Satura rādītājs

1. Neironu tīkli – skaitļošanas paradigma ... 3
   1.1. Neironu tīkls kā skaitļošanas sistēma ... 3
   1.2. Bioloģiskie neironu tīkli ... 4
2. Mākslīgais neirons ... 7
   2.1. Neirona uzbūves un darbības pamatprincipi ... 7
   2.2. Vienkārši neirona piemēri ... 8
      2.2.1. Piemērs #1. Loģiskā UN (x_1 AND x_2) modelēšana ... 8
      2.2.2. Piemērs #2. Funkcijas NOT (x_1 AND x_2) modelēšana ... 9
   2.3. Summēšanas funkciju veidi ... 11
   2.4. Aktivitātes funkciju veidi ... 11
3. Neironu tīklu topoloģija ... 16
   3.1. Lineāras un nelineāras problēmas ... 16
   3.2. XOR modelēšana ar divu slāņu neironu tīklu ... 17
   3.3. Neironu tīklu topoloģiju veidi ... 18
      3.3.1. Vienvirziena tīkli ... 18
      3.3.2. Rekursīvie tīkli ... 21
   3.4. Neironu tīkla un tā darbības notācija vairāku slāņu topoloģijā ... 22
4. Neironu tīkla apmācīšanās ... 24
   4.1. Apmācības process ... 24
   4.2. Apmācīšanās paradigmas ... 26
      4.2.1. Apmācīšanās „ar skolotāju” ... 27
      4.2.2. Neuzraudzītā apmācīšanās ... 28
      4.2.3. Apmācīšanās ar pastiprinājumu ... 28
   4.3. Apmācīšanās likumi ... 29
      4.3.1. Apmācīšanās ar kļūdu labošanu ... 29
      4.3.2. Heba apmācīšanās ... 30
      4.3.3. Apmācīšanās ar konkurenci ... 30
5. Vienslāņa perceptrons ... 32
   5.1. Perceptrona uzbūve ... 32
   5.2. Ar perceptronu risināmu problēmu piemēri ... 33
      5.2.1. Funkcija NOT (x_1 AND x_2) ... 33
      5.2.2. Ciparu atpazīšana ... 34
      5.2.3. Normalizācija un denormalizācija ... 34
   5.3. Perceptrona darbināšana ... 35
6. Vienslāņa perceptrona apmācīšanās ... 37
   6.1. Kvadrātiskās kļūdas metode ... 37
      6.1.1. Algoritma apraksts ... 37
      6.1.2. Dinamiska apmācīšanās parametra izmantošana ... 40
      6.1.3. Apmācīšanās piemērs ... 41
      6.1.4. Teorētiskā bāze ... 42
   6.2. Perceptrona apmācīšanās ar pseido-apgrieztās matricas metodi ... 44
      6.2.1. Algoritma apraksts ... 44
      6.2.2. Apmācīšanās piemērs ... 44
7. Vairākslāņu perceptrons ... 46
   7.1. Vairākslāņu perceptrona uzbūve ... 46
   7.2. Vairākslāņu perceptrona darbināšana ... 47
   7.3. Vairākslāņu perceptrona apmācīšanās ar kļūdu atgriezeniskās izplatīšanās metodi 48
8. Kohonena tīkls ... 51

8.1. Paraugu klāsterizācija un Kohonena tīkls 51
8.2. Ar Kohonena tīklu risināmas problēmas piemērs 52
8.3. Kohonena tīkla uzbūves un darbības principi 53
8.4. Kohonena tīkla darbināšana 55
8.5. Kohonena tīkla apmācīšanās 56
8.5.1. Apmācīšanās algoritms 56
8.5.2. Apmācīšanās procesa parametru kontrole 59

1. Neironu tīkli – skaitļošanas paradigma

1.1. Neironu tīkls kā skaitļošanas sistēma

Datoriem kļūstot aizvien spēcīgākiem un tehnoloģijām aizvien spēcīgākām, rodas iespēja sākt izmantot datorus arī tādu problēmu risināšanai, kas līdz šim bijušas tikai cilvēku kompetencē. Daudzas šādas problēmas cilvēkam ir salīdzinoši viegli paveicamas (piemēram, atpazīt seju vai sašķirot ābolus skaistajos un ne tik skaistos), tai pat laikā datoram tās var būt ļoti smagas. Ņemot vērā to, ka ir tādas problēmas, kuras cilvēki risina daudz vieglāk nekā tradicionālie algoritmi (sekvenciāla, uz loģiku balstīta rēķināšana), varētu padomāt, ka „skaitļošanas mehānismā", kuru izmanto cilvēks, ir „kaut kas tāds", ko būtu vērts izzināt un iestrādāt algoritmos, lai tie kļūtu „gudrāki".

Jau no pašiem pirmsākumiem neironu tīklu izpēti ir motivējusi atziņa, ka cilvēka smadzenes rēķina pilnīgi savādākā veidā, nekā to dara parastie ciparu datori. Smadzenes ir ļoti kompleksa, nelineāra un paralēla informācijas apstrādes sistēma. Smadzenēm ir ļoti apjomīga un spēcīga struktūra un līdz ar to spēja veidot pašām savus noteikumus, ko mēs saucam par pieredzi. [Haykin, 1999]

Vispārīgi runājot, neironu tīkls (jeb, precīzāk, **mākslīgais neironu tīkls** (*artificial neural network*)) ir mašīna, kas ir veidota, lai modelētu smadzeņu darbību noteikta uzdevuma sasniegšanai vai problēmas risināšanai [Haykin, 1999]. Īsi varētu teikt, ka neironu tīkls ir cilvēka smadzeņu modelis. Tomēr jāsaka, ka, salīdzinot ar cilvēka smadzenēm, mākslīgie neironu tīkli pagaidām vēl ir uzskatāmi tikai par ļoti vienkāršotiem un ir tikai divas īpašības (principi), ko ikviens neironu tīkls ir garantēti pārņēmis no smadzenēm:

1. **Apmācība** kā informācijas ieguves veids. Neironu tīkls zināšanas iegūst apmācības ceļā, kaut arī ir zināmas teorijas gatavu neironu tīklu ģenerēšanai;
2. **Konekcionisms**. Neironu tīkls sastāv no liela daudzuma t.s. neironu, un to savstarpējo saišu stiprums nosaka tīklā glabātās zināšanas.

> **Neironu tīkls** ir informācijas apstrādes sistēma, kas sastāv no relatīvi liela skaita skaitļošanas elementu (sauktu par neironiem) un savu spēju veikt noteiktu uzdevumu ieguvusi apmācības ceļā.

Katrs neirons (jeb, precīzāk, **mākslīgais neirons** (*artificial neuron*)) ir relatīvi vienkārša skaitļošanas vienība, tomēr liels daudzums neironu, kas zināmā veidā savienoti viens ar otru, kopā var paveikt ļoti sarežģītus uzdevumus. Bioloģiskā neirona takts frekvence ir aptuveni 6 kārtas zemāka nekā mūsdienu procesoriem (aptuveni 1 ms), tomēr cilvēka smadzeņu potenciālu nodrošina vismaz 10 miljardi neironu, kas katrs savienots ar citiem neironiem ar vairāku tūkstošu saišu starpniecību, un to kopskaits sasniedz 60 triljonus.

Vai ir vērts runāt par smadzeņu modeļiem un cilvēku rīcības atdarināšanu pirms kaut cik manāma (salīdzinot ar smadzenēm) mākslīgo neironu tīkla apjoma sasniegšanas? Neironu tīklu pētniekiem ir ticība un pārliecība, ka var, jo kā gan savādāk viņi attaisnotu savu darbošanos šajā sfērā. Turklāt atsevišķos praktiskos uzdevumos neironu tīkli jau ir pierādījuši savu noderīgumu un pārsvaru pār citām skaitļošanas metodēm.

Neironu tīklu laikmets sākās 1943. gadā ar Makaloka un Pitsa (*McCulloch and Pitts*) darbu *„A logical calculus of the ideas immanent in nervous activity"*.

Svarīgs moments agrīnajā neironu tīklu attīstībā bija Rozenblata (*Rosenblatt*) un viņa līdzstrādnieku izstrādātais t.s. **perceptrons** Masačūsetsas tehnoloģiskajā institūtā (MIT) 1958. gadā.

Trieciens neironu tīklu attīstībā bija 1969. gadā izdotā Minska un Peiperta (*Minsky and Papert*) izdotā grāmata, kurā tika parādīts, ka perceptrona modelis nespēj atrisināt pat tik vienkāršas problēmas kā t.s. **XOR problēma**, kur nu vēl sarežģītākas. Līdz ar šo daudzos neironu tīklu projektos tika samazināts vai noņemts finansējums, un pētījumus turpināja tikai entuziasti.

Tikai 15 gadus vēlāk, kad tika pārkāpti vairāki teorētiski, kā arī tehnoloģiski ierobežojumi neironu tīklu attīstība sākās ar jaunu sparu. Tika izgudrota t.s. neironu tīklu apmācība ar **kļūdu atgriezeniskās izplatīšanās metodi** (*error back-propagation*) un uz šādiem tīkliem vairs neattiecās Minska un Peiperta ierobežojumi. Tieši uz vairākslāņu perceptrona bāzēti neironu tīklu modeļi mūsdienās vairākumā gadījumu tiek asociēti ar „neironu tīkliem".

