ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ანზორ დავარაშვილი

ნეირონული ქსელის იმპლემენტაცია და მისი საშუალებით მკერდის სიმსივნის აღმოჩენა

სამაგისტრო პროგრამა: კომპიუტერული მეცნიერება

ნაშრომი შესრულებულია კომპიუტერული მეცნიერების

მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: რევაზ ქურდიანი

თბილისი 2018

ანოტაცია

ხელოვნური ინტელექტი აქტიურად ცვლის ჩვენს ყოველდღიურ ცხოვრებას.ის ფართოდ გამოიყენება ბიზნესში.მეცნიერებაში,მედიცინაში,განათლებასა და სხვა სფეროებში.მისი მნიშვნელობა ყოველდღიურად იზრდება. ამ ნაშრომში შევეხები ხელოვნური ინტელექტის ერთერთ ყველაზე საინტერესო და პოპულარულ მიმართულებას: **ხელოვნურ ნეირონულ ქსელებს.**

სამაგისტრო ნაშრომში განხილულია ნეირონული ქსელის იმპლემენტაციის დეტალები, ნეირონული ქსელის კონცეფციები და ნეირონის ძირითადი სტრუქტურები, აქტივაციის ფუნქციები, წონა, და დასწავლის ალგორითმები. გარდა ამისა, ნაშრომში წარმოდგენილია საბაზისო ნეირონული ქსელის შექმნის პროცესი Java ენის გამოყენებით დასაწყისიდან დასრულებამდე.

ნაშრომში განიხილება ნეირონული ქსელების დასწავლის პროცესის დეტალები. წარმოდგენილი იქნება სასარგებლო ცნებები, როგორიცაა დასწავლა, ტესტირება და ვალიდაცია.ვაჩვენებ როგორ შეიძლება განახორციელდეს დასწავლის და ვალიდაციის ალგორითმები. ასევე განვიხილავთ ცდომილების შეფასების მეთოდებს.

ასევე შევეხებით პერცეპტონებს, დამკვირვებლიან სწავლას და მათ ძირითად თვისებებს, თქვენ გაეცანით პერცეპტონებს და ზედამხედველობითი სწავლების თვისებებს. წარმოგიდგენთ დასწავლის ალგორითმს ამ ტიპის ნეირონულ ქსელებში.

მკითხველი შეისწავლის, თუ როგორ უნდა განახორციელონ ეს ფუნქციები Java- ში.
ისინი ასევე შეისწავლიან, თუ როგორ უნდა დაამუშაონ მონაცემები მათი ნეირონული ქსელისთვის მიწოდებამდე.
და ბოლოს წარმოვადგენ ამ ქსელის პრაქტიკული გამოყენების მაგალითს.განხილული იქნება კლასიფიკაციის ამოცანა,რომელიც არის დამკვირვებლიანი სწავლების მაგალითი.ქსელის საშუალებით ვახდენთ დაავადებათა დიაგნოსტირებას,
პაციენტის ჩანაწერების მონაცემების გამოყენებით, ნეირონული ქსელი აგებულია როგორც საექსპერტო სისტემას, რომესაც შეუძლია დასვას მკერდის სიმსივნის დიაგნოზი პაციენტების სიმპტომებიდან გამომდინარე.

Abstract

Artificial intelligence is changing how we conduct our daily lives.It's widely ised in bussiness,science,educations,medicine and etc.It's gaining more and more popularity every day.
In theis thesis I'll discuss on of the most popular and interesting field of AI : **Neural Networks**.

The thesis introduces neural networks concepts and shows the very basic neuron structures (Single Layer Perceptrons, Adaline), activation functions, weights,
and learning algorithms. Besides, this work shows the process of creating basic neural networks in Java from start to finish.

The thesis presents the details of the neural network learning process. Useful concepts, such as training, test, and validation are introduced. We show how to implement training and validation algorithms. This work also shows methods for error evaluation.
I'll discuss Perceptrons and Supervised Learning, you will get acquainted with perceptrons and the supervised learning features. I present training algorithms for these types of Neural Network. The reader will learn how to implement these features in Java.They also learn how to apply preprocessing before presenting data to neural networks.

And lastly I'll introduce a classification problem, which is also
encompassed in supervised learning. Using patient records data, a neural network is built to act as an expert system, being capable of giving a breast cancer diagnosis based on patients, symptoms.

სარჩევი

შესავალი

ხელოვნური ინტელექტის სისტემები განსაკუთრებით სწრაფად განვითარდა ბოლო დროს. შეიქმნა რამდენიმე ძალიან პოპულარული ციფრული ასისტენტი: Google Assitant, Siri, Cortana, Alexa. მათ შეუძლიათ ახალი ამბების წაკითხვა, ფოსტის შემოწმება, საოჯახო აპარატურის (ტელევიზორის, კონდენციონერის, ჟალუზების...) მართვა.. ასისტენტების შესაძლებლობები სულ უფრო იზრდება და მალე საოცრად მრავალმხრივი დამხმარეები, და შეიძლება მეგობრებიც, გვეყოლება. უკვე არსებობს მარტოხელა მოხუცთა დამხმარეები, რომელთაც შეუძლიათ წამლის დალევის შეხსენება, წნევის გაზომვაში დახმარება და, საჭიროების შემთხვევაში, სასწრაფოს გამოძახებაც. ელექტრონული მიმტანები მოგემსახურებათ კაფეებში, კონსულტაციას გაგიწევთ ბანკებში იპოთეკური სესხისა და ჯანმრთელობის დაზღვევის შესახებ. გამოჩდნენ თვითმავალი ავტომობილებიც. ისინი გაცილებით უსაფრთხოდ მოძრაობენ ვიდრე ამას ადამიანები ვახერხებთ. მომავალში, უნდა ვივარაუდოთ, რომ ადამიანის საჭესთან დაჯდომა საერთოდ აღარ იქნება საჭირო. ხელოვნური ინტელექტის ასეთი ნახტომისებური განვითარება არის „ნეირონული ქსელების“ მძლავრი მოდელების შექმნის დამსახურება. ხელოვნურ ნეირონულ ქსელებზე დაფუძნებული ტექნოლოგიებით გახდა შესაძლებელი ბიონიკური თვალის და კიდურების პროთეზების დამზადება. ხელოვნურად დამზადებული ბიონიკური თვალით ადამიანი ხედავს, ხოლო თანამედროვე პროთეზები იმდენად სრულყოფილია, რომ მათ ადამიანი თითქმის საკუთარი სხეულის ნაწილად აღიქვამს, გრძნობს შეხებას, ტემპერატურას, ტკივილსაც კი.კიდევ მრავალი სფეროს ჩამოთვლა შეიძლება, სადაც პროგრესის მამოძრავებელი გახდა ხელოვნური ნეირონული ქსელი. მიმდინარე ნაშრომი ხელოვნურ ნეირონულ ქსელებს და მედიცინაში,კერძოდ დაავადებათა დიაგნოსტირებაში მის პრაქტიკულ გამოყენებას ეძღვნება. განხილულია ნეირონული ქსელი და მისი იმპლემენტაციის დეტალები კოდის ფრაგმენტებთან ერთად.გარჩეულია მათი სწავლების მეთოდები . ნაშრომი მაქსიმალურად სრულად წარმოგვიდგენს ხელოვნური ინტელექტის ლამაზ სამყაროს და მასში ნეირონული ქსელების გამორჩეულ როლს ცხადად დაგვანახებს.

თავი 1. ნეირონი და ნეირონული ქსელი

კომპიუტერის გამოგონებამ ადამიანის ცხოვრება რადიკალურად შეცვალა. რიცხვებზე ოპერაციების გარდა, კომპიუტერმა სწრაფად შეითავსა თითქმის ყველა სფეროსთვის საჭირო ფუნქცია. კომპიუტერის სწრაფქმედების გაზრდამ ადამიანს იმდენად მძლავრი ხელსაწყო მოუვლინა მოულოდნელად, რომ მისი რესურსის სრულად გამოყენებისთვის ახლებური ალგორითმების მოფიქრება დღემდე გრძელდება.

მრავალი გამოწვევა არსებობდა კომპიუტერების შემოღების დღიდან საინფორმაციო ტექნოლოგიებში. ერთ-ერთ მიუღწეველ მიზანს გამოსახულების ამოცნობა წარმოადგენდა. ვერ ხერხდებოდა ელემენტარული ობიექტების ამოცნობაც კი. ამ ამოცანას თუ ძველებური მიდგომით განვიხილავთ, უნდა მოვახდინოთ ამოსაცნობი ობიექტის ზუსტი აღწერა და ამ გზით შევეცადოთ მის მოძიებას. ზუსტი აღწერის შედგენა არ არის მარტივი. ვთქვათ, გვინდა მანქანის აღწერის შედგენა. მანქანა ბევრნაირი არსებობს და ყოველი ახალი სურათი შეიძლება რადიკალურად განსხვავდებოდეს წინასგან.მიუხედავად ასეთი რთული ამოცანისა, ადამიანი ადვილად ახერხებს ამოცნობას პრაქტიკულად ნებისმიერ სურათზე, მათ შორის სქემატურ ნახატებზეც კი, რომლის მსგავსი მანქანაც ბუნებაში საერთოდ არ მოიძებნება.ყოველივე ზემოთ თქმულიდან გამომდინარე, ცხადია, რომ ახლებური მიდგომების შემუშავება იყო აუცილებელი.

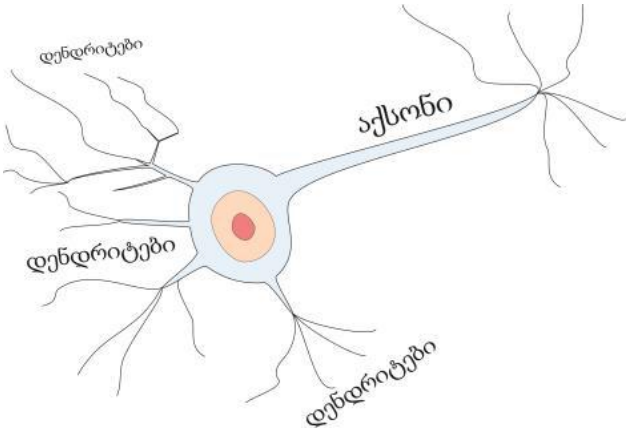
ახალი ალგორითმების შესამუშავებლად ადამიანებმა ისევ დედა ბუნებას მიმართეს. გადაწყვიტეს, გამოეკვლიათ ტვინის მუშაობა და მისი მსგავსი მეთოდებით ეწარმოებინათ ამოცნობა. ადამიანის ტვინი, ერთ-ერთ ურთულეს მექანიზმს წარმოადგენს, სწორედ მისი საშუალებით მივაღწიეთ განვითარების ასეთ მაღალ დონეს და, ცხადია, ინფორმაციის დამუშავების ამოცანებში, მისი ფუნქციონირების გამოყენება დიდი გარღვევის საწინდარი უნდა გამხდარიყო.ასეც მოხდა, ადამიანის ტვინის სტრუქტურის მსგავსი მოწყობილობების გამოჩენამ ტექნიკური პროგრესი ახალ საფეხურზე აიყვანა.

ცნობილია, რომ ადამიანის ტვინი შედგება სპეციფიკური ტიპის უჯრედებისგან - ნეირონებისგან. ნეირონები ერთმანეთთან არის დაკავშირებული. გარკვეულწილად გამოკვლეულია უჯრედებს შორის კავშირის მექანიზმიც. საოცარია, მაგრამ თითქმის იდენტური უჯრედების ნაკრები ახერხებს ანალიტიკურ აზროვნებას და აფორმირებს მეხსიერებასაც. ინფორმაცია ნეირონებს გადაეცემა გარე რეცეპტორებიდან (გრძნობის ორგანოებიდან: სმენა, მხედველობა, ყნოსვა, გემო, ვესტიბულარული აპარატი, ...) და სხვა ნეირონებიდან. ნეირონების მიერ დამუშავებული სიგნალების მიხედვით ხდება ადამიანის აქტივობა .

ძალიან საინტერესო იქნება თუ დავამზადებთ ტვინის ელექტრონულ მოდელს და მისი საშუალებით ვცდით აზროვნებისთვის დამახასიათებელი თვისებების მოდელირებას.სწორედ ასეთი მოდელების შესწავლის კვალობაზე ჩამოყალიბდა მეცნიერების მიმართულება - ნეირონული ქსელები. მიღებულია პრაქტიკული შედეგებიც - ნეირონული ქსელების ბაზაზე აგებული მოწყობილობები უკვე ფართოდ ინერგება ყოფა-ცხოვრებაში. აღსანიშნავია, რომ ადამიანის აზროვნებასთან მიახლოება, ამ სისტემების მიერ, ჯერ ადრეულ სტადიაზეა.

1.1 ბუნებრივი ნეირონი და ნეირონული ქსელი

როგორც ზემოთ აღვნიშნეთ, ბუნებრივი ნეირონი არის ტვინის უჯრედი. ნეირონს სხვა ნეირონებთან აქვს კავშირი. ტვინის ფუნქციონირებისას ერთი ნეირონიდან მეორეს გარკვეული სიგნალები გადაეცემა.ნეირონს აქვს მრავალი არხი, საიდანაც სხვა ნეირონების სიგნალებს იღებს და აქვს ერთი არხი, საიდანაც საკუთარ სიგნალს გადასცემს სხვებს.ნეირონები წარმოადგენს არა მხოლოდ აზროვნების საშუალებას - ადამიანის მეხსიერებასაც.ადამიანის ტვინი 100 მილიარდამდე ნეირონისგან შედგება. ნეირონებს შორის დაახლოებით 60 ტრილიონი კავშირია.დენდრიტების საშუალებით ნეირონში შედის სიგნალები, ხოლო აქსონის საშუალებით ნეირონიდან გამომავალი სიგნალი სხვა ნეირონებს მიეწოდება. სიგნალების გაცვლა-გამოცვლა ელექტროქიმიური ფორმით ხდება. დამუხტული იონები დამაკავშირებელ არხებში გადაადგილდებიან და ნეირონის ბირთვში ხვდებიან. თუ ბირთვში სხვადასხვა დენდრიტიდან მოხვედრილი იონების ჯამური მუხტი გარკვეულ ზღვარს გადააჭარბებს, დამუხტული იონები ზვავისებურად გამოიტყორცნება ნეირონის აქსონიდან და სხვა ნეირონებს გადაეცემა. ნეირონის ფუნქციონირების ეს სქემა გამოყენებული საინფორმაციო ტექნოლოგიებში ხელოვნური ნეირონული ქსელების დამზადებისას.



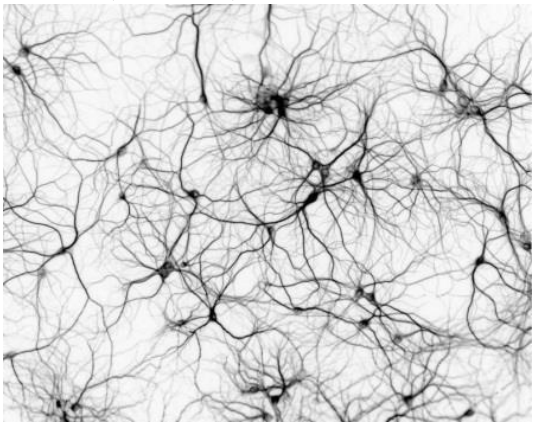
სურათი 1.ბუნებრივი ნეირონი

საოცარია, რომ ასეთი ერთნაირი და მარტივი კომპონენტებისგან შედგება ჩვენი ტვინი. მაღალინტელექტუალური ქმედებების შესაძლებლობას, სწორედ, ეს ერთი შეხედვით მარტივი, მოწყობილობა იძლევა.

ტვინში გარე სენსორებიდან შედის სიგნალები, ეს სიგნალები თავის ტვინის ნეირონებს მიეწოდება. ნეირონებში გარედან შემოსული სიგნალები გადამუშავდება. საბოლოოდ, ნეირონების მიერ ფორმირებული სიგნალი ნერვული დაბოლოებებით გადაეცემა სხეულის სხვადასხვა ორგანოს კონკრეტული მოქმედებების განსახორციელებლად.

ადამიანის სხეული არის დაქსელილი ნერვული არხებით. ზოგიერთი არხით თავის ტვინის ნეირონები იღებენ გარედან შემოსულ ინფორმაციას, ხოლო ზოგი სხეულის მოქმედების განსახორციელებლად არის განკუთვნილი.

ამ თვალსაზრისით თუ განვიხილავთ, ადამიანი არის ძალიან აკურატულად აწყობილი მექანიზმი. ნეიროქირურგებს შეუძლიათ ნერვული დაბოლოებების მიერთება ელექტრულ აპარატურასთან, რაც ხელოვნური პროთეზების დამზადების საშუალებას იძლევა. ასეთი პროთეზი ბუნებრივი კიდურის მსგავსად იმართება ადამიანის ტვინის მიერ და, შესაბამისი სენსორების არსებობის შემთხვევაში, ტვინს კიდურიდან შეგრძნებებიც ეგზავნება.


სურათი 2 ნეირონული ქსელი

თავის ტვინის მიერ ინფორმაციის გადამუშავების და სიგნალების ფორმირების პროცესში ძალიან ბევრი ნეირონი მონაწილეობს. ასევე მრავალი ნეირონია ჩართული ინფორმაციის შენახვის პროცესშიც. რამდენიმე ნეირონის მწყობრიდან გამოსვლა აზროვნების პროცესს არ უშლის ხელს, არც მეხსიერების დაკარგვას იწვევს.ზემოთ აღწერილი სქემა ბუნებრივად გვიბიძგებს საინტერესო შეკითხვებზე პასუხის ძიებისკენ.

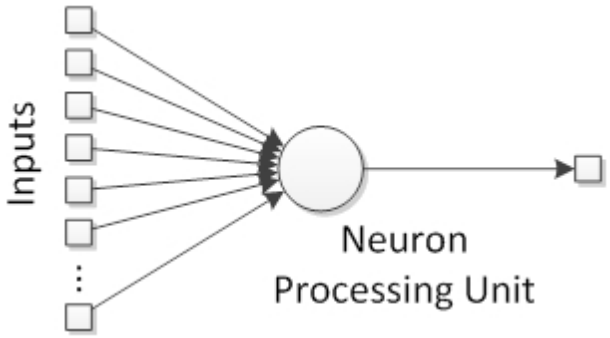
1.1 ხელოვნური ნეირონი და მისი ისტორია

ხელოვნური ნეირონი არის ბუნებრივი ნეირონის მსგავსი მოწყობილობა. ასეთ მოწყობილობებს ერთმანეთთან აერთებენ და იღებენ ხელოვნურ ნეირონულ ქსელს. სხვადასხვა ტიპის ნეირონულ ქსელებს სხვადასხვა პრაქტიკული ამოცანის გადასაწყვეტად იყენებენ.როგორც წინა თავში აღვწერეთ, ბუნებრივ ნეირონს დენდრიტების საშუალებით მიეწოდება დამუხტული იონები. დამუხტული იონები ნეირონის ბირთვში იყრის თავს, თუმცა აქსონის საშუალებით მომენტალურად არ ხდება მათი გამოსვლა ნეირონიდან. აქსონში იონების გამოტყორცნა მოხდება ერთბაშად, მხოლოდ იმ შემთხვევაში, თუ ნეირონში არსებული ჯამური მუხტი გარკვეულ ზღვარს გადააჭარბებს.

1940-იან წლებში ნეიროფიზიოლოგმა უორენ მაკკოლოკმა და მათემატიკოსი ვალტერ პიტსმა შექმნდეს ხელოვნური ნეირონის პირველი მათემატიკური იმპლემენტაცია, რომელიც აერთიანებს ნეირომეცნიერების საფუძვლებს მათემატიკურ ოპერაციებთან ერთად. იმ დროს, ადამიანის ტვინი სწავლობდა აქტიურად,რათა გაეგოთ მისი ფარული და საიდუმლო ქცევები, ჯერ კიდევ ნეირომეცნიერების სფეროში.იმ დროისთვის ცნობილი იყო ბუნებრივი ნეირონის სტრუქტურა და მისი მუშაობის პრინციპი,რომ ის იღებს შემომავალ სიგნალებს სხვა ნეირონებისგან და აქსონი ახდენს სიგნალის გააქტიურებას სხვა ნეირონებზე,როგორც ნაჩვენებია სურათ 1-ზე.

შევქმნათ ბუნებრივი ნეირონის მათემატიკური მოდელი, ანუ ხელოვნური ნეირონი, რომელსაც ექნება ბუნებრივს მსგავსი მექანიზმი.

მაკკოლოკმა და პიტსმა შემოგვთვაზეს ხელოვნური ნეირონის (შემდგომში მას უბრალოდ ნეირონს ვუწოდებთ) მათემატიკური კომპონენტი.ნეირონი ეს არის არის მოწყობილობა(პროცესორი), რომელსაც აქვს რამდენიმე შესასვლელი და ერთი გამოსასვლელი. შესასვლელზე მოდებული სიგნალები ნეირონის მიერ გადამუშავდება და გამოსასვლელზე აღიძვრება გამომავალი სიგნალი:



სურათი 3. ნეირონის მოდელი

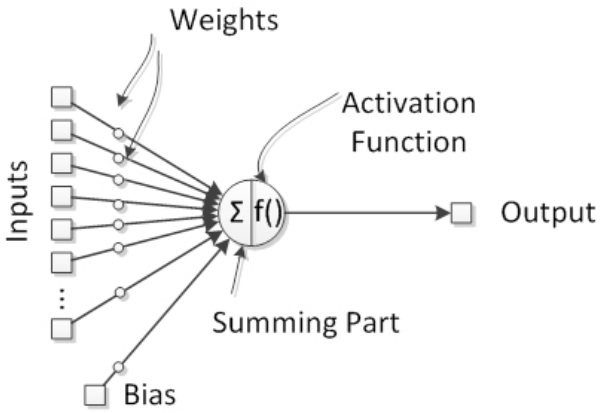
უფრო მეტიც, იმის გათვალისწინებით, რომ ტვინი შედგება მილიარდობით ნეირონისგან, თითოეული მათგანი ურთიერთდაკავშირებულია კიდევ რამდენიმე ათეულ ათას ნეირონთნ, რის შედეგადაც გვაქვს რამდენიმე ტრილიონი კავშირი, ჩვენ ვსაუბრობთ გიგანტური ქსელის სტრუქტურაზე. ამ ფაქტთან დაკავშირებით, მაკკოლოგმა და პიტსმა შექმნეს ნეირონის მარტივი მოდელი ,რომელიც თავდაპირველად ადამიანის ხედვის სიმულაცია ახდენდა. იმ დროისათვის არსებული კალკულატორები ან კომპიუტერები ძალიან იშვიათი იყო, მაგრამ მათ საკმაოდ კარგად შეეძლოთ მათემატიკური ოპერაციების ჩატარება; მეორეს მხრივ, დღევანდელი ამოცანებიც კი, როგორიცაა გამოსახულებისა და ხმის ამოცნობა ადვილი არ არის სპეციალური ფრეიმვორკების გარეშე საპირისპიროდ მათემატიკურ ოპერაციებისა. მიუხედავად ამისა, ადამიანის ტვინის შეუძლია შეასრულოს ხმის და გამოსახულების დამუშავება უფრო ეფექტურად, ვიდრე რთული მათემატიკური

გამოთვლები, და ეს ფაქტი დღესაც აინტრიგებს მეცნიერებსა და მკვლევარებს.თუმცა, ერთ-ერთი ცნობილი ფაქტია, რომ ყველა კომპლექსური საქმიანობა, რომელსაც ადამიანის ტვინი ეფუძნება,დაფუძნებულია ცოდნაზე, ANN არის შექმნილი იმისათვის რათა გადაწყვიტოს ზოგიერთი ამოცანა თვითონ, მონაცემების საფუძველზე:

ამოცანები, რომლებსაც ადვილად წყვიტავს ადამიანის ტვინი	ამოცანები ,რომლებსაც ადვილად წყვიტავს კომპიუტერი
სურათების კლასიფიკაცია	რთული გამოთვლები
ხმის ამოცნობა	სიგნალების დამუშავება
სახის იდენტიფიკაცია	ოპერაციული სისტემის მართვა
ცოდნაზე დაფუძნებით მოვლენების პროგნოზი	გრამატიკული შეცდომების გასწორება

1.2 ხელოვნური ნეირონის სტრუქტურა

მოდით უფრო დეტალურად განვიხილოდ პრაქტიკულად როგორ მუშაობს ხელოვნური ნეირონი.ჩვენ შეგვიძლია ვიფიქროთ ნეირონზე, რომელიც შედგება სიგნალის კოლექტორისგან და აქტივაციის ერთეულისგან, რომელმაც შეიძლება მოგვცეს სიგნალი, რომელიც გადაგზავნილი იქნება სხვა ნეირონებისათვის. სიგნალის გასროლის ფუნქცია განხორციელებულია აქტივაციის ფუნქციის მიერ.ასე რომ, ჩვენ შეგვიძლია განვსაზღვროთ ხელოვნური ნეირონის სტრუქტურა, როგორც ნაჩვენებია შემდეგი სურათზე:

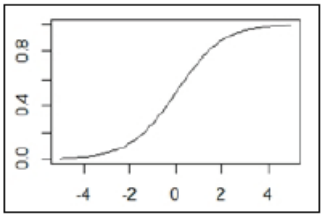
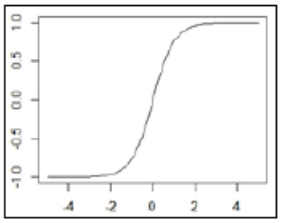
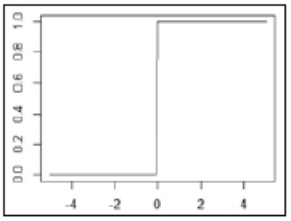
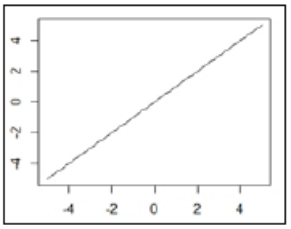


სურათი 4.ხელოვნური ნეირონი

განვიხილოთ ნეირონის შემადგენელი კომპონენტები.პირველი ასეთი არის **აქტივაციის ფუნქცია**.აქტივაციის ფუნქცია არის ის, რაც გვაძლევს ნეირონის გამომავალ სიგნალს ყველა შემომავალი სიგნალის ჯამიდან გამომდინარე. მათემატიკურად იგი სძენს არაწრფივობას ნეირონულ ქსელს ,რაც ძალიან სასარგებლო იქნება ბუნებრივი ნეირონის არაწრფივი ბუნების სიმულაციისთვის. აქტივაციის ფუნქცია, როგორც წესი, ორ მნიშვნელობას შორისა გვაძლევს მნიშვნელობებს .ზოგიერთ შემთხვევაში აქტივაციის ფუნქცია შესაძლოა, წრფივიც იყოს.ოთხი ყველაზე გავრცელებული აქტივაციის ფუნქცია არის შემდეგი:

- Sigmoid(სიგმოიდი)
- Hyperbolic tangent(ჰიპერბოლური ტანგენსი)
- Hard limiting threshold
- Purely linear(წრფივი)

შემდეგ ცხრილში მოცემულია ამ ფუნქციების ფორმულები და გრაფიკები:

Function	Equation	Chart
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	
Hyperbolic tangent	$f(x) = \frac{1-e^{-x}}{1+e^{-x}}$	
Hard limiting threshold	$f(x) = \begin{cases} 0 & if\ x < 0 \\ 1 & if\ x \geq 1 \end{cases}$	
Linear	$f(x) = x$	

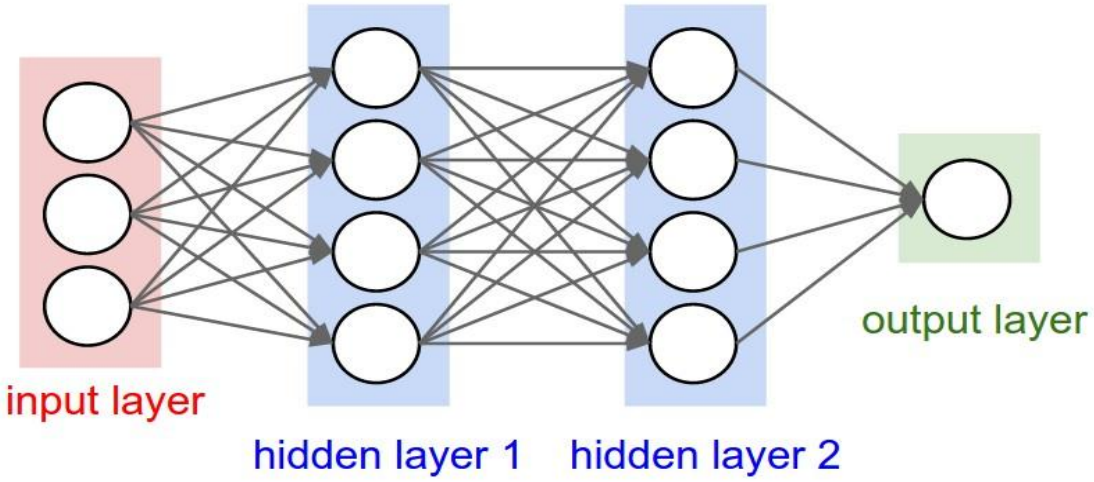
სურათი 5.ძირითადი აქტივაციის ფუნქციები

მიუხედავად იმისა, რომ ნერვული ქსელის სტრუქტურა შეიძლება ფიქსირებული იწოს, **წონა** წარმოადგენს ნეირონებს შორის კავშირებს და მათ აქვთ უნარი გააძლიერონ ან შეასუსტონ შემომავალი ნეირონული სიგნალები, ანუ ახდენენ მათ მოდიფიცირებას და გავლენა აქვთ ნეირონის გამოსავალზე. ამიტომ ნეირონის აქტივაცია არ იქნება დამოკიდებული მხოლოდ შემომავალ სიგნალზე არამედ, წონებზეც. იმ პირობით, რომ სიგნალი მოდის სხვა ნეირონებისგან ან გარე სამყაროსგან წონა ითვლება ნეირონულ ქსელში ნეირონებს შორის კავშირები. მას შემდეგ, რადგან წონა არის ნერვული ქსელის შიდა კომპონენტი და გავლენას ახდენს მისი გამოსავალზე, ისინი შეიძლება ჩაითვალოს როგორც ნეირონული ქსელის ცოდნა.ანი იმ წონების შეცვლით შეცვლება ნეირონული ქსელის შედეგებიც, ანუ ის პასუხობს გარესამყაროს სტიმულებზე.

