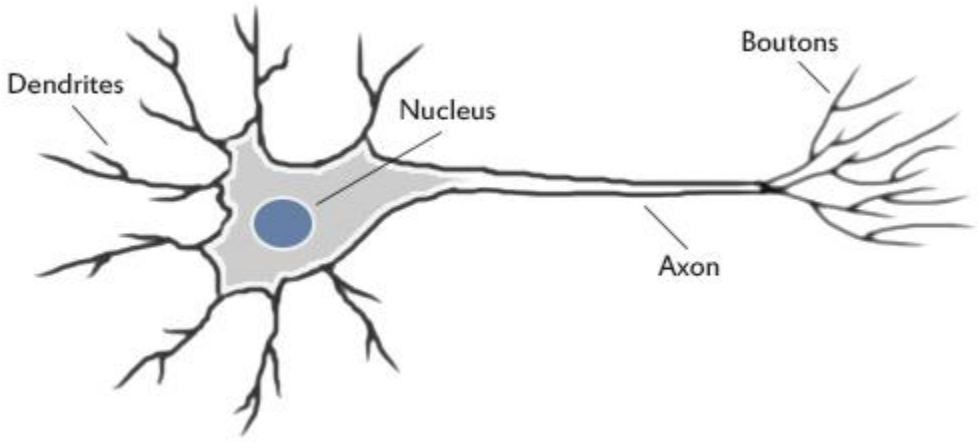
შესავალი ხელოვნურ ნეირონულ ქსელებში

შესავალი

კომპუტერები მოხერხებულია ისეთი პრობლემების გადასაწყვეტად, რომლებიც მათემატიკური ალგორითმებით წარმოიდგინებიან. მაგრამ არსებობენ პრობლემები, რომელთა წარმოდგენა მათემატიკური ალგორითმებით ძნელია, როგორიცაა სახეთა გამოცნობა და ბუნებრივი ენების კომპიუტერული მოდელირება. თუმცა, ამ პრობლემებს ადამიანის ტვინი ადვილად წყვეტს. ხელოვნური ქსელები რეალიზებულია კომპიუტერებზე ისე, რომ ისინი ინფორმაციას ამუშავებენ ადამიანის ტვინის ანალოგიურად, რაც საშულებას იძლევა გადავწყვიტოთ სახეთა გამოცნობისა და სხვა ანალოგიური პრობლემები კომპიუტერის საშუალებით ადამიანის ტვინის მსგავსად.

როგორ მუშაობენ ბიოლოგიური ნეირონები

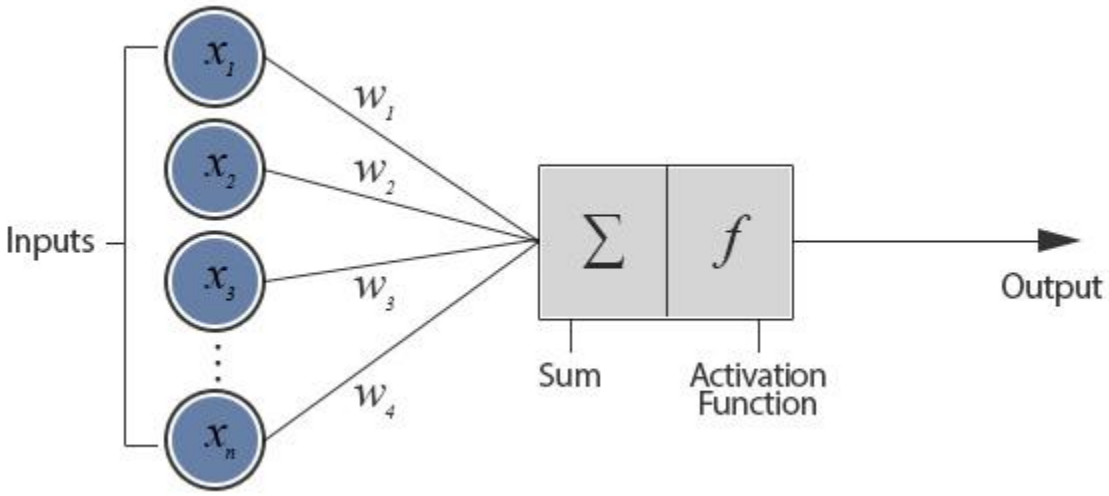
პირველ რიგში გავეცნოთ როგორ გამოიყურებიან ბიოლოგიური ნეირონები.