20. gs. 80.-90. gados tika piedāvāti dažādi neironu tīklu modeļi, piemēram, **Radiālo bāzes funkciju** (RBF) tīkli un t.s. ***support vector machines***, no kuriem pēdējais atsevišķos literatūras avotos tiek parādīts kā ārpus neironu tīklu paradigmas esošs.

Blakus tam, 20. gs. 90. gados ir sākusies, kā šī virziena pārstāvji nereti sauc – 3. paaudzes neironu tīklu jeb **pulsējošo neironu tīklu** (*pulsed/spiking neural networks*) attīstība, kas pretendē uz vairāku līdz šim neironu tīklu modeļos nesastaptu bioloģisku principu pielietošanu.

1.2. Bioloģiskie neironu tīkli

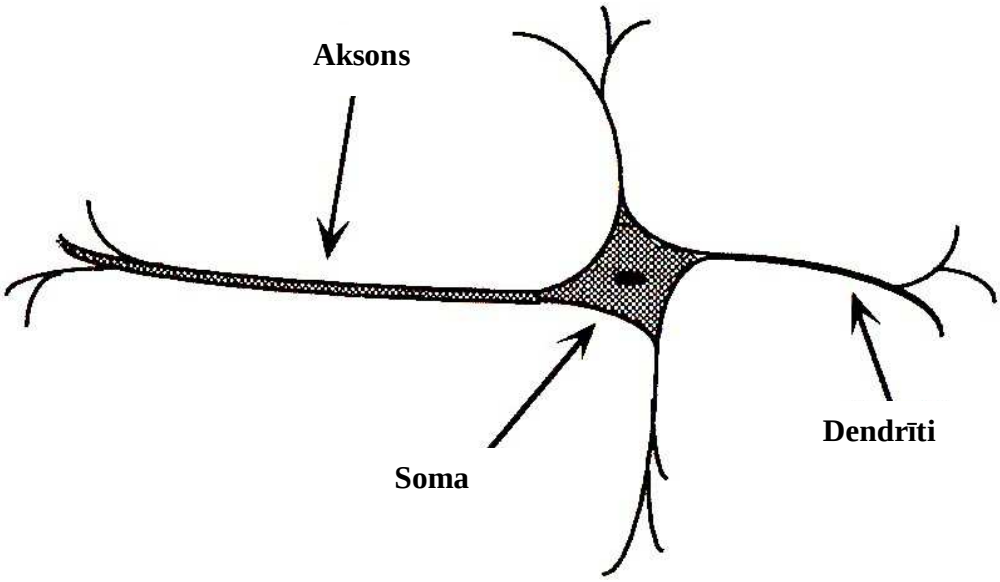
Kaut arī pakāpe, ar kuru dažādi mākslīgo neironu tīklu modeļi atbilst bioloģiskajiem analogiem atšķiras, tomēr bioloģiskie neironu tīkli (visbiežāk cilvēka smadzenes) ir uzskatāmi vismaz par sākotnējo iedvesmas avotu neironu tīklu virziena attīstībai.

Arī starp pētniekiem ir atšķirība – vieni uzskata, ka bioloģiskā atbilstība (*plausibility*) ir būtiska, kamēr citi par svarīgāku vērtē konkrēta modeļa noderīgumu konkrētu problēmu risināšanā.

Neskatoties uz atšķirībām pieejās, tomēr būtu nepieciešamība īsumā saprast galvenais bioloģisko neironu un bioloģisko neironu tīklu uzbūves un darbības principus.

Starp bioloģiskā neirona (piemēram, smadzeņu vai nervu šūnas) un mākslīgā neirona struktūru ir cieša analoģija. Atsevišķo neironu struktūra atšķiras salīdzinoši nedaudz, ja salīdzina dažādas sugas, lielākās atšķirības ir starp sistēmu organizāciju, kurās neirons ir elements.

Neirons jeb nervu šūna ir nervu sistēmas galvenais elements. Kā jau katrai šūnai, arī neironam ir ķermenis, saukts par **somu** (*soma*). No somas iziet neskaitāmi izaugumi, kurus var sadalīt divās grupās – tievie un biezi sazarotie **dendrīti** (*dendrites*) un resnākais **aksons** (*axon*). Ienākošie signāli nervu šūnā ienāk caur t.s. **sinapsēm** (*synapses*), bet izejošais signāls tiek aizvadīts pa aksonu, kas tālāk sazarojas, lai novadītu signālu uz vairākiem citiem neironiem.



Att. 1.1. Bioloģiskā neirona shematiska uzbūve. [Scherer, 1997]

Mākslīgie neironi no bioloģiskajiem neironiem aizguvuši šādas īpašības [Fausett, 1994]:

1. Neirons saņem **daudzus** signālus;
2. Signālu, kas ienāk neironā, var **modificēt** uztverošās sinapses **svars** (svara vērtība);
3. Neirons veic svērto ieejas vērtību **summēšanu**;
4. Noteiktos apstākļos (pietiekoša ievada gadījumā) neirons izejā dod **vienu** signālu;
5. Izejas vērtība no noteikta neirona var nonākt daudzos citos neironos (t.i. aksons **sazarojas**).

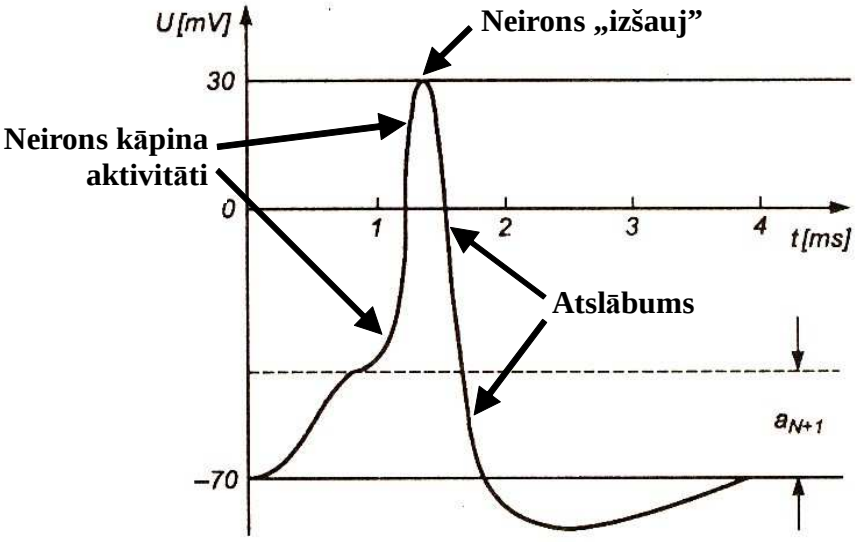
Tālāk tiks uzskaitītas vēl dažas mākslīgo neironu tīklu īpašības, kas radušās, iedvesmojoties no bioloģiskajiem neironiem [Fausett, 1994]:

1. Informācijas apstrāde notiek **lokāli** (kaut arī tādi signālu pārraides līdzekļi, piemēram, hormonu darbība, kas var tikt uztverti kā vispārēja (resp., globāla) procesa kontrole);
2. Atmiņa ir **sadalīta** (*distribited*):
3. ilglaicīgā atmiņa (*long-term memory*) izvietota neironu sinapsēs jeb svaros,
4. īslaicīgā atmiņa (*short-term memory*) atbilst signāliem, kurus izsūta neironi.
5. Sinaptiskie svari var tikt **modificēti**, balstoties uz **pieredzi**;
6. Sinapses var būt gan ar **pastiprinošu**, gan **pavājinošu** iedarbību (*excitatory or inhibitory*).

Vēl viena būtiska īpašība, kur mākslīgie neironu tīkli atbilst to bioloģiskajiem analogiem, ir kļūdu noturība (*fault tolerance*). Bioloģiskās neironu sistēmas ir kļūdu noturīgas divos veidos. Pirmkārt, mēs esam atpazīt ieejas signālus, kas kaut kādā veidā atšķiras no tiem, kurus esam redzējuši iepriekš. Otrkārt, mums ir spējas pārciest pašas bioloģiskās sistēmas bojājumus

Caur sinapsēm ienākošie signāli neironu regulāri kairina. Ja kairinājuma (neirona aktivitātes) līmenis pārsniedz noteiktu slieksni, neirons "izšauj" (*fires*), resp., izejā **dod** noteiktu **signāla impulsu**. Turpretī, ja neirona aktivitātes līmenis nesasniedz slieksni, tad neirons "neizšāvis" atgriežas iepriekšējā stāvoklī. Pēc kārtējā nervu impulsa ģenerēšanas neirons kādu brīdi zaudē

spēju to atkal izdarīt pat pie ļoti spēcīgas kairināmības (iestājas **signālnoturības** (*refractoriness*) (t.i., „noturības nedot signālu") periods jeb, no neirona aktivitātes viedokļa, atslābums). Signālnoturība nodrošina to, ka neirons nedod signālu nepārtraukti, bet kādu laiku „atpūšas" un tādējādi dod iespēju ietekmei citiem neironiem.



Att. 1.2. Tipiska nervu impulsa forma. [Osowski, 2002]

Ir vismaz 3 dažādi aspekti, kuri nodrošina, ka bioloģiskas nervu sistēmas (piemēram, smadzenes) spēj efektīvi risināt sarežģītas problēmas tā, kā tas notiek:

1. Ļoti liels paralēli darbojošos neironu skaits;
2. Fizikāli ķīmiskā vide (šķīdumi, membrānas utt.);
3. Noteikts darbības algoritms (ietverot gan atsevišķu neironu uzbūves un darbības principus, gan kopējo topoloģiju).