დამატებითი პარამეტრი- **ბიასი**.ხელოვნური ნეირონისთვის სასარგებლოა დამოუკიდებელი კომპონენტის არსებობა, რომელიც დამატებით სიგნალს ანიჭებს აქტივაციის ფუნქციას: მას ეწოდება ბიასი. ეს პარამეტრი მოქმედებს, როგორც როგორც ჩვეულებრივი შემავალი მონაცემი,მაგრამ, ის სტიმულირებულია ერთი ფიქსირებული მნიშვნელობით (ჩვეულებრივ 1), რომელიც გამრავლებულია მასთან ასოცირებული წონაზე. ეს ფუნქცია ეხმარება ნეირონული ქსელის ცოდნის წარმოსადგენად, როგორც უფრო წმინდა არაწრფივი სისტემა, იმ პირობით, რომ როდესაც ყველა შემავალი არის ნულოვანი, ნეირონი

აუცილებლად არ გამოიტანს ნულს გამომავალზე,ამის ნაცვლად მან შეიძლება მოგვცეს განსხვავებული მნიშვნელობა, რომელიც დაკავშირებულია ბიასის შესაბამის წონასთან.

ბუნებრივი ნეირონები განლაგებულია **ფენებში(შრეებში),** თითოეული მათგანი უზრუნველყოფს სპეციფიკური დონის დამუშავებას; მაგალითად, შეყვანის ფენა იღებს პირდაპირ სტიმულს გარე სამყაროდან და გამომშვები შრეს გამოაქვს შედეგი,რომელსაც გავლენა აქვს გარე სამყაროზე. ამ ფენებს შორის არსებობს მთელი რიგი ფარული ფენები,ფარული იმ გაგებით, რომ მათ პირდაპირი კავშირი არ აქვთ გარე სამყაროსთან. ხელოვნურ ნეირონულ ქსელებში,ერთ შრეში განლაგებულ ყველა ნეირონს აქვთ ერთნაირი შემავალი მონაცემები და აქტივაციის ფუნქცია,როგორც ნაჩვენებია შემდეგ სურათზე:



სურათი 6.ნეირონული ქსელის შრეები

ნეირონული ქსელები შეიძლება შედგებოდეს რამდენიმე დაკავშირებული შრისაგან, რომლებიც ქმნიან ე.წ.მრავალშრიან ქსელებს. ნეირონული შრეები შეიძლება დაიყოს სამ კლასად:

- **შემავალი შრე-input layer**
- **ფარული შრე-hidden layer**
- **გამომავალი შრე-output layer**

პრაქტიკაში დამატებითი შრე აძლევს ქსელს საშუალებას წარმოადგინოს უფრო რთული ცოდნა.ყველა ნეირონულ ქსელს აქვს მინიმუმ შემავალი / გამომავალი შრე მიუხედავად შრეების რაოდენობისა. მრავალშრიანი ქსელის შემთხვევაში, შემავალ და გამომავალ შრეებს შორის არსებულ შრეებს **ფარული** შრეები ეწოდება.

1.2 ნეირონული ქსელების არქიტექტურა

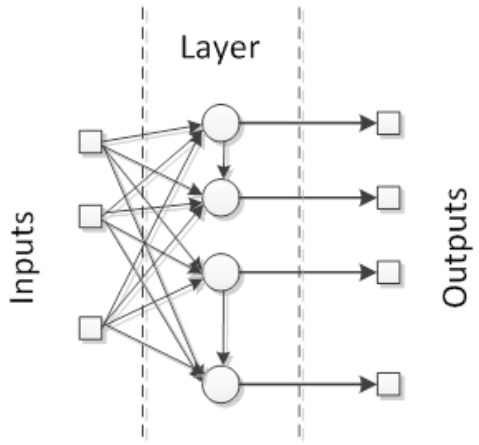
სხვადასხვა ნეირონულ ქსელს შეიძლება ჰქონდეს განსხვავებული არქიტექტურა, რაც დამოკიდებულია ნეირონების და შრეების კავშირის ტიპზე. თითოეული ნეირონული ქსელის არქიტექტურა განკუთვნილია კონკრეტული მიზნისთვის. ნეირონულ ქსელები შეიძლება იყოს მიმართული რიგი პრობლემების გადასაჭრელად, და პრობლემის ბუნებიდან გამომდინარე , ნეირონული ქსელი შეიძლება იყოს დაპროექტებული ამ პრობლემის უფრო ეფექტურად გადასაჭრელად. ნეირონული ქსელის არქიტექტურის ორ ძირითადი ტიპი არსებობს:

- **Neuron connections(ნეირონული კავშირების მიხედვით)**

- Monolayer networks(ერთშრიანი ქსელები)
- Multilayer networks(მრავალშრიანი ქსელები)
  - **Signal flow (სიგნალის მოძრაობის მიხედვით)**
- Feedforward networks
- Feedback networks

მოკლედ შევეხოთ თითოეულ მათგანს:

ერთშრიანი ქსელები : ამ არქიტექტურაში ყველა ნეირონი მოთავსებულია ერთსა და იმავე დონეზე, რომელიც ქმნის ერთ შრეს:ნეირონული ქსელი იღებს შემავალ სიგნალებს და აწოდებს მათ ნეირონებში, რომლებიც თავის მხრივ ითვლიან გამომავალი სიგნალენს. ნეირონები შეიძლება მჭიდროდ უკავშირდებოდნენ ერთმანეთთან ან რეკურენტულად ან მის გარეშე.ამ არქიტექტურის მაგალითები არის ერთი შრიანი პერცეპტონი, ადალაინი, თვითორგანიზებადი რუკა, ელმანის ,და ჰოპფილდის ნეირონულიქსელები.



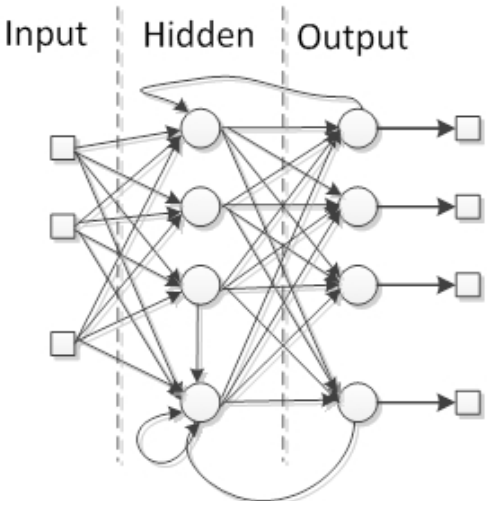
სურათი 7.ერთშრიანი ქსელი

მრავალშრიანი ქსელები: ამ კატეგორიაში ნეირონები განთავსებულია მრავალრიცხოვან შრეებში, თითოეული შრე, რომელსაც შეესაბამება პარალელურად ნეირონების განლაგება, იზიარებს ერთსა იმავე შემავალ მონაცემებს, როგორც ნაჩვენებია სურათი 6-ზე.რადიალური ბაზის ფუნქციები და მრავალშრიანი პერცეპტონი კარგი მაგალითებია ამ არქიტექტურაში. ასეთი ქსელები მართლაც სასარგებლოა რეალური მონაცემების იმ ფუნქციებთან მიახლოების მიზნით, რომლებიც სპეციალურადაა შექმნილი ამ ინფორმაციის წარმოსადგენად.უფრო მეტიც, რადგან მათ აქვთ დამუშავების მრავალი შრე, ეს ქსელები ადაპტირებულია ისწავლოს არაწრფივი მონაცემები, მას შეუძლია გამოყოფა ან უფრო მარტივად ცოდნის განსაზღვ რომელიც შექმნის ან აღიქვამს მონაცემებს.

პირდაპირი გავრცელების ქსელი: ნეირონულ ქსელში სიგნალის ნაკადი შეიძლება იყოს მხოლოდ ერთი მიმართულებით ან რეკურენტულად.პირველი შემთხვევაშ, ჩვენ მას ვუწოდებთ პირდაპირი გავრცელების ქსელს:, რადგან შემავალი სიგნალები შედის შემავალ შრეში; შემდეგ, დამუშავების შემდეგ, ისინი გადაეგზავნება მომდევნო შრეს, როგორც ნაჩვენებია სურათ 6-ზე.-ზე.რადიალური ბაზის ფუნქციები და მრავალშრიანი პერცეპტონი ასევე კარგი მაგალითებია ამ არქიტექტურაში.

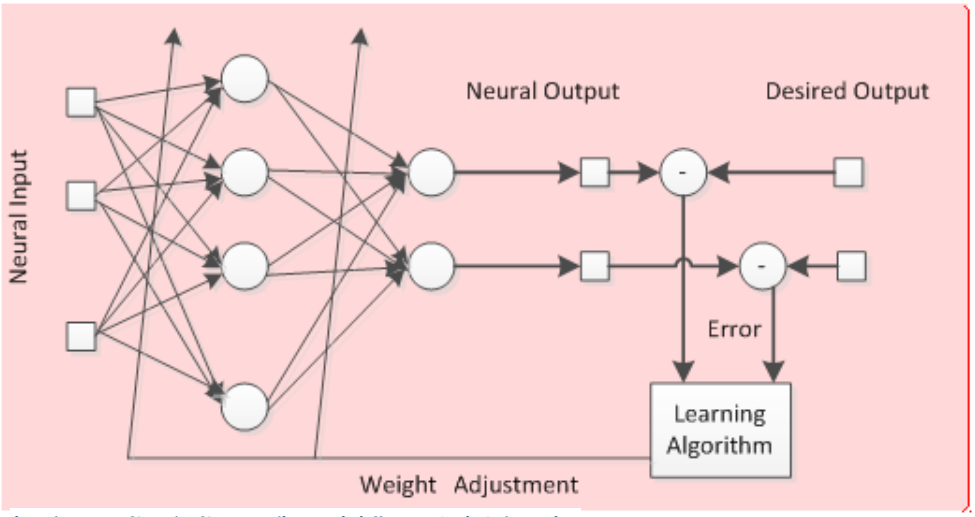
უკუგავრცელების ქსელი: როდესაც ნეირონულ ქსელში არის გარკვეული სახის რეკურენტულობა, ეს ნიშნავს, რომ სიგნალები მიეწოდება უკან იმ ნეირონში ან შრეში,

რომელმაც უკვე მიიღო და დაამუშავა სიგნალი,ასეთი ქსელი არის უკუკავშირის ტიპის.:ქსელში რეკურენტულობის მიზანი არის დინამიური ქცევის გამოწვევა,განსაკუთრებით მაშინ, როდესაც ქსელი წყვიტავს დროის სერიის ან ნიმუშის ამოცნობის პრობლემებს, რომლებიც საჭიროებს შიდა მეხსიერებას სწავლის პროცესის გაძლიერებისათვის თუმცა, ასეთი ქსელები განსაკუთრებით რთული დასატრენინგებელია, რადგან საბოლოო ჯამში სწავლებაში რეკურსიული ქცევა იქნება(მაგალითად, ნეირონი, რომლის გამოსავალიც მიეწოდება შემავალ შრეს), უმეტესობა უკუკავშირის ქსელებისა ერთ შრიანია, როგორიცაა ელმანისა და ჰოპფილდის ქსელები, მაგრამ შესაძლებელია მრავალშრიანი რეკურენტული ქსელის შექმნა , როგორიცაა მრავალშრიანი რეკურენტული პერცეპტონის ქსელები.



სურათი 8.უკუგავრცელების ქსელი

ნეირონული ქსელები სწავლობენ ნეირონებს შორის კავშირების დარეგულირების საშუალებით, კერძოდ, წონების ცვლილებით. როგორც ვთქვით, ნეირონის წონის ცვლილებით მისი გამოსავალი იცვლება,ანუ ნეირონულმა ქსელმა შეიძლება გააუმჯობესოს შედეგები სწავლების წესის მიხედვით წონების ცვლილებით.სქემა გამოსახულია შემდეგ სურათზე:



სურათი 9.ნეირონული ქსელის სწავლების პროცესი

სურათზე გამოსახულია დამკვირვებლიანი სწავლების პროცესი(supervised learning),რადგანაც ჩვენ გვაქვს სასურველი გამოსავალი,მაგრამ არსებობს სწავლება დამკვირვებლის გარეშეც(unsupervised learning).ამ თემას შემდეგ თავში შევეხები.

1.3 ნეირონული ქსელის იმპლემენტაცია

ამ თავში განვიხილავთ ნეირონული ქსელის იმპლემენტაციის დეტალებს.ამ ამოცანისათვის შერჩულია ერთერთი ყველაზე პოპულარული ობიექტზე ორიენტირებული პროგრამირების ენა Java.ამოცანის გადასაწყვეტად გამოყენებულია ობიექტზე ორიენტირებული პროგრამირების პარადიგმები,როგორიცაა მემკვიდროება,ინკაფსულაცია,პოლიმორფიზმი,აბსტრაქცია და კლასი.როგორც ვიცით, ნეირონული ქსელი შედგება შრეებისგან, ნეირონებისგან, წონის, აქტივაციის ფუნქციების და ბიასისფან. არსებობს სამი სახის შრე: შემავალი, დაფარული და გამომავალი. თითოეული ფენის შეიძლება ჰქონდეს ერთი ან მეტი ნეირონი. თითოეული ნეირონი დაკავშირებულია ან სხვა ნეირონის შემევალზე ან გამოსავალზე და ეს კავშირები ცნობილია როგორც წონა.მნიშვნელოვანია ხაზგასმით აღინიშნოს, რომ ნეირონულ ქსელს შეიძლება ჰქონდეს ბევრი ფარული შრე ან საერთოდ არცერთი.ნეირონების რაოდენობა თითოეულ შრეში შეიძლება განსხვავდებოდეს. თუმცა, შემავალ და გამომავალი შრეებს შესაბამისი რაოდენობის ნეირონი აქვთ ნეირონის შესავლების და გამოსავალების რაოდენობების შესაბამისად.

ასე რომ, დავიწყოთ იმპლემენტაცია თავდაპირველად, ჩვენ ვაპირებთ განვსაზღვროთ შემდეგი კლასები:

- **Neuron**: განსაზღვრავს ხელოვნუ ნეირონს
- **NeuralLayer**: აბსტრაქტული კლასი რომელიც განსაზღვრავს ნეირონთა შრეს
- **InputLayer**: განსაზღვრავს შემავალ შრეს
- **HiddenLayer**: განსაზღვრავს დაფარულ შრეს
- **OutputLayer**: განსაზღვრავს გამომავალ შრეს
- **InputNeuron**: განსაზღვრავს შემავალ ნეირონს
- **NeuralNet**: განსაზღვარვს ნეირონულ ქსელს.

გარდა ამ კლასებისა, ჩვენ ასევე უნდა განვსაზღვროთ IActivationFunction ინტერფეისი აქტივაციის ფუნქციისათვის. ეს აუცილებელია იმისათვის, რომ აქტივაციის ფუნქციები მოქმედებენ როგორც მეთოდები, მაგრამ ისინი საჭიროა როგორც ნეირონის თვისებები. ამიტომ, ჩვენ ვაპირებთ, განვსაზღვროთ კლასები აქტივაციის ფუნქციებისთვის,რომლების იმპლემენტაციას გაუკეთებენ ამ ინტერფეისს.

- Linear
- Sigmoid
- Step
- HyperTan

ამით ნეირონული ქსელი თითქმის დასრულებულია. ჩვენ უნდა განვსაზღვროთ კიდევ ორი კლასი. ერთი იქნება გამონაკლისი სიტუაციების დასამუშავებლად (NeuralException) და შემთხვევითი რიცხვების გენერატორი (RandomNumberGenerator).

ადგილის დასაზოგად აქ არ მოვყვები კოდის დეტალების განხილვას ,რადგან მიმაგრებულ პროექტში კოდი კარგად დაკომენტარებულია. ასევე შესაძლებელია javadoc-ის გაცნობაც.NeuralNetConsoleTest კლასში ვქმნით მარტივ ქსელს და მისი საშუალებით კონსოლზე ვბეჭდავთ ქსელის გამოსავალ მნიშვნელობებს.

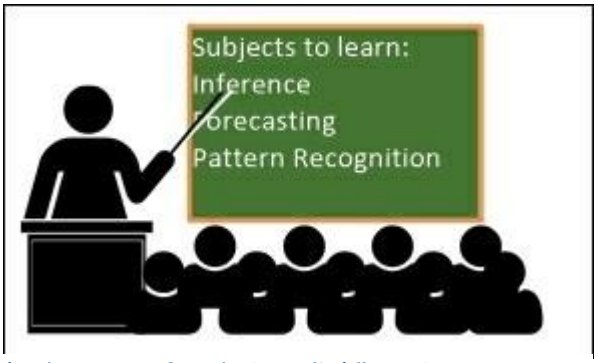
თავი 2. ნეირონული ქსელის სწავლება

ამ თავში შევეხები ნეირონული ქსელების სწავლების ძირითად იდეებს და მათ იმპლემენტაციას.მართლაც საოცარია ნეირონული ქსელებს აქვთ უნარი ისწავლონ გარემოდან, ისევე როგორც მოაქროვნე არსებები. ჩვენ,ადამიანები ვწავლობთ დაკვირვებისა და გარემოების შედეგად სანამ რამე ქცევა არ თემა არ იქნება დასწავლილი. . ფსიქოლოგიური თვალსაზრისით, ადამიანის ტვინში სწავლის პროცესი არის ნერვული კავშირების რეკონფიგურაცია ნეირონებს შორის , რაც ახალი აზროვნების სტრუქტურას აყალიბებს.განსხვავებით ჩვეულებრივი ციფრული კომპიუტერებისგან, რომელთაც შეუძლიათ განახორციელონ მხოლოდ ის ამოცანები, რომლებიც დაპროგრამებულია მათზე, ნეირონულ სისტემებს შეუძლიათ გააუმჯობესონ და განახორციელონ ახალი აქტივობები რაღაც საკმარისობის კრიტერიუმის მიხედით.სხვა სიტყვებით, რომ ვთქვათ ნეირონულ ქსელებს არ ჭირდებათ რომ იყვნენ დაპროგრამებული; ისინი თავად სწავლობენ პროგრამას.იმის გათვალისწინებით, რომ ზოგიერთ ამოცანის შეიძლება ჰქონდეს დიდი რაოდენობით თეორიულად შესაძლებელი ამონახსნი, სწავლის პროცესი ცდილობს ოპტიმალური გადაწყვეტის პოვნას, რაც მოგვცემს დამაკმაყოფილებელი შედეგს. ხელოვნური ნერვული ქსელები (ANN) გამოყენება გამართლებულია რადგან მათ აქვთ შესაძლებლობა ისწავლონ ნებისმიერი ტიპის ცოდნა, მხოლოდ შემავალი დატისა და საჭირო ინფორმაციის მიხედვით. თავდაპირველად , ANN აწარმოებს შემთხვევით შედეგს და მოგვცემს გარკვეულ შეცდომას, და ამ შეცდომაზე დაფუძნებით ANN პარამეტრები შეიცვლება ეს პროცესი გრძელდება იტერაციულად სანამ არ მიიღწევა შეცდომის დამაკმაყოფილებელი სიდიდე ამოცანიდან გამომდინარე.

2.1 ნეირონული ქსელის სწავლების პარადიგმები

არსებობს ძირითადად ორი ტიპის სწავლება ნეირონული ქსელებისთვის, კერძოდ, დამკვირვებლიანი და დამკვირვებლის გარეშე.მაგალითად, ადამიანის გონებაც , ასევე მუშაობს ამ გზით. ჩვენ შეგვიძლია შევიძინოთ ცოდნა დაკვირვების შედეგად ნებისმიერი მიზნის გარეშე (unsupervised) ან შეგვიძლია გვყავდეს მასწავლებელი, რომელიც გვიჩვენებს სწორ გზას. განსხვავება ამ ორ პარადიგმას შორის ძირითადად დამოკიდებულია სამიზნეზე და განსხვავებულია ამოცანებიდან გამომდინარე.

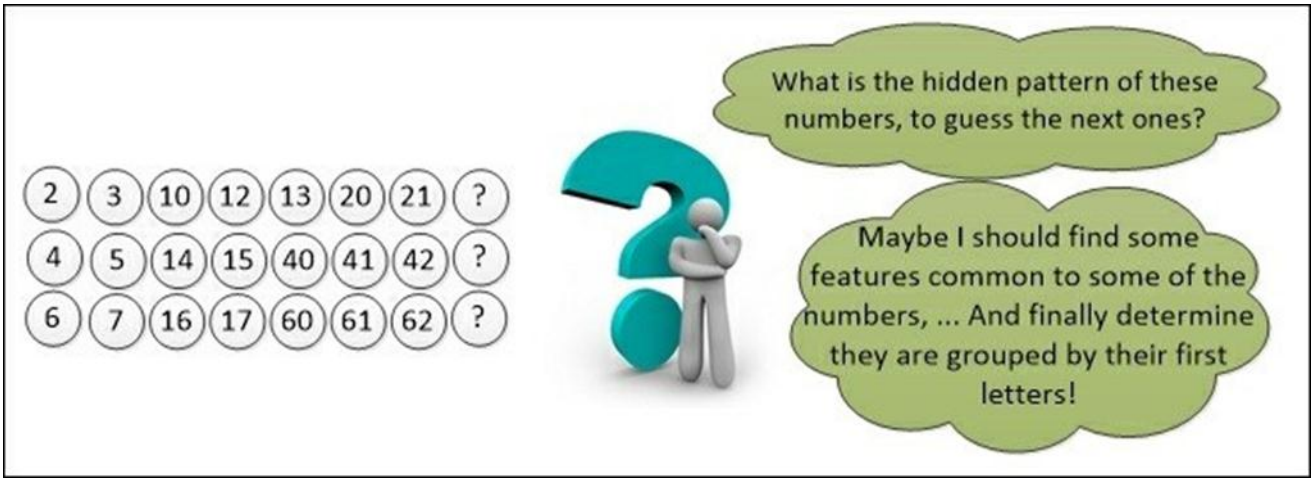
დამკვირვებლიანი სწავლა(Supervised learning)ეს სწავლების ტიპი ეხება წყვილების X- ს (დამოუკიდებელი მნიშვნელობები) და ys (დამოკიდებული მნიშვნელობები) და მიზანი არის მათი დამოკიდებულების შესწავლა. აქ Y მონაცემები არის დამკვირვებელი, სამიზნე სასურველი შედეგები, და X არის წყარო დამოუკიდებელი მონაცემებისა, რომლებიც ერთობლივად აგენერირებენ Y მონაცემებს.ეს ანალოგიურია მასწავლებელისა, რომელიც ასწავლის ვინმეს როგორ უნდა შესრულდეს კონკრეტული დავალება.



სურათი 10.დამკვირვებლიანი სწავლება

ამ სწავლების პარადიგმის ერთ-ერთი თავისებურება ის არის, რომ არსებობს შეცდომის პირდაპირი განსაზღვრის შესაძლებლობა, რომელიც არის შედარება სამიზნე და მიმდინარე ფაქტობრივი შედეგისა. ქსელის პარამეტრები მიეწოდება ღირებულების ფუნქციას(cost function), რომელიც ითვლის აცდენას სასურველ და ფაქტობრივ შედეგებს შორის. ღირებულება ფუნქცია არის მხოლოდ საზომი რომელიც უნდა შემცირდეს ოპტიმიზაციის პრობლემაში. ეს ნიშნავს რომ ქსელის ამოცანაა იპოვოს ისეთი პარამეტრები, რომ ღირებულების ფუნქციამ ყველაზე დაბალი შესაძლო მნიშვნელობა მიიღოს. დამკვირვებლიანი სწავლება განკუთვნილია ისეთი ამოცანების გადასაჭრელად როდესაც გვაქვს ქცევა განსაზღვრული ნიმუშით. ზოგიერთი მაგალითები მოიცავს სურათების კლასიფიკაციას,საუბრუს ამოცნობას, ფუნქციის მიახლოებას და პროგნოზირება. გასათვალისწინებელია , რომ ნეირონული ქსელი უნდა შეიცავდეს ცოდნას ორივე შემავალი დამოუკიდებელი მნიშვნელობებისა (X) და გამომავალი დამოკიდებულi მნიშვნელობებისა (Y). დამოკიდებული გამომავალი მნიშვნელობის არსებობა არის აუცილებელი პირობა ასეთი სწავლების მიდგომის გამოყენებისას.

დამკვირვებლის გარეშე(unsupervised learning,) სწავლისას, ჩვენ მხოლოდ მონაცემები გაგვაჩნია ყოველგვარი დამატებითი ინფორმაციისა და კლასიფიკაციის გარეშე. ამის ნაცვლად, ჩვენ ვცდილობთ დასკვნა გავაკეთოთ და ამოვიღოთ საჭირო ცოდნა მხოლოდ დამოუკიდებელი მონაცემების გათვალისწინებით X:



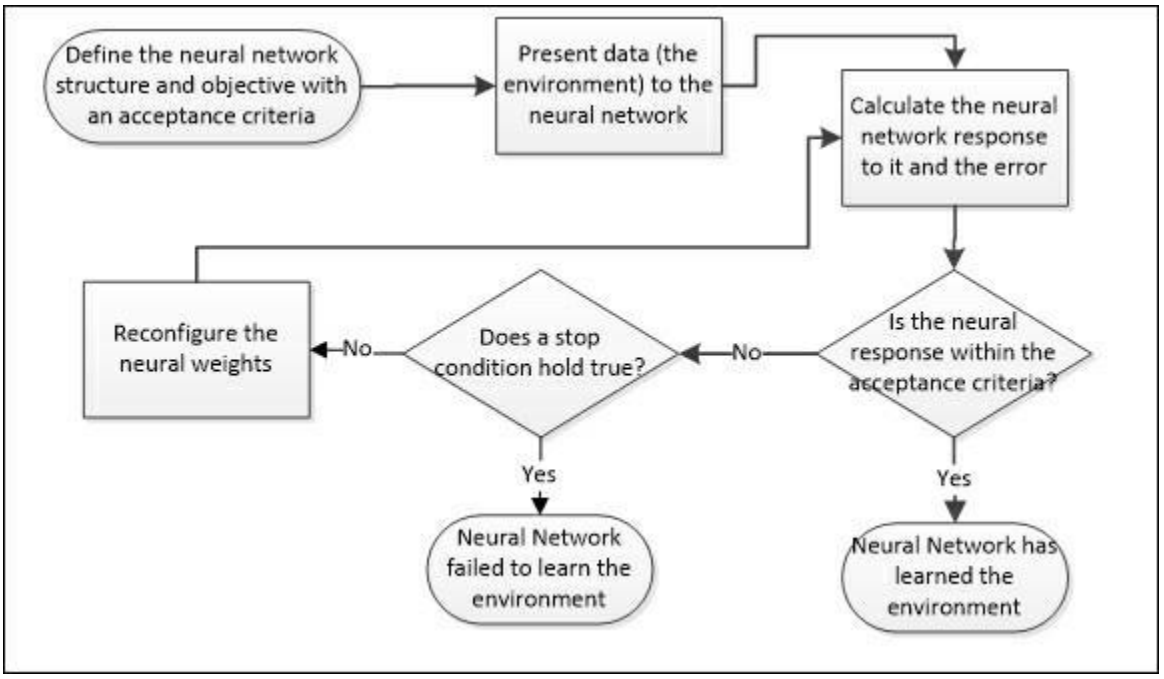
სურათი 11.დამკვირვებლიანი სწავლება

ეს არის თვითსწავლების ანალოგი, როდესაც ვინმე საკუთარ გამოცდილებას და გარკვეულ კრიტერიუმებს ითვალისწინებს. არადამკვირვებლიანი სწავლისას, ჩვენ არ გვაქვს განსაზღვრული სასურველი შედეგი; თუმცა ნეირონულ ქსელს შეუძლია ამ მონაცემების საფუძველზე დამოკიდებული (Y) მონაცემების დადგენა ყოველგვარი დამკვირვებლის გარეშე.არადამკვირვებლიანი სწავლების მაგალითებია, დაკლასტერება, მონაცემთა შეკუმშვა,სტატისტიკური მოდელირება და სხვა.რადგანაც ამ ნაშრომში ძირითადად განვიხილავთ დამკვირვებლიან სწავლებას არადამკვირვებლიან სწავლებას ნაკლებად შევეხები.