ჩვენი ტვინი იყენებს ძალიან დიდ რაოდენობა ურთიერთ დაკავშირებულ ნეირონულ ქსელებს, რომ მოდელირება გაუკეთოს სამყაროს, რომელშიაც ჩვენ ვცხოვრობთ. ელექტრონული იმპულსები გადაეცემათ ქსელის შესასვლელს და მათი დამუშავების შედეგად გამოდის გამოსასვლელში. ნეირონი აგროვებს შესავალ ინფორმაციას სტრუქტურის საშუალებით, რომელსაც ჰქვია დენდრიტები (Dendrites). ნეირონი აჯამებს შესასვლელ ინფორმაციას და თუ მისი მნიშვნელობა მეტია ზღვარზე (treshold), მაშინ მოხდება გასროლა. როცა ნეირონი გაისვრის იგი გადასცემს ელექტრონულ იმპულსებს აქსონების (axon) საშუალებით ბუტონებს (boutons). ეს ბუტონები ქსელურად დაკავშირებული არიან ათასობით სხვა ნეირონებთან კავშირით, რომელსაც ჰქვია სინაპსები. ადამიანის ტვინი შედგება დაახლოებით ას მილიარდამდე ნეირონისაგან. თითოეულ მათგანს აქვს ათასამდე სინაპტიკური დაკავშირება.

ხელოვნური ნეირონების მოდელირება

ხელოვნური ნეირონული მოდელები არიან ბიოლოგიურ ნეირონებზე დაფუძნებული გამარტივებული მოდელები. ისინი ახორციელებენ ბიოლოგიური ნეირონების არსებით ფუნქციებს. ჩვენ ვუწოდებთ ასეთ ხელოვნურ ნეირონებს პერსეპტრონებს (perceptrons). ახლა, ვნახოთ როგორ გამოიყურება ასეთი პერსეპტრონი:



როგორც ნაჩვენებია დიაგრამაზე, პერსეპტრონს აქვს მრავალი შესასვლელი და თითოეულ შესასვლელს აქვს თავისი წონა. წონები ამცირებენ ან ზრდიან შესასვლელი სიგნალის მნიშვნელობას. მაგალითად, თუ შესასვლელის მნიშვნელობა არის ერთი და წონა 0.2., მაშინ მნიშვნელობა იქნება 0.2. ეს შეწონილი შესასვლელები შემდეგ იკრიბება და გადაეცემა აქტივაციის ფუნქციას. იგი გამოიყენება, რომ გადააქციოს შესასვლელი უფრო სასარგებლო გამოსასვლელად. არსებობენ სხვადასხვა ტიპის აქტივაციის ფუნქციები, მაგრამ ყველაზე მარტივია ნაბიჯი ფუნქცია. ამ ფუნქციას გამოაქვს 1, თუ შესასვლელი მეტია ზღვარზე და 0 წინააღმდეგ შემთხვევაში.

განვიხილოთ მაგალითი:

შესასვლელი **1** (X_1) **= 0.6**

შესასვლელი **2** (X_2) **= 1.0**

წონა 1 (W_1) = 0.5

წონა2 2 (W_2) = 0.8

ზღვარი = 1.0

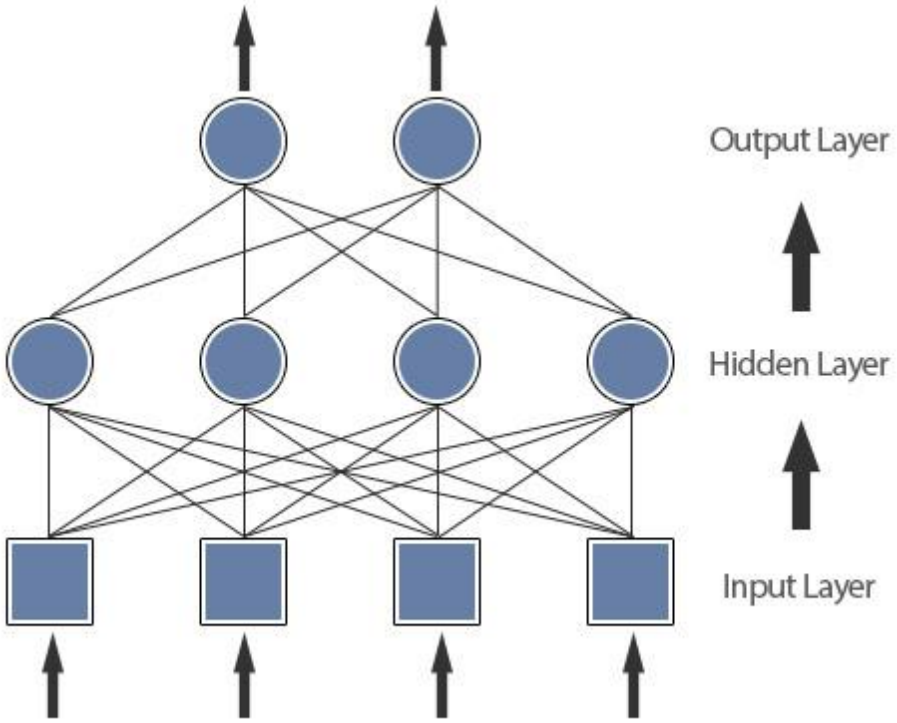
შემდეგ ვასრულებთ მოქმედებას:

$X_1W_1 + X_2W_2 = (0.6 \times 0.5) + (1 \times 0.8) = 1.1$

რადგან შედეგი მეტია 1-ზე (ზღვარზე) , პერსეპტრონი აენთება ე.ი. გამოსასვლელია 1.

ხელოვნური ნეირონული ქსელების რეალიზაცია

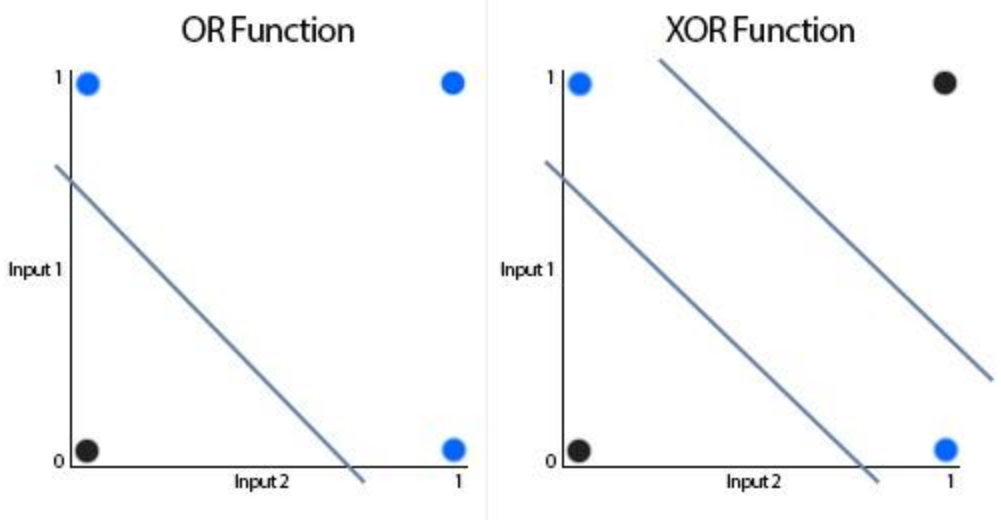
ახლა ვნახოთ რა არის ხელოვნური ნეირონული ქსელი და როგორ ამუშავებს იგი ინფორმაციას. ჩვენ ვაპირებთ გავეცნოთ feedforward ქსელებს და როგორ არიან ისინი დაკავშირებული პერსეპტრონებთან. მაგრამ მანამდე, გავეცნოთ რა არის feedforward ქსელი.