Mākslīgajos neironu tīklos fizikāli ķīmiskā vide ir pilnīgi savādāka un arī neironu skaits, salīdzinot ar dabiskajām sistēmām, ir uzskatāms par niecīgu.

Atliek vienīgi cerēt, ka, izveidojot pietiekoši labus algoritmus, ir iespējams vismaz daļēji iegūt priekšrocības, kuras bauda bioloģiskās nervu sistēmas salīdzinājumā ar tradicionālajiem datoriem un algoritmiem. Esošā neironu tīklu izmantošanas pieredze to apstiprina un rāda, ka esošie algoritmi pat pie salīdzinoši neliela neironu skaita dod jūtamu ieguldījumu daudzu problēmu risināšanā.

2. Mākslīgais neirons

Neironu tīkls vispārīgā gadījumā sastāv no relatīvi liela daudzuma neironu. Neirons (*neuron, unit, processing unit, node, processing element, cell*) ir salīdzinoši vienkāršs (pret visu neironu tīklu) skaitļošanas elements.

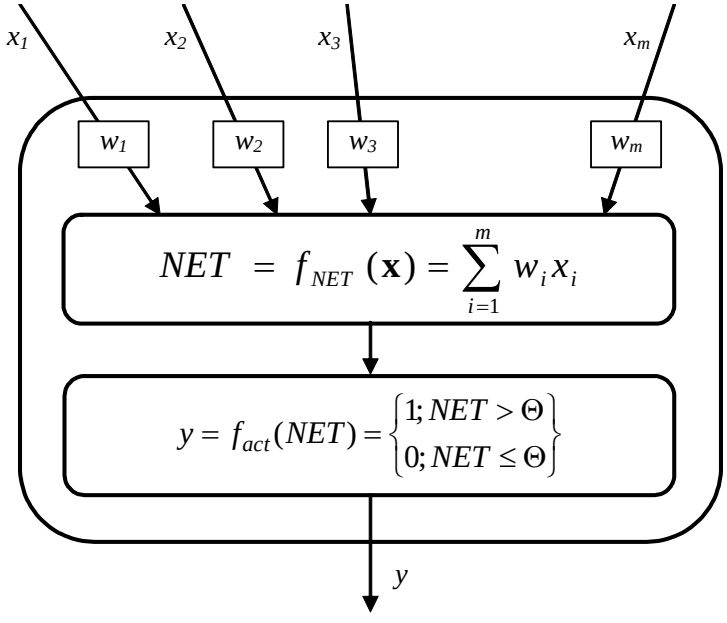
Atšķirībā no tradicionālajām skaitļošanas sistēmām, neironu tīkls ir drīzāk līdzīgs melnajai kastei – katra atsevišķa neirona loma kāda konkrēta uzdevuma risināšanā nav precīzi nosakāma. Šī neironu tīklu īpašība krietni apgrūtina to konstruēšanu, testēšanu un ekspluatēšanu, tādējādi neironu tīkli nebūtu uzskatāmi par ērtu līdzekli jebkuras problēmas risināšanai. Tieši otrādi – neironu tīklu piegājiens ir salīdzinoši grūts un neērts, un neironu tīkli būtu izmantojami tikai atsevišķos gadījumos, piemēram, kad tradicionālie algoritmi nedod efektīvu risinājumu vai pieejamā informācija par problēmu nav pilnīga, resp., īstajā vietā nevis „par katru cenu".

Neirons ir salīdzinoši vienkāršs elements, tai pat laikā atsevišķa neirona uzbūve var būtiski ietekmēt visa neironu tīkla skaitļošanas spēju. Kaut arī nopietni par neironu tīkliem var runāt tikai tad, ja tajos ir daudzi neironi, tomēr arī viena neirona (vai dažu neironu) sistēmas izzināšana var dot lielu priekšstatu par šo datorzinātņu un mākslīgā intelekta sfēru un pat būt derīga vienkāršu problēmu risināšanai.

2.1. Neirona uzbūves un darbības pamatprincipi

Neirona sastāvā ietilpst šādi galvenie konstruktīvie elementi:

1. Svari;
2. Summēšanas jeb izplatīšanās funkcija;
3. Aktivitātes funkcija.



Att. 2.1. Mākslīgā neirona uzbūves shēma ar sliekšņveida aktivitātes funkciju (x_i – ieejas; w_i – svari; *NET* – summēšanas vērtība; Θ – slieksnis; y – izeja)

Svari (jeb sinaptiskie svari, *[synaptic] weights*) ir vērtības, caur kurām neironā ienāk ienākošie signāli un tādējādi tiek pamainīti.

- Tieši svari ir tie, kas visbūtiskāk atšķir neironu no neirona, jo izplatīšanās un aktivitātes funkcijas parasti noteiktai neironu grupai vai pat visam neironu tīklam ir vienādas vai ļoti līdzīgas.
- Tieši svari ir tie, uz kuriem iedarbojas apmācības process, tos modificējot un tādējādi nodrošinot, ka neironu tīkls apmācības procesa beigās ir ieguvis spēju risināt noteiktu problēmu.

Neironā parasti ir tik daudz svaru, cik ieeju (katrai ieejai atbilst tieši viens svars), atsevišķos gadījumos par vienu vairāk (papildus svars). Formulās svarus ļoti bieži apzīmē ar burtu *w* (no angļu valodas vārda *weight*) un indeksu, kas norāda svara kārtas numuru neironā, piemēram, w_2. Lai apzīmētu visa neironu tīkla svaru kopumu, var lietot kortežu vai matricu notāciju (t.i., pierakstu), piemēram, **w** vai **W** (ja ir runa par vairāku neironu svariem).

Summēšanas funkcija (*propagation function*) $f_{NET}(\cdot)$, kombinējot ieejas signālus un svarus, izrēķina vienu vērtību, kas tālāk tiek padota aktivitātes funkcijai. Summēšanas funkcijas izrēķināto vērtību parasti apzīmē ar burtu virkni *NET*. Tipiskākā summēšanas funkcija ir summa no ieeju un attiecīgo svaru reizinājuma (parādīta Att. 2.1).

Aktivitātes funkcija (*activation function*) $f_{act}(\cdot)$, izmantojot summēšanas funkcijas vērtību *NET*, izrēķina neirona izejas vērtību *y*. Aktivitātes funkcija ir svarīgākā neirona darbības raksturotāja, kas var norādīt uz neirona spēju risināt noteiktas sarežģītības uzdevumus (piemēram, „nelineārs neirons" nozīmē, ka neironā ir nelineāra aktivitātes funkcija). Aktivitātes funkciju piemēri ir: lineāra, sliekšņveida, sigmoidāla. Att. 2.1 redzama sliekšņveida aktivitātes funkcija.

2.2. Vienkārši neirona piemēri

2.2.1. Piemērs #1. Loģiskā UN (x_1 AND x_2) modelēšana

Apskatīsim neironu, kas aprēķina vienkāršu funkciju – loģisko UN (x_1 AND x_2; Tab. 2.1).

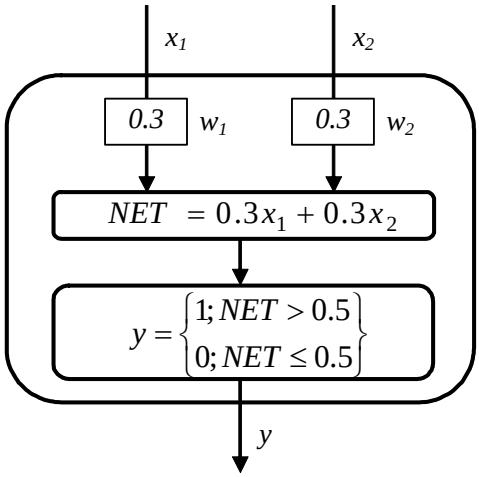
Tab. 2.1. Funkcija „Loģiskais UN" (x_1 AND x_2).

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Loģiskā UN modelēšanai nepieciešama sistēma (t.i. neirons) ar divām ieejām un (protams) vienu izeju.

Šim nolūkam izmantosim Att. 2.1 parādīto neirona modeli (ar sliekšņveida aktivitātes funkciju) ar divām ieejām un sliekšņa vērtību $\Theta=0.5$.

Tik vienkāršam piemēram dotajā neirona modelī nav grūti izdomāt, ka svara vērtības var būt, piemēram, attiecīgi 0.3 un 0.3 (Att. 2.2).



Att. 2.2. Neirons loģiskā UN modelēšanai

Tab. 2.2. Loģiskā UN modelēšana ar neironu, kas parādīts Att. 2.2 (NET, y – neirona izrēķinātās vērtības; d – „vēlamā" (*desired*) funkcijas vērtība).

x_1	x_2	NET	y	$d = x_1$ AND x_2
0	0	0.0	0	0
0	1	0.3	0	0
1	0	0.3	0	0
1	1	0.6	1	1

2.2.2. Piemērs #2. Funkcijas NOT (x_1 AND x_2) modelēšana

Otrajā piemērā apskatīsim funkciju kas pretēja loģiskajam UN (NOT (x_1 AND x_2); Tab. 2.3).