2.2 ნეირონული ქსელის სწავლების პროცესი

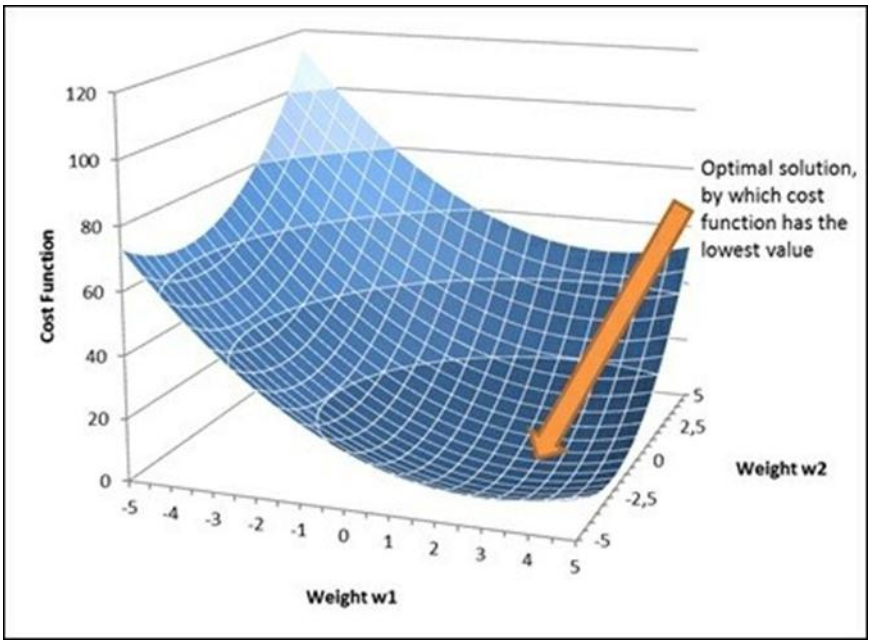
ჯერჯერობით ჩვენ თეორიულად განვსაზღვრეთ სწავლების პროცესი და როგორ ხორციელდება იგი. მაგრამ პრაქტიკაში,ჩვენ უფრო ღრმად უნდა ჩავწვდეთ მათემატიკურ ლოგიკას რათა იმპლემენტაცია გავუკეთოთ სწავლების ალგორითმს. სწავლების ალგორითმი არის პროცედურა, რომელიც ახორციელებს ნეირონული ქსელების სწავლების პროცესს, და ეს მკაცრად არის

განსაზღვრული ნეირონული ქსელის არქიტექტურით. მათემატიკური თვალსაზრისით, ალგორითმი ცდილობს იპოვოს ოპტიმალური წონის W, რომელსაც შეუძლია მიიყვანოს ღირებულებათა ფუნქცია C (X, Y) ყველაზე დაბალი შესაძლო მნიშვნელობაზე.თუმცა, ხანდახან სწავლის პროცესი ვერ პოულობს კარგ წონებს,რომელიც სასურველ კრიტერიუმებს აკმაყოფილებს, ამიტომ საჭიროა გაჩერების პირობა დადგინდეს, რომ თავიდან იქნას აცილებული ნეირონული ქსელის სამუდამოდ სწავლა ,რაც გამოიწვევს კომპიუტერის პარალიზებას.ზოგადად პროცესი მიდის შემდეგნაირად,როგორც შემდეგ სქემაზეა გამოსახული:



სურათი 12.სწავლების ალგორითმის სქემა

ახლა დეტალურად გავარკვიოთ, რა როლს ასრულებს ღირებულების ფუნქცია. მოდით წარმოვიდგინოთ ღირებულებათა ფუნქცია როგორც ორი ცვლადის ფუნქცია,რომელიც გვაძლევს ზედაპირს. სიმარტივისთვის, განვიხილოთ მხოლოდ ორი წონა (ორი განზომილებიანი სივრცე პლუს სიმაღლე წარმოადგენს ღირებულებათა ფუნქციას). დავუშვათ ჩვენს ღირებულების ფუნქციას აქვს შემდეგი ფორმა,როგორც სურათზეა:



სურათი 13.ღირებულების ფუნქცია

ვიზუალურად, ჩვენ ვხედავთ, რომ არსებობს ექსტრემუმი, რომლის მეშვეობითაც ღირებულების ფუნქცია უხეშად უახლოვდება ნულს. მაგრამ როგორ შეგვიძლია ამის გაკეთება პროგრამულად? პასუხი არის მათემატიკურ ოპტიმიზაციაში სადაც ღირებულებათა ფუნქცია განისაზღვრება, როგორც ოპტიმიზაციის პრობლემა:

$$\min_{X,Y,W \in R^N} C(X,Y,W)$$

როგორც ვიცით ფერმას თეორემის მიხედვით ექსტრემუმის არსებობისთვის აუცილებელი პირობაა, რომ ამ წერტილში კერძო წარმოებული უნდა იყოს ნულოვანი და ისინი უნდა იყვნენ კრებადი (მინიმალური შემთხვევაში). ვთქვათ თავიდან ვიწყებთ რაღაც W წონებით ვიწყებთ ძებნას იმ მიმართულებით სადაც ზედაპირის სიმაღლე მცირდება. ეს არის ე.წ. გრადიენტის მეთოდი

ღირებულების ფუნქციის მიხედვით,წონების განახლების წესი განსაზღვრავს თუ როგორ უნდა შეიცვალოს ქსელის პარამეტრები,ისინი შეიცვლება ისე რომ ღირებულებათა ფუნქცია ექნება უფრო დაბალი მნიშვნელობა:

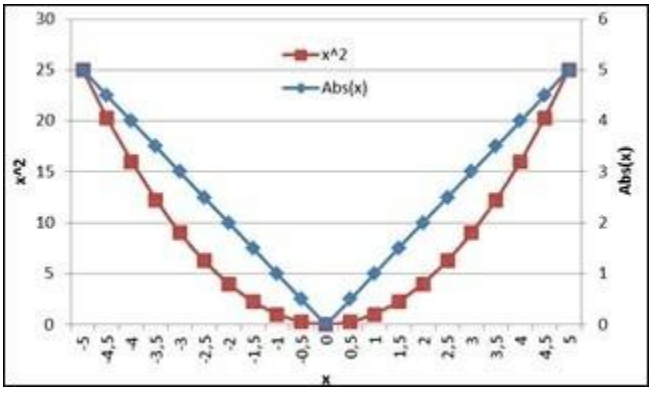
$$W(k+1) = W(k) + \Delta W$$

აქ k ნიშნავს k-ურ იტერატიაც და W (k) ეხება ნეირონულ წონენს k-ურ იტერაციაზე.k + 1 მიუთითებს მომდევნო იტერაციას.წონის განახლების ოპერაცია შეიძლება შესრულდეს სათითაოდ ან ერთბაშად . სათითაო აქ გულისხმობს, რომ წონა განახლდება თითოეული ჩანაწერის შემდეგ. Batch განახლება ნიშნავს, რომ ჯერ ყველა ჩანაწერი მიეწოდება ქსელს ,სანამ იგი იწყებს წონების განახლებას. ეს საკითხი დეტალურად იქნება აღწერილი კოდში.როდესაც ნერვული ქსელი სწავლობს, იგი იღებს მონაცემებს გარემოსგან და წონებს ანახლებს შესაბამისად. ეს მონაცემები იწოდება როგორც სატრენინგო მონაცემთა ნაკრები და მას აქვს რამდენიმე ნიმუში. სიტყვა ტრენინგის უკან დგას ნეირონული წონების ადაპტირების პროცესი. როდესაც ნეირონული ქსელი სწავლობს, არსებობს ცდომილება სამიზნე შედეგსა (Y) და ნეირონულ შედეგებს შორის, ზედამხედველობასთან დაკავშირებით: დამკვირვებლიან სწავლებაში გვექნება:

$$e = Y - \hat{Y}$$

იმის გათვალისწინებით, რომ ტრენინგის მონაცემთა ნაკრებს აქვს მრავალი მნიშვნელობა, საბოლოოდ გვექმენა შეცდომების N მნიშვნელობა.

მაგრამ როგორ უნდა მივიღოთ საერთო შეცდომა? ერთი ინტუიციური მიდგომაა ავიღოთ საშუალო შეცდომა, მაგრამ ეს შეცდომაა. შეცდომის ვექტორმა შეიძლება მიიღოს როგორც დადებითი ისე უარყოფითი მნიშვნელობები.შესაძლებელია ავიღოთ შეცდომის აბსოლუტური მნიშვნელობა ,მაგრამ ეს მიდგომაც არაა კარგი რადგან ასეთი ფუნქცია არაა უწყვეტი 0 წერტილში და მისი წარმოებული არ არსებობს ამ წერტილში.ეს ნაჩვენების სურათზე:



სურათი 14.აბსოლუტური და კვადრატული ფუნქციები

ასე რომ, გონივრული ვარიანტი იქნება, რომ გამოვიყენოთ შეცდომის კვადრატული ჯამი, რომელსაც ეწოდება საშუალო კვადრატული შეცდომა(mean squared error) MSE:

$$MSE(Y,\hat{Y}) = \frac{1}{N}\sum_{i}^{N}(Y_i - \hat{Y}_i)^2$$

როდესაც ქსელს გვაქვს ბევრი გამომავალი გვექმენა შეცდომის მატრიცა:

$$E = \begin{pmatrix} (Y_{kj} - \hat{Y}_{kj}) & \dots & (Y_{kn} - \hat{Y}_{kn}) \\ \vdots & \ddots & \vdots \\ (Y_{Nj} - \hat{Y}_{Nj}) & \dots & (Y_{Nn} - \hat{Y}_{Nn}) \end{pmatrix}$$

შესაბამისად უფრო მოსახერხებელია დავითვალოთ საერთო შეცდომა,რომელსაც აქვს შემდეგი სახე:

$$e_k = \frac{1}{2}\sum_{j}^{n}(Y_{kj} - \hat{Y}_{kj})^2$$

შეიძლება ნეირონულმა ქსელმა სამუდამოდ ისწავლოს? როდის უნდაა შეჩერედეს ის?როდესაც დასწავლის პროცესი მიმდინარეობს, ნეირონული ქსელი უნდა გვაძლევდეს შედეგებს უფროდაუფრო ახლოს მოსალოდნელთან, სანამ საბოლოო ჯამში არ დაკმაყოფილდება კრიტერიუმი ან არ მიიღწევა იტერაციების მაქსიმალური რაოდენობა. სწავლის პროცესი დასრულებულად ითვლება მაშინ, როდესაც ამ ორი პირობიდან ერთერთი მაინც სრულდება.:

- **მიიღწევა საკმარისობის კრიტერიუმი**
- **მიიღწევა ეპოჩების(იტერაციების) მაქსიმალური რაოდენობა**

2.3 სწავლების ალგორითმის იმპლემენტაცია

ვიწყებთ დასწავლის ალგორითმის პროგრამულ რეალიზაციას.ამისთვის შევქმნათ აბსტრაქტული სუპერკლასი კლასი LearningAlgorithm learn პაკეტში,რომელიც საშუალებას მისცემს ნებისმიერ მსურველს იმპლემენტააცია გაუკეთოს სხვადასხვა ალგორითმებს.ასევე შექმნილი გვაქვს დამხმარე კლასები მონაცემებთან სამუშაოდ.ეს კლასები მოთავსებულია data პაკეტში. NeuralInputDat, NeuralOutputData კლასები,რომლებიც NeuralDataSet კლასის შვილები არიან.

LearningAlgorithm კლასს შემდეგი ატრიბუტები და მეთოდები გააჩნია:

```
public abstract class LearningAlgorithm {

protected NeuralNet neuralNet;

public enum LearningMode {ONLINE,BATCH};

protected enum LearningParadigm {SUPERVISED,UNSUPERVISED};


//…

protected int MaxEpochs=100;

protected int epoch=0;

protected double MinOverallError=0.001;
```

```
protected double LearningRate=0.1;

protected NeuralDataSet trainingDataSet;

protected NeuralDataSet testingDataSet;

protected NeuralDataSet validatingDataSet;

public boolean printTraining=false;

public abstract void train() throws NeuralException;

public abstract void forward() throws NeuralException;

public abstract void forward(int i) throws NeuralException;

public abstract Double calcNewWeight(int layer,int input,int neuron) throws


NeuralException;

public abstract Double calcNewWeight(int layer,int input,int neuron,double error) throws NeuralException;

//…

}
```

სრული კოდის სანახავად გთხოვთ ნახოთ პროექტის ფაილები.

2.2 ნეირონული ქსელის სწავლება დელტა წესით და მისი იმპლემენტაცია

ეს ალგორითმი ანახლებს წონებს ღირებულების ფუნქციის მიხედვით,რომელიც მიჰყვება გრადიენტის მეთოდს. ჩვენ გვინდა გავიგოთ, თუ რომელმა წონებმა შეიძლება წაიყვანონ ღირებულებათა ფუნქცია უფრო დაბალი მნიშვნელობისკენ. გაითვალისწინეთ, რომ ჩვენ შეგვიძლია ვიპოვოთ ღირებულებათა ფუნქციის თითოეული კერძო წარმოებულები წონების მიხედვით. უფრო მარტივად გასაგებად განვიხილოთ ერთი მარტივი მაგალითი მხოლოდ ერთი ნეირონით, ერთი წონით და ერთი ბიასით,და ასევე ერთი შემავალით. გამოსავალი იქნება შემდეგი:

$$\hat{Y} = g(X \cdot w + b)$$

აქ g არის აქტივაციის ფუნქცია, X არის ვექტორის შემცველი x მნიშვნელობებისა და Y არის გამომავალი ვექტორი გენერირებული ნეირონული ქსელის მიერ. ზოგადი შეცდომა k-ური მონაცემისა ასეთი იქნება:

$$E_k = y_k - \hat{y}_k$$

თუმცა, ამ შეცდომის განსაზღვრა შესაძლებელია კვადრატული შეცდომა: MSE. მაგრამ, სიმარტივისათვის, განვიხილოთ მარტივი შეცდომის განსხვავება. საერთო შეცდომა იქნება:

$$C(X, Y, \hat{Y}) = \frac{1}{N}\sum_{k}^{N} E_k^2$$

წონა და ბიასი დელტა წესის შესაბამისად განახლდება, რომელიც კერძო წარმოებულებს განიხილავს:

$$\partial C\frac{\left(X,Y,\hat{Y}\right)}{\partial w}\ and\ \partial C\frac{\left(X,Y,\hat{Y}\right)}{\partial b}$$

Batch სწავლების დროს , X და E ვექტორებია:

$$\Delta w = \alpha\frac{\partial C(X,Y,\hat{Y})}{\partial w} = \alpha E^T X g'(X\cdot w+b)^T$$

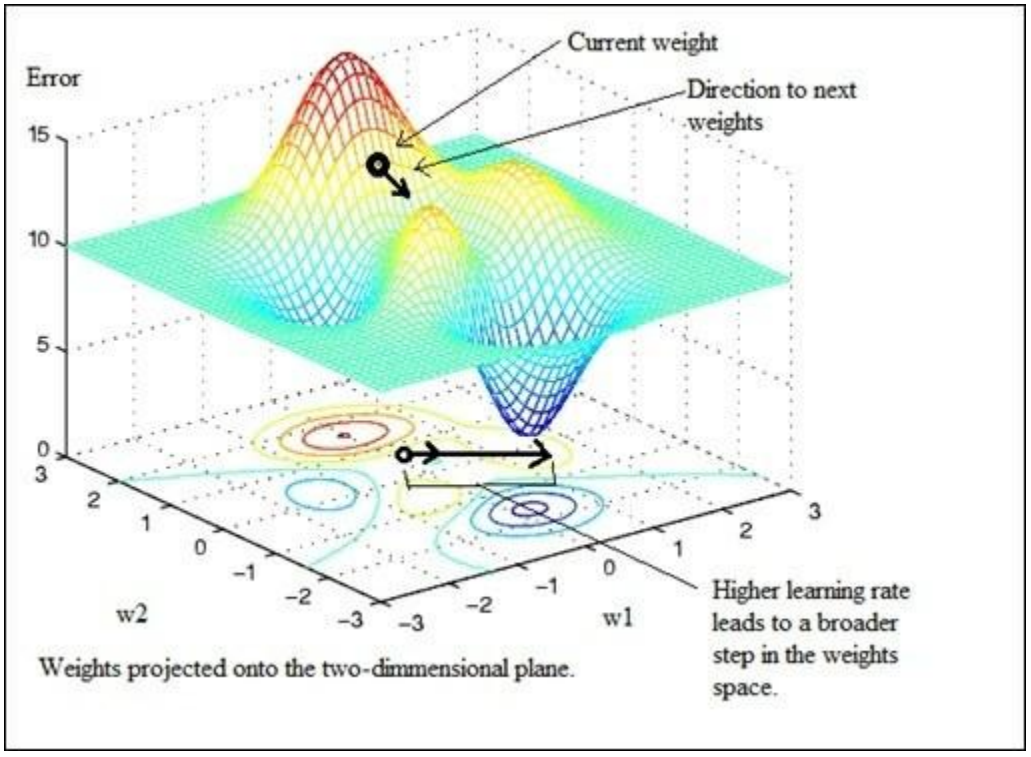
$$\Delta b = \alpha\frac{\partial C(X,Y,\hat{Y})}{\partial b} = \alpha E^T \begin{pmatrix}1\\ \vdots\end{pmatrix} g'(X\cdot w+b)^T$$

სათითაო სწავლების დროს , X და E ვექტორებია:

$$\Delta w = \alpha\frac{\partial C(X,Y,\hat{Y})}{\partial w} = \alpha E_k X_k g'(X_k\cdot w+b)$$

$$\Delta b = \alpha\frac{\partial C(X,Y,\hat{Y})}{\partial b} = \alpha E_k g'(X_k\cdot w+b)$$

შევნიშნოთ რომ წინა ტოლებებში α მიუთითებს სწავლის ფასს(learning rate). იგი მნიშვნელოვან როლს ასრულებს წონის განახლებაში, ვინაიდან მას შეუძლია მიგვიყვანოს უფრო სწრაფად ან უფრო ნელა ღირებულებათა ფუნქციის მინუმუმზე. ეს ნაჩვენების შემდეგ ნახაზზე სადაც მხოლოდ ორი წონაა.



სურათი 15.learning rate სწავლების ფასი

სწავლების ფასი გვეუბნება წონების განახლებისას რა სიდიდის პროპორციულად უნდა შეიცვალონ წონების მნიშვნელობები.უხეშად რომ ვთქვათ, ეს არის ნაბიჯების სიგრძე ზედაპირზე სიარულისას.

ახლა შევუდგეთ მის იმპლემენტაციას. ჩვენ იმპლემენტაციას გავუკეთებდ დელტა წესს DeltaRule კლასში,რომელიც LearningAlgorithm კლასის მემკვიდრეა.სანამ კლასის განხილვაზე გასავალთ მანამდე საჭიროა ორიოდე სიტყვა ვთქვათ ცდომილების შესახებ. ნეირონულ ქსელს შეიძლება ჰქონდეს მრავალი გამოსავალი. ჩვენ უნდა გაუმკლავდეთ მრავალ გამომავალ შედეგს,ამ დროს ცდომილების ვექტორის ნაცვლად, ჩვენ ცდომილების მატრიცა გვექნება:

$$E = \begin{pmatrix} (Y_{kj} - \hat{Y}_{kj}) & \dots & (Y_{kn} - \hat{Y}_{kn}) \\ \vdots & \ddots & \vdots \\ (Y_{Nj} - \hat{Y}_{Nj}) & \dots & (Y_{Nn} - \hat{Y}_{Nn}) \end{pmatrix}$$

ასეთ შემთხვევებში შეიძლება დიდი რაოდენობის ცდომილებებთან მოგვიწიოს მუშაობა: ერთი კონკრეტული გამოსავალის ცდომილება, კონკრეტული ჩანაწერის, ან მთელი მონაცემთა ნაკრების. უფრო მარტივად რომ გავიგოთ , მოდით ვუწოდო თკონკრეტული ჩანაწერის ცდომილებას ზოგადი შეცდომა, რომლითაც ყველა გამომავალ ცდომილებას ექნება ერთი სკალარული მნიშვნელობა და ცდომილება რომელიც განსაზღვარვს მთელი მონაცემების შეცდომას საერთო შეცდომა.ზოგადი შეცდომა ერთ გამოსავალიანი ქსელისთვის იქნება მხოლოდ განსხვავება სამიზნესა და გამომავალს შორის, მაგრამ მრავალი გამომავალის შემთხვევაში, მასში უნდა იყოს თითოეული გამომავალის შეცდომა. როგორც ვნახეთ, კვადრატული ცდომილება კარგი საშუალებაა ამის დასათვლელად.ამიტომ ზოგად შეცდომას ექნება სახე:

$$e_k = \frac{1}{2}\sum_j^n (Y_{kj} - \hat{Y}_{kj})^2$$

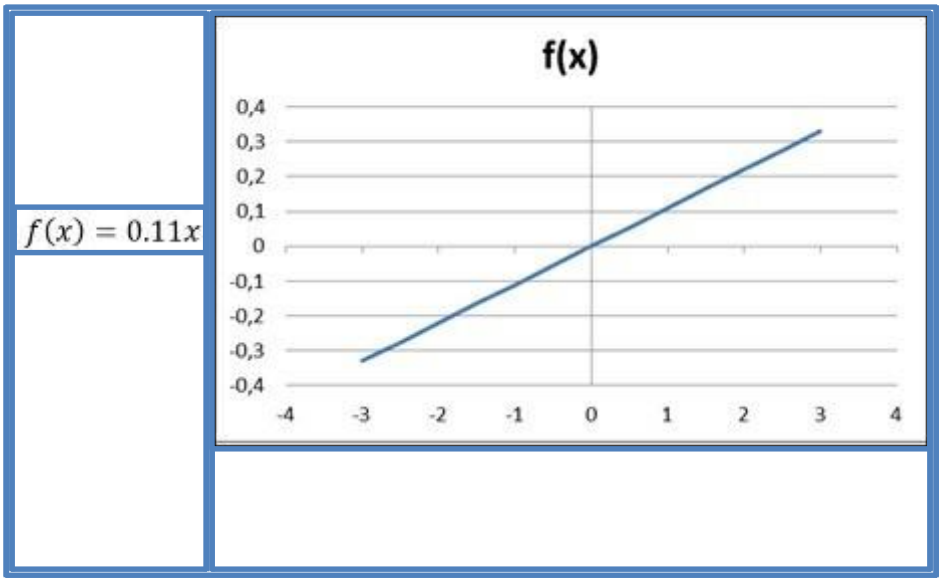
რაც შეეხება საერთო შეცდომას.ის განიხილავს ზოგად ცდომილებას მთელი მონაცემთა ერთობისათვის.რადგან მონაცემი შეიძლება იყოს მრავალი უმჯობესია იგი დავითვალოთ,როგორც MSE ზოგადი ცდიმილებების კვადრატებისა. ზოგადი და საერთო ცდომილებები(general and overall errors) განხილულია DeltaRule- ის კლასში, რადგან დელტა წესის ალგორითმი განიხილავს ამ შეცდომებს სწავლებისას. ისინი მასივები არიან, რადგან იქნება ზოგადი შეცდომა თითოეული ჩანაწერისთვის და იქნება საერთო შეცდომა თითოეული გამომავალისათვის. ატრიბუტი overallGeneralError იღებს ღირებულებათა ფუნქციის მნიშვნელობას, ანუ საერთო შეცდომას ყველა გამოსავალისა და ჩანაწერისა. მატრიცა სახელად error , ინახავს შეცდომებს თითოეული გამომავალი ჩანაწერის კომბინაციისას.

```
public class DeltaRule extends LearningAlgorithm {

public ArrayList<ArrayList<Double>> error;

public ArrayList<Double> generalError;

public ArrayList<Double> overallError;

public double overallGeneralError;

public double degreeGeneralError=2.0;

public double degreeOverallError=0.0;

public enum ErrorMeasurement {SimpleError, SquareError,NDegreeError,MSE}

public ErrorMeasurement

generalErrorMeasurement=ErrorMeasurement.SquareError;

public ErrorMeasurement overallErrorMeasurement=ErrorMeasurement.MSE;

private int currentRecord=0;

private ArrayList<ArrayList<ArrayList<Double>>> newWeights;

//…
```

}

ეს კლასი ასევე რამდენიმე შესაძლებლობას იძლევა ზოგადი და საერთო ცდომილების გამოსათვლელად. ატრიბუტებმა generalErrorMeasurement და overallErrorMeasurement შეუძლიათ მიიღონ ერთერთი მნიშვნელობა მარტივი შეცდომა, კვადრატული შეცდომა, N-ური ხარისხის შეცდომა (კუბური, მეოთხე, და აშ),ან MSE..ნაგულისხმევი იქნება მარტივი შეცდომა ზოგადი შეცდომისათვის და MSE საერთოსათვის. აღსანიშნავია ორი მნიშვნელოვანი ატრიბუტი ამ კლასში: ესაა currentRecord რომელიც მიუთითებს მინდინარე ჩანაწერის ინდექსს,რომელიც მიეწოდება ნეირონულ ქსელს ტრენინგის დროს და newWeights კუბური მატრიცა სადაც არის ყველა ახალი წონების მნიშვნელობები. currentRecord ატრიბუტი სასარგებლოა ონლაინ ტრეინინგში, და newWeights მატრიცა ეხმარება ნეირონულ ქსელს შეინარჩუნოს ყველა მისი ორიგინალური წონები სანამ ახალები არ გამოანგარიშდება,ეს საშუალებას გვაძლევს ქსელის სისწრაფე იყოს ბევრად სწრაფი ვიდრე ყოველ ჯერზე წონების განახლება.ამ კლასში აღწერილია ძირითადი მეთოდები როგორიცაა: forward,train,calcNewWeights, რომლების შესაბამისად აკეთებენ ქსელისთვის მონაცემის მიწოდებას და მნიშვნელობების გამოთვლას,მეთოდეი შეიცავს ციკლს ,რომელიც ტრიალებს სანამ პირობა არ მიიღწევა ან მაქსიმალური იტერაციის რიცხვი არ მიიღწევა ან საეთო შეცდომა არ გახდება მისაღები,ეს მეთოდი ითვლის ახალ წონებს და იყენებს აქტივაციის ფუნქციის ინტერფეისში განსაზღვრულ derivative() მეთოდს რომელსაც იმპლემენტაციას უკეთებს თითოეული აქტივაციის ფუნქციის კლასი.ერთობლივი სწავლის დროს ვიყენებთ derivativeBatch(),რომელიც თითოეული სკალარის ნაცვლად აბრუნებს წარმოებულების მასივს. newWeights ში ვინახავთ ახალ წონებს,რათა გავლენა არ მოახდინოს მიმდინარე სწავლის პროცესზე და წონები განახლდება მხოლოდ იტერაციის დასრულების შემდეგ.დელტა წესის გარდა არსებობს რამდენიმე სწავლების ალგორითმი მაგალითად ჰებიანის და ადალაინის,მაგრამ მათ აქ არ შევეხებით და მსურველს შეუძლია ინფორმაცია მოიძიოს გამოყენებულ ლიტერატურაში.

