МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І.І. МЕЧНИКОВА
Факультет математики, фізики та інформаційних технологій

Звіт
з дисципліни: «Комп’ютерні системи штучного інтелекту»
Лабораторна робота № 3
за темою: «Бінарний перцептрон»

Виконав: студент денної форми навчання
Лавров Владислав
спеціальності: Комп’ютерна інженерія
кафедри: КСТ
курс: 4
Викладач:
Гожий О.П.

Одеса – 2024

Лабораторна робота №3. Бінарний перцептрон

Завдання: Програмно реалізувати наведений алгоритм. Використовуючи дані таблиці 1.1 і результат пункту 1, побудувати таблицю 1.2. Алгоритм реалізувати таким чином: на вхід поступово подати всі образи, коригуючи паралельно вагові коефіцієнти. Якщо для всіх образів $\delta = 0$, то кінець, якщо ні, то на вхід знову ж таки подати всі образи, але із скоригованими ваговими коефіцієнтами.

Таблиця 1.1 – Початкові дані

№ варіанта	**Логічна функція**	**Можливі значення вагових коефіцієнтів**		
7	$x_1 \wedge (x_2 \rightarrow x_3)$	0,2	0,02	0,002

Таблиця 1.2 – Таблиця подання результатів роботи програми

w_1	w_2	w_3	θ	x_1	x_2	x_3	a	Y	T	$\eta(T-Y)$	δw_1	δw_2	δw_3	$\delta\theta$

Контрольні питання:

1. Коротка історична довідка про розвиток теорії нейронних мереж.

Теорія нейронних мереж почала формуватися у 1940-х роках за роботами Маккаллоха та Піттса. У 1980-х виник backpropagation, що відкрило нові можливості для навчання мереж. В 1990-х нейронні мережі стали використовуватися в різних областях, зокрема, у розпізнаванні образів та обробці природної мови. У 2010-х роках з'явилося глибоке навчання, що прискорило розвиток нейронних мереж та дало змогу досягнути значних успіхів у штучному інтелекті.

2. Будова біологічного нейрона.

Біологічний нейрон складається з трьох основних частин: дендритів, соми та аксону.

1. Дендрити: Це відростки нейрона, які приймають сигнали від інших нейронів або від зовнішніх джерел. Дендрити функціонують як вхідні канали для інформації, яка потім передається до соми.

2. Сома (клітинне тіло): Це центральна частина нейрона, де обробляється та інтегрується вхідна інформація. Сома містить ядро та інші органели, необхідні для життєдіяльності клітини.

3. Аксон: Це довгий відросток нейрона, який передає сигнали від соми до інших нейронів, м'язів або залоз через електричні імпульси. Аксони можуть бути довжиною від кількох мікрометрів до декількох метрів у деяких випадках (наприклад, в аксонах моторних нейронів людини).

Крім цих основних компонентів, біологічний нейрон також має аксонні термінали, які випускають нейротрансмітери - хімічні речовини, що передають сигнали до сусідніх нейронів або до ефекторних клітин (таких як м'язи або залози) через простір між нейронами, відомий як синапс.

3. Штучний нейрон та активаційні функції.

Штучний нейрон - це базовий елемент штучної нейронної мережі, який моделює біологічний нейрон. Він приймає вхідні сигнали, обробляє їх та генерує вихідний сигнал. Основні компоненти штучного нейрона включають ваги, суматор, активаційну функцію та поріг.

1. Ваги: Кожен вхідний сигнал має свою вагу, яка визначає його важливість для роботи нейрона. Ваги регулюються під час процесу навчання нейронної мережі.

2. Суматор: Суматор обчислює зважену суму всіх вхідних сигналів, помножених на їх вагу, і додає їх разом. Це значення передається наступному кроку - активаційній функції.

3. Активаційна функція: Активаційна функція приймає вихід суматора та генерує вихід нейрона. Вона може бути лінійною або нелинійною та відповідати за активацію нейрона певним чином, наприклад, вона може визначати, коли нейрон має вогні або припиняти свою активність.