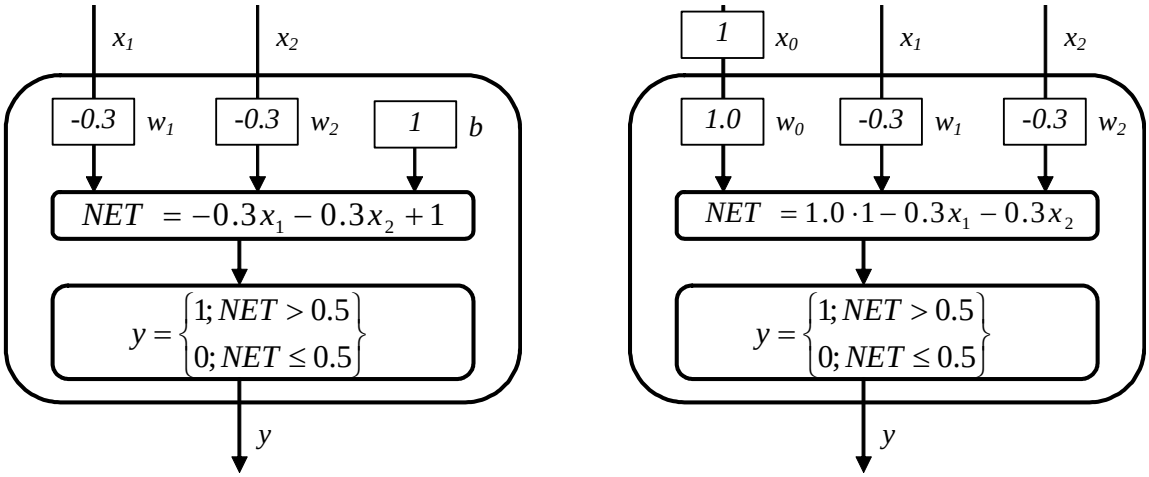
Tab. 2.3. Funkcija NOT (x_1 AND x_2).

x_1	x_2	NOT (x_1 AND x_2)
0	0	1
0	1	1
1	0	1
1	1	0

Iepriekš pielietotais modelis (Att. 2.1), kurš tika lietots loģiskā UN modelēšanai, šeit neder, jo, piemēram, ne pie kādiem svariem nav iespējams dabūt, ka ar ieejām attiecīgi 0 un 0 izejā varētu dabūt 1.

Modelis ir jāpapildina, un to veic, pievienojot papildus svara jeb nobīdes (*bias*) jēdzienu.

Papildus svaru parasti apzīmē ar *b* vai w_0. Otrajā gadījumā tiek formāli uzskatīts, ka ir tāds 0-tais ievads, pa kuru neironā vienmēr ienāk vērtība 1 – šāda interpretācija atvieglo neirona realizāciju datorprogrammā.

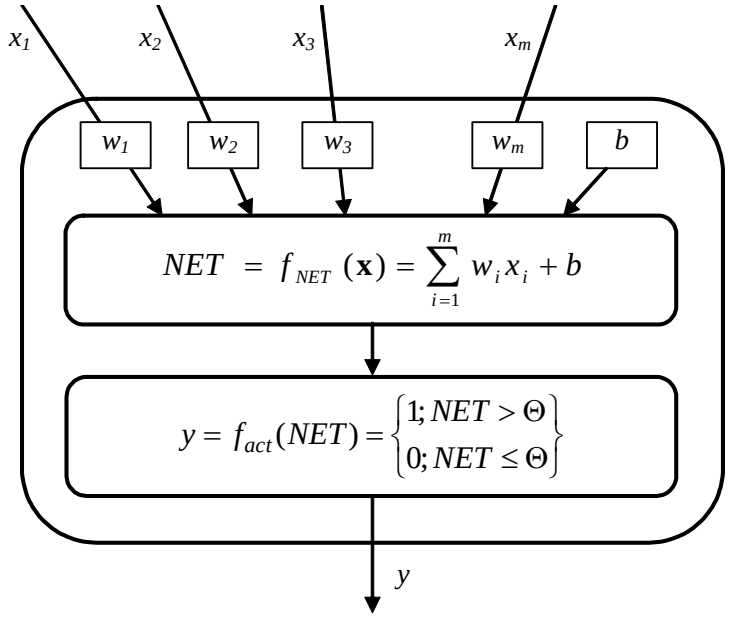


Att. 2.3. Neirons funkcijas NOT (x_1 AND x_2) modelēšanai. Abi varianti atšķiras pēc papildus svara pieraksta veida, bet ir funkcionāli identiski

Tab. 2.4. Funkcijas NOT (x_1 AND x_2) modelēšana ar neironu, kas parādīts Att. 2.3.

x_1	x_2	NET	y	d = NOT (x_1 AND x_2)
0	0	1.0	1	1
0	1	0.7	1	1
1	0	0.7	1	1
1	1	0.4	0	0

Ieviešot papildus svaru, ir attiecīgi jāpapildina neirona uzbūves shēma (sk. Att. 2.4).



Att. 2.4. Mākslīgā neirona uzbūves shēma ar papildus svaru (salīdzināt ar Att. 2.1)

2.3. Summēšanas funkciju veidi

Summēšanas (izplatīšanās) funkcija $f_{NET}(\cdot)$, kombinējot ieejas signālus un svarus, izrēķina vienu vērtību, kas tālāk tiek padota aktivitātes funkcijai. Summēšanas funkcijas izrēķināto vērtību parasti apzīmē ar burtu virkni *NET*.

Viena no biežāk lietotajām summēšanas funkcijām ir summa no ieeju un attiecīgo svaru reizinājuma (Formula (2.1)).

$$NET = \sum_{i=1}^{m} w_i x_i + b \tag{2.1}$$

Kā jau minēts iepriekšējos piemēros, papildus svara *b* iesaistīšanos summēšanas funkcijā var pierakstīt arī kā w_0, papildus pieņemot, ka x_0=1 (Formula (2.2)).

$$NET = \sum_{i=0}^{m} w_i x_i \tag{2.2}$$

(2.2) ir ērti pierakstīt arī matricu notācijā (Formula (2.3)).

$$NET = \sum_{i=0}^{m} \mathbf{xw} = \begin{pmatrix} x_0 & x_1 & \dots & x_m \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_m \end{pmatrix} \tag{2.3}$$

Par summēšanas funkciju dažos neironu tīklu modeļos lieto arī Eiklīda attālumu (Formula (2.4)) vai tā kvadrātu.

$$NET = \|\mathbf{x} - \mathbf{w}\| = \sqrt{\sum_{i=1}^{m} (x_i - w_i)^2} \tag{2.4}$$

Ir neironu modeļi, kur tiek izmantotas arī „eksotiskākas" summēšanas/izplatīšanās funkcijas (Formulas (2.5), (2.6), (2.7)).

$$NET = \prod_{i=1}^{m} w_i x_i \tag{2.5}$$

$$NET = \max \{w_i x_i\}_{i=1}^{m} \tag{2.6}$$

$$NET = \min \{w_i x_i\}_{i=1}^{m} \tag{2.7}$$

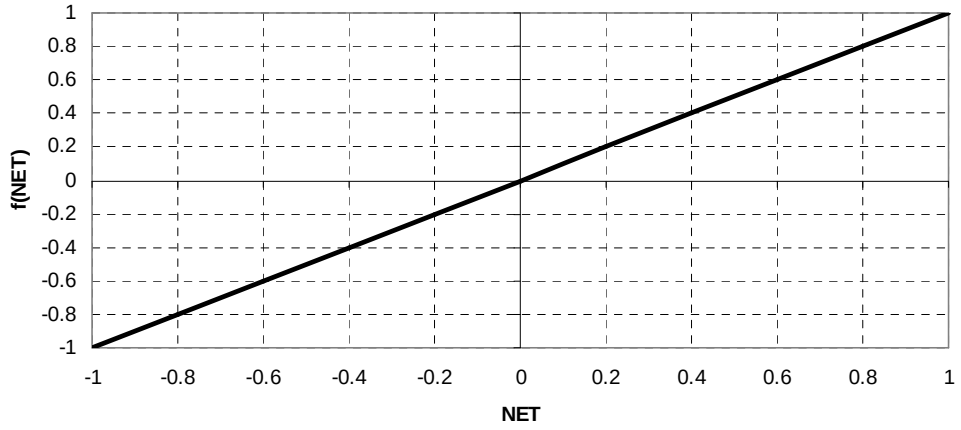
2.4. Aktivitātes funkciju veidi

Aktivitātes funkcija $f_{act}(\cdot)$, izmantojot summēšanas funkcijas vērtību *NET*, izrēķina neirona izejas vērtību *y*. Aktivitātes funkcija ir svarīgākā neirona darbības raksturotāja, kas var norādīt

uz neirona spēju risināt noteiktas sarežģītības uzdevumus (piemēram, „nelineārs neirons" nozīmē, ka neironā ir nelineāra aktivitātes funkcija).

