Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Прикладные интеллектуальные системы и экспертные системы»

Обучение нейронной сети

Студент Мастылина А.А.

Группа М-ИАП-22

Руководитель Кургасов В.В.

Липецк 2022 г.

Ход работы

Краткая теоретическая справка

Обучение нейронной сети — это процесс обучения нейронной сети выполнению задачи. Нейронные сети обучаются путем первичной обработки нескольких больших наборов размеченных или неразмеченных данных. На основе этих примеров сети могут более точно обрабатывать неизвестные входные данные.

Для того, чтобы нейронная сети была способна выполнить поставленную задачу, ее необходимо обучить (см. рис. 1). Различают алгоритмы обучения с учителем и без учителя.

Нейрон принимает несколько входов, выполняет над ними кое-какие математические операции, а потом выдает один выход. Нейрон с двумя входами:

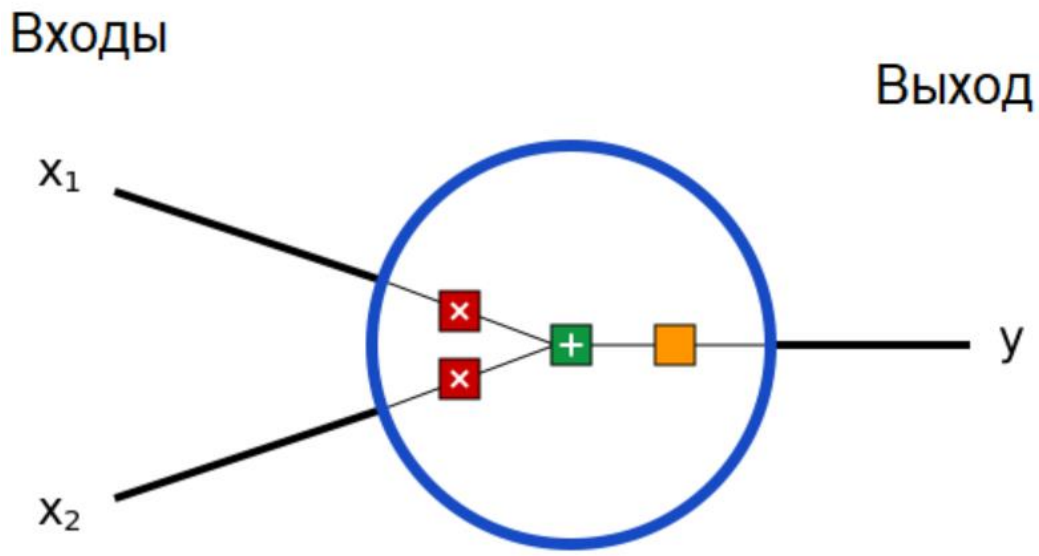

Рисунок 1 – Нейрон с двумя входами

Внутри нейрона происходят три операции. Сначала значения входов умножаются на веса:

$$x_1 \to x_1 * w_1,\ x_2 \to x_2 * w_2$$

Затем взвешенные входы складываются, и к ним прибавляется значение порога b:

$$x_1 * w_1 + x_2 * w_2 + b$$

Полученная сумма проходит через функцию активации:

$$= f(x_1 * w_1 + x_2 * w_2 + b)$$

Функция активации преобразует неограниченные значения входов в выход, имеющий ясную и предсказуемую форму. Одна из часто используемых функций активации – сигмоида.

Сигмоида выдает результаты в интервале (0, 1). Можно представить, что она «упаковывает» интервал от минус бесконечности до плюс бесконечности в (0, 1): большие отрицательные числа превращаются в числа, близкие к 0, а большие положительные – к 1.

Вот как может выглядеть простая нейронная сеть:

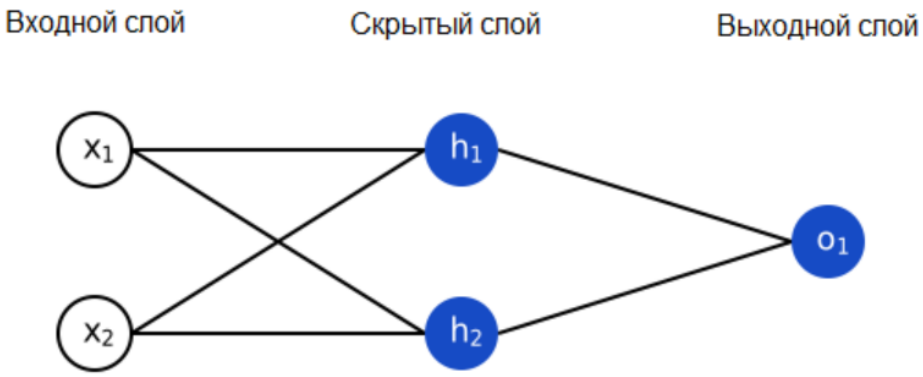

Рисунок 2 – Простая нейронная сеть

Скрытый слой – это любой слой между входным (первым) слоем сети и выходным (последним). Скрытых слоев может быть много.

Прежде чем обучать нашу нейронную сеть, нам нужно как-то измерить, насколько "хорошо" она работает, чтобы она смогла работать "лучше". Это измерение и есть потери (loss).

Мы используем для расчета потерь среднюю квадратичную ошибку (mean squared error, MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

n – это количество измерений, в нашем случае 4 (Алиса, Боб, Чарли и Диана).

y - представляет предсказываемое значение, пол.

ytrue – истинное значение переменной ("правильный ответ").

ypred – предсказанное значение переменной. Это то, что выдаст наша нейронная сеть.

$(ytrue-ypred)^2$ - называется квадратичной ошибкой.

В ходе работы использовалась numPy — библиотека с открытым исходным кодом для языка программирования Python. Возможности: поддержка многомерных массивов; поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Пример выполнения задания

```
Обучение нейронной сети:
Эпоха 0 потери: 0.135
Эпоха 10 потери: 0.061
Эпоха 20 потери: 0.035
Эпоха 30 потери: 0.024
Эпоха 40 потери: 0.017
Эпоха 50 потери: 0.014
Эпоха 60 потери: 0.011
Эпоха 70 потери: 0.009
Эпоха 80 потери: 0.008
Эпоха 90 потери: 0.007
Эпоха 100 потери: 0.006
Эпоха 110 потери: 0.006
Эпоха 120 потери: 0.005
Эпоха 130 потери: 0.005
Эпоха 140 потери: 0.004
Эпоха 150 потери: 0.004
Эпоха 160 потери: 0.004
Эпоха 170 потери: 0.003
Эпоха 180 потери: 0.003
Эпоха 190 потери: 0.003
Эпоха 200 потери: 0.003
Эпоха 210 потери: 0.003
Эпоха 220 потери: 0.003
Эпоха 230 потери: 0.002
Эпоха 240 потери: 0.002
Эпоха 250 потери: 0.002
Эпоха 260 потери: 0.002
Эпоха 270 потери: 0.002
Эпоха 280 потери: 0.002
Эпоха 290 потери: 0.002
Эпоха 300 потери: 0.002
Эпоха 310 потери: 0.002
Эпоха 320 потери: 0.002
Эпоха 330 потери: 0.002
Эпоха 340 потери: 0.002
Эпоха 350 потери: 0.002
Эпоха 360 потери: 0.002
```

```
Эпоха 370 потери: 0.001
Эпоха 380 потери: 0.001
Эпоха 390 потери: 0.001
Эпоха 400 потери: 0.001
Эпоха 410 потери: 0.001
Эпоха 420 потери: 0.001
Эпоха 430 потери: 0.001
Эпоха 440 потери: 0.001
Эпоха 450 потери: 0.001
Эпоха 460 потери: 0.001
Эпоха 470 потери: 0.001
Эпоха 480 потери: 0.001
Эпоха 490 потери: 0.001
Эпоха 500 потери: 0.001
Эпоха 510 потери: 0.001
Эпоха 520 потери: 0.001
Эпоха 530 потери: 0.001
Эпоха 540 потери: 0.001
Эпоха 550 потери: 0.001
Эпоха 560 потери: 0.001
Эпоха 570 потери: 0.001
Эпоха 580 потери: 0.001
Эпоха 590 потери: 0.001
Эпоха 600 потери: 0.001
Эпоха 610 потери: 0.001
Эпоха 620 потери: 0.001
Эпоха 630 потери: 0.001
Эпоха 640 потери: 0.001
Эпоха 650 потери: 0.001
Эпоха 660 потери: 0.001
Эпоха 670 потери: 0.001
Эпоха 680 потери: 0.001
Эпоха 690 потери: 0.001
Эпоха 700 потери: 0.001
Эпоха 710 потери: 0.001
Эпоха 720 потери: 0.001
Эпоха 730 потери: 0.001
Эпоха 740 потери: 0.001
Эпоха 750 потери: 0.001
Эпоха 760 потери: 0.001
Эпоха 770 потери: 0.001
Эпоха 780 потери: 0.001
```

```
Эпоха 790 потери: 0.001
Эпоха 800 потери: 0.001
Эпоха 810 потери: 0.001
Эпоха 820 потери: 0.001
Эпоха 830 потери: 0.001
Эпоха 840 потери: 0.001
Эпоха 850 потери: 0.001
Эпоха 860 потери: 0.001
Эпоха 870 потери: 0.001
Эпоха 880 потери: 0.001
Эпоха 890 потери: 0.001
Эпоха 900 потери: 0.001
Эпоха 910 потери: 0.001
Эпоха 920 потери: 0.001
Эпоха 930 потери: 0.001
Эпоха 940 потери: 0.001
Эпоха 950 потери: 0.001
Эпоха 960 потери: 0.001
Эпоха 970 потери: 0.001
Эпоха 980 потери: 0.001
Эпоха 990 потери: 0.001
Emily: 0.982
Frank: 0.02373
```

