [1, 1, 1] the predictions is: [ True], expected: True
```

In [33]:
```
for input_stat, correct_predict in train:
    print("For input: {} the predictions is: {}, expected: {}".format(
        str(input_stat),
        str(network.predict(np.array(input_stat))),
        str(correct_predict == 1)))
```

```
For input: [0, 0, 0] the predictions is: [0.12632291], expected: False
For input: [0, 0, 1] the predictions is: [0.94819071], expected: True
For input: [0, 1, 0] the predictions is: [0.00086696], expected: False
For input: [0, 1, 1] the predictions is: [0.04049946], expected: False
For input: [1, 0, 0] the predictions is: [0.94775381], expected: True
For input: [1, 0, 1] the predictions is: [0.97485136], expected: True
For input: [1, 1, 0] the predictions is: [0.04129157], expected: False
For input: [1, 1, 1] the predictions is: [0.92509022], expected: True
```

In [34]:
```
network.weights_0_1
```

Out[34]:
```
array([[ 3.04360128, -3.39422547,  2.40166377],
       [-2.35639727,  2.83297422, -2.63419664]])
```

In [35]:
```
network.weights_1_2
```

Out[35]:
```
array([[ 3.72478343, -7.59252229]])
```

Задание

- Обучить свою нейронную сеть, состоящую из 4 входных нейронов и 3 нейронов скрытого слоя.