Lineāra aktivitātes funkcija (Formula (2.8)) ir triviāls aktivitātes funkcijas variants, kas nenodrošina iespēju risināt „sarežģītākas" problēmas, tomēr dod iespēju veikt apstrādi (šajā gadījumā – neirona vai neironu tīkla apmācību), izmantojot tradicionālas lineāras metodes.

$$f_{lin}(NET) = NET \tag{2.8}$$

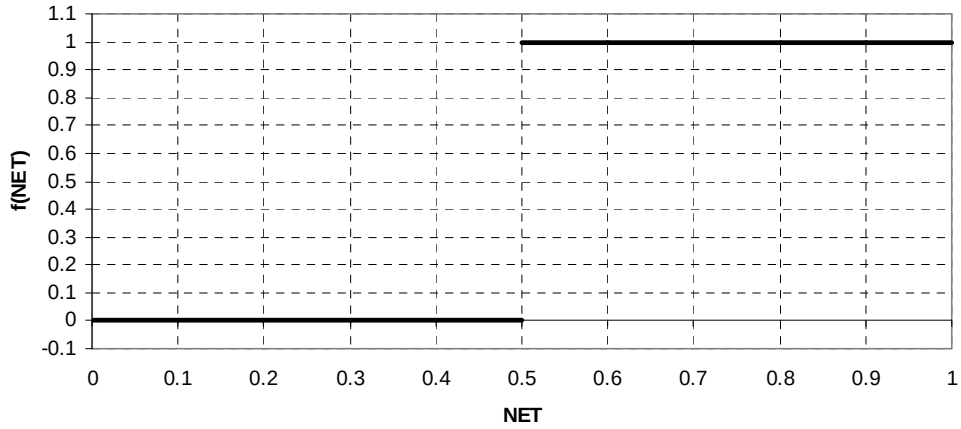


Att. 2.5. Lineāra aktivitātes funkcija

Ja modelējamā funkcija ir bināra, t.i., tās izejā ir, piemēram, tikai 0 vai 1, bet nav starpvērtību, var tikt lietota **sliekšņveida** aktivitātes funkcija, kas ir vienkāršākā nelineārā funkcija.

$$f_{\Theta}(NET) = \begin{cases} B; & NET > \Theta \\ A; & NET \leq \Theta \end{cases}, \tag{2.9}$$

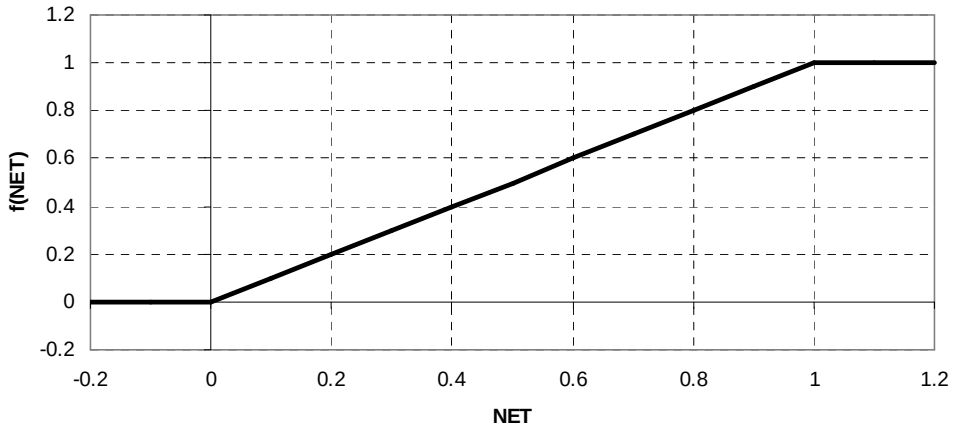
kur Θ – slieksnis (*threshold*).



Att. 2.6. Sliekšņveida aktivitātes funkcija (Θ=0.5; A=0.0; B=1.0)

Praksē parasti ērtāka ir **„pseidolineāra”** aktivitātes funkcija (Formula (2.10)) – intervālā ([A, B]) ierobežota lineārā aktivitātes funkcija, kas ir sava veida kompromiss starp lineāro un sliekšņveida funkciju. Kaut arī tā pēc būtības nav lineāra funkcija, tomēr kaut kādā tuvinājumā to var pieņemt par tādu, un ērtība šeit izpaužas apstāklī, ka ir iespējams pielietot lineāras metodes neirona vai neironu tīkla apmācībā.

$$\mathrm{f}_{\mathrm{pseudo\_lin}}\ (NET\ ) = \left\{ \begin{array}{ll} B; & NET\ > B \\ A; & NET\ < A \\ NET\ ; & else \end{array} \right\} \tag{2.10}$$



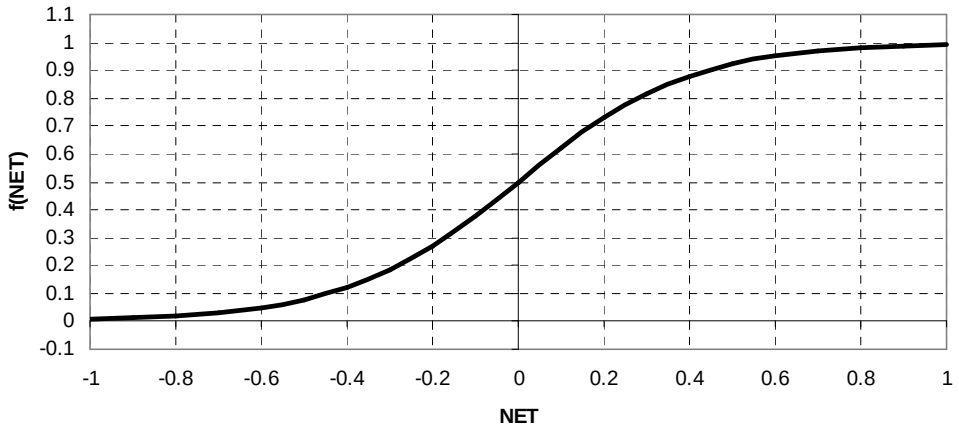
Att. 2.7. Lineāra aktivitātes funkcija, kas ierobežota intervālā (A=0.0; B=1.0)

Sigmoidālās funkcijas veido S-veida grafiku. Sigmoidālās aktivitātes funkcijas ir ne vien „skaistas”, bet arī bioloģiski pamatotas. Bez tam tās ir arī praktiski ērtas, jo ļauj ierobežot izejas vērtību noteiktā intervālā, kam ir ļoti vērtīgais normalizācijas efekts. Sigmoidālās funkcijas ir nelineārās, bet, atšķirībā no sliekšņveida funkcijām – arī atvasināmas.

Viena no populārākajām aktivitātes funkcijām ir t.s. **loģistiskā** aktivitātes funkcija (Formula (2.11)). Tās izejas vērtība ir ierobežota intervālā [0, 1].

$$\mathrm{f}_{\log}\ (NET\ ) = \frac{1}{1 + e^{-\frac{1}{g} NET}}, \tag{2.11}$$

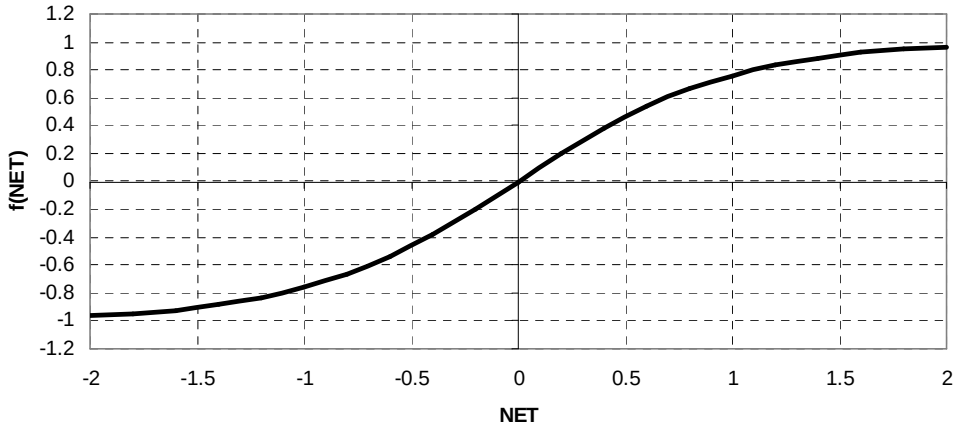
kur g – līknes slīpuma koeficients (*gain*).



Att. 2.8. Loģistiskā aktivitātes funkcija (g=0.2)

Loģistiskajai funkcijai ļoti līdzīgs ir **hiperboliskais tangenss**, kas izeju ierobežo intervālā [-1, 1] (Formula (2.12)).

$$f_{tanh}(NET) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.12}$$



Att. 2.9. Hiperboliskais tangenss

Abu sigmoidālo funkciju vērtīga īpašība ir tā, ka to **atvasinājums ir izsakāms ar pašas funkcijas vērtību**, kas ir ļoti nozīmīgi tāda neironu tīklu modeļa, kā perceptrons apmācībā.

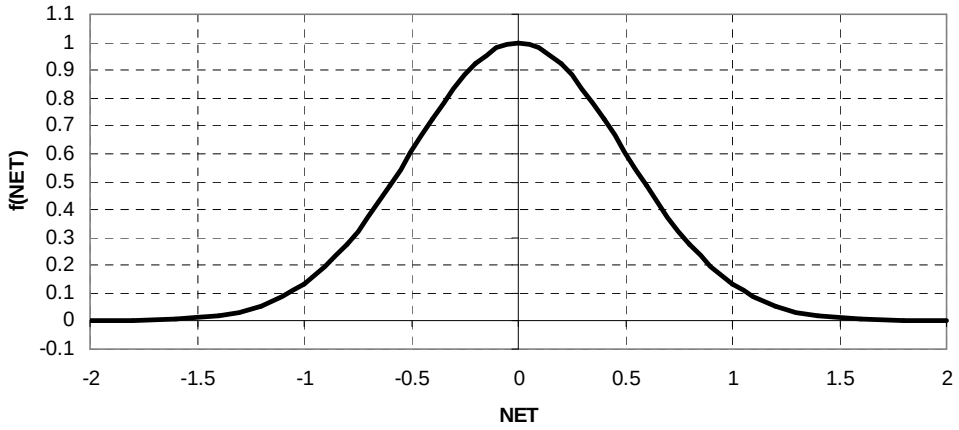
$$f_{log}' = f_{log}(1 - f_{log}) \tag{2.13}$$

$$f'_{tanh} = 1 - f^{2}_{tanh} \tag{2.14}$$

Vairākos neironu tīklu modeļos tiek lietota **Gausa** funkcija (Formula (2.15)).

$$f_{Gauss}(NET) = e^{-\frac{NET^2}{2\sigma^2}}, \tag{2.15}$$

kur σ (*sigma*) – līknes slīpuma koeficients.