Рисунок 1 – Пример выполнения задания

Создадим 7 тестовых заданий величин для предсказания по теме:

```
# Make some predictions
emily = np.array([-7, -3, -3])
frank = np.array([20, 2, 3])
franki = np.array([3, 1, 2])
anna = np.array([-2, -1, -1])
lili = np.array([0, 0, -1])
lulu = np.array([-4, -9, -3])
sasha = np.array([5, 3, 4])
artem = np.array([6, 0, 4])


print("Emily: %.3f" % network.feedforward(emily))
print("Frank: %.3f" % network.feedforward(frank))
print("Franki: %.3f" % network.feedforward(frankf))
print("Anna: %.3f" % network.feedforward(anna))
print("Lili: %.3f" % network.feedforward(lili))
print("Lulu: %.3f" % network.feedforward(lulu))
print("Artem: %.3f" % network.feedforward(artem))
```

Рисунок 2 – Величины для предсказаний

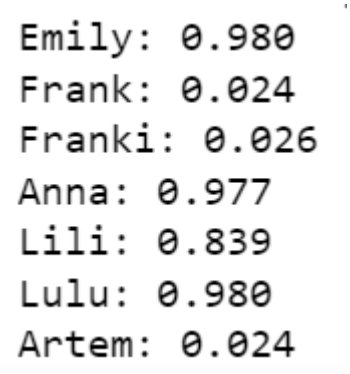

Рисунок 3 – Предсказанные значения

1 – женский род

0 – мужской род

Код программы

```
def sigmoid(x):
  # Sigmoid activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))

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

def mse_loss(y_true, y_pred):
  # y_true and y_pred are numpy arrays of the same length.
  return ((y_true - y_pred) ** 2).mean()
class OurNeuralNetwork:
  '''
  A neural network with:
    - 2 inputs
    - a hidden layer with 2 neurons (h1, h2)
    - an output layer with 1 neuron (o1)

  *** DISCLAIMER ***:
  The code below is intended to be simple and educational, NOT optimal.
  Real neural net code looks nothing like this. DO NOT use this code.
  Instead, read/run it to understand how this specific network works.
  '''
  def __init__(self):
    # Weights
    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()
  self.w7 = np.random.normal()
  self.w8 = np.random.normal()

  # Biases
  self.b1 = np.random.normal()
  self.b2 = np.random.normal()
  self.b3 = np.random.normal()

def feedforward(self, x):
  # x is a numpy array with 2 elements.
  h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.w7 * x[2] + self.b1)
  h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.w8 * x[2] + self.b2)
  o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
  return o1

def train(self, data, all_y_trues):

  learn_rate = 0.1
  epochs = 1000 # number of times to loop through the entire dataset
  print("Обучение нейронной сети:")
  for epoch in range(epochs):
    for x, y_true in zip(data, all_y_trues):
      # --- Do a feedforward (we'll need these values later)
      sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.w7 * x[2] + self.b1
      h1 = sigmoid(sum_h1)

      sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.w8 * x[2] + self.b2
```

```
h2 = sigmoid(sum_h2)


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