მოდით განვიხილოთ ძალიან მარტივი საილუსტრაციო მაგალითი. დავუშვათ, რომ გვინდა ერთი ნეირონის ნეირონულ ქსელს ვასწავლოთ მარტივი წრფივი ფუნქცია:



ჩვენ ვაპირებთ ავაგოთ მონაცემები ამ ქსელისთვის შემდეგი კოდის საშუალებით, რომელიც შეგიძლიათ იპოვოთ NeuralNetDeltaRuleTest კლასის main მეთოდში :

```
Double[][] _neuralDataSet = {
{1.2 , fncTest(1.2)}
, {0.3 , fncTest(0.3)}
, {-0.5 , fncTest(-0.5)}
, {-2.3 , fncTest(-2.3)}
, {1.7 , fncTest(1.7)}
, {-0.1 , fncTest(-0.1)}
, {-2.7 , fncTest(-2.7)} };
int[] inputColumns = {0};
int[] outputColumns = {1};
NeuralDataSet neuralDataSet =
newNeuralDataSet(_neuralDataSet,inputColumns,outputColumns);
```

fncTest განსაზღვრულია შემდეგნაირად:

```
public static double fncTest(double x){
return 0.11*x;
}
```

აღსანიშნავია , რომ ჩვენ ვიყენებთ კლასს NeuralDataSet-მონაცემთა სწორად სტრუქტურირებისათვის. ახლა მოდით დავუკავშიროთ ამ მონაცემთა ნაკრებს ნეირონული ამ ქსელს აქვს ერთი ნეირონი გამომავალში. მოდით გამოვიყენოთ არაწრფივი აქტივაციის ფუნქცია როგორიცაა ჰიპერბოლური ტანგენსი კოეფიციენტით 0.85:

```
int numberOfInputs=1;
int numberOfOutputs=1;
HyperTan htAcFnc = new HyperTan(0.85);
NeuralNet nn = new NeuralNet(numberOfInputs,numberOfOutputs,
htAcFnc);
```

ახლა შევქმნათ DeltaRule ობიექტი და მასთან დავაკავშიროთ ნეირონული ქსელი. შემდეგ. ამის შემდეგ განვსაზღვროთ სწავლის პარამეტრები, როგორიცაა სწავლის ფასი, მინიმალური საერთო შეცდომა და მაქსიმალური რაოდენობა ეპოჩები:

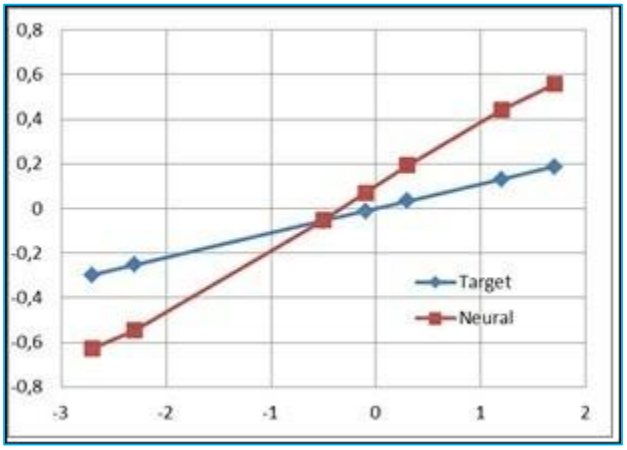
```
DeltaRule deltaRule=new
DeltaRule(nn,neuralDataSet.LearningAlgorithm.LearningMode.ONLINE);
deltaRule.printTraining=true;
deltaRule.setLearningRate(0.3);
deltaRule.setMaxEpochs(1000);
deltaRule.setMinOverallError(0.00001);
```

ახლა მივიღოთ, დაუტრენინგებელი ნეირონული ქსელის პირველი ნეირონული შედეგი deltaRule ობიექტზე forward( ) მეთოდის გამოძახებით:

```
deltaRule.forward();
neuralDataSet.printNeuralOutput();
```

```
Getting the first output of the neural network
Neural:
Neural Output[0]={ 0.44224768996697406}
Neural Output[1]={ 0.19297592401999314}
Neural Output[2]={ -0.053052819745845956}
Neural Output[3]={ -0.5457392112113837}
Neural Output[4]={ 0.5582946870603158}
Neural Output[5]={ 0.07104196712543694}
Neural Output[6]={ -0.6270608140997245}
```

ავაგოთ მიღებული შედეგის გრაფიკი:



სურათი 16.forward მეთოდის გამოძახების შემდეგ

ჩვენ ვიწყებთ ქსელის სწავლებას ონლაინ რეჟიმში. ვასწავლით. ჩვენ ვაყენებთ printTraining ატრიბუტს trueზე, ასე რომ ჩვენ მივიღებთ ეკრანზე პროცესის მსვლელობას:

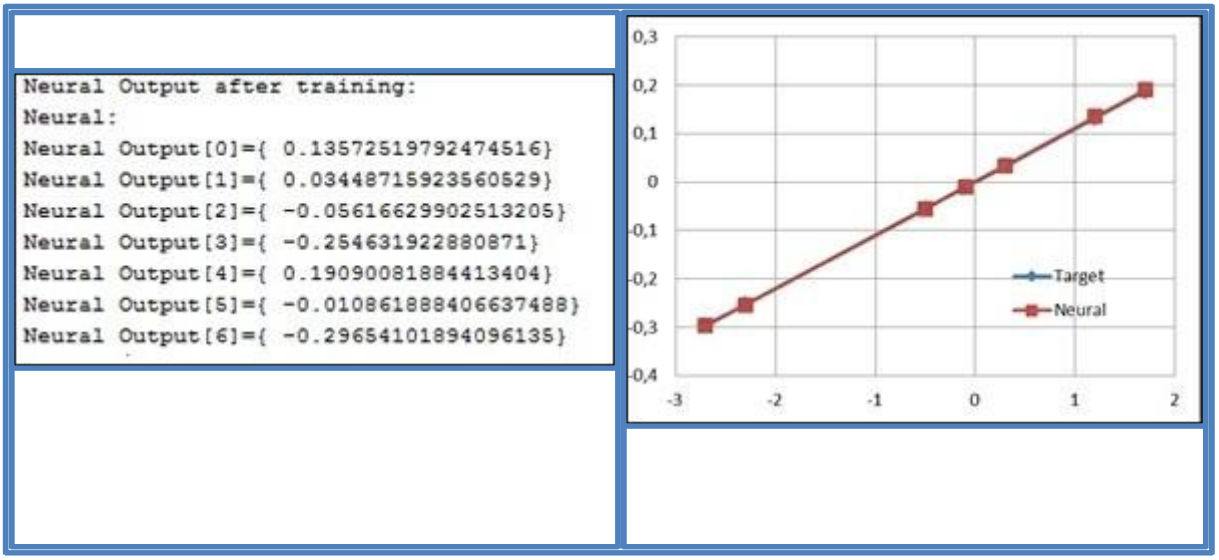
```
1.  System.out.println("Beginning training");
2.  deltaRule.train();
3.  System.out.println("End of training");
4.  if(deltaRule.getMinOverallError()>=deltaRule.getOverallGeneralError()){
5.  System.out.println("Training succesful!");
6.  }else{
7.  System.out.println("Training was unsuccesful");
8.  }
```

```
Beginning training
Epoch=0; Record=0; Overall Error=0.06586810467152877
Epoch=0; Record=1; Overall Error=0.0642007959986327
Epoch=0; Record=2; Overall Error=0.06427692265805388
Epoch=0; Record=3; Overall Error=0.06102840935680022
Epoch=0; Record=4; Overall Error=0.04880202611839414
Epoch=0; Record=5; Overall Error=0.04823124833461199
Epoch=0; Record=6; Overall Error=0.03388528328163059
Epoch=1; Record=0; Overall Error=0.021528920861072397
```

სწავლება იწყება და ცდომილების შესახებ ინფორმაცია გამოდის ყოველი იტერაციის შემდეგ.შევნიშნოთ რომ ცდომილება თანდათან მცირდება:

```
Epoch=4; Record=4; Overall Error=1.8051182588340976E-5
Epoch=4; Record=5; Overall Error=1.6070522271176285E-5
Epoch=4; Record=6; Overall Error=1.3418163790928809E-5
Epoch=5; Record=0; Overall Error=8.344050623786828E-6
End of training
Training succesful!
Overall Error:8.344050623786828E-6
Min Overall Error:1.0E-5
Epochs of training:5
```

5 იტარაციის შემდეგ ცდომილება დადის ჩვენთვის მისაღებ დონეზე.შემდეგ სურათებზე მოცემულია ქსელის შედეგი და მისი შესაბამისი გრაფიკი:



როგორც ვხედავთ ქსელმა წარმატებით ისწავლა! მოდით ვნახოთ წონისა და ბიასის მნიშვნელობა:

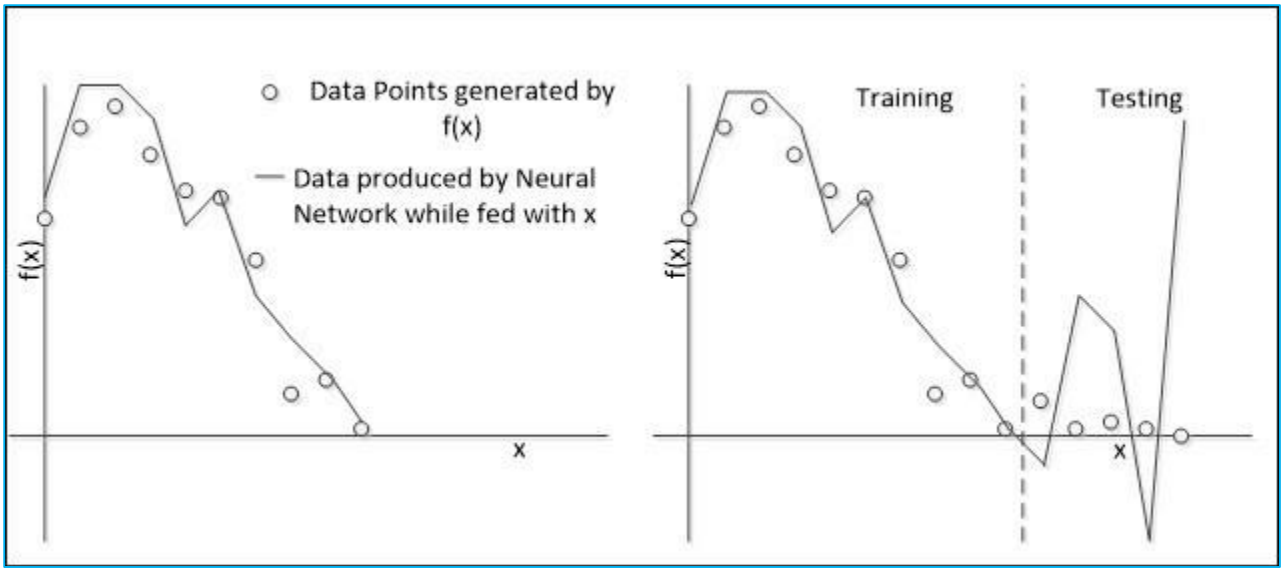
```
1.  weight = nn.getOutputLayer().getWeight(0, 0);
2.  bias = nn.getOutputLayer().getWeight(1, 0);
3.  System.out.println("Weight found:"+String.valueOf(weight));
4.  System.out.println("Bias found:"+String.valueOf(bias));
5.  //Weight found:0.2668421011698528
6.  //Bias found:0.0011258204676042108
```

2.3 ნეირონული ქსელის ტესტირება სწავლების შემდეგ

შედეგის მიღებასთან ერთად არანაკლებ მნიშვნელოვანია მისი ტესტირება.როგორ შეიძლება გავზომოთ რამდენად ეფექტურია ჩვენი ქსელი. ისევე, როგორც გამოცდების დროს სტუდენტებს, ჩვენ უნდა შეამოწმოთ ქსელის პასუხები.სწავლების შემდეგ,მაგრამ სავარაუდოდ, მასწავლებელი არ ჩაუტარებს გამოცდას იგივე კითხვებში რაც უკვე წარადგინა კლასში.არ არსებობს აზრი ვინმეს სწავლის შეფასებისას იმავე მაგალითებით რაც უკვე ცნობილია მოსაწავლისათვის, შესაძლოა მოსწავლეს დაზეპირებული ჰქონდეს მასალა მისის სწავლის ნაცვლად.

ახლა ავხსნათ ეს ნაწილი. ნეირონული ქსელის სწავლის შემდეგ უნდა შევამოწმოთ ნამდვილად ისწავლა თუ არა მან. ტესტირებისთვის ქსელს უნდა მივაწოდოთ განსხვავებული მონაცემები იგივე გარემოდან. ეს აუცილებელია, რადგან, როგორც მოსწავლე, ნეირონულმა ქსელმაც შეიძლება უპასუხოს სწორად მხოლოდ იმ მონაცემებს, რომლებიც მან ისწავლა; ამ მოვლენას უწოდებენ overtraining ჭარბი სწავლება.იმისათვის რომ გადავამოწმოთ სწორად ისწავლა თუ არა ქსელმა, ჩვენ მას უნდა მივცეთ განსხვავებული მონაცემები.

შემდეგი სურათი ასახავს ზედმეტი ტრენინგის პრობლემას. წარმოიდგინეთ, რომ ჩვენი ქსელი განკუთვნილია რაღაც ფუნქციის f (x), მისაახლოებლად რომლის განსაზღვრებაც უცნობია. ნეირონულმა ქსელმა მიიღო რაღაც მონაცემები ამ ფუნქციისაგან და მოგვცა შედეგი მარცხნივ არსებულ ნახაზზე. მაგრამ როდესაც გავაფართოებთ განსაზღრის არეს, მაგალითად, ტესტირების მონაცემთა ნაკრებს დავამატებთ , ჩვენ დავინახავთ, რომ ქსელის პასუხი არ მიყვება მარჯვნივ გამოსახულ ფუნქციას:



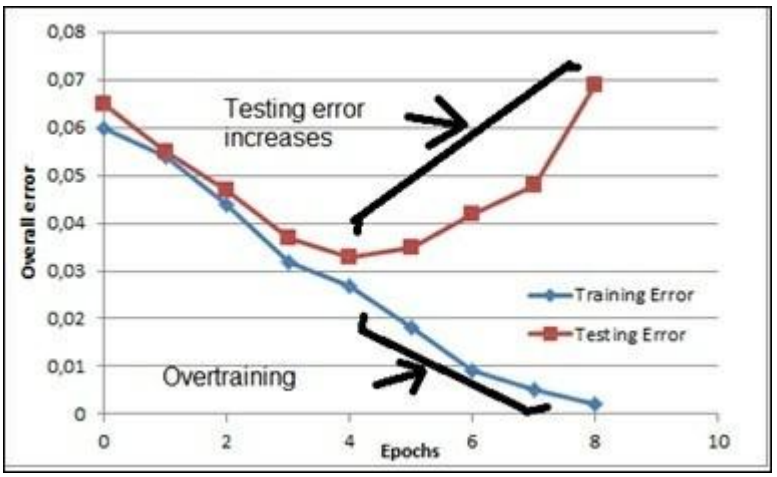
სურათი 17.overtraining

ამ შემთხვევაში, ჩვენ ვხედავთ, რომ ნეირონულმა ქსელმა ვერ ისწავლა მთელი გარემო (ფუნქცია f (x)).ეს ხდება რიგი მიზეზების გამო:

- **ნეირონულ ქსელს გარემოსგან საკმარისი ინფორმაცია არ მიუღია**
- **მონაცემები გარემოდან არის არადეტერმინისტული**
- **.ტრეინინგისა და ტესტირების მონაცემები ცუდად არის განსაზღვრული.**
- **ნეირონულმა ქსელმა იმდენად ისწავლა სატრენინგო მონაცემები ,რომ "დაავიწყდა" ტესტირების მონაცემები**

Overfitting and overtraining:

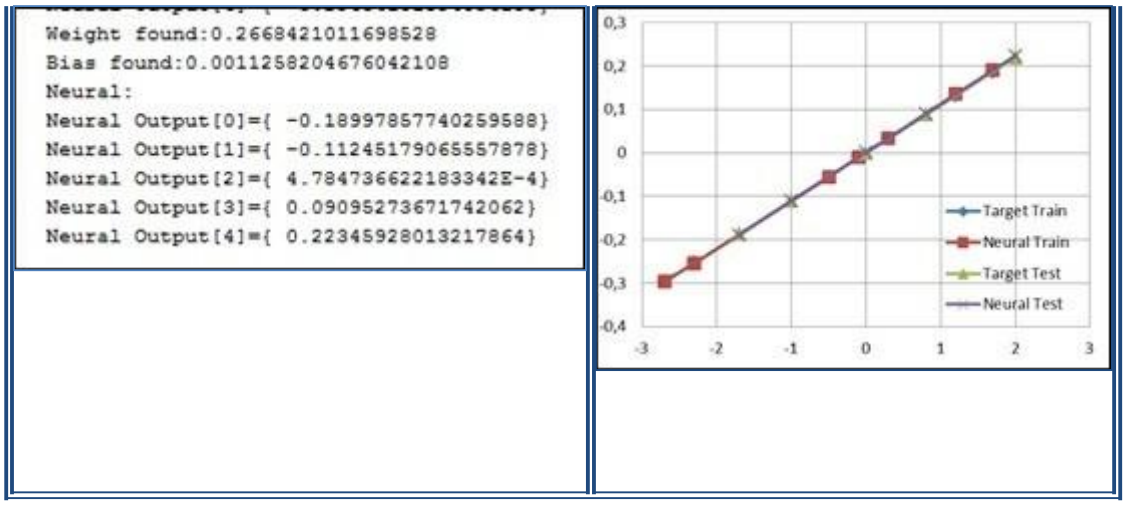
ჩვენს წინა მაგალითში, ნეირონულმა ქსელმა საკმაოდ კარგად ისწავლა. თუმცა, არსებობს overfitting და overtraining-ის რისკი. განსხვავება ამ ორ კონცეფციას შორის საკმაოდ დახვეწილია. overfitting ხდება, როდესაც ნეირონული ქსელი სწავლობს პრობლემის ქცევას ისე ,რომ მას შეუძლია მოგვცეს კარგი შედეგები მხოლოდ სატრენინგო წერტილებზე, რის შედეგადაც ის განზოგადების უნარს კარგავს. Overtraining, რომელიც შეიძლება იყოს მიზეზი overfitting-სა, ხდება, როდესაც ტრენინგის შეცდომა ხდება ბევრად უფრო მცირე, ვიდრე ტესტირების ცდომილება, ან როდესაც ტესტირების შეცდომა იწყებას გაზრდას როდესაც ქსელი განაგრძობს სწავლას.



სურათი 18.Overfitting and overtraining

ერთერთი გზა, რათა თავიდან იქნას აცილებული overtraining და overfitting არის ტესტირების შეცდომის შემოწმება, როდესაც ტრენინგის პროცესი მიმდინარეობს. როდესაც ტესტირების შეცდომა იწყება გაზრდას, დროა შევჩრდეთ.მოდი ვნახოთ, თუ არსებობს ეს მოვლენები ჩვენს მაგალითში. დავამატოთ მეტი მონაცემები და გავტესტოთ:

```
1.  Double[][] _testDataSet ={
2.  {-1.7 , fncTest(-1.7) }
3.  , {-1.0 , fncTest(-1.0) }
4.  , {0.0 , fncTest(0.0) }
5.  , {0.8 , fncTest(0.8) }
6.  , {2.0 , fncTest(2.0) }
7.  };
8.  NeuralDataSet testDataSet = new NeuralDataSet(_testDataSet, ....inputColumns,
9.  outputColumns);
10. deltaRule.setTestingDataSet(testDataSet);
11. deltaRule.test();
12. testDataSet.printNeuralOutput();
```



როგორც გრაფიკიდან ჩანს, ნეირონულმა ქსელმა კარგად შეძლო განზოგადება განზოგადებას. მიუხედავად ამ მაგალითის სიმარტივისა, ჩვენ შეგვიძლია დავინახოთ ნეირონული ქსელის დასწავლის უნარი.

თავი 3. პერცეპტონები და დამკვირვებლიანი სწავლება

ამ თავში, ჩვენ უფრო დეტალურად განვიხილავთ დამკვირვებლიან სწავლებას, რაც ძალიან სასარგებლოა ორ მონაცემთა სიმრავლეს შორის ურთიერთდამოკიდებულების დასადგენად. გარდა ამისა, ჩვენ წარმოვადგენთ პერცეპტონს, ძალიან პოპულარულ ნეირონული ქსელის არქიტექტურას, დამკვირვებლიან სწავლებაში. ამ თავში ასევე წარმოდგენილია მათი გაფართოებული და განზოგადებული ვერსია, ე.წ. მრავალშრიანი პერცეპრონი და მათი ალგორითმი და პარამეტრები.ასევე ვაჩვენებთ მის პროგრამულ რეალიზაციას და განვიხილავთ მაგალითს როგორ შეიძლება მისი მეშვეობით პრობლემების გადაჭრა.

3.1 დამკვირვებლიანი სწავლა(Supervised learning)

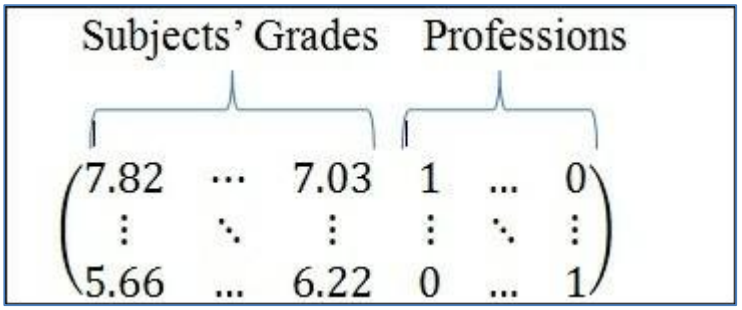
წინა თავში ჩვენ შევისწავლეთ სწავლების პარადიგმები, რომლებიც ეხება ნეირონულ ქსელებს, სადაც დამკვირვებლიანი სწავლება გულისხმობს, რომ არსებობს მიზანი . პრაქტიკაში ჩვენ წარმოვადგენთ შეყვანის მონაცემებს X და სასურველი გამომავალი მონაცემებს YT, მაშინ ჩვენ ვაფასებთ ღირებულებათა ფუნქციას, რომლის მიზანია შემცირდეს შეცდომა შორის ქსელის გამომავალ Y და სამიზნე გამომავალ YT მნიშვნელობებს შორის.დამკვირვებლიან სწავლაში არსებობს ორი ძირითადი კატეგორიის ამოცანა: **კლასიფიკაცია და რეგრესია.**

კლასიფიკაცია - შესაბამისი კლასის პოვნა

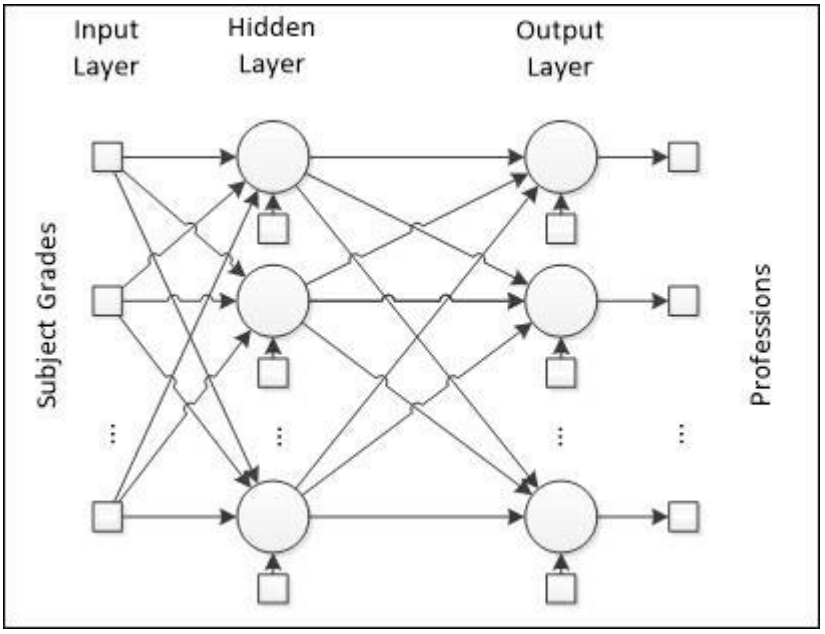
ნეირონული ქსელები ასევე მუშაობენ კატეგორიული მონაცემებით. კლასების და მონაცემთა ნაკრების სიის გათვალისწინებით,ჩვენ გვსურს დავაკლასიფიციროთ ისინი ისტორიული მონაცემების მიხედვით, რომლებიც შეიცავს ჩანაწერებს და მათ შესაბამის კლასს. ქვემოთ მოყვანილი ცხრილი გვიჩვენებს ამ მონაცემთა ნაკრების მაგალითს, მოცემულია საგნების საშუალო ქულები 0-დან 10-მდე:

სტუდენტის Id	საგნები								პროფესია
	English	**Math**	**Physics**	**Chemistry**	**Geography**	**History**	**Literature**	**Biology**	
89543	7.82	8.82	8.35	7.45	6.55	6.39	5.90	7.03	**Electrical Engineer**
93201	8.33	6.75	8.01	6.98	7.95	7.76	6.98	6.84	**Marketer**
95481	7.76	7.17	8.39	8.64	8.22	7.86	7.07	9.06	**Doctor**
94105	8.25	7.54	7.34	7.65	8.65	8.10	8.40	7.44	**Lawyer**
96305	8.05	6.75	6.54	7.20	7.96	7.54	8.01	7.86	**School Principal**
92904	6.95	8.85	9.10	7.54	7.50	6.65	5.86	6.76	**Programmer**

ერთი მაგალითია პროფესიის პროგნოზირება, რომელიც დაფუძნებულია სკოლის ქულებზე. მოდი განვიხილოთ მონაცემთა ნაკრები ყოფილი სტუდენტებისა, რომლებიც ახლა მუშაობენ. ჩვენ შევკრიბეთ თითოეული სტუდენტის საშუალო ქულა თითოეულ საგანში და მისი ამჟამინდელი პროფესია. გაითვალისწინეთ, რომ გამომავალი იქნება პროფესიის სახელი,რომელიც ნეირონულ ქსელს არ შეუძლია მოგვცეს პირდაპირ. ამის ნაცვლად, უნდა დავამატოთ ერთი სვეტი (გამომავალი) თითოეული ცნობილი პროფესიისთვის. თუ სტუდენტმა აირჩია გარკვეული პროფესია, სვეტი,რომელიც შეესაბამება ამ პროფესიას,მისი მნიშვნელობა იქნება ერთი, წინააღმდეგ შემთხვევაში სხვა სვეტების მნიშვნელობა იქნება ნულოვანი.



ახლა ჩვენ გვინდა მოვძებნოთ მოდელი -გავაკეთოთ პროგნოზი რომელ პროფესიისა იქნება სტუდენტი სავარაუდოდ მისი ქულებიდან გამომდინარე, ა. ამ მიზნით, ჩვენ ვაგებთ ნეირონულ ქსელს,რომელსაც აქვს სასკოლო საგნები შემავალ მონაცემებად და ცნობილი პროფესიები გამოსავლად.სამეცნიერო სუბიექტების რაოდენობას, როგორც შედარებითი და ცნობილი პროფესიების რაოდენობას და არჩვითი რაოდენობის ნეირონების რაოდენობა ფარულ შრეში.



