Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики і обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Самостійна робота

з курсу «*Розпізнавання та класифікація в управлінні*»

«Багатошаровий персептрон. Розпізнавання геометричних фігур»

Виконали:

студенти групи ІА-33

Прудніков А. О.

Сокол А. В.

Перевірив:

Дорогий Я.Ю.

Київ – 2016

Ціль роботи

Реалізувати багатошаровий персептрон та використати його для розпізнавання геометричних.

Хід роботи

Персептрон являє з себе найпростішу нейронну мережу, відповідно потребує три базові компоненти для своєї роботи:

1. Декілька шарів нейронів
   В нашому випадку ми маємо базовий багатошаровий персептрон, що складається з 3-х шарів: вхідний, прихований та вихідний.

2. Навчаюча множина
   Навчаюча множина генерується програмно. Випадковим чином вибираються точки у прямокутному зображенні заданого розміру та формують геометричну фігуру. В нашому випадку це трикутник, прямокутник та лінія.

3. Алгоритм корекції помилки
   Використаний класичний алгоритм зворотного поширення помилки (backpropagation), що зустрічається у більшості простий нейронних мереж.

Також слід перерахувати кілька важливих характеристик розробленої нами нейронної мережі:

- Функція активації
  Використовується стандартна функція – сигмоїд
  $$OUT = \frac{1}{1 + \exp\left(- \alpha Y\right)}$$
  Де α = 1

- Функція градієнту
  Для вихідного шару
  $$\delta = OUT(1 - OUT)(T - OUT)$$
  Де:
  - OUT – вихід нейрона
  - T – очікувана вихідна величина
  - δ – просто змінна, що тримає в собі коефіцієнт. Використовується для корекції ваги як у поточному шарі, так і в наступному

Для прихованих шарів

$$\delta_n = OUT_{n-1}(1 - OUT)\sum_{k}^{i} \omega_i \delta_i$$

Де:

- ω – вага зв’язку між двома нейронами (поточного та попереднього шару)
- n – номер поточного шару
- i – ітератор

Нарешті:

$$\omega_{n+1} = \omega_n + \eta \delta_n \omega_{n-1}$$

- Вхідний шар
  Зображення перетворюється у двовимірний масив з нулів та одиниць розмірністю 20x20 для зручнішої візуалізації у консолі. Навчаюча множина відразу формується у цьому вигляді.
  Для того, щоб подати дані на вхід, масив перетворюється з двовимірного у одновимірний.
  У приведеній програмі вхідний шар містить 400 нейронів (20*20)

- Прихований шар
  Складається з 50-ти нейронів

- Вихідний шар
  3 нейрона. 1 – трикутник, 2 – прямокутник, 3 – лінія.

- Коефіцієнт навчання
  Підлаштовується динамічно, початкове значення – 0.1. Під час навчання, якщо значення помилки залишається менше заданого рівня, коефіцієнт трохи зменшується. Це дозволяє точніше налаштувати ваги.

Код написаний на мові Javascript ES6

Перше завантаження триває кілька хвилин, але розпізнавання працює миттєво.