Активаційні функції різних типів можуть бути використані для різних завдань і можуть включати в себе сигмоїдальні функції, функції ReLU (Rectified Linear Unit), tanh (гіперболічний тангенс) та інші. Вони допомагають нейронній мережі навчатися та адаптуватися до складних залежностей у вхідних даних.

4. Будова та алгоритм функціонування перцептрона.

Перцептрон - це проста форма штучного нейрона, яка складається з вхідних зв'язків, ваг, функції сумування та функції активації. Вхідні зв'язки мають свої ваги, які відображають їхню важливість для виходу перцептрона. Функція сумування обчислює взважену суму всіх вхідних значень, а функція активації визначає, чи буде відповідний вихід активованим. Алгоритм функціонування перцептрона включає ініціалізацію ваг, подачу вхідних даних, обчислення взваженої суми, застосування функції активації та оновлення ваг у разі необхідності для навчання. Цей процес повторюється протягом багатьох ітерацій з метою налаштування ваг для правильної класифікації вхідних даних.

5. Проблема лінійного поділу досліджувальної області.

Проблема лінійного поділу досліджуваної області полягає в тому, що не всі області даних можуть бути коректно розділені за допомогою простих лінійних моделей чи алгоритмів. У деяких випадках дані можуть бути складними та мають нелінійну структуру, що ускладнює їх класифікацію чи прогнозування. Наприклад, якщо дані мають складну взаємозв'язок або неоднорідний розподіл, то лінійні моделі можуть давати неприйнятні результати. Ця проблема стає актуальною у задачах машинного навчання, де метою є класифікація об'єктів або прогнозування значень на основі вхідних даних.

Для вирішення цієї проблеми часто використовуються більш складні моделі, такі як не лінійні класифікатори або нейронні мережі, які здатні до адаптації до більш складних структур даних. Наприклад, нейронні мережі можуть виявити та використовувати нелінійні залежності в даних для кращого класифікації або прогнозування. Такі моделі можуть бути більш гнучкими та ефективними в роботі з різноманітними типами даних і допомагають уникнути обмежень лінійних моделей.

Хід роботи:

Розрахуємо значення *T* для логічної функції за наданим варіантом (рис 1).

```
func valuesOfFunc(X [][3]float64) [][]float64 {
	values := make([][]float64, len(X))
	for i, vInput := range X {
		result := 0.0
		if vInput[0] != 0 && vInput[1] == 0 || vInput[2] != 0 {
			result = 1.0
		}
		values[i] = []float64{vInput[0], vInput[1], vInput[2],
result}
	}
	return values
}

values := valuesOfFunc([][3]float64{{1, 1, 1}, {0, 1, 1}, {1, 0, 1}, {0,
0, 1}, {1, 1, 0}, {0, 1, 0}, {1, 0, 0}, {0, 0, 0}})
```

```
[[1 1 1 1] [0 1 1 1] [1 0 1 1] [0 0 1 1] [1 1 0 0] [0 1 0 0] [1 0 0 1] [0 0 0 0]]
```

Рисунок 1 – Результат розрахунку *T*

Далі за прикладом було виконано 5 ітерацій. Код алгоритму наведений у додатку А.

№	w_1	w_2	w_3	x_1	x_2	x_3	a	Y	T	η(T-Y)	Δw_1	Δw_2	Δw_3
1	0,2	0,02	0	**1**	**1**	**1**	0,22	1	**1**	0	0	0	0
2	0,2	0,02	0	**0**	**1**	**1**	0,02	1	**1**	0	0	0	0
3	0,2	0,02	0	**1**	**0**	**1**	0,2	1	**1**	0	0	0	0
4	0,2	0,02	0	**0**	**0**	**1**	0	1	**1**	0	0	0	0
5	0,2	0,02	0	**1**	**1**	**0**	0,22	1	**0**	-1	-0,1	-0,1	0
6	0,1	-0,08	0	**0**	**1**	**0**	-0,08	0	**0**	0	0	0	0
7	0,1	-0,08	0	**1**	**0**	**0**	0,1	1	**1**	0	0	0	0
8	0,1	-0,08	0	**0**	**0**	**0**	0	0	**0**	0	0	0	0