Att. 2.10. Gausa funkcija (σ=0.5)

3. Neironu tīklu topoloģija

3.1. Lineāras un nelineāras problēmas

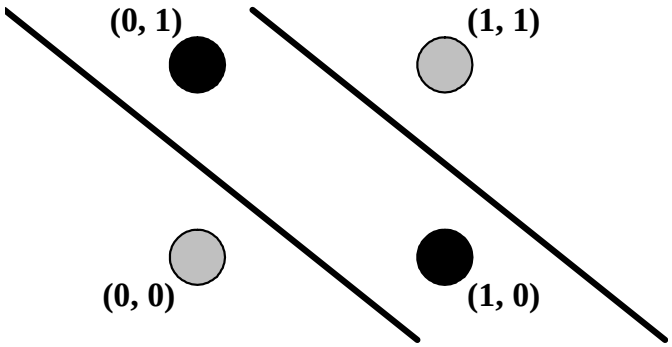
Apskatīsim bināru loģisko funkciju – izslēdzošo VAI (x_1 XOR x_2; Tab. 3.1).

Tab. 3.1. Funkcija „Izslēdzošais VAI" (x_1 XOR x_2).

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Lai kā mēs censtos, izmantojot iepriekš aprakstīto neirona modeli (Att. 2.4), mums neizdosies to nomodelēt.

Izrādās, ka XOR problēma pieder t.s. **nelineārajām problēmām**, t.i., kuras nevar atrisināt, novelkot vienu līniju (vienkāršoti sakot, vajadzīgas vismaz divas).



Att. 3.1. XOR – vienkāršākā nelineārā klasifikācijas problēma. Lai sadalītu punktus divās klasēs, nepieciešamas divas taisnes (ar vienu līniju tas nav iespējams)

Izrādās, ka ar vienu vienīgu neironu nevar atrisināt nelineāras problēmas, kaut arī šī neirona aktivitātes funkcija būtu nelineāra.

Tab. 3.2. Lineāri atdalāmu n-argumentu Būla funkciju skaits [Scherer, 1997].

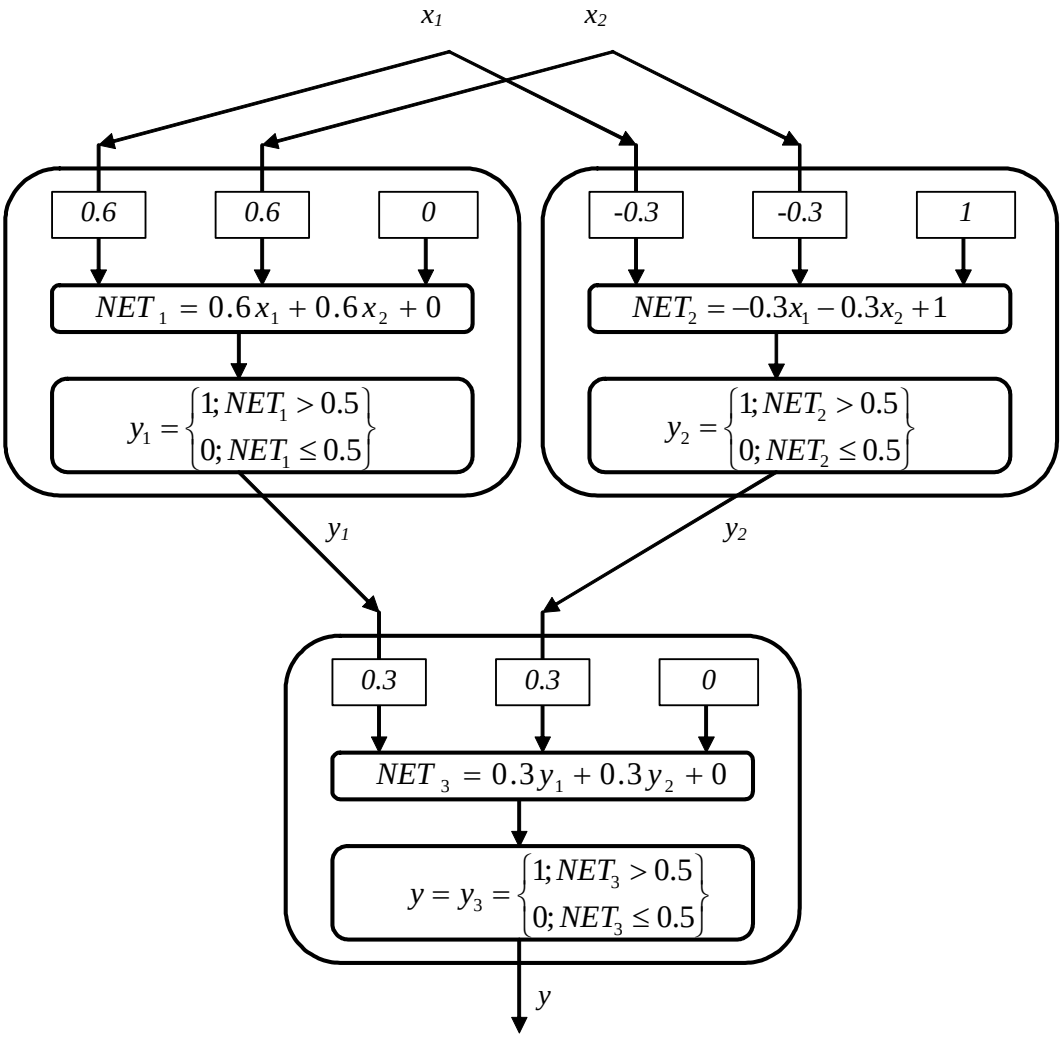
n	Skaits kopā	Lineāri atdalāmo funkciju skaits
1	4	4
2	16	14
3	256	104
4	65'536	1'772
5	$4.3*10^9$	94'572
6	$1.8*10^{19}$	5'028'134

Cik liela nozīme ir neironu tīkla spējai risināt nelineāras problēmas? Izrādās, ka ļoti liela, un tas tā ir nelineāro problēmu lielā īpatsvara dēļ. Ja starp 16 divu argumentu Būla funkcijām tikai divas ir nelineāras (XOR un ekvivalence), tad, palielinoties argumentu skaitam, nelineāru funkciju īpatsvars strauji pieaug (Tab. 3.2).

3.2. XOR modelēšana ar divu slāņu neironu tīklu

XOR iespējams modelēt ar trīs neironiem, kas izvietoti divos slāņos (slāņi ir tādi neironu kopumi, kas izvietoti viens aiz otra).

Funkcija x_1 XOR x_2 tiek modelēta (Att. 3.2) kā (x_1 OR x_2) AND NOT (x_1 AND x_2), un katra no formulā ietilpstošajām trim lineārajām funkcijām (OR, NOT AND, AND) tiks modelēta ar vienu neironu, kas konstruēts pēc shēmas, kas parādīta Att. 2.4.



Att. 3.2. Neironu tīkls funkcijas (x_1 XOR x_2) modelēšanai

Lai varētu **modelēt nelineāru problēmu**, neironu tīklam jāatbilst šādām minimālajām prasībām:

1. Neironu tīklā jābūt vismaz vienam t.s. slēptajam slānim (tādam, kura izeja reizē nav visa neirona izeja; dotajā piemērā pie slēptā slāņa pieder neironi #1 un #2);

2. Neironu aktivitātes funkcijai jābūt nelineārai (vismaz slēptajā slānī); dotajā piemērā tiek lietota sliekšņveida aktivitātes funkcija, kas ir nelineāra.

Tab. 3.3. Neironu tīkla (Att. 3.2) pārbaude ($y = x_1$ XOR x_2).

x_1	x_2	NET_1	y_1	NET_2	y_2	NET_3	y
0	0	0	0	1	1	0.44	0
0	1	0.6	1	0.7	1	0.74	1
1	0	0.6	1	0.7	1	0.74	1
1	1	1.2	1	0.4	0	0.44	0

3.3. Neironu tīklu topoloģiju veidi

> **Neironu tīkla topoloģija** jeb arhitektūra ir neironu izvietojums tīklā un to savstarpējās saites.

Topoloģija, blakus neirona aktivitātes funkcijai, ir viena no būtiskiem aspektiem, kas nosaka neironu tīkla skaitļošanas spēju.

Neironi neironu tīklā parasti tiek grupēti slāņos.

> **Neironu slānis** (*layer*) ir neironu kopums, kas topoloģiski izvietoti kopā un kuru darbināšana un apmācība parasti notiek pēc viena algoritma.

