Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

Отчет по лабораторной работе №4

По дисциплине “Методы и алгоритмы принятия решений”

Тема: “Нелинейные ИНС в задачах прогнозирования. Адаптивный шаг обучения” **Вариант №8**

**Выполнил**:

Крук Д.Н.

студент 2 курса группы ПО-7

**Проверил:**

Крощенко А.А.

Брест 2021

**Цель работы:**

Изучить обучение и функционирование нелинейной ИНС при решении задач распознавания образов.

**Задание:**

Написать на любом ЯВУ программу моделирования прогнозирующей нелинейной ИНС. Для тестирования использовать функцию: 𝑦 = 0. 4𝑐𝑜𝑠(0.2 \* 𝑥) + 0. 07𝑠𝑖𝑛(0.2 \* 𝑥) Для прогнозирования использовать многослойную ИНС с одним скрытым слоем. В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного – линейную. При этом необходимо использовать алгоритм обучения многослойной ИНС с адаптивным шагом.

**Выполнение:**

**Код программы:**

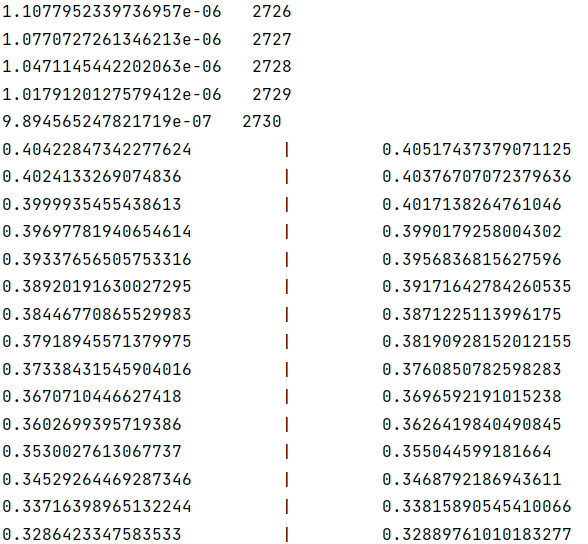
**Main.py:**

**from** layer **import** \*  
  
INPUT\_SIZE = 6  
HIDDEN\_SIZE = 2  
EXPRESSION\_STEP = 0.2  
E = 1e-6  
  
input\_layer = Layer(INPUT\_SIZE, HIDDEN\_SIZE)  
hide\_layer = LastLayer(HIDDEN\_SIZE)  
  
  
**def** expression(x):  
 a, b, c, d = 0.4, 0.2, 0.07, 0.2  
 **return** a \* math.cos(b \* x) + c \* math.sin(d \* x)  
  
  
**def** training\_data():  
 vectors = []  
 start\_x = 0  
 **for** \_ **in** range(15):  
 vector = []  
 **for** i **in** range(INPUT\_SIZE):  
 vector.append(expression(start\_x + i \* EXPRESSION\_STEP))  
 vectors.append([vector, expression(start\_x + INPUT\_SIZE \* EXPRESSION\_STEP)])  
 start\_x += EXPRESSION\_STEP  
 **return** vectors  
  
  
**def** predict(vector):  
 input\_layer.set\_values(vector)  
 hide\_layer.set\_values(input\_layer.output\_values)  
 **return** hide\_layer.result  
  
  
**def** education():  
 train\_data = training\_data()  
 error = E  
 k = 0  
 **while** abs(error) >= E:  
 error = 0  
 **for** data **in** train\_data:  
 e = predict(data[0]) - data[1]  
 error += e \*\* 2 / 2  
 hide\_layer.weights\_change(e)  
 input\_layer.weights\_change(hide\_layer.error)  
 k += 1  
 print(error, **' '**, k)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 education()  
 train\_data = training\_data()  
 **for** data **in** train\_data:  
 print(**'{0:<28}{1:<10}{2:<30}'**.format(predict(data[0]), **'|'**, data[1]))