Лістинг

Персептрон:

```
function sigmoid(x) {
    return 1 / (1 + Math.exp(-x))
}

function randomWeight() {
    return (Math.random() - 0.5) * 2
}

class LogovDendrit {
    constructor(from, to) {
        this.weight = randomWeight();
        this.from = from;
        this.to = to;
    }
}

class LogovNeuron {
    constructor() {
        this.dendrits = [];
        this.aggregation = 0;
        this.delta = 0;
        this.out = 0;
        this.connections = [];
    }
}

class LogovLayer {
    constructor(neuronCount) {
        this.neurons = [];

        for (let i = 0; i < neuronCount; i++) {
            this.neurons.push(new LogovNeuron())
        }
    }

    connectTo(layer) {
        layer.neurons.forEach(neuron => {
            this.neurons.forEach(thisNeuron => {
                var dendrit = new LogovDendrit(neuron, thisNeuron);
                neuron.connections.push(dendrit);
                thisNeuron.dendrits.push(dendrit);
            });
        });
    }
}

class LogovPerceptron {
    constructor(iNeurons, hNeurons, oNeurons) {
        this.iLayer = new LogovLayer(iNeurons);
        this.hLayer = new LogovLayer(hNeurons);
        this.oLayer = new LogovLayer(oNeurons);

        this.hLayer.connectTo(this.iLayer);
        this.oLayer.connectTo(this.hLayer);

        this.activation = sigmoid;

        this.learnRate = 0.1;
    }

    getSumForDelta(neuron) {
        var sum = 0;
        neuron.connections.forEach(connection => {
            sum += connection.weight * connection.to.delta;
        });
```

```
        return sum;
    }

    train(pattern, test) {
        this.exec(pattern);

        this.oLayer.neurons.forEach((neuron, i) => {
            neuron.dendrits.forEach(dendrit => {
                neuron.delta = neuron.out * (1 - neuron.out) * (test[i] - neuron.out);
                dendrit.weight = dendrit.weight + this.learnRate * neuron.delta * dendrit.
from.out;
            });
        });
        this.hLayer.neurons.forEach(neuron => {
            neuron.dendrits.forEach(dendrit => {
                neuron.delta = neuron.out * (1 - neuron.out) * this.getSumForDelta(neuron)
;
                dendrit.weight = dendrit.weight + this.learnRate * neuron.delta * dendrit.
from.out;
            });
        });
    }

    exec(pattern) {
        this.iLayer.neurons.forEach((neuron, i) => {
            neuron.out = pattern[i];
            // neuron.out = this.activation(pattern[i]);
            neuron.connections.forEach(connection => {
                connection.to.aggregation += connection.weight * neuron.out;
            })
        });
        this.hLayer.neurons.forEach(neuron => {
            neuron.out = this.activation(neuron.aggregation);
            neuron.aggregation = 0;
            neuron.connections.forEach(connection => {
                connection.to.aggregation += connection.weight * neuron.out;
            })
               });
               var outArr = [];
               this.oLayer.neurons.forEach(neuron => {
                   neuron.out = this.activation(neuron.aggregation);
                   neuron.aggregation = 0;
                   outArr.push(neuron.out);
               });

               return outArr;
           }
       }
```