Neironu tīkla topoloģiju nosaka:

- Neironu slāņu savstarpējais izvietojums un to neironu sasaiste;
- Neironu sasaiste slāņa ietvaros.

Tipiska vairāku slāņu neironu arhitektūra ir, ka slāņi ir izvietoti virknē viens aiz otra, turklāt divu blakusesošo slāņu neironi ir savienoti katrs ar katru.

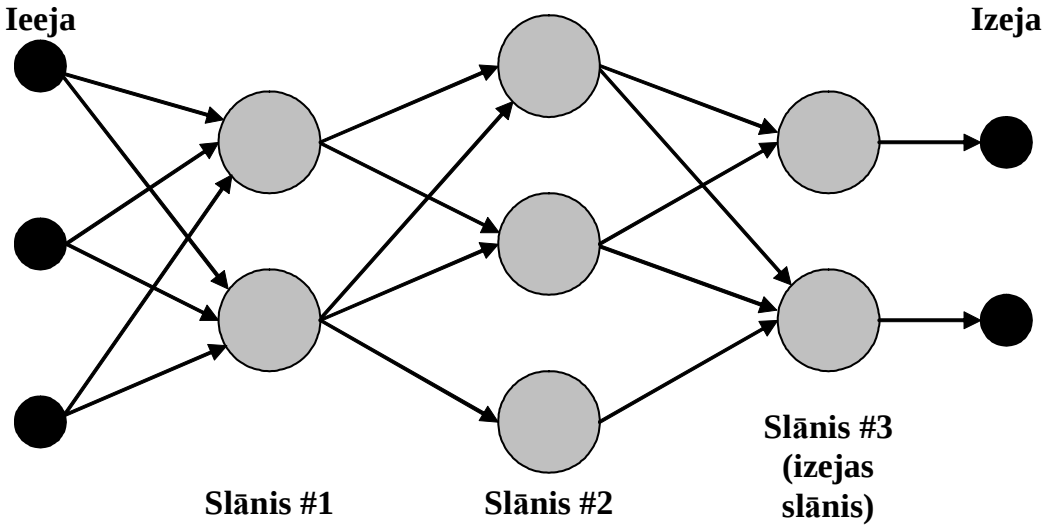
Pēc neironu slāņu izkārtojuma neironu tīkli iedalās divos galvenajos veidos:

- **Vienvirziena** tīkli (feedforward networks);
- **Rekursīvie** tīkli (recurrent/feedback networks).

3.3.1. Vienvirziena tīkli

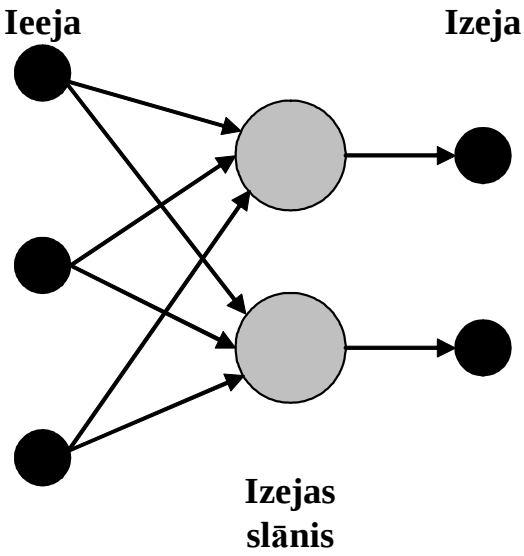
Vienvirziena tīklos slāņi ir izkārtoti virknē pēc kārtas, līdz ar to tos var numurēt pēc kārtas. Tīkla darbināšana notiek, sākot ar ieejas slāni (nr. 1) un beidzot ar izejas slāni (nr. m). Saites starp neironiem ir tikai virzienā no slāņa ar mazāku numuru uz slāni ar lielāku numuru. Neironu tīkla vienvirziena topoloģija ar 3 slāņiem parādīta Att. 3.3. Beidzamais no slāņiem ir izejas slānis.

> **Izejas slānis** (*output layer*) ir neironu slānis, kas tieši saistīts ar apkārtējo vidi, kur tiek aizvadītas izejas vērtības. Neironu slānis, kura neironu izejas ir arī visa neironu tīkla izejas.



Att. 3.3. Vienvirziena neironu tīkls (saites tikai starp blakus slāņiem)

Att. 3.3 ir pilnīgi skaidrs, ka izejas slānis ir slānis #3, bet ar ieejas slāņa definēšanu nav tik vienkārši, jo, izmantojot šo pieeju, neironu tīklam ir ieejas, bet ieejas slāņa nav (ir tikai izejas slānis un citi slāņi). Šajā izpratnē **vienslāņa** (*single-layer*) neironu tīkls ir tāds tīkls, kuram ir tikai viens – izejas slānis (Att. 3.5).



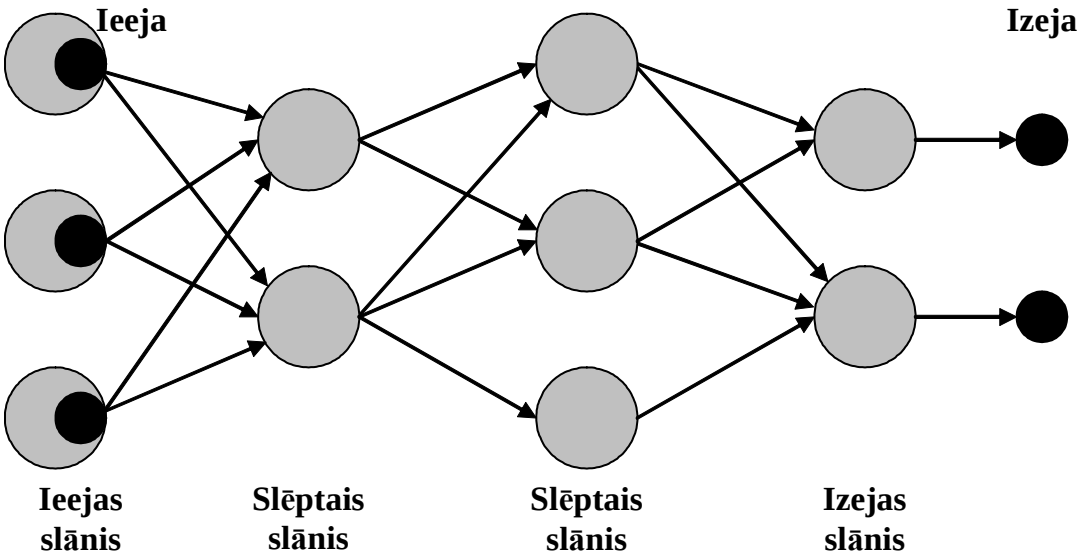
Att. 3.4. Vienslāņa vienvirziena neironu tīkls ar diviem neironiem

Lai atvieglotu neironu tīkla realizāciju (datorprogrammā), bieži vien neironu tīkla slāņu struktūra tiek reprezentētā citā izpratnē, resp., tiek mākslīgi nodefinēts ieejas slānis, kura neironi „neko nedara", bet uz kuru izejām tiek padotas neironu tīkla ieejas vērtības (Att. 3.5).

Šajā izpratnē neironu tīklam sanāk par vienu slāni vairāk, un visi slāņi, kas nav ne izejas, ne ieejas ir slēptie slāņi..

> **Ieejas slānis** (*input layer*) ir tāds neironu slānis (iespējams, mākslīgi veidots), uz kura neironu izejām tiek uzstādītās visa neironu tīkla ieejas.

> **Slēptais slānis** (*hidden layer*) ir tāds neironu slānis, kas nav ne ieejas, ne izejas slānis.

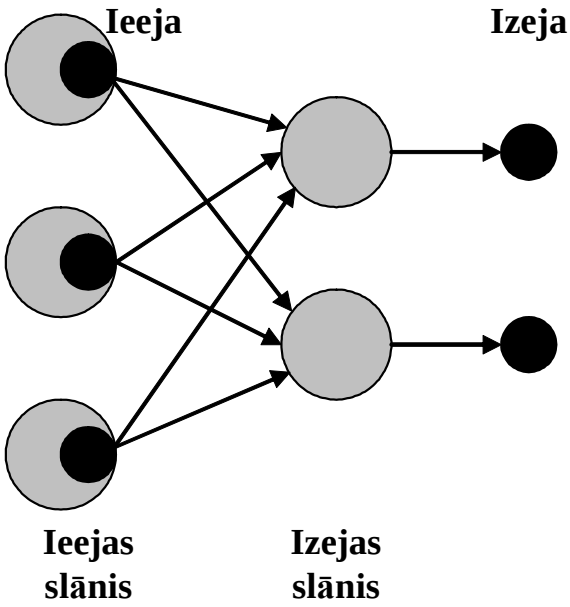


Att. 3.5. Pirmās kārtas vienvirziena neironu tīkls, iesaistot ieejas slāņa jēdzienu

Var teikt, ka vienvirziena neironu tīkliem ieejas slānis vienmēr ir „mākslīgs" jeb pseido-slānis. Lai vienas vai otras pieejas piekopšana neironu tīkla arhitektūras definēšanā neradītu sajukumu, tad parasti saka – nevis, piemēram, 3 vai 4 slāņu neironu tīkls (salīdzināt Att. 3.3 un Att. 3.5), bet gan neironu tīkls ar 2 slēptajiem slāņiem.