როგორც დიაგრამიდან ჩანს, იგი შედგება 3 შრისაგან: შესასვლელი შრე; ფარული შრე და გამოსასვლელი შრე. შესასვლელი შრის ყოველი კვანძიდან ინფორმაცია გადაეცემა ფარული შრის ყოველ კვანძს და იქიდან გამოსასვლელი შრის ყოველ კვანძს. შეიძლება არსებობდეს მრავალი ფარული შრეები და შეიძლება, ასევე, სიგნალები მოძრაობდნენ ორივე მიმართულებით. ჩვენს შემთხვევაში სიგნალები მოძრაობენ ერთი მიმართულებით შესასვლელიდან გამოსასვლელის მიმართულებით, ამიტომაც მას ეწოდება feedforvard(წინკვება) ქსელი. ქსელებს, როცა სიგნალები მოძრაობენ ორივე მიმართულებით ეწოდებათ feedback(უკუკვება) ქსელები.

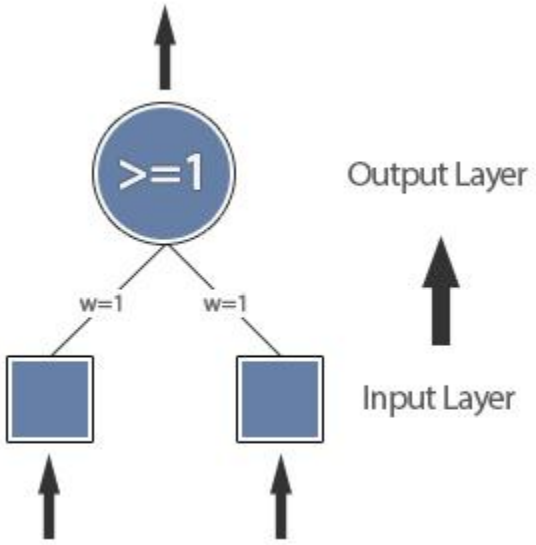
წრფივი განცალკევადობა

იმისათვის, რომ ვაჩვენოთ, რატომ არის საჭირო ფარული შრეები, განვიხილოთ მაგალითები:

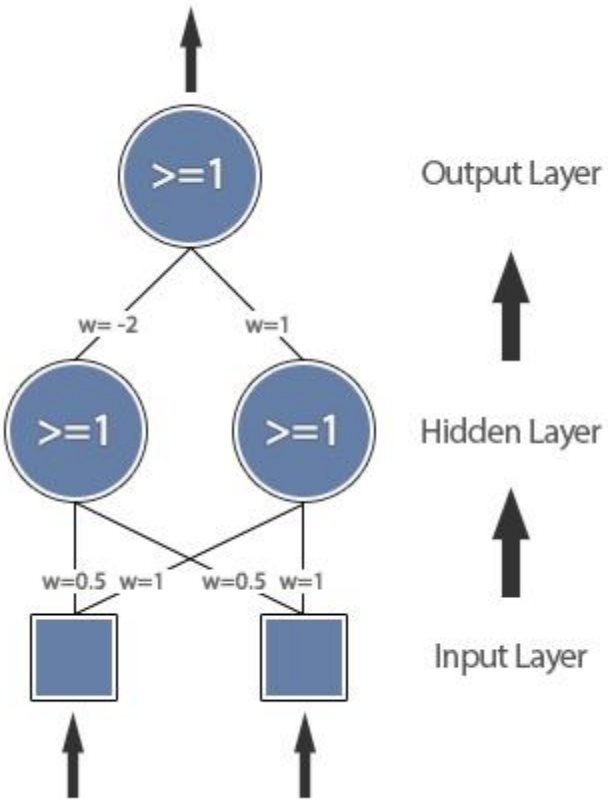


.