Рисунок 2 – Початок роботи

1	0,1	-0,08	0	**1**	**1**	**1**	0,02	1	**1**	0	0	0	0
2	0,1	-0,08	0	**0**	**1**	**1**	-0,08	0	**1**	1	0	0,1	0,1
3	0,1	0,02	0,1	**1**	**0**	**1**	0,2	1	**1**	0	0	0	0
4	0,1	0,02	0,1	**0**	**0**	**1**	0,1	1	**1**	0	0	0	0
5	0,1	0,02	0,1	**1**	**1**	**0**	0,12	1	**0**	-1	-0,1	-0,1	0
6	0	-0,08	0,1	**0**	**1**	**0**	-0,08	0	**0**	0	0	0	0
7	0	-0,08	0,1	**1**	**0**	**0**	0	0	**1**	1	0,1	0	0
8	0,1	-0,08	0,1	**0**	**0**	**0**	0	0	**0**	0	0	0	0

Рисунок 3 – Друга ітерація

1	0,1	-0,08	0,1	**1**	**1**	**1**	0,12	1	**1**	0	0	0	0
2	0,1	-0,08	0,1	**0**	**1**	**1**	0,02	1	**1**	1	0	0	0
3	0,1	-0,08	0,1	**1**	**0**	**1**	0,2	1	**1**	0	0	0	0
4	0,1	-0,08	0,1	**0**	**0**	**1**	0,1	1	**1**	0	0	0	0
5	0,1	-0,08	0,1	**1**	**1**	**0**	0,02	1	**0**	-1	-0,1	-0,1	0
6	0	-0,18	0,1	**0**	**1**	**0**	-0,18	0	**0**	0	0	0	0
7	0	-0,18	0,1	**1**	**0**	**0**	0	0	**1**	1	0,1	0	0
8	0,1	-0,18	0,1	**0**	**0**	**0**	0	0	**0**	0	0	0	0

Рисунок 4 – Третя ітерація

1	0,1	-0,18	0,1	**1**	**1**	**1**	0,02	1	**1**	0	0	0	0
2	0,1	-0,18	0,1	**0**	**1**	**1**	-0,08	0	**1**	1	0	0,1	0,1
3	0,1	-0,08	0,2	**1**	**0**	**1**	0,3	1	**1**	0	0	0	0
4	0,1	-0,08	0,2	**0**	**0**	**1**	0,2	1	**1**	0	0	0	0
5	0,1	-0,08	0,2	**1**	**1**	**0**	0,02	1	**0**	-1	-0,1	-0,1	0
6	0	-0,18	0,2	**0**	**1**	**0**	-0,18	0	**0**	0	0	0	0
7	0	-0,18	0,2	**1**	**0**	**0**	0	0	**1**	1	0,1	0	0
8	0,1	-0,18	0,2	**0**	**0**	**0**	0	0	**0**	0	0	0	0

Рисунок 5 – Четверта ітерація

П’ята ітерація подана у вигляді таблиці подання результатів, адже вонає кінцевою, тому що значення похибки у всіх рядках дорівнює нулю.

w_1	w_2	w_3	θ	x_1	x_2	x_3	a	Y	T	$\eta(T-Y)$	δw_1	δw_2	δw_3	$\delta\theta$
0.1	-0,18	0.2	0	1	1	1	0.12	1	1	0	0	0	0	0
0.1	-0,18	0.2	0	0	1	1	0.02	1	1	0	0	0	0	0
0.1	-0,08	0.2	0	1	0	1	0.3	1	1	0	0	0	0	0
0.1	-0,08	0.2	0	0	0	1	0.2	1	1	0	0	0	0	0
0.1	-0,08	0.2	0	1	1	0	-0.08	0	0	0	0	0	0	0
0.1	-0,18	0.2	0	0	1	0	-0,18	0	0	0	0	0	0	0
0.1	-0,18	0.2	0	1	0	0	0,1	1	1	0	0	0	0	0
0.1	-0,18	0.2	0	0	0	0	0	0	0	0	0	0	0	0