Генератори фігур:

```
var generateTriangle = function(width, height, type) {
    var x1 = Math.random(),
        y1 = Math.random(),
        x2 = Math.random(),
        y2 = Math.random(),
        x3 = Math.random(),
        y3 = y1,
        arr = [];
    while (!(y1 - y2 > 0.4 &&
        x3 - x1 > 0.4)) {
        x1 = Math.random();
        x2 = Math.random();
        x3 = Math.random();
        y1 = Math.random();
        y2 = Math.random();
        y3 = y1;
    }

    x1 = x1 * width;
    y1 = y1 * height;
    x2 = x2 * width;
    y2 = y2 * height;
    x3 = x3 * width;
    y3 = y3 * height;

    for (let i = 0; i < height; i++) {
        arr.push([]);
        for (let j = 0; j < width; j++) {
            arr[i].push(0);
        }
    }

    for (let x = 0; x < width; x++) {
        var line1Y = ((x2*y1-x1*y2)-(y1-y2)*x)/(x2-x1);
        if (line1Y >= Math.min(y1,y2) && line1Y <= Math.max(y1,y2)) {
            arr[Math.floor(line1Y)][x] = 1
        }
        var line3Y = ((x3*y2-x2*y3)-(y2-y3)*x)/(x3-x2);
        if (line3Y >= Math.min(y2,y3) && line3Y <= Math.max(y2,y3)) {
            arr[Math.floor(line3Y)][x] = 1
        }
    }
    for (let y = 0; y < height; y++) {
        var line1X = ((x2*y1-x1*y2)-(x2-x1)*y)/(y1-y2);
        if (line1X >= Math.min(x1,x2) && line1X <= Math.max(x1,x2)) {
            arr[y][Math.floor(line1X)] = 1
        }
        var line3X = ((x3*y2-x2*y3)-(x3-x2)*y)/(y2-y3);
        if (line3X >= Math.min(x2,x3) && line3X <= Math.max(x2,x3)) {
            arr[y][Math.floor(line3X)] = 1
        }
    }
    for (let i=Math.floor(x1); i <= Math.floor(x3); i++) {
        arr[Math.floor(y1)][i] = 1
    }

    if (type == 'debug') {
        let str = '';
        for (let i = 0; i < height; i++) {
            for (let j = 0; j < width; j++) {
                str += arr[i][j];
            }
            str += '\n';
        }
        console.log(str);
    }

    return arr;
```

```
};

var generateRect = function(width, height, type) {
    var x1 = Math.random(),
        y1 = Math.random(),
        x2 = Math.random(),
        y2 = Math.random(),
        arr = [];
    while (!(Math.abs(x1) - Math.abs(x2) > 0.4 &&
    Math.abs(y1) - Math.abs(y2) > 0.4)) {
        x1 = Math.random();
        y1 = Math.random();
        x2 = Math.random();
        y2 = Math.random();
    }

    x1 = Math.min(Math.round(x1 * width), width - 1);
    y1 = Math.min(Math.round(y1 * height), height - 1);
    x2 = Math.min(Math.round(x2 * width), width - 1);
    y2 = Math.min(Math.round(y2 * height), height - 1);

    for (let i = 0; i < width; i++) {
        arr.push([]);
        for (let j = 0; j < height; j++) {
            arr[i].push(0);
        }
    }
    for (let x = Math.min(x1, x2); x <= Math.max(x1, x2); x++) {
        arr[x][Math.min(y1, y2)] = 1;
        arr[x][Math.max(y1, y2)] = 1;
    }
           for (let y = Math.min(y1, y2); y <= Math.max(y1, y2); y++) {
               arr[Math.min(x1, x2)][y] = 1;
               arr[Math.max(x1, x2)][y] = 1;
           }

           if (type == 'debug') {
               let str = '';
               for (let i = 0; i < height; i++) {
                   for (let j = 0; j < width; j++) {
                       str += arr[j][i];
                   }
                   str += '\n';
               }
               console.log(str);
           }

           return arr;
       };

       var generateLine = function(width, height, type) {
         var x1 = Math.random(),
           y1 = Math.random(),
           x2 = Math.random(),
           y2 = Math.random(),
           arr = [];
         while (!(Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2)) > 0.5)) {
           x1 = Math.random();
           y1 = Math.random();
           x2 = Math.random();
           y2 = Math.random();
         }

         x1 = x1 * width;
         y1 = y1 * height;
         x2 = x2 * width;
         y2 = y2 * height;

         for (let i = 0; i < width; i++) {
           arr.push([]);
           for (let j = 0; j < height; j++) {
```

```
            arr[i].push(0);
          }
        }
        for (let x = 0; x < width; x++) {
          var line1Y = Math.round(((x - x1) * (y2 - y1)) / (x2 - x1) + y1);
          if (line1Y >= Math.min(y1, y2) && line1Y < Math.max(y1, y2)) {
            arr[x][line1Y] = 1
          }
        }
        for (let y = 0; y < height; y++) {
          var line1X = Math.round(((y - y1) * (x2 - x1)) / (y2 - y1) + x1);
          if (line1X >= Math.min(x1, x2) && line1X < Math.max(x1, x2)) {
            arr[line1X][y] = 1
          }
        }

        if (type == 'debug') {
          let str = '';
          for (let i = 0; i < height; i++) {
            for (let j = 0; j < width; j++) {
              str += arr[j][i];
            }
            str += '\n';
          }
          console.log(str);
        }

        return arr;
      };
```