შევნიშნოთ, რომ OR ფუნქცია(ტოლია ნულის, როცა ორივე შესასვლელი ნულია. სხვა შემთხვევებში ერთია) გრაფზე შეიძლება განცალკევდეს ერთი სწორი ხაზით, მაშინ როცა XOR ფუნქციას(ტოლია ნულის, როცა ორივე შესასვლელი ნულია ან ერთია, სხვა შემთხვევებში ერთია) სჭირდება ორი სწორი ხაზი (შავი ბურთულები აღნიშნავენ ნულს და ლურჯი ბურთულები აღნიშნავენ ერთს). ეს ნიშნავს, რომ OR ფუნქცია წრფივად განცალკევებადია და შეიძლება მოდელირებულ იქნეს ფარული შრის გარეშე. მაგალითად, ერთი პერსეპტრონით:



მაგრამ XOR ფუნქციის მოდელირებისათვის გვჭირდება დამატებითი შრე:



უმეტეს შემთხვევებში, გვჭირდება ერთი ან ორი შრე, მაგრამ საჭიროა ექსპერიმენტები, რომ ვიპოვოთ კვანძების ოპტიმალური რაოდენობა ყოველ კონკრეტულ შემთხვევაში.

სწავლება ხელოვნურ ნეირონულ ქსელებში

ახლა, ჩვენ განვიხილავთ, თუ როგორ შეიძლება შემოვიტანოთ სწავლება ხელოვნურ ნეირონულ ქსელებში. რატომ არის მნიშვნელოვანი სწავლება ნეირონულ ქსელებში და რა ტიპის სწავლებები არსებობენ. პირველ რიგში, განვიხილავთ ერთშრიანი პერსეპტრონების სწავლებას პერსეპტრონის სწავლების წესით. სწავლებას განვიხილავთ, როგორც საშუალებას, რომ კომპიუტერმა უკეთესად შეასრულოს გადასაწყვეტი ამოცანები. სწავლების მეთოდები განსაკუთრებით სასარგებლოა მაშინ, როცა პრობლემის გადაწყვეტა ალგორითმებით ძალზე რთულია და სწავლების მეთოდებით მათი გადაწყვეტა უფრო იოლია.

სწავლების ტიპები

არსებობს სხვადასხვა ტიპის ალგორითმები, რომლებიც შეიძლება გამოყენებულ იქნეს ხელოვნური ნეირონული ქსელების ვარჯიშის დროს. ყოველ მათგანს გააჩნია თავისი უპირატესობები და ნაკლოვანებები. სწავლების პროცესი არის წონების ცვლილების შედეგი გარკვეული სახეობის სწავლების ალგორითმით. მიზანი არის ვიპოვოთ ისეთი წონების მატრიცების სიმრავლე, რომელიც გადასახავს შესასვლელს სწორ გამოსასვლელში. სწავლების

ტიპი, რომელზედაც ჩვენ გავამახვილებთ ყურადღებას, არის მართული სწავლება. მაგრამ, სანამ მასზე გადავიდოდეთ, განვიხილოთ სამი უმთავრესი სწავლების პარადიგმები.

მართული სწავლება

სწავლების ტიპი ეკუთვნის ამ კატეგორიას, თუ სწავლების დროს შესასვლელით გათვალისწინებულია სასურველი გამოსასვლელი. მიღებული გამოსასვლელით ჩვენ შეგვიძლია შევაფასოთ შეცდომა და მის მიხედვით შევასწოროთ წონები.

უმართავი სწავლება

ნეირონული ქსელის ასეთი პარადიგმის დროს, მოცემულია მხოლოდ შესასვლელები და ქსელმა თვითონ უნდა იპოვოს სწორი გამოსავლელი. ასეთი მიდგომა ძირითადად გამოიყენება ინფორმაციის მოძიების დროს მონაცემთა ბაზებში. ამისათვის არსებობს სხვადასხვა ალგორითმები.