Висновки:

В ході виконання даної лабораторної роботи було вивчено принципи функціонування і навчання перцептрона. Було отримано практичні навички реалізації алгоритма бінарного перцептрону для визначення логічної функції за варіантом. Процес виконання роботи можна побачити на рис.1-5. Результат подано у вигляді таблиці.

Додаток А – Код програми

Лістинг 1 – Реалізація бінарного перцептрону

```
package main
import (
     "fmt"
)
type BinaryPerceptron struct {
     inputSize int
     weights   [3]float64
}
func NewBinaryPerceptron(inputSize int) *BinaryPerceptron {
     return &BinaryPerceptron{
          inputSize: inputSize,
          weights:   [3]float64{0.2, 0.02, 0.002},
     }
}
func (bp *BinaryPerceptron) predict(inputs [3]float64) (float64,
int) {
     weightedSum := bp.weights[0]*inputs[0] +
bp.weights[1]*inputs[1] + bp.weights[2]*inputs[2]
     prediction := 0
     if weightedSum > 0 {
          prediction = 1
     }
     return weightedSum, prediction
}
func (bp *BinaryPerceptron) train(inputs [][3]float64, targets
[]float64, learningRate float64, epochs int) {
     for epoch := 0; epoch < epochs; epoch++ {
          fmt.Println("Iteration #", epoch+1)
          isError := false
          for i, input := range inputs {
               sum, prediction := bp.predict(input)
               error := targets[i] - float64(prediction)
               if error != 0 {
                    isError = true
               }
               prevWeights := bp.weights
               for j := range prevWeights {
                    bp.weights[j] += learningRate * error *
```

```
input[j]
                }
                fmt.Printf("#%d w_1=%.2f w_2=%.2f w_3=%.2f x_1=%.2f
x_2=%.2f x_3=%.2f a=%.2f Y=%d T=%.2f error=%.2f delta_w_1=%.2f
delta_w_2=%.2f delta_w_3=%.2f\n",
                     i+1, prevWeights[0], prevWeights[1],
prevWeights[2], input[0], input[1], input[2], sum, prediction,
targets[i], error,
                     learningRate*error*input[0],
learningRate*error*input[1], learningRate*error*input[2])
          }
          fmt.Println()
          if !isError {
               fmt.Println("Training is over")
               break
          }
     }
}
func valuesOfFunc(X [][3]float64) [][]float64 {
     values := make([][]float64, len(X))
     for i, vInput := range X {
          result := 0.0
          if vInput[0] != 0 && vInput[1] == 0 || vInput[2] != 0 {
               result = 1.0
          }
          values[i] = []float64{vInput[0], vInput[1], vInput[2],
result}
     }
     return values
}
func main() {
     values := valuesOfFunc([][3]float64{{1, 1, 1}, {0, 1, 1}, {1,
0, 1}, {0, 0, 1}, {1, 1, 0}, {0, 1, 0}, {1, 0, 0}, {0, 0, 0}})
     fmt.Println(values)
     fmt.Println()

     inputs := make([][3]float64, len(values))
     targets := make([]float64, len(values))

     for i, val := range values {
          inputs[i] = [3]float64{val[0], val[1], val[2]}
          targets[i] = val[3]
     }

     perceptron := NewBinaryPerceptron(3)

     perceptron.train(inputs, targets, 0.1, 20)

     // Testing data
     newData := [][3]float64{{1, 1, 1}, {0, 0, 0}, {1, 1, 0}, {0,
1, 1}}
     for _, data := range newData {
          _, prediction := perceptron.predict(data)
          fmt.Println("Prediction for", data, ":", prediction)
     }
}
```