სურათი 19.ნეირონული ქსელი პროფესიების პროგნოზირებისათვის

კლასიფიკაციის პრობლემებისათვის, როგორც წესი, მხოლოდ ერთი კლასია თითოეული მონაცემისთვის. ასე რომ გამომავალ შრეში,ნეირონები მოგვცემენ ნულს ან ერთს,ამ დროდ უმჯობესია გამოვიყენოთაქტივაციის ფუნქციები რომელიც გვაძლევენ მნიშვნელობებს 0სა და 1ს შორის. თუმცა, ჩვენ უნდა გავითვალისწინოთ შემთხვევა ,როდესაც ერთზე მეტი ნეირონი გვაძლევს ერთს ანუ გვაძლევს ორ კლასს, ამის თავიდან ასაცილებლად არსებობს მთელი რიგი მექანიზმები მაგალითად, როგორიცაა softmax ფუნქცია აქ გამარჯვებულს მიაქვს ყველა. ამ თემას შევეხებით ჩვენი დაავადების დიაგნოზის კლასიფიკაციის ამოცანის განხილვისას.ტრენინგის შემდეგ, ნეირონულმა ქსელმა გაიგო, თუ რა იქნება ყველაზე სავარაუდო პროფესია მოცემული სტუდენტისთვის მის ქულებთან შესაბამისობაში.

რეგრესია - mapping real inputs to outputs

რეგრესიის იდეა მდგომარეობს გარკვეული ფუნქციის აღმოჩენაში, რომელიც გადაიყვანს შემავალ მონაცემებს გამომავალში. ქვემოთ მოყვანილი ცხრილი გვიჩვენებს დატასეტს რომელიც შედგება k ჩანაწერისგან , რომლებიც შეიცავს m დამოუკიდებელ შემავალ X-ს და მასზე დამოკიდებულ n გამოსავალ მნიშვნელობებს.

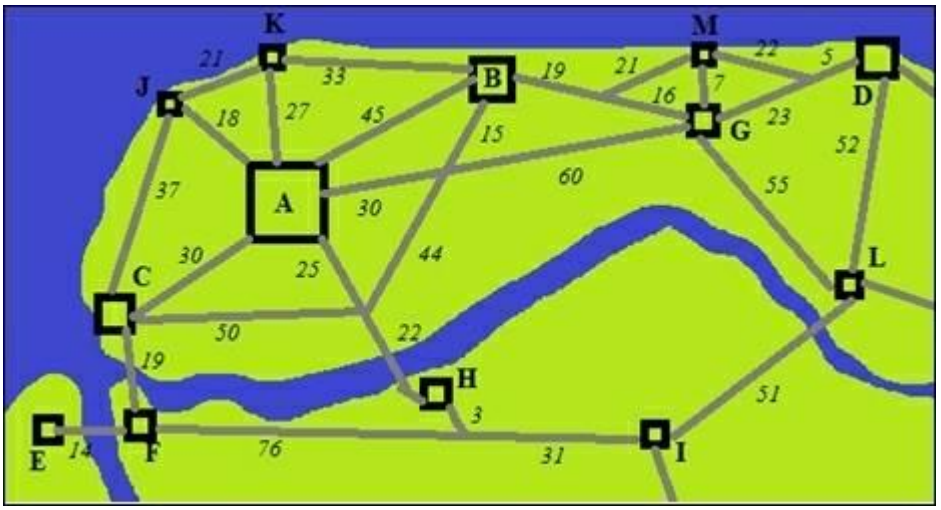
Input independent data				**Output dependent data**			
X1	X2	…	XM	T1	T2	…	TN
x1[0]	x2[0]	…	xm[0]	t1[0]	t2[0]	…	tn[0]
x1[1]	x2[1]	…	xm[1]	t1[1]	t2[1]	…	tn[1]
…	…	…	…	…	…	…	…
x1[k]	x2[k]	…	xm[k]	t1[k]	t2[k]	…	tn[k]

შემდეგი ცხრილი მატრიცულად შემდეგნაირად ჩაიწერება:

$$X_{k,m} = \begin{pmatrix} x_1(0) & x_2(0) & \cdots & x_m(0) \\ x_1(1) & x_2(1) & \cdots & x_m(1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(k) & x_2(k) & \cdots & x_m(k) \end{pmatrix}$$

$$T_{k,n} = \begin{pmatrix} t_1[0] & t_2[0] & \cdots & t_n[0] \\ t_1[1] & t_2[1] & \cdots & t_n[1] \\ \vdots & \vdots & \ddots & \vdots \\ t_1[k] & t_2[k] & \cdots & t_n[k] \end{pmatrix}$$

კლასიფიკაციისგან განსხვავებით, გამომავალი მნიშვნელობები რიცხვითია და არა კლასები ან იარლიყები.აქაც არსებობს ისტორიული მონაცემთა ბაზა, რომელიც შეიცავს ზოგიერთი ქცევის ჩანაწერებს, და ჩვენ გვინდა ნეირონულ ქსელს ვასწავლოთ იგი.ერთი მაგალითია ავტობუსის ბილეთის ფასების პროგნოზირება ორ ქალაქს შორის. ამ მაგალითში ჩვენ ვაგროვებთ ინფორმაციას ქალაქების ჩამონათვალში და ავტობუსების მიმდინარე ბილეთის ფასებს რომლებიც გადიან ერთიდან და ჩადიან სხვა ქალაქში. ჩვენ ვაწვდით შემავალ მონაცემებად ქალაქის თვისებებს როგორიცაა მანძილის ან / და დროს მათ შორის და გამომავალ მონაცემად ბილეთის ფასს.



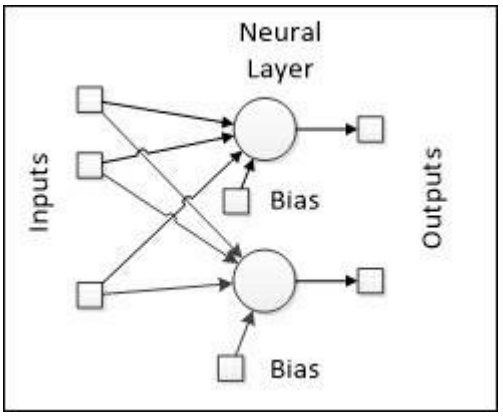
სურათი 20.ბილეთის ფასის პროგნოზირება ქალაქებს შორის

მონაცემთა ბაზის სტრუქტურირების შემდეგ, ჩვენ უნდა განვსაზღვროთ ნეირონული ქსელი, რომელიც შეიცავს იგივე რაოდენობის თვისებებს (გამრავლებული ორზე, ორი ქალაქის გამო) პლუს მარშრუტი შეყვანის მონაცემებში, ერთი გამომავალი, და არჩევითი რაოდენობის ნეირონები ფარულ შრეში. წინა შემთხვევიდან გამომდინარე აქ არ გვიწევს მონაცემების გარდაქმნა,რადგან რიცხვით მონაცემებთან გვაქვს საქმე.ეს ნეირონული ქსელი მოგვცემს სავარაუდო ფასს ორ ქალაქს შორის, რომელსაც არ ემსახურება არცერთი სატრანსპორტო კომპანია.როგორც ცხრილიდან ჩანს გვექნება 9 შემავალი მონაცემი:

Features city of origin			Features city of destination			Features of the way between			Ticket fare
Population	**GDP**	**Route s**	**Population**	**GDP**	**Routes**	**Distance**	**Time**	**Stops**	
500,000	4.5	6	45,000	1.5	5	90	1,5	0	15
120,000	2.6	4	500,000	4.5	6	30	0,8	0	10
30,000	0.8	3	65,000	3.0	3	103	1,6	1	20
35,000	1.4	3	45,000	1.5	5	7	0.4	0	5
…									
120,000	2.6	4	12,000	0.3	3	37	0.6	0	7

3.2 პერცეპტონი

პერცეპტონი არის ყველაზე მარტივი ნეირონული ქსელის არქიტექტურა. 1957 წელს ფრანკ როზენბლატის მიერ დაპროექტებული, მას ნეირონების მხოლოდ ერთი შრე აქვს, იღებს შემავალ მონაცემებს და ითვლის გამოსავალს. ეს იყო ერთერთი პირველი ნეირონული ქსელის წარმოდგება,რომელმაც ყურადღება მიიპყრო, განსაკუთრებით მათი სიმარტივის გამო:



სურათი 21.მარტივი პერცეპტონი

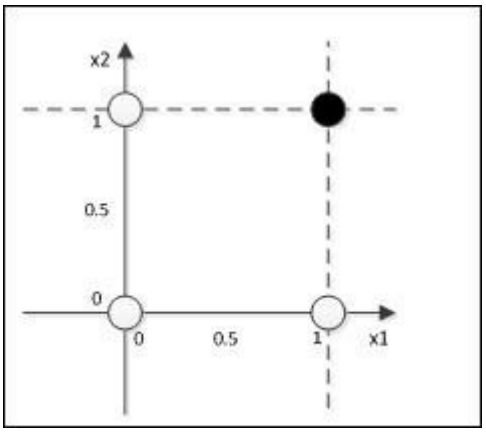
ჩვენს Java იმპლემენტაციაში, ეს ილუსტრირებულია ერთი ნეირონული შრით (გამომავალი შრე). მომდევნო კოდი ქმნის პერცეპტონს სამი შემავლით და ორი გამომავალით, რომელსაც აქვს წრფივი გამომავალი ფუნქცია:

```
1.  int numberOfInputs=3;
2.  int numberOfOutputs=2;
3.  Linear outputAcFnc = new Linear(1.0);
4.  NeuralNet perceptron = new NeuralNet(numberOfInputs,numberOfOutputs,outputAcFnc);
```

თუმცა, მეცნიერებს არ დასჭირდათ დიდი დრო იმის მისახვედრად , რომ პერცეპტონის ნეირონული ქსელი შეიძლება იყოს გამოყენებული მხოლოდ მარტივი ამოცანების ამოსახსნელად. იმ ხანად, ნეირონული ქსელები მარტივი კლასიფიკაციის პრობლემების გადასაჭრელად გამოიყენებოდა, მაგრამ პერცეპტონი, როგორც წესი, ვერაფერს უხერხებდა

შედარებით რთულ მონაცემებს.მოდით განვიხილოთ ერთი ძალიან მარტივი მაგალითი (და AND ფუნქცია) ამ პრობლემის გასაგებად.

მაგალითად შედგება და ფუნქციისაგან, რომელიც იღებს ორი მონაცემს, x1 და x2. ეს ფუნქცია შეიძლება განისაზღვროს გრაფიკულად შემდეგნაირად:



სურათი 22.და ფუნქცია

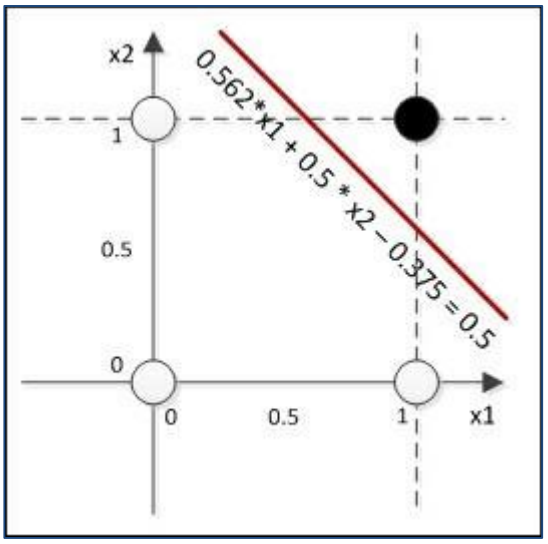
ახლა მოდით ვნახოთ თუ როგორ სწავლობს ნეირონულ ქსელი პერცეპტონის მეშვეობით.განვიხილოთ წყვილი ორი წონისა, W1 და W2, თავდაპირველად ისინი უდრიან 0.5-ს, და ბიასიც აგრეთვე 0.5 . ვივარაუდოთ.სწავლის ფასი η შეადგენს 0.2:

Epoch	**x1**	**x2**	**w1**	**w2**	**b**	**y**	**t**	**E**	**Δw1**	**Δw2**	**Δb**
1	0	0	0.5	0.5	0.5	0.5	0	-0.5	0	0	-0.1
1	0	1	0.5	0.5	0.4	0.9	0	-0.9	0	-0.18	-0.18
1	1	0	0.5	0.32	0.22	0.72	0	-0.72	-0.144	0	-0.144
1	1	1	0.356	0.32	0.076	0.752	1	0.248	0.0496	0.0496	0.0496
2	0	0	0.406	0.370	0.126	0.126	0	-0.126	0.000	0.000	-0.025
2	0	1	0.406	0.370	0.100	0.470	0	-0.470	0.000	-0.094	-0.094
2	1	0	0.406	0.276	0.006	0.412	0	-0.412	-0.082	0.000	-0.082
2	1	1	0.323	0.276	-0.076	0.523	1	0.477	0.095	0.095	0.095
…	…										
89	0	0	0.625	0.562	-0.312	-0.312	0	0.312	0	0	0.062

89 ეპოჩის შემდეგ, ჩვენ ქსელი გვაძლევს სასურველ მნიშვნელობასთან ახლო შედეგს. ამ მაგალითში გამომაცალი შედეგები არის ორობითი (ნულოვანი ან ერთი), ჩვენ შეგვიძლია ვიგულისხმოთ, ქსელის მიერ წარმოებული ნებისმიერი მნიშვნელობა, რომელიც 0.5-ზე ნაკლებია 0-ით,და ნებისმიერი მნიშვნელობა 0.5-ზე მეტი 1-ით.მას აქვს შემდეგი სახე.

$$Y = x_1 w_1 + x_2 w_2 + b = 0.5$$

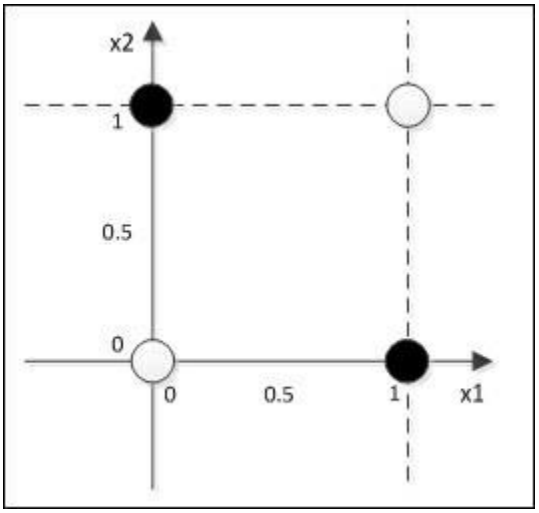
საბოლოო წონები და ბიასი არის w1 = 0.562, w2 = 0.5 და b = -0.375, რომელიც გამოსახულია შემდეგ გრაფიკზე:



სურათი 23.და ფუნქციის სწავლება

ეს საზღვარი არის ქსელის მიერ მოცემული კლასიფიკაციის განმარტება. თქვენ ხედავთ, რომ საზღვარი არის წრფივი, იმის გათვალისწინებით, რომ ფუნქცია ასევე წრფივია. ამდენად, პერცეპტონის ქსელი მართლაც შესაფერისია პრობლემებისა, რომლელთა გაყოფაც წრფივად შეიძლება.

XOR- ის შემთხვევა

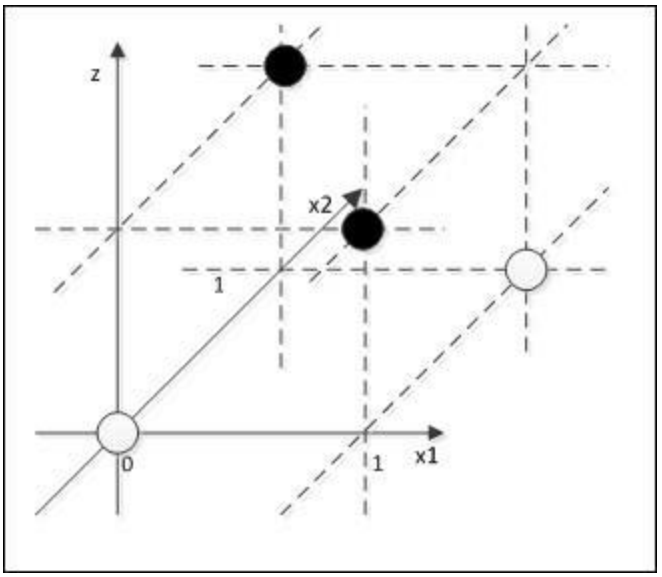


სურათი 24.XOR შემთხვევა

ჩვენ ვხედავთ, რომ ორ განზომილებაში შეუძლებელია გავავლოთ ხაზი რომ გამოვყოთ ეს ორი კლასი. რა მოხდება, თუ შევეცდებით ამ ფუნქციის დასწავლას ერთ შრიანი პერცეპტონის მიერ? დავუშვათ, შევეცადეთ, ვნახოთ, რა მოხდა შემდეგ ცხრილში:

Epoch	x1	x2	w1	w2	b	y	t	E	Δw1	Δw2	Δb
1	0	0	0.5	0.5	0.5	0.5	0	-0.5	0	0	-0.1
1	0	1	0.5	0.5	0.4	0.9	1	0.1	0	0.02	0.02
1	1	0	0.5	0.52	0.42	0.92	1	0.08	0.016	0	0.016
1	1	1	0.516	0.52	0.436	1.472	0	-1.472	-0.294	-0.294	-0.294
2	0	0	0.222	0.226	0.142	0.142	0	-0.142	0.000	0.000	-0.028
2	0	1	0.222	0.226	0.113	0.339	1	0.661	0.000	0.132	0.132
2	1	0	0.222	0.358	0.246	0.467	1	0.533	0.107	0.000	0.107
2	1	1	0.328	0.358	0.352	1.038	0	-1.038	-0.208	-0.208	-0.208
…	…										
127	0	0	-0.250	-0.125	0.625	0.625	0	-0.625	0.000	0.000	-0.125

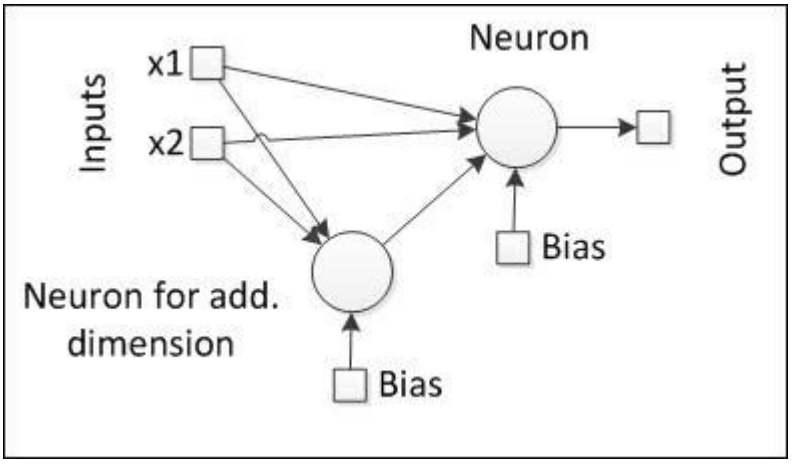
პერცეპტონმა ვერ იპოვა წყვილი წონებისა, რომელიც შეამცირებდა შეცდომას 0.625 .ეს შეიძლება აიხსნად მათემატიკურად, რადგან უკვე აღვნიშნეთ ჩარტში, რომ ეს ფუნქცია არ შეიძლება იყოს წრფივად სეპარირებადი 2 განზომილებაში. რა მოხდება თუ დავამატებთ სხვა განზომილებას? მოდით ვნახოთ ჩარტში სამი განზომილება:



სურათი 25.XOR შემთხვევა 3 განზომილებაში

სამგანზომილებიან სივრცეში შესაძლებელია ზედაპირის გავლება,რომელიც გამოყოფს ამ კლასებს, იმ პირობით, რომ ეს დამატებითი განზომილება სწორად გარდაქმნის შეყვანილ მონაცემებს. ეს კარგია, მაგრამ ახლა არის დამატებითი პრობლემა: როგორ შეგვიძლია მივიღოთ ეს დამატებითი განზომილება, ჩვენ ხომ გვაქვს მხოლოდ ორი შემავალი ცვლადი? ერთი მიდგომა აშკარაა, რომ დავამატოთ მესამე ცვლადი, როგორც წარმოებული იქნება ამ ორი ორიგინალი

შემავალი ცვლადიდან. და ამ მესამე ცვლადის (წარმოებული) დამატებით, ჩვენი ნეირონული ქსელი მიიღებს შემდეგი ფორმას:

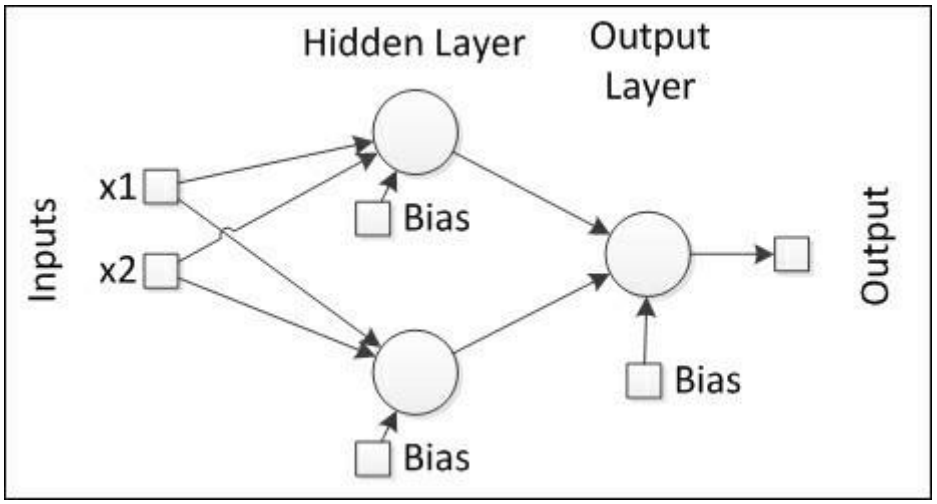


სურათი 26.დამატებული მესამე ნეირონი

ახლა პერცეპტონს აქვს სამი შემავალი, ერთი მათგანი შედგენილია დანარჩენი ორისაგან. ამას ასევე მივყავართ ახალ კითხვასთან: როგორ უნდა დამუშავდეს ეს კომპოზიცია? ჩვენ ვხედავთ, რომ ამ კომპონენტმა შეიძლება იმოქმედოს ნეირონად, რაც ნეირონულ ქსელს აძლევს ჩაწყობილი არქიტექტურას. თუ ასეა, კიდევ ერთიახალი კითხვა ისმება: როგორ შეიძლება ამ ახალი ნეირონის წონების სწავლა , რადგან შეცდომა არის გამომავალ ნეირონზე?

3.3 მრავალშრიანი პერცეპტონი

როგორც ვნახეთ წინა მაგალითში შემთხვევა ,როდესაც ნიმუშები არ არის წრფივად განცალკევებადი.ეს გვიბიძგებს უფრო ღრმად ჩავიდეთ ნეირონული ქსელის არქიტექტურის დეტალებში.პრობლემა გადავჭერით მრავალშრიანი პერცეპტონით.როგორც თავიდან ვთქვით ნეირონული ქსელები შედგება შრეებისაგან.თითოეული შრე იღებს და აბსტრაქციას უკეთებს ინფორმაციას და გარდაქმნის მას სხვა განზომილებისა ან/და სხვა ფორმის მონაცემებში.XOR-ის შემთხვევაში ჩვენ ვიპოვეთ გამოსავალი მესამე კომპონენტის დამატებაში.რომელიც საშუალებას იძლევა წრფივი განცალკევებისა.მაგრამ რჩება რამდენიმე შეკითხვა იმაზე, თუ როგორ გამოითვლება ეს მესამე კომპონენტია. ახლა განვიხილოთ იგივე გადაწყვეტა, როგორც ორი შრიანი პერცეპტონი:



სურათი 27.ორშრიანი პერცეპტონი

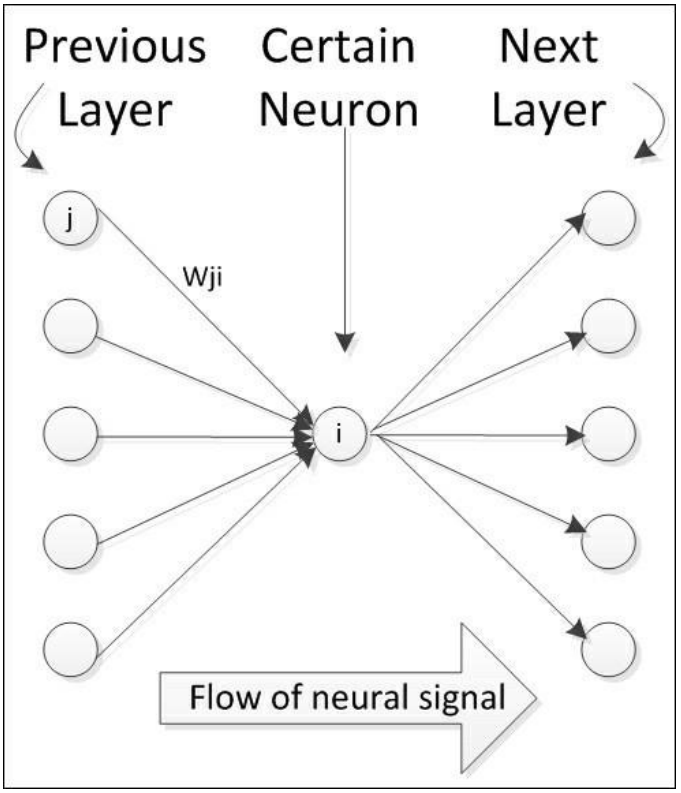
ახლა ჩვენ გვაქვს სამი ნეირონი ერთის ნაცვლად, მაგრამ გამომავალში ინფორმაცია გადაცემული წინა შრიდან გარდაიქმნება სხვა განზომილებასა ან ფორმაში.თეორიულად შესაძლებელი იქნება რომ ამ წერტილებს შორის წრფივი თანაფარდობა დავამყაროთ. თუმცა, კითხვა პირველი შრის ნეირონების წონების პოვნაზე უპასუხოდ რჩება , შეგვიძლია ჩვენ იმავე სასწავლო წესით მივუდგეთ

ამ ნეირონებსაც? ამ საკითხის მოგვარება განზოგადებულ დელტას წესის მეშვეობითაა შესაძლებელი.მრავალშრიან პერცეპტონებს შეიძლება ჰქონდეთ ნებისმიერი რაოდენობის შრეები და ასევე ნებისმიერი რაოდენობის ნეირონები თითოეულ შრეში.აქტივაციის ფუნქციები შეიძლება განსხვავდებოდეს ნებისმიერი შრის მიხედვით. MLP ქსელი ჩვეულებრივ შედგება მინიმუმ ორი შრისაგან, ერთი გამომავალი და ერთი ფარული შრე.*ზოგიერთ ლიტერატურაში შემავალი შრეც ჩვეულებრივ შრედ ითვლება,მაგრამ აქ ჩვენ მას განვიხილავთ მხოლოდ, როგორც ინფორმაციის მიღების საშუალება და მას წონები არ გააჩნია.სატრენინგოდ ეფექტურ შრეებად მიჩნეულია მხოლოდ ფარული და გამომავალი შრე,*

ფარული შრე იმიტომ ოწოდება ასე , რომ ის, ფაქტობრივად, მალავს მისი შედეგებს გარე სამყაროსგან. ფარული შრეები შეიძლება დაკავშირებულ იქნეს სერიებად ნებისმიერი რაოდენობით, რითაც ქმნიან ღრმა ნეირონულ ქსელს. თუმცა,რაც უფრო მეტი შრეების ნეირონულ ქსელში , ნელი იქნება, როგორც სწავლა ისე მისი მუშაობა. მათემატიკურად შესწავლილია რომ ნეირონული ქსელი ერთი ან ორი ფარული შრით თითქმის ისევე კარგად სწავლობს ,როგორც ღრმა ნეირონული ქსელები ათობით ფარული შრით. ეს დამოკიდებულია რამდენიმე ფაქტორზე.

რეკომენდირებულია რომ აქტივაციის ფუნქცია ფარული შრეებში იყოს არაწრფივი. განსაკუთრებით როცა გამომავალი შრის აქტივაციის ფუნქცია წრფივია,. წრფივი ალგებრის თანახმად, წრფივი აქტივაციის ფუნქციის ქონა ყველა შრეში ნიშნავს მხოლოდ ერთ შრეს, რადგან დამატებითი ცვლადები შრეებში იქნება წრფივი კომბინაცია წინა შრეებისა. ჩვეულებრივ, გამოიყენება აქტივაციის ფუნქციები, როგორიცაა ჰიპერბოლური ტენგენსი ან სიგმოიდი, რადგანი ისინი წარმოებადი არიან.