გაძლიერებული სწავლება

გაძლიერებული სწავლება ჰგავს მართულ სწავლებას იმაში, რომ ხდება ქსელის დასაჩუქრება იმის მიხედვით, თუ რამდენად წარმატებული იყო. ეს მიდგომა ჰგავს ცხოველების გაწრთვნის პროცესს.

მართული სწავლების რეალიზაცია

იდეა მდგომარეობს იმაში, რომ მოვამარაგოთ ქსელის შესასვლელი და გამოსასვლელი მაგალითებით და ქსელმა უნდა იპოვოს ფუნქცია, რომელიც მოცემულ შესასვლელს გადაიყვანს შესატყვის გამოსასვლელში. როცა, ქსელი სწორად გადაიყვანს საკმარის რაოდენობა შესასვლელებს შესატყვის გამოსასვლელებში, მაშინ ქსელის ვარჯიში მთავრდება. ჩვენ უნდა მივცეთ ახალი შესასვლელი ქსელს და მან უნდა მოგვცეს სწორი გამოსასვლელი. ახლა, ჩვენ განვიხილავთ ერთშრიანი პერსეპტრონების სწავლებას.

პერსეპტრონის სწავლების წესი

პერსეპტრონის სწალების წესი ცდილობს გაარკვიოს რა იყო მცდარი და ღებულობს ზომებს, რომ არ მოხდეს იგივე შეცდომა მომავალში. პირველ რიგში, ვიღებთ აქტუალურ გამოსასვლელს და ვადარებთ სწორ გამოსასვლელს. თუ არ ემთხვევა, მაშინ ვცდილობთ შევასწოროთ წონები, რომ დაემთხვეს გამოსასვლელი სწორ გამოსასვლელს. განვიხილოთ კონკრეტული მაგალითი. პირველ რიგში ჩვენ გვჭირდება გამოვითვალოთ პერსეპტრონის გამოსასვლელი მისი ყოველი გამოსასვლელი კვანძისათვის:

output = f(input1 × weight1 + input2 × weight2 + ...)

ახლა, უნდა შევადაროთ გამოსასვლელი მიზან გამოსასვლელს, რომ ვიპოვოთ შეცდომა:

error = target output - output

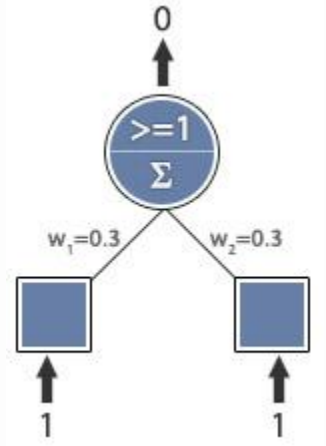
ახლა, უნდა გამოვიყენოთ პერსეპტრონის შეცდომა, რომ დავაზუსტოთ წონები:

weight change = learning rate × error × input

ჩვენ უნდა მოვახდინოთ მცირე ცვლილებები, რომ არ გამოგვრჩეს სწორი ამონახსნი:

weight change = learning rate × (target output - actual output) × input

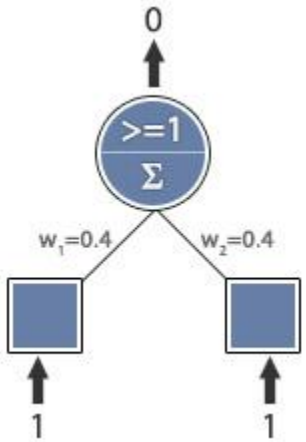
ახლა, ვნახოთ ლოგიკური AND მაგალითზე, თუ როგორ მუშაობს ასეთი მიდგომა:



სწავლების სიდიდე = 0.1
მოსალოდნელი გამოსასვლელი = 1
რეალური გამოსასვლელი = 0
შეცდომა = 1

წონის განახლება:
$w_i = r E x + w_i$
$w_1 = 0.1 \times 1 \times 1 + w_1$
$w_2 = 0.1 \times 1 \times 1 + w_2$