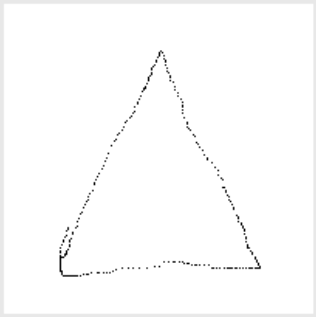
Допоміжні функції:

```
var array2dTo1d = function(arrToConvert) {
  var newArr = [];
  for (var i = 0; i < arrToConvert.length; i++) {
    newArr = newArr.concat(arrToConvert[i]);
  }
  return newArr;
};

function getMaxIndex(arr) {
  var max = 0,
    index = 0;
  arr.forEach((item, i) => {
    if (item > max) {
      max = item;
      index = i;
    }
  });
  return index;
}

function present(pattern) {
  var res = perc.exec(pattern);
  switch (getMaxIndex(res)) {
    case 0:
      return 'triangle';
    case 1:
      return 'rectangle';
    case 2:
      return 'line';
  }
}
```

Головна частина програми:

```
var perc = new LogovPerceptron(400, 100, 3);

var error = 0,
  goodIt = 0,
  frontier = 1;
for (let i = 0; i < 10000; i++) {
  error += perc.train(array2dTo1d(generateTriangle(20, 20)), [1, 0, 0]);
  error += perc.train(array2dTo1d(generateRect(20, 20)), [0, 1, 0]);
  error += perc.train(array2dTo1d(generateLine(20, 20)), [0, 0, 1]);
  if (error < frontier) {
    goodIt++;
    if (goodIt == 15) {
        perc.learnRate /= 1.5;
      frontier /= 1.2;
        console.log(error);
      console.log(i);
    }
  } else {
    goodIt = 0;
  }
  error = 0;
}
```

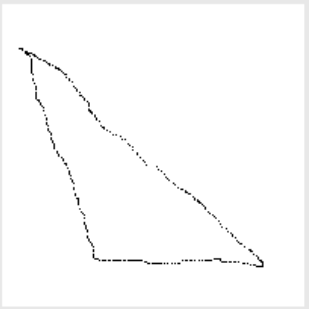
Результат роботи програми



Clear display

run with drawing run with triangle run with rectangle run with line

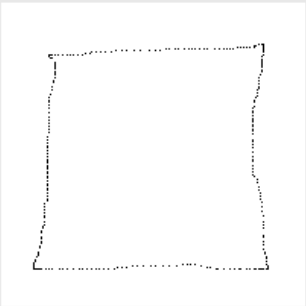
Result:
triangle



Clear display

run with drawing run with triangle run with rectangle run with line

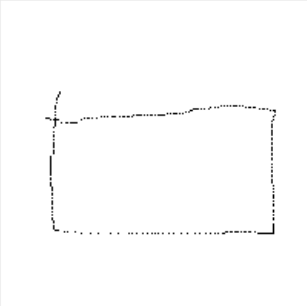
Result:
triangle



Clear display

run with drawing run with triangle run with rectangle run with line

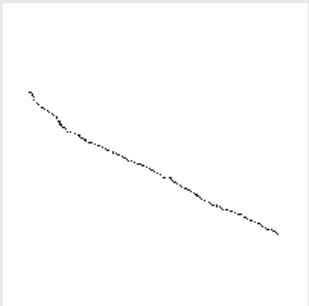
Result:
rectangle



Clear display

run with drawing run with triangle run with rectangle run with line

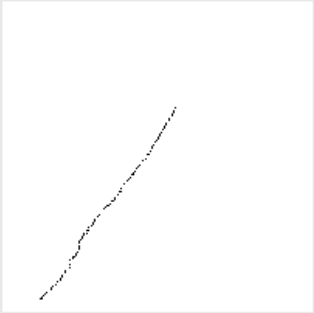
Result:
rectangle



Clear display

run with drawing run with triangle run with rectangle run with line

Result:
line



Clear display

run with drawing run with triangle run with rectangle run with line

Result:
line