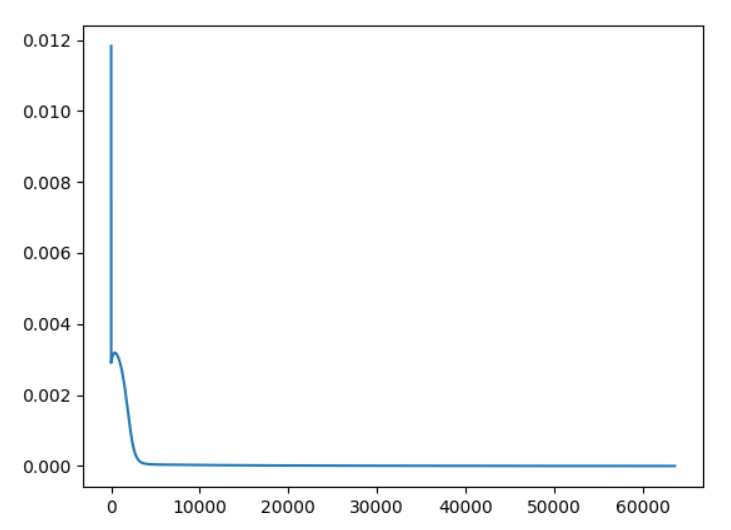
**layer.py:**

**import** math  
**import** random  
**import** numpy **as** np  
  
TRAINING\_STEP = 0.2  
  
  
**class** Layer:  
 **def** \_\_init\_\_(self, in\_size, nl\_size):  
 self.\_\_training\_step = TRAINING\_STEP  
 self.\_\_input\_values = []  
 self.output\_values = []  
 self.\_\_weights = []  
 self.error = []  
 self.\_\_T = [random.uniform(-math.sqrt(in\_size), math.sqrt(in\_size)) **for** \_ **in** range(nl\_size)]  
 **for** \_ **in** range(nl\_size):  
 self.\_\_weights.append([random.uniform(-math.sqrt(in\_size), math.sqrt(in\_size)) **for** \_ **in** range(in\_size)])  
  
 **def** set\_values(self, input\_values):  
 self.\_\_input\_values = input\_values  
 self.output\_values = [1 / (1 + math.exp(-s)) **for** s **in** self.\_\_get\_sum()]  
  
 **def** \_\_get\_sum(self):  
 S = []  
 **for** r, W **in** enumerate(self.\_\_weights):  
 S.append(sum([w \* self.\_\_input\_values[c] **for** c, w **in** enumerate(W)]) - self.\_\_T[r])  
 **return** S  
  
 **def** \_\_error\_init(self, error: list):  
 weights = np.array(self.\_\_weights).T.tolist()  
 self.error = [sum([error[ind] \* self.output\_values[ind] \* (1 - self.output\_values[ind]) \* w **for** ind, w **in** enumerate(weight)]) **for** weight **in** weights]  
  
 **def** \_\_adaptive\_step(self):  
 numerator = 4 \* sum(  
 [self.error[ind] \*\* 2 \* self.\_\_input\_values[ind] \* (1 - self.\_\_input\_values[ind]) **for** ind **in** range(len(self.error))])  
 denominator = (1 + sum([y \*\* 2 **for** y **in** self.output\_values])) \* (sum(  
 [(self.error[ind] \* self.\_\_input\_values[ind] \* (1 - self.\_\_input\_values[ind])) \*\* 2 **for** ind **in** range(len(self.error))]))  
 self.\_\_training\_step = numerator / denominator  
  
 **def** weights\_change(self, error: list):  
 self.\_\_error\_init(error)  
 self.\_\_adaptive\_step()  
 **for** r, W **in** enumerate(self.\_\_weights):  
 **for** c, w **in** enumerate(W):  
 self.\_\_weights[r][c] -= self.\_\_training\_step \* error[r] \* self.output\_values[r] \* \  
 self.\_\_input\_values[c] \* (1 - self.output\_values[r])  
  
 **for** ind **in** range(len(self.\_\_T)):  
 self.\_\_T[ind] += self.\_\_training\_step \* error[ind] \* self.output\_values[ind] \* (1 - self.output\_values[ind])  
  
  
**class** LastLayer:  
 **def** \_\_init\_\_(self, in\_size):  
 self.\_\_training\_step = TRAINING\_STEP  
 self.result = **None** self.error = []  
 self.\_\_input\_values = []  
 self.\_\_weights = [random.uniform(-math.sqrt(in\_size), math.sqrt(in\_size)) **for** \_ **in** range(in\_size)]  
 self.\_\_T = random.uniform(-1, 1)  
  
 **def** set\_values(self, input\_values):  
 self.\_\_input\_values = input\_values  
 self.result = sum([w \* self.\_\_input\_values[ind] **for** ind, w **in** enumerate(self.\_\_weights)]) - self.\_\_T  
  
 **def** \_\_error\_init(self, error: list):  
 self.error = [error \* self.result \* (1 - self.result) \* w **for** w **in** self.\_\_weights]  
  
 **def** \_\_adaptive\_step(self):  
 self.\_\_training\_step = 1 / (1 + sum([x \*\* 2 **for** x **in** self.\_\_input\_values]))  
  
 **def** weights\_change(self, error):  
 self.\_\_error\_init(error)  
 self.\_\_adaptive\_step()  
 **for** ind **in** range(len(self.\_\_weights)):  
 self.\_\_weights[ind] -= self.\_\_training\_step \* error \* self.\_\_input\_values[ind]  
 self.\_\_T += self.\_\_training\_step \* error

**Результат программы:**

****

**График изменения ошибки:**

****

**Вывод:**

Научился использовать адаптивный шаг обучения для персептрона с одним скрытым слоем. Адаптивный шаг обучения помог значительно (в 4 раза) улучшить скорость обучения ИНС.