**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение**

**Высшего профессионального образования**

**Московский технический университет связи и информатики**

**Лабораторная работа №1**

**Вариант 1**

Выполнил:

Саитов Р.М.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

**Обучение однослойного персептрона методом коррекции по ошибке через дельта-правило**

**Цель работы**. Изучить алгоритм обучения однослойного персептрона методом коррекции по ошибке через дельта-правило.

**Задание.** В соответствии с вариантом, необходимо обучить нейронную сеть распознавать указанные 4 символа. На каждый символ необходимо подготовить 4 обучающих образа с использованием разных шрифтов. Соответственно, всего будет 16 обучающих образов: 4 образа первым шрифтом, 4 образа, вторым шрифтом и т.д. Тестовая выборка должна содержать по 1 образу на каждый из 4-х символов. Символы должны быть написаны другим шрифтом, который не был использован в обучающей выборке.

**Теоретические сведения.** Алгоритм обучения с положительным и отрицательным подкреплением можно представить в более общей форме – в виде дельта-правила. Если за *dj* обозначить желаемый выходной сигнал *j*-го нейрона (от слов desire response, что в переводе с английского означает «желаемый отклик»), то на каждой эпохе (итерации) обучения можно рассчитывать разницу между желаемым *dj* ответом *j*-го нейрона и реальным значением *yj*, вычисляемым на его выходе. Соответственно, ошибкой выхода персептрона называется следующее выражение:

Тогда относительно алгоритма обучения персептрона с положительным и отрицательным подкреплением:

- случай *εj = 0* соответствует шагу 4а;

- случай *εj > 0* соответствует шагу 4б;

- случай *εj < 0* соответствует шагу 4в.

Идея алгоритма обучения персептронного слоя с помощью правила Хебба сохранится, если итерационный процесс корректировки весов вести по формулам:

|  |
| --- |
|  |
|  |

где *wi,j(t)* и *wi,j(t+1)* – старое и новое значения синаптических весов персептрона, *Δwi,j(t + 1)* – новое значение величины коррекции синаптических весов персептрона, *i* – номер входного сигнала, *j* – номер нейрона. Приведенные формулы называются дельта-правилом.

Кроме того, можно получить аналогичную итерационную формулу для подстройки нейронного смещения *b*, если учесть, что его можно интерпретировать как вес *w0,j* дополнительного входа *x0,j*, значение которого всегда равно 1:

|  |
| --- |
|  |
|  |

В итерационные формулы полезно ввести коэффициент скорости обучения *η*, с помощью которого можно управлять величиной коррекции синаптических весов и нейронного смещения:

|  |
| --- |
|  |
|  |

При слишком больших значениях коэффициента *η* обычно теряется устойчивость процесса обучения, тогда как при слишком малых – увеличивается время обучения. На практике коэффициент скорости обучения *η* обычно задают в пределах от 0.05 до 1.

Алгоритм обучения персептрона с использованием этих формул известен под названием обучения с коррекцией по ошибке или дельта-правило.

В общем виде, алгоритм обучения с коррекцией по ошибке через дельта-правило, представлен ниже.

***Алгоритм обучения*** ***методом коррекции по ошибке через дельта-правило***

Шаг 1. Подготовить обучающую выборку, каждый элемент которой будет состоять из пар *(X, D)m (m=1,…q)* – обучающего вектора *X = (x1,…,xn)* (*i=1,…,n)* с вектором желаемых значений *D = (d1,…,dk) (j=1,…,k*) выходов персептрона.

Шаг 2. Генератором случайных чисел всем синаптическим весам *wij* и нейронным смещениям *w0,j* (*i=1,…,n; j=1,…,k*) присваиваются некоторые малые случайные значения.

Шаг 3. Из обучающей выборки *(X, D)1,…,(X, D)q*, взять следующий по счету вектор *Xт = (x1,…,xn)* и подать его на входы персептрона *x1,…,xn*. Сигналам нейронных входов смещения *x0*присваиваются единичные значения: *x0 = 1*.

Шаг 4. Для каждого *j*-го нейрона вычислить взвешенную сумму входных сигналов *netj* и выходной сигнал *yj* на основании функции активации *f*:

|  |
| --- |
|  |
|  |

Шаг 5. Для каждого *j*-го нейрона вычислить его ошибку:

где *dj* – вектор правильных (желаемых) ответов персептрона.

Шаг 6. Произвести коррекцию синаптических весов *j*-го нейрона и нейронных смещений:

|  |
| --- |
|  |
|  |

где t – номер итерации.

Шаги 4-6 повторяются для всех нейронов персептронного слоя при подаче конкретного образа.

Шаги 3-6 выполняются последовательно для каждого входного образа, на котором обучается персептрон.

Шаг 7. После подачи последнего обучающего вектора, проверить критерий останова обучения, если он выполняется, то завершить обучение. В противном случае – возврат к шагу 3.

Критерии останова алгоритма обучения могут быть следующими:

1. Значения синаптических весов *wi,j* перестают изменяться;
2. Перестают появляться неправильные выходные сигналы *yj*;
3. Превышен установленный лимит количества эпох (либо итераций).

Указанные критерии останова обучения используются как по отдельности, так и вместе. Важным замечанием будет то, что 3-ий критерий, при задании недостаточного количества эпох, может привести к недообучению сети, поэтому его использование в промышленном применении не рекомендуется, но в учебных целях он показателен и удобен.

**Ход выполнения работы**

**1. Подготовка данных**

Для начала было разработано несколько функций для генерации и визуализации бинарных изображений символов. Используя библиотеку Pillow (PIL), были созданы изображения текстовых символов, которые затем переведены в бинарный вид и сохранены в виде массива numpy.