# --- Calculate partial derivatives.
# --- Naming: p_L_p_w1 stands for "partial L partial w1"
p_L_p_ypred = -2 * (y_true - y_pred)

# Neuron o1
p_ypred_p_w5 = h1 * deriv_sigmoid(sum_o1)
p_ypred_p_w6 = h2 * deriv_sigmoid(sum_o1)
p_ypred_p_b3 = deriv_sigmoid(sum_o1)

p_ypred_p_h1 = self.w5 * deriv_sigmoid(sum_o1)
p_ypred_p_h2 = self.w6 * deriv_sigmoid(sum_o1)

# Neuron h1
p_h1_p_w1 = x[0] * deriv_sigmoid(sum_h1)
p_h1_p_w2 = x[1] * deriv_sigmoid(sum_h1)
p_h1_p_w7 = x[2] * deriv_sigmoid(sum_h1)
p_h1_p_b1 = deriv_sigmoid(sum_h1)

# Neuron h2
p_h2_p_w3 = x[0] * deriv_sigmoid(sum_h2)
p_h2_p_w4 = x[1] * deriv_sigmoid(sum_h2)
p_h2_p_w8 = x[2] * deriv_sigmoid(sum_h2)
p_h2_p_b2 = deriv_sigmoid(sum_h2)
```

```
        # --- Update weights and biases
        # Neuron h1
        self.w1 -= learn_rate * p_L_p_ypred * p_ypred_p_h1 * p_h1_p_w1
        self.w2 -= learn_rate * p_L_p_ypred * p_ypred_p_h1 * p_h1_p_w2
        self.w7 -= learn_rate * p_L_p_ypred * p_ypred_p_h1 * p_h1_p_w7
        self.b1 -= learn_rate * p_L_p_ypred * p_ypred_p_h1 * p_h1_p_b1

        # Neuron h2
        self.w3 -= learn_rate * p_L_p_ypred * p_ypred_p_h2 * p_h2_p_w3
        self.w4 -= learn_rate * p_L_p_ypred * p_ypred_p_h2 * p_h2_p_w4
        self.w8 -= learn_rate * p_L_p_ypred * p_ypred_p_h2 * p_h2_p_w8
        self.b2 -= learn_rate * p_L_p_ypred * p_ypred_p_h2 * p_h2_p_b2

        # Neuron o1
        self.w5 -= learn_rate * p_L_p_ypred * p_ypred_p_w5
        self.w6 -= learn_rate * p_L_p_ypred * p_ypred_p_w6
        self.b3 -= learn_rate * p_L_p_ypred * p_ypred_p_b3

      # --- Calculate total loss at the end of each epoch
      if epoch % 10 == 0:
        y_preds = np.apply_along_axis(self.feedforward, 1, data)
        loss = mse_loss(all_y_trues, y_preds)
        print("Эпоха %d потери: %.3f" % (epoch, loss))

# Define dataset
data = np.array([
  [-2, -1, -1],  # Alice
  [25, 6, 3],   # Bob
  [17, 4, 1],   # Charlie
```

```
  [-15, -6, -3], # Diana
  [-10, -3, -2], #katya
  [-1, -1, -1], #Sveta
  [20, 5, 3], #Oleg
  [12, 3, 0], #Ruslan
  [-5, -3, -2], #Olga
  [-3, -2, -2], #Anya
  [10, 2, 1], #Artem
])
all_y_trues = np.array([
  1, # Alice
  0, # Bob
  0, # Charlie
  1, # Diana
  1,
  1,
  0,
  0,
  1,
  1,
  0
])

# Train our neural network!
network = OurNeuralNetwork()
network.train(data, all_y_trues)

# Make some predictions
emily = np.array([-7, -3, -3]) # 128 pounds, 63 inches, 38sm
frank = np.array([20, 2, 3])  # 155 pounds, 68 inches, 44sm
```

```
print("Emily: %.3f" % network.feedforward(emily))
print("Frank: %.3f" % network.feedforward(frank))
```

Вывод

В ходе выполнения индивидуального домашнего задания мы получили базовые навыки работы с языком python и набором функций для анализа и обработки данных.