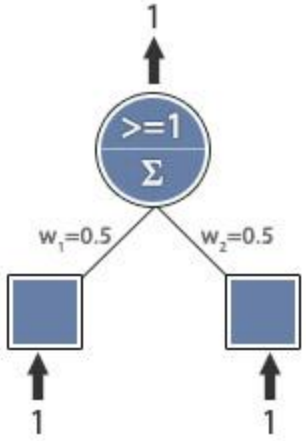
ახალი წონები:
$w_1 = 0.4$
$w_2 = 0.4$



სწავლების სიდიდე = 0.1
მოსალოდნელი გამოსასვლელი = 1
რეალური გამოსასვლელი = 0
შეცდომა = 1

წონის განახლება:
$w_i = r E x + w_i$
$w_1 = 0.1 \times 1 \times 1 + w_1$
———————————————$w_2 = 0.1 \times 1 \times 1 + w_2$

ახალი წონები:
$w_1 = 0.5$
$w_2 = 0.5$



სწავლების სიდიდე = 0.1
მოსალოდნელი გამოსასვლელი = 1
რეალური გამოსასვლელი = 1
შეცდომა = 0

სწავლება დასრულდა.

პერსეპტრონის სწავლების წესის რეალიზაცია

პერსეპტრონის სწავლების სრულად შესწავლის მიზნით, განვიხილოთ java ენაზე დაწერილი პროგრამა:
პირველ რიგში განვსაზღვროთ ზღვარი, სწავლების სიდიდე და წონები.

```
double threshold = 1;
double learningRate = 0.1;
double[] weights = {0.0, 0.0};
```

შემდეგ, ჩვენ გვჭირდება პერსეპტრონის საწვრთნელი მონაცემები. ამ მაგალითში პერსეპტრონს ვასწავლით AND ფუნქციას.

```
// AND function Training data
int[][][] trainingData = {
    {{0, 0}, {0}},
    {{0, 1}, {0}},
    {{1, 0}, {0}},
    {{1, 1}, {1}},
};
```

ჩვენ გვჭირდება ციკლი, რომელიც დაამთავრებს მუშაობას, როცა პერსეპტრონი უშეცდომოდ შეასრულებს ყველა სავარჯიშო მონაცემს. ასევე, გვჭირდება მეორე ციკლი ყოველი შესასვლელისათვის.

```
// Start training loop
while(true){
    int errorCount = 0;
    // Loop over training data
    for(int i=0; i < trainingData.length; i++){
        System.out.println("Starting weights: " + Arrays.toString(weights));
    }
```

შემდეგ, უნდა გამოვითვალოთ შესასვლელების შეწონილი ჯამები და გამოვიტანოთ გამოსასვლელი.

```
// Calculate weighted sum of inputs
double weightedSum = 0;
for(int ii=0; ii < trainingData[i][0].length; ii++) {
    weightedSum += trainingData[i][0][ii] * weights[ii];
}

// Calculate output
int output = 0;
if(threshold <= weightedSum){
    output = 1;
}

System.out.println("Target output: " + trainingData[i][1][0] + ", "
    + "Actual Output: " + output);
```

შემდეგი ნაბიჯია შეცდომის გამოთვლა და წონების დაზუსტება:

```
// Calculate error
int error = trainingData[i][1][0] - output;

// Increase error count for incorrect output
if(error != 0){
    errorCount++;
}

// Update weights
for(int ii=0; ii < trainingData[i][0].length; ii++) {
    weights[ii] += learningRate * error * trainingData[i][0][ii];
}

System.out.println("New weights: " + Arrays.toString(weights));
System.out.println();
```

ბოლოს, თუ ვიპოვეთ ამოხსნა დავამთავროთ ციკლი.

```
    // If there are no errors, stop
    if(errorCount == 0){
        System.out.println("Final weights: " + Arrays.toString(weights));
        System.exit(0);
    }
}
```