**2. Обучение**

**Параметры:**

* **Размер изображений:** 20x20 пикселей.
* **Шрифты:** Arial, Verdana, Times New Roman и LiberationSerif.
* **Скорость обучения:** 0.1
* **Эпохи:** 10000
* **Функция активации:** Сигмоида
* **Инициализация весов:** Случайные значения из нормального распределения с использованием np.random.randn.

**3. Обучающий процесс**

Для каждого символа из списка ['A', 'B', 'C', 'D'] были сгенерированы бинарные образы с использованием разных шрифтов. Эти данные использовались для создания обучающей выборки. Обоучающая выборка разделена на входные данные и целевые выходы.

Обратное распространение ошибок выполнялось с использованием Delta-правила для обновления весов однослойного персептрона, при этом каждая эпоха состояла из полного цикла по всем обучающим данным.

### Тестирование

После завершения обучения были протестированы новые данные, полученные с использованием новой шрифта (Arial). Для каждого тестового символа сеть выдавала предсказание, которое сравнивалось с реальным символом:

**Вывод.**

Однослойный персептрон успешно обучился распознавать символы 'A', 'B', 'C' и 'D', показав ассоциативные свойства и хорошие результаты на тестовых данных. Однако, стоит учесть, что корректные ответы могли быть достигнуты не во всех случаях.

**Исходный код программы**

import numpy as np  
from PIL import Image, ImageDraw, ImageFont  
import os  
import matplotlib.pyplot as plt  
  
# Генерация бинарного представления символа  
def generate\_symbol\_image(symbol, font\_path, image\_size=(20, 20)):  
 image = Image.new('L', image\_size, color=255)  
 draw = ImageDraw.Draw(image)  
  
 if not os.path.isfile(font\_path):  
 raise ValueError(f"Файл шрифта '{font\_path}' не найден.")  
  
 font = ImageFont.truetype(font\_path, 15)  
 text\_bbox = draw.textbbox((0, 0), symbol, font=font)  
 text\_width, text\_height = text\_bbox[2] - text\_bbox[0], text\_bbox[3] - text\_bbox[1]  
 text\_position = ((image\_size[0] - text\_width) // 2, (image\_size[1] - text\_height) // 2)  
 draw.text(text\_position, symbol, fill=0, font=font)  
 binary\_image = np.array(image).flatten() // 255  
 return binary\_image  
  
# Пример генерации и визуализации символа  
def visualize\_symbol(symbol, font\_path):  
 binary\_image = generate\_symbol\_image(symbol, font\_path)  
 image = binary\_image.reshape((20, 20))  
 plt.imshow(image, cmap='gray')  
 plt.title(symbol)  
 plt.show()  
  
# Список символов  
symbols = ['A', 'B', 'C', 'D']  
  
# Проверьте, что пути к шрифтам указаны корректно. Замените на ваши реальные пути!  
fonts\_directory = "/Users/rmsaitov/wrs/mtuci/nn/lab1"  
fonts = [  
 os.path.join(fonts\_directory, 'Arial.ttf'),  
 os.path.join(fonts\_directory, 'Verdana.ttf'),  
 os.path.join(fonts\_directory, 'TimesNewRoman.ttf'),  
 os.path.join(fonts\_directory, 'LiberationSerif-Regular.ttf')  
]  
  
for font in fonts:  
 if not os.path.isfile(font):  
 print(f"Ошибка: файл шрифта '{font}' не найден. Укажите правильный путь к шрифту.")  
 exit(1)  
  
training\_data = []  
outputs = []  
  
# Создаем обучающую выборку  
for i, symbol in enumerate(symbols):  
 for font in fonts:  
 binary\_image = generate\_symbol\_image(symbol, font)  
 training\_data.append(binary\_image)  
 output = [0] \* len(symbols)  
 output[i] = 1  
 outputs.append(output)  
  
training\_data = np.array(training\_data)  
outputs = np.array(outputs)  
  
# Инициализация весов  
np.random.seed(0)  
n\_inputs = len(training\_data[0])  
n\_outputs = len(symbols)  
weights = np.random.randn(n\_inputs, n\_outputs)  
learning\_rate = 0.1  
epochs = 10000  
  
  
def activation(x):  
 return 1 / (1 + np.exp(-x)) # Сигмоида  
  
  
def predict(inputs):  
 return activation(np.dot(inputs, weights))  
  
  
# Алгоритм обучения  
for epoch in range(epochs):  
 for inputs, target in zip(training\_data, outputs):  
 outputs\_pred = predict(inputs)  
 error = target - outputs\_pred  
 weights += learning\_rate \* np.dot(inputs[:, None], error[None, :])  
  
 if epoch % 1000 == 0:  
 total\_error = np.sum(np.square(outputs - predict(training\_data)))  
 print(f'Epoch {epoch}, Total Error: {total\_error}')  
  
print("Обучение завершено")  
  
visualize\_symbol('A', os.path.join(fonts\_directory, 'Arial.ttf'))  
visualize\_symbol('B', os.path.join(fonts\_directory, 'Verdana.ttf'))  
visualize\_symbol('C', os.path.join(fonts\_directory, 'TimesNewRoman.ttf'))  
visualize\_symbol('D', os.path.join(fonts\_directory, 'LiberationSerif-Regular.ttf'))  
  
# Тестирование на новых данных  
test\_font = os.path.join(fonts\_directory, 'Arial.ttf') # Шрифт, который не использовался в обучающей выборке  
test\_data = [generate\_symbol\_image(symbol, test\_font) for symbol in symbols]  
  
for test\_case, symbol in zip(test\_data, symbols):  
 output = predict(test\_case)  
 predicted\_symbol\_index = np.argmax(output)  
 predicted\_symbol = symbols[predicted\_symbol\_index]  
 print(f'Test case: {symbol}, Predicted: {predicted\_symbol}')