MLP feedforward ქსელში ერთ კონკრეტულ ნეირონი i იღებს მონაცემებს წინა შრის ნეირონი j-დან და გადასცემს გამომავალ მნიშვნელობას შემდეგი შრის ნეირონ k-ში როგოც სურათზეა:



სურათი 28. MLP feedforward

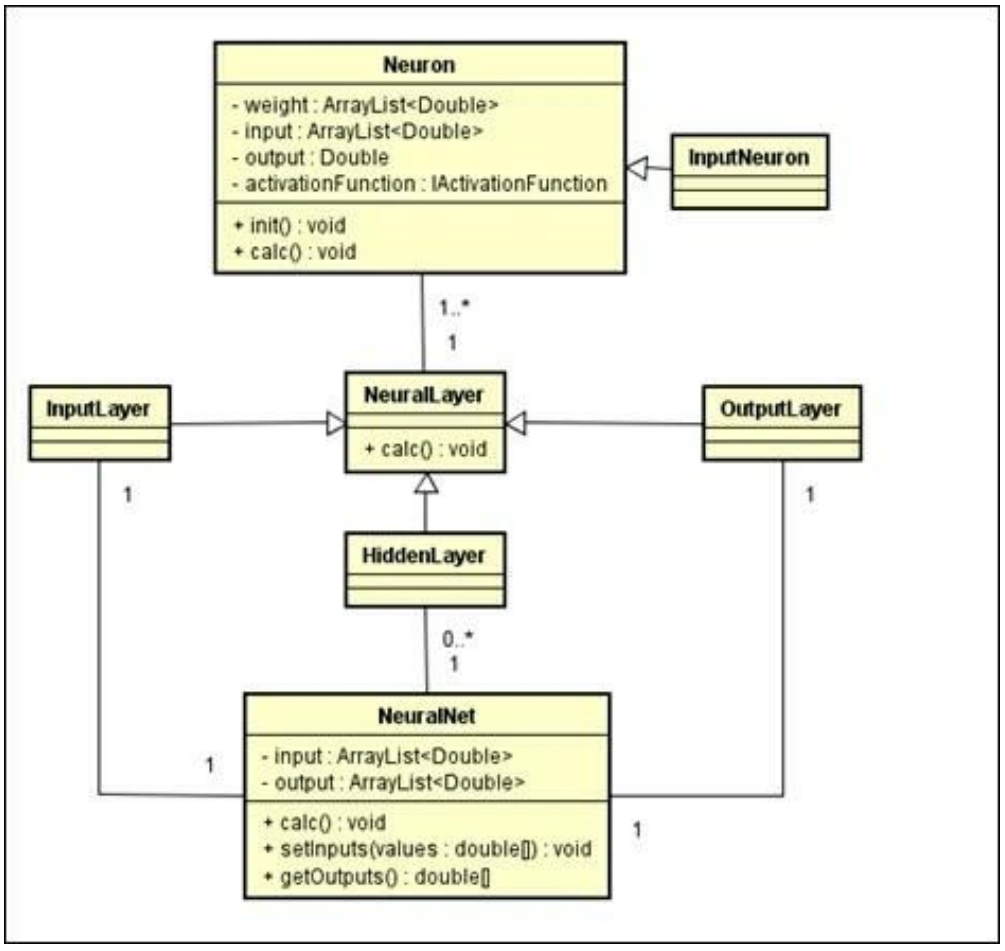
მათემატიკურად ეს გვაძლევს შემდეგ რეკურსიას:

$$y_o = f_o\left(\sum_{i=1}^{n_{h_l}} w_i f_i\left(\sum_{j=1}^{n_{h_{l-1}}} w_{ij} f_j\left(\sum_{k=1}^{n_{h_{l-2}}} w_{jk} f_k(\cdots) + b_j\right) + b_i\right) + b_o\right)$$

აქ, yo არის ქსელის გამომავალი (თუ მრავალი გამოსავალია, ჩვენ შეგვიძლია yo ჩაანაცვლოთ y- ით,რომელიცწარმოადგენს ვექტორს); fo არის გამომავალის აქტივაციის ფუნქცია; l არის ფარული შრეების რაოდენობა; nhi არის ნეირონების რიცხვი ფარული შრეშიI; wi არის წონა დამაკავშირებელი ბოლო ფარული შრის i -ური ნეირონისა გამოსავალთან; fi არის i ნეირონის აქტივაციის ფუნქცია; და bi არის i ნეირონის ბიასი. შეიძლება დავინახოთ, რომ ეს განტოლება უფრო იზრდება როცა, იზრდება შრეების რაოდენობა. ბოლო ჯამში იქნება შენავალი xi მნიშვნელობები.

MLP- ში ნეირონებს შეუძლიათ სიგნალების გადასცენ არა მხოლოდ ნეირონების მომდევნო შრეებში feedforward network), არამედ ნეირონებს იმავე ან წინა შრეებში(feedback or recurrent). ეს ქცევა საშუალებას იძლევა ნეირონულმა ქსელმა შეინარჩუნოს მდგომარეობა რაღაც მონაცემთა მიმდევრობაზე. ეს თვისება განსაკუთრებით მნიშვნელოვანია , როდესაც საქმე ეხება დროის სერიებს ან ხელნაწერის ამოცნობას. რეკურენტული ქსელები ჩვეულებრივრ უფრო რთული დასატრენინგებელია და საბოლოოდ კომპიუტერს ამოეწურება მეხსიერებია მისი შესრულებისას. არსებობს რეკურენტული ქსელების არქიტექტურა უფრო უკეთესი, ვიდრე MLP- ები, როგორიცაა ელმან, ჰოპფილდი,Bidirectional RNNs (რეკურენტული ნეირონული ქსელები და სხვა).

MLP-ს იმპლემენტაცია: ჯერჯერობით ჩვენ ავაგეთ კლასების შემდეგი იარერქია:

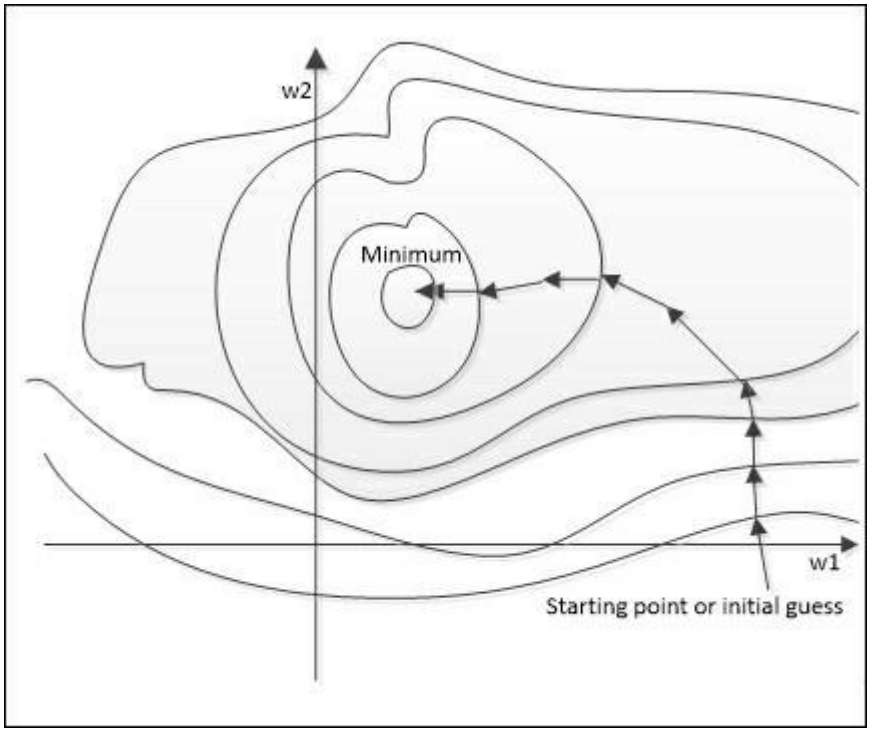


მოვიყვანოთ კოდი,როგორ შეიძლება შევქმნათ მრავალშირანი ნეირო

```
int numberOfInputs=3; int numberOfOutputs=1;
int[] numberOfHiddenNeurons={5};
Linear outputAcFnc = new Linear(1.0);
Sigmoid hiddenAcFnc = new Sigmoid(1.0);
NeuralNet neuralnet = new NeuralNet(numberOfInputs, numberOfOutputs,
numberOfHiddenNeurons, hiddenAcFnc, outputAcFnc);
```

MLP-ს სწავლება:

მრავალშრიანი პერცეპტონის ქსელი სწავლობს დელტა წესზე დაფუძლებით, რომელიც ასევე შთაგონებულია გრადიენტური ოპტიმიზაციის მეთოდით. გრადიენტური მეთოდი ფართოდ გამოიყენება მოცემული ფუნქციის მინიმუმის ან მაქსიმუმის დასადგენად :



სურათი 29.გრადიენტის მეთოდი

ეს მეთოდი გულისხმობს სიარულს იმ მიმართულებით, სადაც ფუნქციის გამოსავალი უფრო მაღალია ან უფრო დაბალი რაც დამოკიდებულია შესაბამის კრიტერიუმზე. ეს იდეა განიხილება დელტა წესში:

$$\Delta w_i = \eta\left(t(k) - y(k)\right)x_i[k]\,g'\left(h_i(k)\right)$$

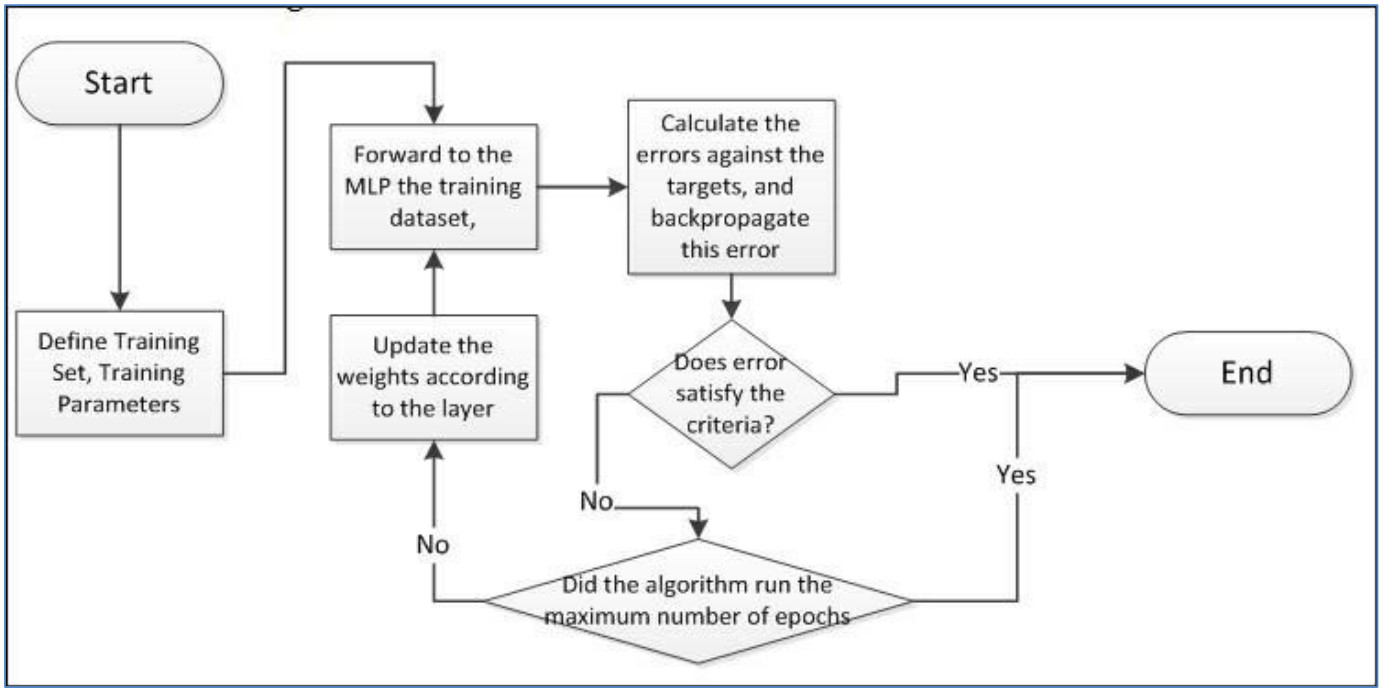
ფუნქცია, რომლის მინიმიზაციისასაც ცდილობს დელტა წესი, არის ცდომილება ნეირომული ქსელის გამოსავალსა და სამიზნე მნიშვნელობას შორის.საძიებელი პარამეტრები არიან ნეირონული წონები. ეს არის უფრო კარგი სწავლების ალგორითმი შედარებით პერცეპტონთან, რადგან იგი ითვალისწინებს აქტივაციის ფუნქციის წარმოებულს g '(h), რომელიც მიუთითებს იმ მიმართულებაზე, სადაც ფუნქცია ყველაზე მეტად.

3.4 უკუგავრცელების ალგორითმი (backpropagation)

მიუხედავად იმისა, რომ დელტა წესი კარგად მუშაობს ნეირონულ ქსელებზე, რომელსაც მხოლოდ გამომავალი და შემავალი შრეები აქვს. MLP ქსელებისათვის , სუფთა დელტა წესი არ შეიძლება გამოყენებული ფარული შრეების ნეირონების გამო. ამ საკითხის დასაძლევად, 1980-იან წლებში, რუმელჰარტმა და სხვებმა ახალი ალგორითმით წარმოადგინეს,რომელიც ასევე შთაგონებული იყო გრადიენტის მეთოდით,რომელიც უკუგავრცელები მეთოდი ეწოდება.ეს ალგორითმი არის MLP- ებისთვის დელტა წესის განზოგადება.დამატებითი შრეები გვაძლევს მეტი მონაცემების აბსტრაქციის საშუალებას გარემოსგან.ამან უბიძგა მეცნიერებს განევითარებინათ სასწავლო ალგორითმი, რომელმაც შეიძლება სწორად შეცვალოს წონები

ფარულ შრეში. გრადიენტის მეთოდზე, დაყრდნობით გამომავალი შეცდომა იქნება (უკან) გავრცელებული წინა შრეებზე, რაც შესაძლებელ ხდის წონები განახლდეს იმავენაირად , როგორც დელტა წესში.

ალგორითმს სქემატურად შემდეგნაირად აღიწერება:



სურათი 30.უკუგავრცელების ალგორითმის სქემა

მეორე ნაბიჯი არის თვითონ უკუგავრცელება.ის აკეთებს წონის კორექტირებას გრადიენტის მიხედვით, რომელიც დელტას წესის საფუძველია:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial h_i}\frac{\partial h_i}{\partial w_{ji}} = (t-y)\,f'(h_i)\,x_j$$

აქ არის E შეცდომა, wji არის წონა ნეირონ i და j-ს შორის, oi არის გამომავალი i-ური ნეირონის და hi არის შეწონილი ჯამი, რომამ ნეირონის შემავლებისა ,სანამ ის გადაეცემა აქტივაციის ფუნქციას. შევნიშნოთ რომ oi = f (hi), f აქტივაციის ფუნქციაა.

ფარული შრეების განახლებისთვის ეს ოპერაცია ცოტა უფრო გართულებულია, რადგან ცდომილებას განვიხილავთ , როგორც ყველა ნეირონის ფუნქცია,გასაახლებელ წონასა და გამოსავალს შორის. ამ პროცესის ხელშესაწყობად ,ჩვენ უნდა გამოვიკვლიოთ მგრძნობელობა უკუგავრცელების ცდომილებისა:

$$\delta_i = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial h_i}$$

წონის განახლება იქნება შემდეგნაირად:

$$\Delta w_{ji} = -\eta\frac{\partial E}{\partial w_{ji}} = \eta\delta_i x_j$$

უკუგავრცელების ცდომილების გამოთვლა განსხვავდება გამომავალი და ფარული შრეებისათვის

Backpropagation გამომავალი შრისათვის:

$$\delta_i = (o_i - t_i) f'(h_i)$$

აქ არის oi i-ური გამოსავალი, ti არის სასურველი გამოსავალი, f '(hi) არის წარმოებული გამომავალი შრის აქტივაციის ფუნქცისა და hi არის შეწონილი ჯამი i-ური ნეირონისა.

Backpropagation ფარული შრისათვის:

$$\delta_i = \sum_l \delta_l w_{il} f'(h_i)$$

აქ l არის შემდეგი შრის ნეირონი და wil არის წონა, რომელიც აერთებს მიმდინარე i-ურ ნეირონს შემდეგი შრის l-ურ ნეირონთან.

სიმარტივისათვის , ჩვენ აქ არ აღვწერთ სრულად თუ როგორ გამოდის უკუგავრცელების განტოლებებს. დაინტერესებული მკითხველისათვის კონსულტაციისთვის შეიძლება მივმართოთ გამოყენებულ ლიტერატურაში არსებულ წყაროებს.

მომენტი:

როგორც ნებისმიერი გრადიენტის მეთოდი, არსებობს რისკი რომ მოვხვდებით ლოკალურ მინიმუმში.. ამ რისკის შესამცირებლად,ჩვენ შეგვიძლია დაამატოთ კიდევ ერთი ტერმი წონის განახლებისას , რომელსაც ქვია მომენტი.იგი ითვალისწინებს წონის ბოლო ვერსიას :

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}(k) + \mu \Delta w_{ji}(k-1)$$

აქ, μ არის მომენტის მნიშვნელობა და $\Delta w_{ji}(k-1)$ არის ბოლო დელტა წონა. ეს აძლევს დამატებით ბიჯს განახლებისას, რის შედეგადაც ხდება ოსცილაციებისთვის თავის არიდება ცდომილების ჰიპერსივრცეში.

ახლა გადავიდეთ ალგორითმის იმპლემენტაციაზე: განვსაზღვროთ კლასის backpropagation neural.learn პაკეტი-ში.რადგან ეს ალგორითმი არის DeltaRule- ის განზოგადება, ამ კლასს შეუძლია მემკვიდრეობით მიიღოს და გადაფაროს,რომელიც განსაზღვრულია დელტა წესით. ამ კლასში შედის სამი დამატებითი ატრიბუტი:მომენტის მაჩვენებელი, დელტა ნეირონი და ბოლო დელტა წონების მასივი:

```
1.  public class Backpropagation extends DeltaRule { private double MomentumRate=0.7;
2.  public ArrayList<ArrayList<Double>> deltaNeuron;
3.  public ArrayList<ArrayList<ArrayList<Double>>> lastDeltaWeights;
4.  ....
5.  }
```

კონსტრუქტორს ექნება იგივე არგუმენტები, რაც DeltaRule კლასს.მასში ემატება მეთოდების გამოძახება deltaNeuron და lastDeltaWeights მასივების ინიციალიზაციისათვის.

```
1.  public Backpropagation(NeuralNet _neuralNet, NeuralDataSet _trainDataSet,
2.  DeltaRule.LearningMode _learningMode){
3.  super(_neuralNet,_trainDataSet,_learningMode);
4.  initializeDeltaNeuron();
5.  initializeLastDeltaWeights();
6.  }
7.
```

train() მეთოდი იმუშავებს ანალოგიურად, როგორც DeltaRule კლასში; დამატებითი კომპონენტი არის უკუგავრცელების ბიჯი , რომლითაც ხდება ცდომილების უკუგავრცელება ნეირონულ შრეებში შემავალ ფენამდე:

```
1.  @Override
2.  public void train() throws NeuralException{ neuralNet.setNeuralNetMode(NeuralNet.NeuralNetMode
    .TRAINING); epoch=0; // initialization of epoch
3.  int k=0; // first training record
4.  currentRecord=0; // this attribute keeps track of which record
5.  // is currently under processing in the training
6.  forward();// initial forward step to determine the error
7.  forward(k); // forward for backpropagation of first record error
8.  while(epoch<MaxEpochs && overallGeneralError>MinOverallError){
9.  backward(); // backward step switch(learningMode){
10. case BATCH:
11. if(k==trainingDataSet.numberOfRecords-1) applyNewWeights(); // batch update
12. break; case ONLINE:
13. applyNewWeights(); //online update
14. }
15. currentRecord=++k; // moving on to the next record if(k>=trainingDataSet.numberOfRecords){ //i
    f it was the last
16. k=0;
17. currentRecord=0; // reset to the first
18.
19. epoch++;    // and increase the epoch
20. }
21. forward(k); // forward the next record
22. }
23. neuralNet.setNeuralNetMode(NeuralNet.NeuralNetMode.RUN);
24. }
```

უკუგავრცელების ბიჯის როლი არის დელტა წონების განსაზღვრა შეცდომის უკუგავრცელებით, გამომავალი შრიდან ქვემოთ პირველი ფარულ შრემდე:

```
1.  public void backward(){
2.  int numberOfLayers=neuralNet.getNumberOfHiddenLayers();
3.  for(int l=numberOfLayers;l>=0;l--){
4.  int numberOfNeuronsInLayer=deltaNeuron.get(l).size();
5.  for(int j=0;j<numberOfNeuronsInLayer;j++){
6.  for(int i=0;i<newWeights.get(l).get(j).size();i++){
7.  // get the current weight of the neuron
8.  double currNewWeight = this.newWeights.get(l).get(j).get(i);
9.  //if it is the first epoch, get directly from the neuron
10. if(currNewWeight==0.0 && epoch==0.0)
11. if(l==numberOfLayers)
12. currNewWeight=neuralNet.getOutputLayer().getWeight(i, j);
13. else
14. currNewWeight=neuralNet.getHiddenLayer(l).
15. getWeight(i, j);
16. // calculate the delta weight
17. double deltaWeight=calcDeltaWeight(l, i, j);
18. //store new calculated weight for subsequent update
19. newWeights.get(l).get(j).set(i,currNewWeight+deltaWeight);
20.      }
21.    }
22.   }
23. }
```

უკუგავრცელების ნაბიჯი ხორციელდება მეთოდით calcDeltaWeight (). მომენტი დაემატება მხოლოდ წონების განახლების დაწყებამდე, რადგან მან უნდა გამოიყენოს ბოლო დელტა წონა:

```
1.  public Double calcDeltaWeight(int layer,int input,int neuron) {
2.  Double deltaWeight=1.0;
3.  NeuralLayer currLayer;
4.  Neuron currNeuron;
5.  double _deltaNeuron;
6.  if(layer==neuralNet.getNumberOfHiddenLayers()){ //output layer
7.  currLayer=neuralNet.getOutputLayer();
8.  currNeuron=currLayer.getNeuron(neuron);
9.  _deltaNeuron=error.get(currentRecord).get(neuron)*currNeuron.derivative(currLayer.getInputs())
    ;
10. }
11. else{ //hidden layer
12. currLayer=neuralNet.getHiddenLayer(layer);
13. currNeuron=currLayer.getNeuron(neuron);
14. double sumDeltaNextLayer=0;
15. NeuralLayer nextLayer=currLayer.getNextLayer();
16.
17. for(int k=0;k<nextLayer.getNumberOfNeuronsInLayer();k++){
18. sumDeltaNextLayer+=nextLayer.getWeight(neuron, k)*deltaNeuron.get(layer+1).get(k);
19. }
20. _deltaNeuron=sumDeltaNextLayer*currNeuron.derivative(currLayer.getInputs());
21.
22. }
23.
24. deltaNeuron.get(layer).set(neuron, _deltaNeuron);
25. deltaWeight*=_deltaNeuron;
26. if(input<currNeuron.getNumberOfInputs()){
27. deltaWeight*=currNeuron.getInput(input);
28. }
29.
30. return deltaWeight;
31. }
```

შევნიშნოთ _deltaNeuron- ის გაანგარიშება განსხვავებულია გამომავალი და ფარულ შრეებისგან, მაგრამ ორივე მათგანისთვის გამოყენებულია წარმოებული. ამ მიზნისთვის , ჩვენ დავამატეთ derivative() მეთოდი Neuron კლასში.

წონების განახლება ხორციელდება applyNewWeights() მეთოდის გამოყენებით. ქვემოთ მოცემულია კოდის ძირითადი ნაწილი:

```
1.  HiddenLayer hl = this.neuralNet.getHiddenLayer(l);
2.  Double lastDeltaWeight=lastDeltaWeights.get(l).get(j).get(i);
3.  // determine the momentum
4.  double momentum=MomentumRate*lastDeltaWeight;
5.  //the momentum is then added to the new weight
6.  double newWeight=this.newWeights.get(l).get(j).get(i)-momentum;
7.  this.newWeights.get(l).get(j).set(i,newWeight);
8.  Neuron n=hl.getNeuron(j);
9.  // save the current delta weight for the next step
10. double deltaWeight=(newWeight-n.getWeight(i));
11. lastDeltaWeights.get(l).get(j).set(i,(double)deltaWeight);
12. // finally the weight is updated
13. hl.getNeuron(j).updateWeight(i, newWeight);
```

ეს კლასი შეიძლება გამოყენებულ იქნას ზუსტად ისე, როგორც DeltaRule:

```
1.  int numberOfInputs=2;
2.  int numberOfOutputs=1;
3.  int[] numberOfHiddenNeurons={2};
4.  Linear outputAcFnc = new Linear(1.0);
5.  Sigmoid hdAcFnc = new Sigmoid(1.0);
6.  IActivationFunction[] hiddenAcFnc={hdAcFnc };
7.  NeuralNet mlp = new NeuralNet(numberOfInputs,numberOfOutputs
8.  ,numberOfHiddenNeurons,hiddenAcFnc,outputAcFnc);
9.  Backpropagation backprop = new Backpropagation(mlp,neuralDataSet,
10. LearningAlgorithm.LearningMode.ONLINE);
```

3.5უკუგავრცელების და დელტა წესის შედარება

ახლა მოდით XOR-ის მაგალითზე განვიხილოთ როგორ იმუშავებენ დელტა წესი და უკუგავრცელების ალგორითმი. ვნახოთ მრავალშრიანი პერცეპტონი მოქმედებაში. Xortest.java კლასში მოცემულია მაგალითი რომელიც ქმნის ორ ნეირონულ ქსელს შემდეგ მახასიათებლებით:

Ne ural Ne twork	**Perceptron**	**Multilayer Percepetron**
Inputs	2	2
Outputs	1	1
Hidden Layers	0	1
Hidden Neurons in each layer	0	2
Hidden Layer Activation Function	Non	Sigmoid
Output Layer Activation Function	Linear	Linear
Training Algorithm	Delta Rule	Backpropagation
Learning Rate	0.1	0.3 Momentum 0.6
Max Epochs	4000	4000
Min. overall error	0.1	0.01

კოდს ექნება შემდეგი სახე:

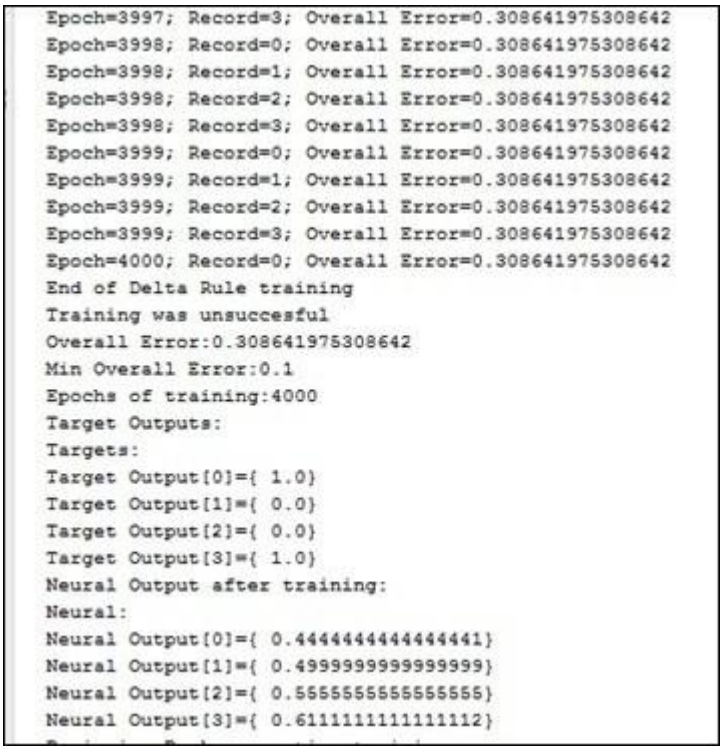
```
1.  public class XORTest {
2.  public static void main(String[] args){
3.  RandomNumberGenerator.seed=0;
4.  int numberOfInputs=2;
5.  int numberOfOutputs=1;
6.  int[] numberOfHiddenNeurons={2};
7.  Linear outputAcFnc = new Linear(1.0); Sigmoid hdAcFnc = new Sigmoid(1.0);
8.  IActivationFunction[] hiddenAcFnc={hdAcFnc};
9.  NeuralNet perceptron = new NeuralNet(numberOfInputs, numberOfOutputs,outputAcFnc);
10. NeuralNet mlp = new NeuralNet(numberOfInputs,numberOfOutputs,numberOfHiddenNeurons,hiddenAcFnc
    ,outputAcFnc);
11. }
12. }
```