ახლა შევაერთოთ ყველაფერი ერთად:

```
package perceptron;
import java.util.Arrays;

public class PerceptronLearningRule {
    public static void main(String args[]){
        double threshold = 1;
        double learningRate = 0.1;
        // Init weights
        double[] weights = {0.0, 0.0};

        // AND function Training data
        int[][][] trainingData = {
            {{0, 0}, {0}},
            {{0, 1}, {0}},
            {{1, 0}, {0}},
            {{1, 1}, {1}},
        };

        // Start training loop
        while(true){
            int errorCount = 0;
            // Loop over training data
            for(int i=0; i < trainingData.length; i++){
                System.out.println("Starting weights: " + Arrays.toString(weights));
                // Calculate weighted input
                double weightedSum = 0;
                for(int ii=0; ii < trainingData[i][0].length; ii++) {
                    weightedSum += trainingData[i][0][ii] * weights[ii];
                }

                // Calculate output
                int output = 0;
                if(threshold <= weightedSum){
                    output = 1;
                }

                System.out.println("Target output: " + trainingData[i][1][0] + ", "
                        + "Actual Output: " + output);

                // Calculate error
                int error = trainingData[i][1][0] - output;

                // Increase error count for incorrect output
                if(error != 0){
                    errorCount++;
                }
```

```
                // Update weights
                for(int ii=0; ii < trainingData[i][0].length; ii++) {
                    weights[ii] += learningRate * error * trainingData[i][0][ii];
                }

                System.out.println("New weights: " + Arrays.toString(weights));
                System.out.println();
            }

            // If there are no errors, stop
            if(errorCount == 0){
                System.out.println("Final weights: " + Arrays.toString(weights));
                System.exit(0);
            }
        }
    }
}
```

Bias(გადახრის) ერთეულები

წინა მაგალითში ზღვარი იყო 1, რაც ნიშნავდა იმას, რომ შეწონილი ჯამები უნდა ყოფილიყო 1 ან მეტი, რომ გამოსასვლელი ყოფილიყო 1. ის საკმარისი იყო AND ფუნქციისათვის, მაგრამ, XOR ფუნქციისათვის გამოსასვლელი უნდა იყოს 1, როცა ორივე შესასვლელი ნულია. ეს ნიშნავს იმას, რომ თუ ზღვარი ერთია არ არსებობს წონების კომბინაცია, რომელიც იქნება ჭეშმარიტი:

$$x_1 = 0$$
$$x_2 = 0$$
$$1 <= x_1w_1 + x_2w_2$$

ეს შეიძლება მარტივად გასწორდეს გადახრის ერთეულით. გადახრის ერთეული არის ნეირონი მუდმივი გამოსასვლელით, ჩვეულებრივ 1. გადახრის ერთეულები არიან შეწონილები მსგავსად სხვა ერთეულებისა, განსხვავებაა მხოლოდ ის, რომ მათ გამოაქვთ 1 მიუხედავად იმისა, თუ როგორია მათი შესასვლელი წინა შრიდან. ამიტომაც, მათ ჰქვიათ გადახრა. გადახრის შესასვლელები ეფექტურად რთავენ ნებას ნეირონს შეისწავლოს ზღვრული მნიშვნელობა. განვიხილოთ წინა მაგალითი, რომელსაც დამატებული აქვს გადახრის შესასვლელი x_0.

$$x_0 = 1$$
$$x_1 = 0$$
$$x_2 = 0$$
$$1 <= x_0w_0 + x_1w_1 + x_2w_2$$

ახლა, ჩვენ შეგვიძლია დავაკმაყოფილოთ განტოლება. ქვემოთ მოცემულია შესატყვისი java კოდი:

```
// Init weights
double[] weights = {0.0, 0.0, 0.0};

// NOR function training data
int[][][] trainingData = {
    {{1, 0, 0}, {1}},
    {{1, 0, 1}, {0}},
    {{1, 1, 0}, {0}},
    {{1, 1, 1}, {0}},
};
```

Related Articles

Autobots and Beyond
Workplace Automation
Creating a genetic algorithm for beginners
Simulated Annealing for beginners
Solving the Traveling Salesman Problem Using Google Maps and Genetic Algorithms