ახლა განვსაზღვროთ დატასეტი და სწავლების ალგორითმი:

```
Double[][] _neuralDataSet = {
{0.0 , 0.0 , 1.0 }
,   {0.0 , 1.0 , 0.0 }
,   {1.0 , 0.0 , 0.0 }
,   {1.0 , 1.0 , 1.0 }
};

int[] inputColumns = {0,1}; int[] outputColumns = {2};
NeuralDataSet neuralDataSet = new NeuralDataSet(_neuralDataSet,inputColumns,outputColumns);
DeltaRule deltaRule=new DeltaRule(perceptron,neuralDataSet,LearningAlgorithm.LearningMode.ONLINE);
deltaRule.printTraining=true; deltaRule.setLearningRate(0.1); deltaRule.setMaxEpochs(4000); deltaRule.setMinOverallError(0.1);

Backpropagation backprop = new Backpropagation(mlp,neuralDataSet,LearningAlgorithm.LearningMode.ONLINE);
backprop.printTraining=true;
backprop.setLearningRate(0.3);
backprop.setMaxEpochs(4000);
backprop.setMinOverallError(0.01);
backprop.setMomentumRate(0.6);
```

დასწავლა მიმდინარეობს ორივე ალგორითმით.რადგან ნეირონული ქსელი არაა წრფივად სეპარირებადი ნეირონული ქსელი დეტლა წესით ისწავლის მაგრამ წარუმატებლად.შედეგები ნაჩვენებია შემდეგ სურათებზე:

```
deltaRule.train();
```



როგორც ვხედავთ 4000 ეპოჩის შემდეგაც ქსელმა ვერ შეძლო XOR-ის დასწავლა დელტა წესის გამოყენებით,

უკუგავრცელების ალგორითმის მუშაობის შემთხვევაში კი სასურველი შედეგი მიიღწევა უკვე 39-ე იტერაციაზე.შემდეგ სურათზე მოცემულია ქსელის მუშაობის შედეგი:

```
backprop.train();
```

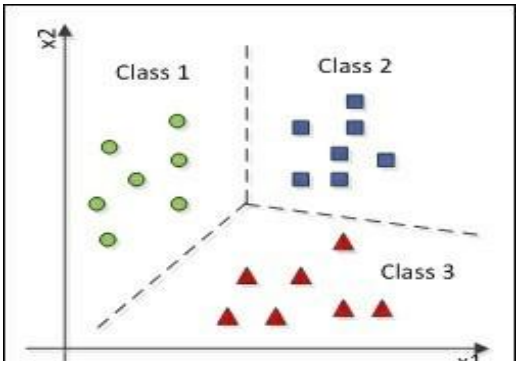
```
Epoch=37; Record=3; Overall Error=0.014316276276702164
Epoch=38; Record=0; Overall Error=0.014205076880259711
Epoch=38; Record=1; Overall Error=0.013967510242490638
Epoch=38; Record=2; Overall Error=0.012818601428508655
Epoch=38; Record=3; Overall Error=0.011524501836923705
Epoch=39; Record=0; Overall Error=0.011442871267615572
Epoch=39; Record=1; Overall Error=0.011242993333301898
Epoch=39; Record=2; Overall Error=0.010319591111492887
Epoch=39; Record=3; Overall Error=0.00923709464092058
End of training
Training successful!
Overall Error:0.00923709464092058
Min Overall Error:0.01
Epochs of training:39
Target Outputs:
Targets:
Target Output[0]={ 1.0}
Target Output[1]={ 0.0}
Target Output[2]={ 0.0}
Target Output[3]={ 1.0}
Neural Output after training:
Neural:
Neural Output[0]={ 0.8635135822257722}
Neural Output[1]={ -0.034317801910536794}
Neural Output[2]={ -0.004429142261391461}
Neural Output[3]={ 0.8635135822257722}
```

თავი 4. დაავადებათა დიაგნოზის კლასიფიკაცია

ჯერჯერობით, ჩვენ შევეხეთ დამკვირვებლიანი სწავლების მაგალითებს რიცხვითი მნიშვნელობების პროგნოზირებისათვის; თუმცა, რეალურ სამყაროში, რიცხვები მხოლოდ მონაცემების საკმაოდ მცირე ნაწილია. რეალური ცვლადები შეიცავს კატეგორიულ მნიშვნელობებს, რომლებიც არ არიან წმინდა რიცხვები, მაგრამ აღწერენ იმ მნიშვნელოვან თვისებებს, რომლებიც გავლენას ახდენენ იმ პრობლემებზე რომელიც გვინდა რომ ნეირონულმა ქსელებმა გადაჭრან. ქვემოთ შევეხები ძალიან დიდაქტურ, მაგრამ საინტერესო ამოცანს, რომელიც მოიცავს კატეგორიულ მნიშვნელობებს და კლასიფიკაციას: დაავადების დიაგნოსტირება. ამ თავში განვიხილავ კლასიფიკაციის პრობლემას დეტალურად და ვნახავთ თუ როგორ შეიძლება წარმოდგინდეს კატეგორიული მონაცემები, ასევე ვაჩვენებთ თუ როგორ უნდა შევიმუშაოთ კლასიფიკაციის ალგორითმი ნეირონული ქსელების გამოყენებით.

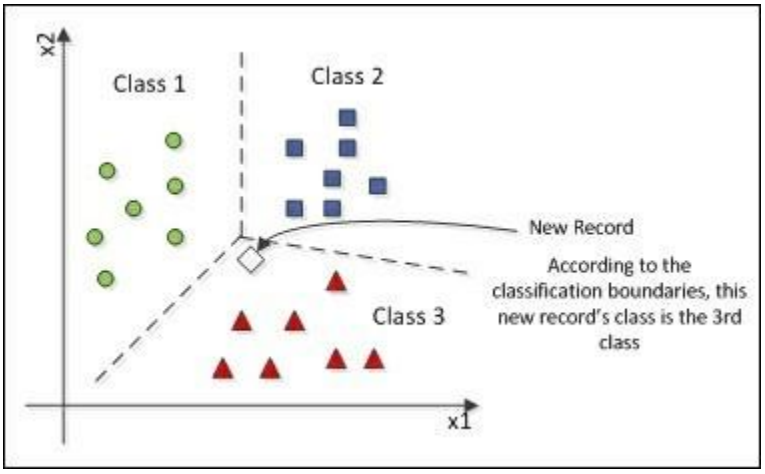
4.1 კლასიფიკაციის ამოცანა

ერთი რამ რაშიც ნეირონული ქსელები მართლაც კარგია, ეს არის მონაცემების კლასიფიკაცია. საკმაოდ მარტივი პერცეპტონის ქსელმა შეიძლება მოახდინოს კლასების ერთმანეთისგან გამოყოფა რეგიონების მიხედვით:რეგიონი ნიშნავს კლასს. ვიზუალურად ეს შემდეგნაირად გამოიყურება :



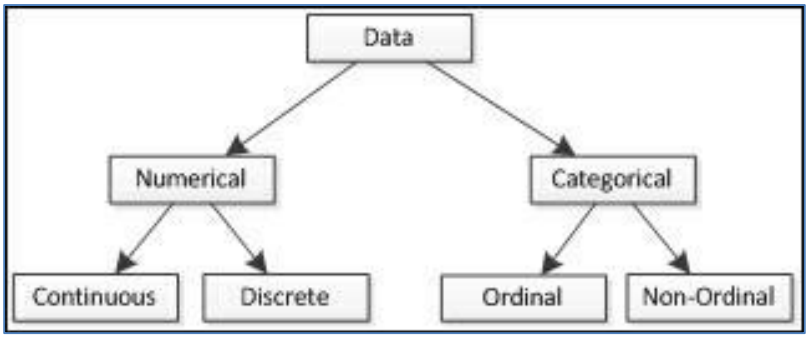
სურათი 31.კლასიფიკაციის მაგალითი

კლასიფიკაციის ალგორითმი ცდილობს მოიძიოს საზღვრები ამ კლასებს შორის მონაცემთა ჰიპერპსივრცეში მას შემდეგ, რაც კლასიფიკაციის საზღვრები განისაზღვრება, ახალი მონაცემის წერტილი, უცნობი კლასით, იღებს კლასის დასახელებას კლასიფიკაციის ალგორითმით განსაზღვრული საზღვრების მიხედვით. ქვემოთ მოყვანილი სურათი გვიჩვენებს, თუ როგორ ხდება ახალი ჩანაწერის კლასიფიკაცია:



სურათი 32.ახალი ჩანაწერის კლასიფიკაცია

აპლიკაციებს ძირითადად უწევთ შემდეგი ტიპის მონაცემებთან გამკლავება:



სურათი 33.მონაცემთა ტიპები

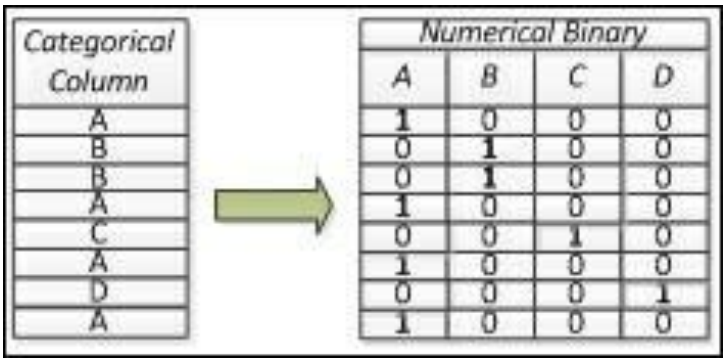
მონაცემები შეიძლება იყოს რიცხვითი ან კატეგორიული ან უბრალოდ სიტყვებით რომ ვთქვათ , სიტყვები ან რიცხვები. რიცხვითი მონაცემები წარმოდგენილია რიცხვითი მნიშვნელობით, რომლებიც შეიძლება იყოს უწყვეტი ან დისკრეტული. ამ ტიპის მონაცემები ჩვენ უკვე გამოვიყენეთ. კატეგორიული მონაცემები არის მონაცემთა ფართო კლასი, რომელიც მოიცავს სიტყვებს, წერილებს ან რიცხვებს, მაგრამ საკმაოდ განსხვავებული მნიშვნელობით. მიუხედავად იმისა, რომ რიცხვით მონაცემებზე შესაძლებელია არითმეტიკული ოპერაციები, კატეგორიული მონაცემები მხოლოდ აღმწერია და არ შეიძლება დამუშავდეს რიცხვების მსგავსად, მაშინაც კი, თუ მნიშვნელობა არის რიცხვითი. მაგალითად დაავადების სიმძიმის ხარისხი მასშტაბით (ნულიდან ხუთამდე მაგ). კატეგორიული მონაცემების კიდევ ერთი თვისებაა , რომ გარკვეული ცვლადს აქვს მნიშვნელობების სასრული რაოდენობის; სხვა სიტყვებით რომ ვთქვათ, მხოლოდ განსაზღვრული მნიშვნელობათა სიმრავლეში შეიძლება განისაზღვროს კატეგორიული ცვლადის მნიშვნელობა. კატეგორიების შიგნით ქვეკლასი არის ორდინალური მონაცემები. კლასი განსაკუთრებულია, რადგან შეიძლება განისაზღვროს განსაზღვრული მნიშვნელობების მიმდევრობა რაღც წესით წინასწარ. მაგალითი არის ზედსართავები, რომლებიც მიუთითებენ ხარისხზე (ცუდი, ნორმალური, კარგი, შესანიშნავი):

შემდეგ ცხრილში მოცემულია მონაცემების ტიპები:

რიცხვითი		კატეგორიული	
მხოლოდ რიცხვები		რიცხვები,სიტყვები,ასოები,სიმბოლოები	
შესაძლებელია არითმეტიკული ოპერაციები		არაა შესაძლებელი არითმეტიკული ოპერაციები	
უსასრულო ან განუსაზღვრელი მნიშვნელობათა სიმრავლე		სასრული ან განსაზღვრული მნიშვნელობათა სიმრავლე	
უწყვეტი	დისკრეტული	ორდინალური	არა ორდინალური
ნამდვილი მნიშვნელობები	მთელი რიცხვები,წილადები	შეიძლება დახარისხება	შეუძლებელია დახარისხება
ყველა შესაძლო მნიშვნელობა	წინსწარგანსაზღვრული მნიშვნელობა	შეუძლება მივანიჭოთ რიცხვები	ყველა შესაძლო მნიშვნელობა არის flag

- შევნიშნოთ რომ ჩვენ აქ შევეხეთ მხოლოდ სტრუქტურირებულ მონაცემებს,მაგრამ მონაცემთა უმეტესობა რეალურ სამყაროში არის არასტრუქტურიზებული და საჭიროა ძალისხმევა მათი სტრუქტურიზაციისთვის.

სტრუქტურირებული მონაცემეთა ფაილები, როგორიცაა CSV ან Excel ან SQL ცხრილები ,როგორც წესი, შეიცავენ რიცხვითი და კატეგორიული მონაცემების სვეტებს.ჩვენ გვაქვს შექმნილიLoadCsv CSV ფაილების ჩასატვირთად და კლასი DataSet (CSV- ის მონაცემების შენახვისთვის), მაგრამ ეს კლასები გამოიყენებიან მხოლოდ რიცხვით მონაცემებთან მუშაობისთვის.კატეგორიული მნიშვნელობების წარმოდგენის ყველაზე მარტივი გზაა თითოეული შესაძლო მნიშვნელობის ორობითი სვეტში გარდაქმნა, რომლის მიხედვითაც თუ მოცემული მნიშვნელობა წარმოდგენილია თავდაპირველ სვეტში შესაბამისი ორობითი სვეტის მნიშვნელობა იქნება ერთი , წინააღმდეგ შემთხვევაში ის იქნება ნულოვანი:

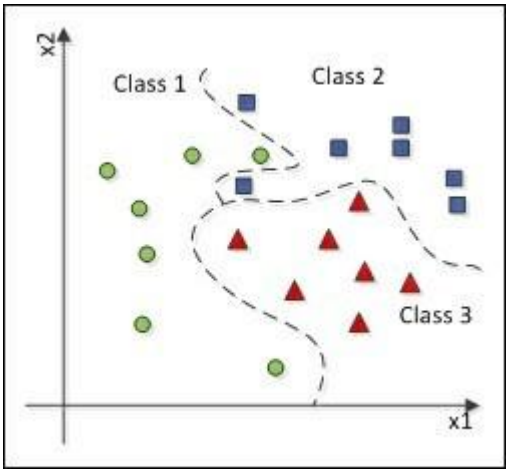


სურათი 34.კატეგორიული მონაცემის გარდაქმნა

4.2 ლოჯისტიკური რეგრესია

განვიხილეთ რომ ნეირონული ქსელები მუშაობენ როგორც მონაცემების კლასიფილატორები, რომლებიც განსაზღვრავენ გადაწყვეტილებების საზღვრებს მონაცემთა ჰიპერსივრცეში. ეს საზღვარი შეიძლება იყოს წრფივი, პერცეპტონების შემთხვევაში, ან არაწრფივი, როდესაც სხვა ნეირონულ არქიტექტურას ვიყენებთ , როგორიცაა MLPs, Kohonen, ან Adaline. წრფივი შემთხვევა ეფუძნება წრფივ რეგრესიას, რომელშიც კლასიფიკაციის საზღვარი ფაქტიურად არის ხაზი,

როგორც ვნახეთ წინა შემთხვევებში. იმ შემთხვევაში, თუ მონაცემები გაფანტულის ისე რომ წრფივი რეგრესია ვერ გვიშველის , მაშინ არაწრფივი კლასიფიკაციის საზღვრებია საჭიროა:

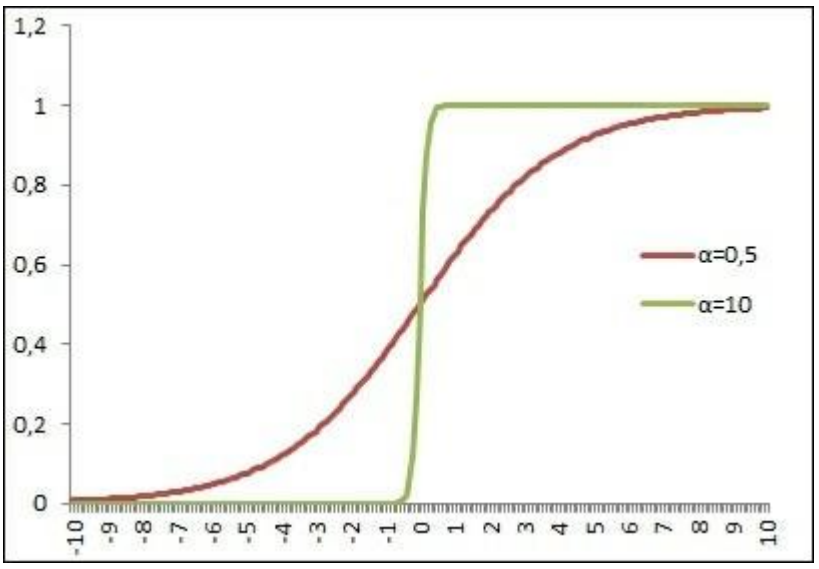


სურათი 35.არაწრფივი კლასიფიკაცია

ნერვული ქსელები კარგი არაწრფივი კლასიფიკატორები არიან და ეს მიიღწევა არაწრფივი აქტივაციის ფუნქციების გამოყენებით. ერთი არაწრფივი ფუნქცია, რომელიც რეალურად კარგად მუშაობს არაწრფივი კლასიფიკაციისათვის, არის სიგმოიდული ფუნქცია, ხოლო ამ ფუნქციის გამოყენებით კლასიფიკაციის პროცედურას ლოჯისტიკური რეგრესია ეწოდება:

$$f(x) = \frac{1}{1+e^{-ax}}$$

ეს ფუნქცია დააბრუნებს მნიშვნელობებს ნულსა და ერთს შორის. ამ ფუნქციის α პარამეტრი აღნიშნავს, თუ რამდენად რთულად ხდება ნულიდან და 1 მდე გადასვლა შემთხვევა. შემდეგი სქემა გვიჩვენებს განსხვავებას:

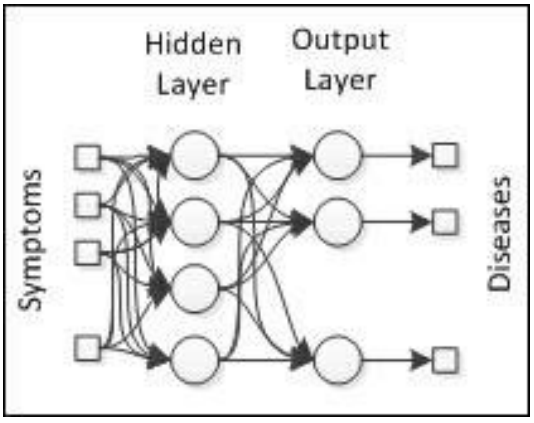


სურათი 36.სიგმოიდური ფუნქცია

როგორც ჩანს რაც უფრო დიდია α პარამეტრი მით უფრო ნაკლებად გლუვად ხდება 0-დან 1-ზე გადასვლა.

კლასიფიკაციის ამოცანები, როგორც წესი, განიხილავენ მრავალ კლასებს , სადაც თითოეულ კლასს ენიჭება შესაბამისი იარლიყი. თუმცა, ბინარული კლასიფიკაციის სქემა სასარგებლოა ნეირონულ ქსელში. ეს იმიტომ, რომ გამომავალი შრეში ლოჯისტიკური ფუნქციის მქონე

ნეირონული ქსელი მხოლოდ 0-დან 1-ს გვაძლევს , რაც გვეუბნება რომ ის ეკუთვნის (1) ამ კლასს ან არა(0) .მიუხედავად ამისა, არსებობს მიდგომა მრავალი კლასის კლასიფიკაციისა ორობითი ფუნქციებით. განვიხილოთ, რომ ყველა კლასი წარმოდგენილია გამომავალ შრეზე საკუთარი ნეირონით და როდესაც ეს ნეირონი მოგვცემს 1-ს შესაბამისი კლასის იარლიყი მიესადაგება შემევალი მონაცემის ჩანაწერს. მოდით დავუშვათ ქსელი დაავადების დიაგნოსტირებას ახდენს; თითოეული ნეირონი გამომავალზე შეესაბამება დაავადებას რაღაც სიმპტომების საფუძველზე:



სურათი 37.დაავადებათა დიაგნოსტირება ნ.ქსელით

შევნიშნოთ რომ ამ ქსელმა შეიძლება ერთზე მეტ გამოსავალზე მოგვცეს ერთი.ამ შემთხვევაში შეგვიძლია ვიმოქმედოთ შეჯიბრებითობის ალგორითმით.

Confusion matrix-გაბნეულობის მარტიცა

არ არსებობს სრულყოფილი კლასიფიკაციის ალგორითმი; ყველა მათგანს აქვს ცდომილებები; თუმცა, მოსალოდნელია , რომ კლასიფიკაციის ალგორითმმა სწორად განაცალკევოს ჩანაწერების 70-90%. ძალიან მაღალი მაჩვენებლით სწორი კლასიფიკაცია ყოველთვის არ არის სასურველი, იმის გამო, რომ შესაძლებელია შეტანილი მონაცემები იყოს მიკერძოებული ერთი ან რამდეიმე კლასისკენ,რომელმაც შეიძლება გავლენა იქონიოს კლასიფიკაციის პროცესზე და ასევე არსებობს რისკი overtraining-სა როდესაც მხოლოდ სასწავლო მონაცემები არის სწორად კლასიფიცირებული.გაბნეულობის მატრიცა აჩვენებს, თუ რამდენად სწორადაა მოცემული კლასის ჩანაწერები კლასიფიცირებული და შესაბამისად, რამდენი იყო არასწორად კლასიფიცირებული. შემდეგი ცხრილი ასახავს ერთერთ ასეთ მატრიცას:

Actual Class	Inferred Class							
	A	B	C	D	E	F	G	Total
A	92%	1%	0%	4%	0%	1%	2%	100%
B	0%	83%	5%	6%	2%	3%	1%	100%
C	1%	3%	85%	0%	2%	5%	4%	100%
D	0%	3%	0%	92%	2%	1%	1%	100%
E	0%	10%	2%	1%	78%	1%	8%	100%
F	22%	2%	2%	3%	3%	65%	3%	100%
G	9%	6%	0%	16%	0%	3%	66%	100%

სურათი 38.გაბნეულობის მატრიცა

აღსანიშნავია, რომ მთავარი დიაგონალს აქვს უფრო დიდი მნიშვნელობები. ყველა სტრიქონის ჯამი უნდა იყოს 100% -იანი, რადგან მოცემული კლასის ყველა ელემენტი კლასიფიცირდება ერთ-ერთი ხელმისაწვდომ კლასში.გაითვალისწინეთ, რომ ზოგმა კლასმა შეუძლია მიიღოს მეტი კლასიფიკაცია ვიდრე მოსალოდნელია.რაც უფრო მეტად გავს გაბნეულობის მატრიცა ერთეულოვან მატრიცას,მით უფრო უკეთესა კლასიფიკაციის ალგორითმი. როდესაც კლასიფიკაცია ორობითია, გაბნეულობის მატრიცა არის მარტივი 2x2 მატრიცა და, შესაბამისად, მისი პოზიციების დაასახელებები შემდეგი ცხრილის შესაბამისია:

Actual Class	Inferred Class	
	Positive (1)	**Negative (0)**
Positive (1)	True Positive	**False Negative**
Negative (0)	**False Positive**	**True Negative**

დაავადების დიაგნოსტიკაში, ორობითი გაბნეულობის მატრიცა გვეუბნება რომ, არასწორი დიაგნოზი შეიძლება იყოს არასწორი დადებითი ან არასწორი უარყოფითი. ცრუ შედეგების მაჩვენებელი შეიძლება შეფასდეს მგრძნობელობისა და სპეციფიკაციის ინდექსებით. მგრძნობელობა ნიშნავს ჭეშმარიტ დადებით მაჩვენებელს; ის ადგენს, თუ რამდენი ჩანაწერი სწორად არის კლასიფიცირებული:

$$Sensitivity = \frac{Number\ of\ True\ Positivies}{Total\ of\ \ Actual\ Positive\ Records}$$

სპეციფიკაციის ინდექსი თავის მხრივ ნიშნავს ჭეშმარიტი უარყოფითი პასუხების პროპორციას მთელს ჩანაწერებში:

$$Specificity = \frac{Number\ of\ True\ Negatives}{Total\ of\ \ Actual\ Negative\ Records}$$

ორივე ინდექსისათვის სასურველია მაღალი შედეგების მიღება.თუმცა ისინი შეიძლება განსხვავდებოდნენ ზოგიერთ შემთხვევაში. მაგალითად განვიხილოთ ორი გამომავალი შრის ნეირონი: თუგამომავალი არის 10, მაშინ ემატება დიახ გაბნეულობის მატრიცაში,ხოლო თუ გამომავალი არის 01, მაშინ ემატება არა.ასევე ხდება შედარება სამიზნე გამოსავალტან რათქმაუნდა. NeuralOutputData კლასში იმპლემენტირებულია კიდევ ერთი მეთოდი : calculatePerformanceMeasures. იგი იღებს პარამეტრად გაბნეულობის მატრიცას და ითვლის და ბეჭდავს სხვადასხვა კლასიფიკაციის პროცესის წარმადობის პარამეტრებს.

- Positive class error rate
- Negative class error rate
- Total error rate
- Total accuracy
- Precision
- Sensibility
- Specificity

ამ მეთოდის იმპლემენტაცია მოცემულია ქვემოთ:

```
1. public void calculatePerformanceMeasures(double[][] confMat) {
2. double errorRatePositive = confMat[0][1] / (confMat[0][0]+confMat[0][1]);
3. double errorRateNegative = confMat[1][0] / (confMat[1][0]+confMat[1][1]);
4. double totalErrorRate = (confMat[0][1] + confMat[1][0]) / (confMat[0][0] +
5. confMat[0][1] + confMat[1][0] + confMat[1][1]);
6. double totalAccuracy = (confMat[0][0] + confMat[1][1]) / (confMat[0][0] +
7. confMat[0][1] + confMat[1][0] + confMat[1][1]);
8. double precision = confMat[0][0] / (confMat[0][0]+confMat[1][0]);
9. double sensibility = confMat[0][0] / (confMat[0][0]+confMat[0][1]);
10. double specificity = confMat[1][1] / (confMat[1][0]+confMat[1][1]);
11. System.out.println("### PERFORMANCE MEASURES ###");
12. System.out.println("positive class error rate: "+
13. (errorRatePositive*100.0)+"%");
14. System.out.println("negative class error rate: "+
15. (errorRateNegative*100.0)+"%");
16. System.out.println("total error rate: "+(totalErrorRate*100.0)+"%");
17. System.out.println("total accuracy: "+(totalAccuracy*100.0)+"%");
18. System.out.println("precision: "+(precision*100.0)+"%");
19. System.out.println("sensibility: "+(sensibility*100.0)+"%");
20. System.out.println("specificity: "+(specificity*100.0)+"%");
```

4.3 დაავადებათა დიაგნოსტირება ნეირონული ქსელებით

კლასიფიკაციის ამოცანები შეიძლება გაკეთდეს ზედამხედველობითი ნერვული ქსელების მიერ ამ წიგნში. თუმცა, რეკომენდირებულია უფრო კომპლექსური არქიტექტურის გამოყენება, როგორიცაა MLP.

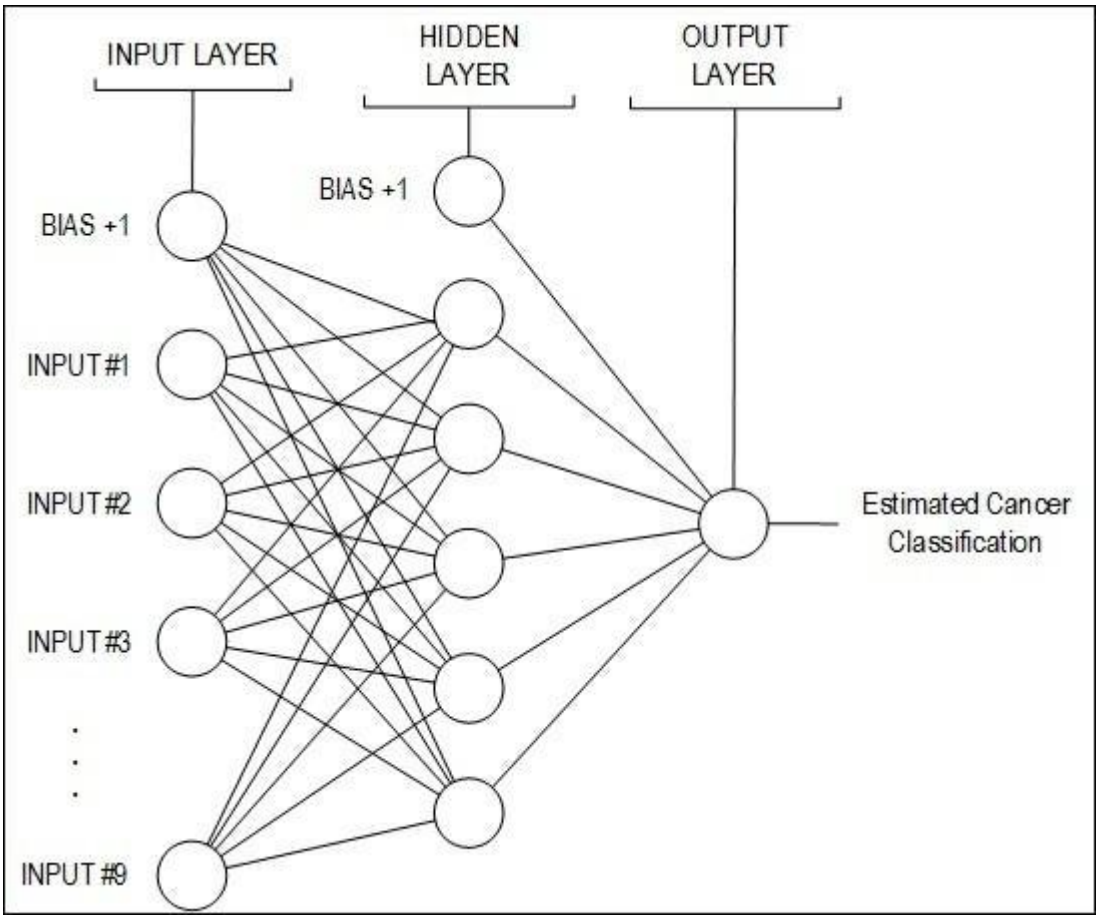
ამ თავში, ჩვენ ვაპირებთ გამოიყენოთ NeuralNet კლასის MLP- ის აშენება ერთი ფარული ფენისა და Sigmoid ფუნქციის გამოსავლენად. ყოველი გამომავალი ნეირონს ნიშნავს კლასი.მაგალითის გატარების კოდი ძალიან ჰგავს გამოცდის კლასს (BackpropagationTest). თუმცა, კლასი დიაგნოზის მაგალითია ითხოვს, თუ რომელი ციფრს სურს მომხმარებლის გამოყენება და სხვა ნორმალური ქსელური პარამეტრები, როგორიცაა ეპოქების რაოდენობა, ნეირონების რიცხვი დაფარული ფენისა და სწავლის მაჩვენებელი(learning rate).

დაავადების დიაგნოსტიკისთვის ჩვენ ვაპირებთ გამოვიყენოთ უფასო მონაცემთა ბაზა proben1, რომელიც ხელმისაწვდომია ინტერნეტში შემდეგ მისამართზე (http://www.filewatcher.com/m/proben1.tar.gz.1782734-0.html). Proben1 არის საკმაოდ კარგი მონაცემთა ნაკრები,რადგან შეიცავს მონაცემებს ბევრი სფეროდნ. ჩვენ ვაპირებთ გამოიყენოთ მკერდის სიმსივნისა და დიაბეტის მონაცემები. ექსპერიმენტის ჩასატარებლად ვამატებთ კლასს: DiagnosisExample.

ძუძუს კიბოს მონაცემთა ბაზა შედგება 10 ცვლადისგან, რომელთაგან ცხრა არის შემავალი და ერთი არის ორობითი გამომავალი. მონაცემთა ნაკრებს აქვს 699 ჩანაწერი, მაგრამ ჩვენ მათგან გამოვრიცხეთ 16 , რომლებიც აღმოჩნდნენ არასრულყოფილი, ამიტომ ჩვენ გამოვიყენებთ 683 ჩანაწერს ქსელის დასატრენინგებლად.პირველი ეტაპია მონაცემთა დამუშავება:სტრუქტურირება და ნორმალიზება(ეს თემა განხილულია გამოყენებულ ლიტერატურაში).რეალურ პრაქტიკულ პრობლემებში, საკმაოდ ხშირია როდესაც გვაქვს დაკარგული ან არასწორი მონაცემები. იდეალურია თუ , კლასიფიკაციის ალგორითმი უმკლავდება ასეთ ჩანაწერებს, მაგრამ ხანდახან რეკომენდირებულია, რომ ისინი გამორიცხოთ, რათა არ მოგვცენ არაზუსტი შედეგები.ქვემოთ მოყვანილი ცხრილი გვიჩვენებს მონაცემთა ბაზის კონფიგურაციას:

Variable Name	Type	Maximum Value and Minimum Value
Diagnosis result	OUTPUT	[0; 1]
Clump Thickness	INPUT #1	[1; 10]
Uniformity of Cell Size	INPUT #2	[1; 10]
Uniformity of Cell Shape	INPUT #3	[1; 10]
Marginal Adhesion	INPUT #4	[1; 10]
Single Epithelial Cell Size	INPUT #5	[1; 10]
Bare Nuclei	INPUT #6	[1; 10]
Bland Chromatin	INPUT #7	[1; 10]
Normal Nucleoli	INPUT #8	[1; 10]
Mitoses	INPUT #9	[1; 10]

შესაბამის ქსელის ტოპოლოგის ექნება შედეგი სახე:



სურათი 39.ნეირონული ქსელის ტოპოლოგია

მონაცემთა ბაზა შეიცავს შემდეგ ჩანაწერებს:

- **სატრენინგო**: 549 ჩანაწერი (80%);
- **სატესტო**: 134 ჩანაწერი (20%)

ახლა ვიწყებთ ბევრ ექსპერიმენტი,რათა მივაგნოთ ქსელის საუკეთესო პარამეტრებს იმის ზუსტად გასარკვევად, არის თუ არა სიმსივნე კეთილთვისებიანი თუ ავთვისებიანი . ჩავატარე 12 სხვადასხვა ექსპერიმენტი (1,000 ეპეჩი თითო ექსპერიმენტში), სადაც MSE და სიზუსტის პარამეტრები იქნა გაანალიზებული. ამის შემდეგ გამოვითვალეთ გაბნეულობის მატრიცა, მგრძნობელობა და სპეციფიკა სატესტო მონაცემებზე და გაკეთდა საბოლოო ანალიზი.ექსპერიმენტებში გამოყენებული ნეირონული ქსელები ნაჩვენებია შემდეგ ცხრილში:

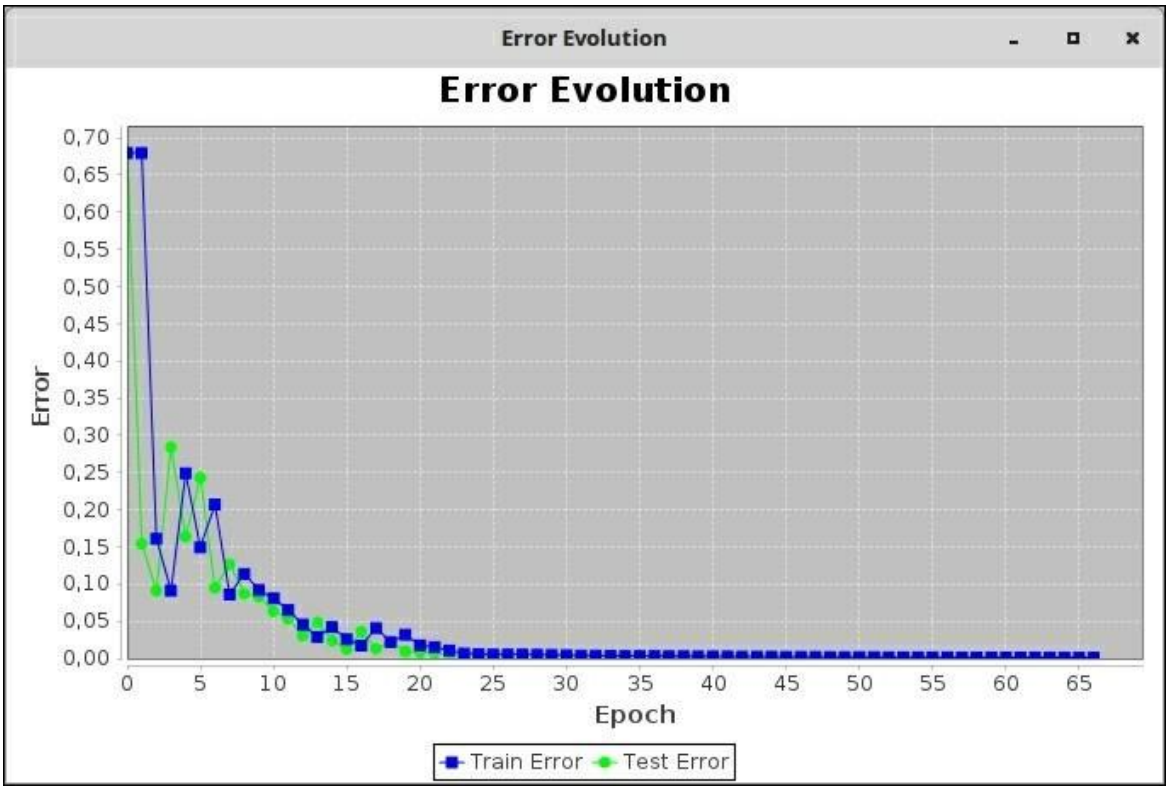
Experiment	Number of neurons in hidden layer	Learning rate	Activation Function
#1	3	0.1	Hidden Layer: SIGLOG Output Layer: LINEAR
#2			Hidden Layer: HYPERTAN Output Layer: LINEAR
#3		0.5	Hidden Layer: SIGLOG Output Layer: LINEAR
#4			Hidden Layer: HYPERTAN Output Layer: LINEAR
#5		0.9	Hidden Layer: SIGLOG Output Layer: LINEAR
#6			Hidden Layer: HYPERTAN Output Layer: LINEAR
#7	5	0.1	Hidden Layer: SIGLOG Output Layer: LINEAR
#8			Hidden Layer: HYPERTAN Output Layer: LINEAR
#9		0.5	Hidden Layer: SIGLOG Output Layer: LINEAR
#10			Hidden Layer: HYPERTAN Output Layer: LINEAR
#11		0.9	Hidden Layer: SIGLOG Output Layer: LINEAR
#12			Hidden Layer: HYPERTAN Output Layer: LINEAR

სურათი 40.ექსპერიმენტების კონფიგურაცია

თითოეული ექსპერიმენტის შემდეგ, ჩვენ ამოვიწერეთ MSE ღირებულებები (იხ .ცხრილი); ექსპერიმენტები # 4, # 8, # 9, # 10 და # 11 იყო ეკვივალენტურები, რადგან მათ ქონდათ დაბალი MSE მნიშვნელობები და საერთო ჯამური სიზუსტე (92.25%). აქედან გამომდინარე, ჩვენ შევარჩიეთ ექსპერიმენტი # 4 და # 11, რადგან მათ აქვთ დაბალი MSE- ის მნიშვნელობები ზემოთ ჩამოთვლილ ხუთივე ექსპერიმენტს შორის:

Experime nt	**MSE training rate**	**Total accuracy**
#1	0.01067	96.29%
#2	0.00443	98.50%
#3	9.99611E-4	97.77%
#4	9.99913E-4	99.25%
#5	9.99670E-4	96.26%
#6	9.92578E-4	97.03%
#7	0.01392	98.49%
#8	0.00367	99.25%
#9	9.99928E-4	99.25%
#10	9.99951E-4	99.25%
#11	9.99926E-4	99.25%
#12	NaN	3.44%

გრაფიკულად, MSE-ის ცვლილება დროში ხდება ძალიან სწრაფად, როგორც ჩანს, შემდეგ გრაფიკზე, რომელიც შექმნილია JfreeCharts ბიბლიოთეკით. იგი ასახავს მეოთხე ექსპერიმენტს. მიუხედავად იმისა, რომ 1000 ეპოჩის გავლა შეგვეძლო, ექსპერიმენტი შეწყდა უფრო ადრე , რადგან მინიმუმალური საერთო შეცდომა (0.001) მიიღწეულ იქნა უფრო ადრე:

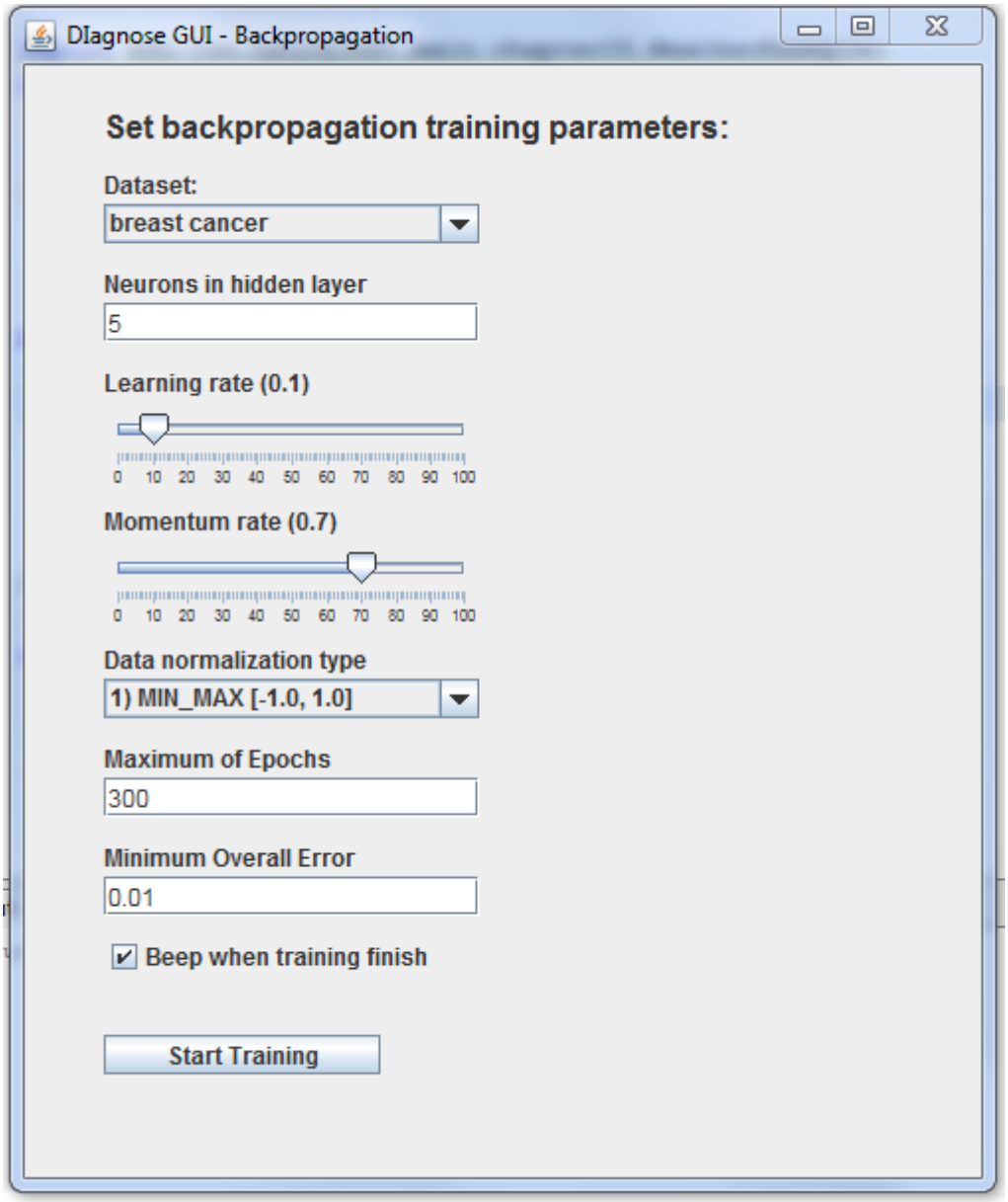


სურათი 41.MSE შეცდომის ევოლუცია

ახლა შევხედოთ გაფანტულობის მატრიცას მას საკმაოდ თანაბარი შედეგები აქვს:

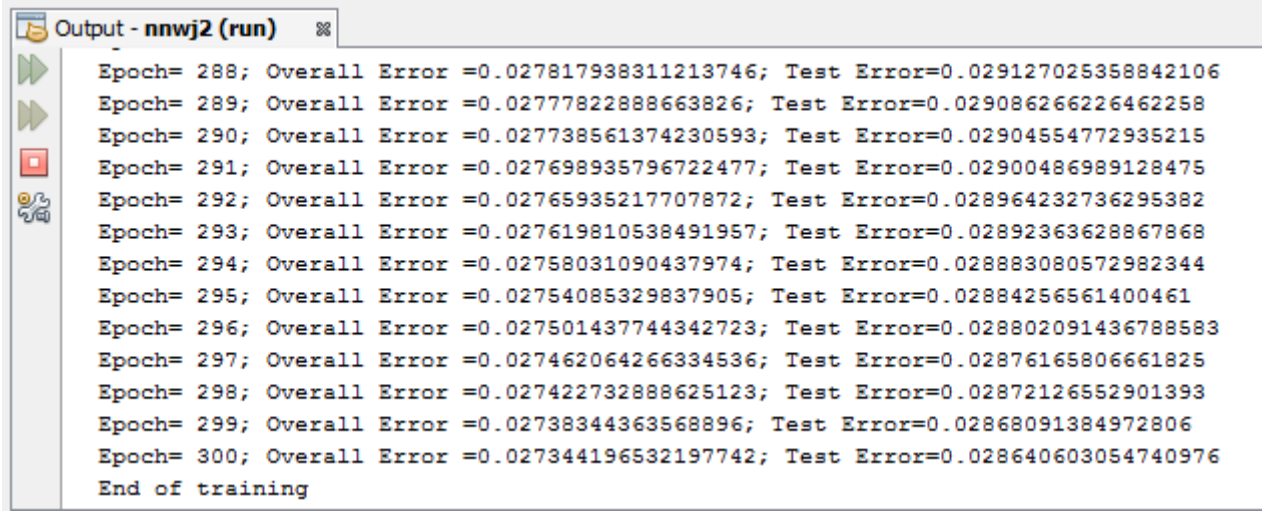
Experiment	Confusion Matrix	Sensibility	Specificity
#3	[[35.0, 12.0] [25.0, 79.0]]	74.46%	75.96%
#9	[[34.0, 12.0] [26.0, 78.0]]	73.91%	75.00%

ექსპერიმენტების ჩატარება რომ უფრო მოსახერხებელი იყოს პარამეტრების დასაყენებლად შემუშავებულია მომხმარებლის გრაფიკული ინტერდეისი,საიდანაც მათ შეუძლიათ მომართონ პარამეტრების მოსახერხებლად და გაუშვან ქსელი მუშაობაზე.გამოვა დინამიური გრაფიკი,რომელზეც შეიძლება დავაკვირდეთ როგორ იცვლება დროში MSE ცდომილება როგორც სასწავლო ისე სატესტო მონაცემებზე,როგორც 41 სურათზეა ნაჩვენები.Swing- ზე შექმნილ პარამეტრების ასარჩევ ფანჯარას აქვს შემდეგი სახე:



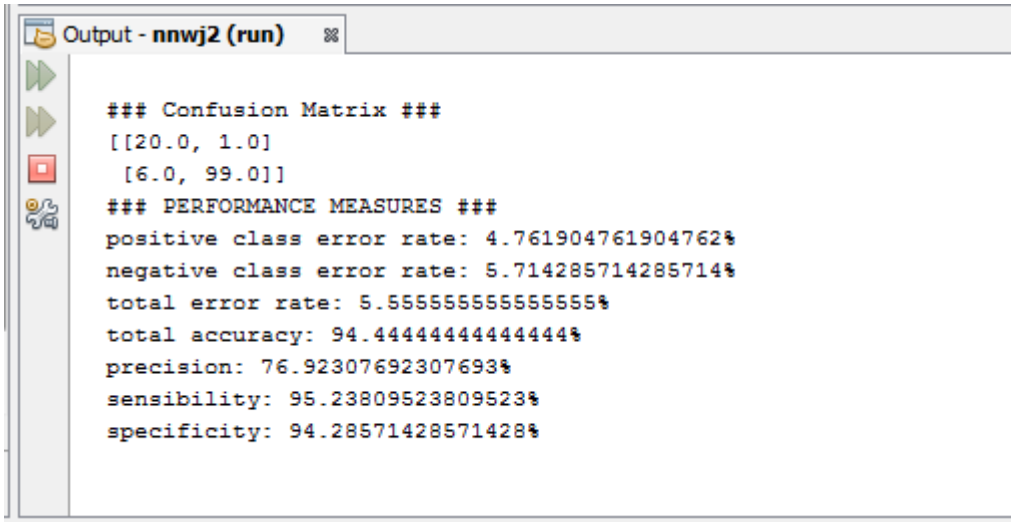
სურათი 42.პარამეტრების ასარჩევი ფანჯარა

შედეგები იბჭდება ეკრანზე თითოეული ეპოჩის შემდეგ:



სურათი 43.შედეგების ბეჭდვა

ტრენინგის დასრულების შედეგ პარამეტრები იბეჭდება შემდეგნაირად:



სურათი 44.შედეგები დასრულების შემდეგ

დიაბეტის მონაცემების შემთხვევაც იდენტურია განსხვავდება მხოლოდ 8 შემავალი მონაცემი.მკითხველს შეუძლია ანალოგიური ექპერიმენტების ჩატარება და მიღებული შედეგების ანალიზი.

დასკვნა

წინამდებარე ნაშრომში განხილული იყო ნეირონული ქსელების თეორია და მისი იმპლემენტაცია ჯავა ენაზე,რის მეშვეობითაც მოვახდინეთ დაავადებათა დიაგნოსტირება, კერძოდ მკერდის სიმსივნის დიაგნოსტირება.შედეგად ჩვენ კარგი პრაქტიკული შედეგი მივიღეთ. ნაშრომში განხილული იქნა ნეირონული ქსელის ძირითადი ცნებები და არქიტექტურის ძირითადი ტიპები.ასევე შევეხეთ სწავლების პარადიგმებს და განვიხილეთ ორი ძირითადი ალგორითმი მაგალითების საფუძველზე.ნაშრომში აქცენტი გაკეთებულია თეორიული საკითხების კარგ გააზრებაზე,რადგან მის გარეშე იმპლემენტაციას აზრი არ აქვს,თავის მხრივ კოდი არის კარგად დაკომენტარებული და მკითხველს შეუძლია თეორიული საკითხების პარალელურად უპრობლემოდ მიჰყვეს და გაარჩიოს კოდი. აღსანიშნავია ,რომ იმპლემენტაციისას არ იქნა გამოყენებული არცერთი მზა ბიბლიოთეკა , რათა მკითხველმა ნახოს რეალურად როგორ მუშაობს ნეირონული ქსელი კულისებს მიღმა. ეს ნაშრომი გამოადგება ყველას,ვისაც აინტერესებს ნეირონული ქსელების თეორია და მისი პრაქტიკაში გამოყენება,იგი განსაკუთრებით სასარგებლო იქნება სტუდენტებისთვის, რომლებიც ეუფლებიან ხელოვნურ ინტელექტს. ისინი შეისწავლიან თეორიის პრაქტიკაში გადაყვანას,განივითარებენ აზროვნებისა და ანალიზის უნარს.ნებისმიერ მსურველს შეუძლია აიღოს კოდი და განავითაროს იგი,დაამატოს ახალი ალგორითმების იმპლემენტაცია,დაემატოს უდამკვირვებლო სწავლების შესაძლებლობაც,რაც არანაკლებ მნიშვნელოვანია.ასევე შეიძლება არსებული პროექტის უფრო მეტ პრობლემაზე მორგება და მისი წარმადობის დახვეწა.კარგი იქნება ნეირონული ქსელის მუშაობის შედეგის შენახვა და მისი პროგრამული ანალიზი.შესაძლებელია ვიზუალიზაციის დახვეწა.მცირე მოდიფიკაციის შემდეგ ამ ნაშრომის გამოყენება შეიძლება სრულ ერთსემესტრიან საუნივერსიტერო სასწავლო კურსად პრაქტიკულ ნეირონულ ქსელებში,რაც საკმაოდ საინტერესო იქნება სტუდენტებისათვის.მიღებული შედეგები გვაძლევს საფუძველს განვავითაროთ ნეირონული ქსელები მედიცინაში დაავადებათა დიაგნოსტირებისათვის ,რაც

საკმაოდ მნიშვნელოვანია მით უფრო, რომ დღეს გვაქვს უზარმაზარი რაოდენობის მონაცემი სამედიცინო ისტორიებისა,რისი გამოყენებაც შეიძლება დიაგნოსტირების ამოცანის გადასაჭრელად.

გამოყენებული ტექნოლოგიები:

- Java 8
- JfreeChart
- JCommon
- Netbeans IDE

გამოყენებული ლიტერატურა

- Priddy, Kevin L., Keller, Paul. E., Artificial Neural Networks: An introduction, SPIE Press, ISBN-13 9780819459879, Jan, 1, 2005.
- Levenick, J., Simply Java: An introduction to Java Programming, Charles River Media; 1 ed., ISBN- 13 9781584504269, Sep, 8, 2005.
- Sejnowski, Terrence, J., Neural Network Learning Algorithms, Neural Computers Vol. 41, Springer Study Edition, pp. 291-300, 1989.
- Hguyen, Derrick H., Widrow, Bernard, Neural Networks for Self-Learning Control Systems, IEEE Control Systems Magazine, April 1990.
- Haykin, Simon O., Neural Networks and Leaning Machines, Prentice Hall, 3rd ed., ISBN-13 9780131471399, Nov, 28, 2008.
- Rumelhart, David E., Hinton, Geoffrey E., Williams, Ronald J., Learning Representations by back-propagating errors", Nature v. 323 (6088), pp. 533-536, Oct, 8, 1986.
- Yusuke Sugomori :Deep Learning: Practical Neural Networks with Java: Packt Publishing.2017. ISBN 139781788470315
- Levenberg K., A Methor for the Solution of Certain Non-Linear Problems in Least Squares, Quaterly of Applied Mathematics, vol 2, pp. 164-168, 1944.
- Marquardt, D., An Algorithm for Least-Squares Estimation of Nonlinear Parameters, SIAM Journal on Applied Mathematics, vol 11 (2), pp. 431-441, 1963.
- Huang, Guang B., Zhu, Qin Y., Siew, Chee K., Extreme learning machine: A new learning scheme of feedforward neural networks, Proceedings of IEEE International Joint Conference on Neural Networks, 2004.
- Altman, Edward I., Marco, Giancarlo, Varetto, Franco, Corporate distress diagnosis: Comparison using linear discriminant analysis and neural networks (the Italian experience), Journal of Banking and Finance v. 18, pp. 505-529, 1994.
- Bishop, C.M. Neural Networks for Pattern Recognition, Oxford University Press, ISBN-10 0198538499, 1995.
- Al-Shayea, Qeethara K., Artificial Neural Networks in Medical Diagnosis, International Journal of Computer Science Issues, Vol. 8, Issue 2, pp. 150-154, March 2011.
- Freedman, David A., Statistical Models: Theory and Practice, Cambrige University Press, 2009.
- Fawcett, Tom, An Introduction to ROC Analysis, Pattern Recognition Letters, vol. 27, is. 8 pp. 861-874, 2006.
- Simon Haykin.Neural Networks: A Comprehensive Foundation. Prentice Hall; 2 edition (July 16, 1998) ISBN-10: 0132733501

- Richard M. Reese, Jennifer L. Reese, Alexey Grigorev: Java: Data Science Made Easy, Packt Publishing 2017 ISBN 139781788475655