Tādējādi, vienslāņa neironu tīkls var tikt definēts kā tīkls, kurā nav slēpto slāņu (Att. 3.6).

> **Vairākslāņu** (*multi-layer*) neironu tīkls ir tāds neironu tīkls, kuram ir vismaz viens slēptais slānis.

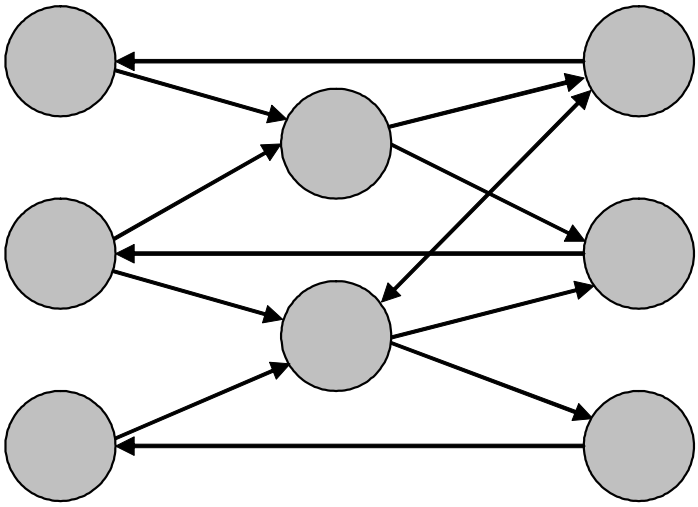


Att. 3.6. Vienslāņa vienvirziena neironu tīkls, iesaistot ieejas slāņa jēdzienu

3.3.2. Rekursīvie tīkli

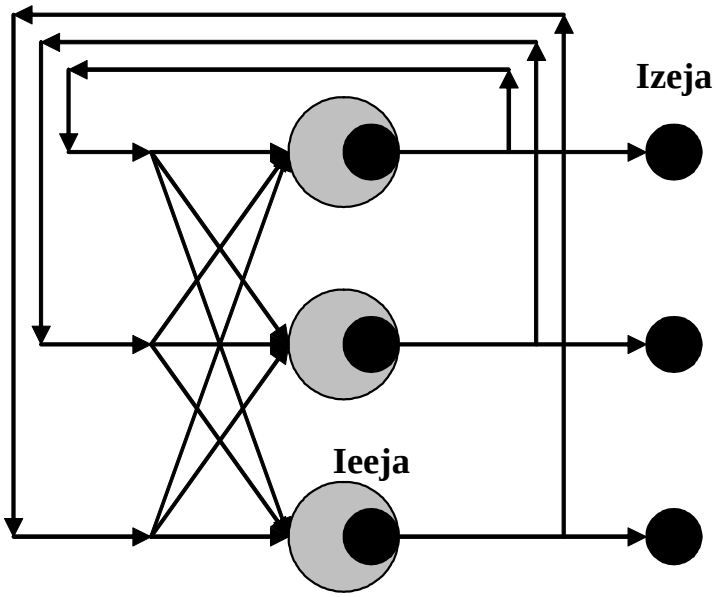
Rekursīvajos tīklos neironu papildus var tikt saistīti viens ar otru šādos veidos:

- no slāņa ar lielāku numuru uz slāni ar mazāku numuru (Att. 3.7);
- slāņa ietvaros (Att. 3.8).



Att. 3.7. Rekursīvs neironu tīkls

Vienslāņa rekursīvajam neironu tīklam, kas parādīts Att. 3.8, ieejas un izejas slānis ir viens un tas pats slānis (salīdzināt ar vienslāņa vienvirziena neironu tīklu Att. 3.4 un Att. 3.6, kur ir tikai izejas slānis, bet īsta ieejas slāņa nemaz nav.



Att. 3.8. Vienslāņa rekursīva neironu tīkla piemērs (Hopfīlda modelis)

3.4. Neironu tīkla un tā darbības notācija vairāku slāņu topoloģijā

Aprakstot neironu tīkla darbību, tiek lietoti dažādi apzīmējumi un notācijas.

Formulās svarus ļoti bieži apzīmē ar mazo burtu *w* un indeksu, kas norāda svara kārtas numuru neironā, piemēram, w_2.

Ja formula attiecas nevis uz vienu neironu, bet uz visu slāni, tad ir nepieciešami divi indeksi – neironam un slānim, piemēram, $w_{3,5}$ ir 3-ā neirona 5-tais svars.[1]

Atsevišķos gadījumos tiek lietots vēl trešais indekss – neironu slāņa numuram (piemēram, $w_{1,3,5}$), bet praktiski (it sevišķi, realizējot datorprogrammās) šādu pierakstu lieto retāk, jo, ja visos viena slāņa neironos svaru skaits parasti ir vienāds, tad dažādos slāņos neironu skaiti parasti atšķiras, līdz ar ko visa neironu tīkla svaru kopumu datorprogrammā ir grūti realizēt kā vienu masīvu, bet realizācija ir ērtāk veicama katram slānim atsevišķi.

Vēl sarežģītāk ir ar ieejām un izejām, jo kas vienam slānim ir izeja, tas otram var būt ieeja (sk. vērtības y_1 un y_2 attēlā Att. 3.2, tāpēc nav skaidrs, kā būtu pareizi kodēt – ar *x* kā ieeju vai *y* kā izeju.

Lai atrisinātu šo notācijas problēmu, var darīt šādi[2]:

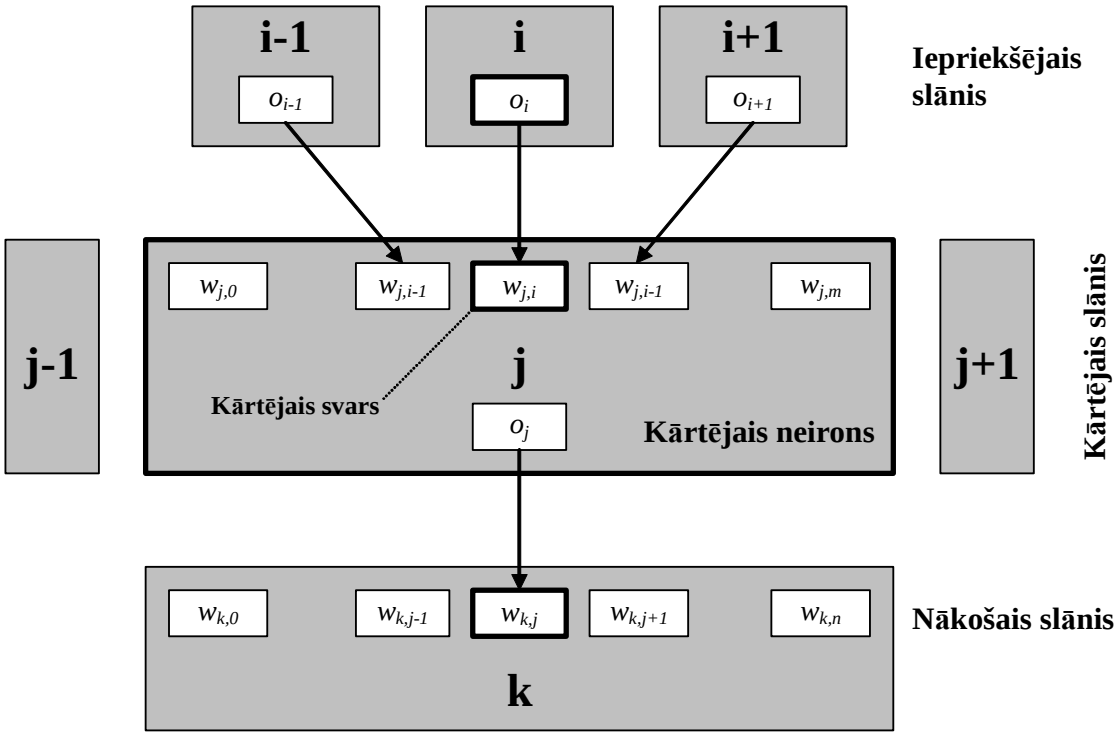
- pieņem, ka noteiktā laika momentā darbība (t.i., aprēķini, notiek fiksētā slānī un fiksētā neironā un atsevišķos gadījumos arī svars ir fiksēts);

[1] Dažos literatūras avotos secība ir apgriezta, t.i., pirmais indekss norāda svara numuru neironā, bet otrs – neirona numuru slānī.

[2] Tiek pieņemts, ka blakusesošo slāņu neironi tiek savienoti katrs ar katru līdz ar ko viena slāņa neironu skaits ir vienāds ar nākošā slāņa svaru skaitu katrā no neironiem.

- kārtējā slāņa kārtējā neirona numurs (un attiecīgi, ja nepieciešams, nākošā slāņa fiksēta neirona svara numurs) tiek apzīmēts ar j, iepriekšējā slāņa fiksēta neirona numurs (un attiecīgi, ja nepieciešams, kārtējais svars) ar i, bet nākošā slāņa fiksēta neirona numurs ar k.[3]
- ieeju un izeju kodēšanai lieto simbolu *o* (*output*).

Ļoti līdzīgi apzīmējumi tiek lietoti arī citos avotos, tomēr katrā avotā parasti ir nelielas izklāsta nianses, tāpēc ir svarīgi precizēt izmantoto apzīmējumu lietojumu.



Att. 3.9. Neironu tīkla komponenšu notācija

[3] Ievērojiet, ka šo (indeksiem izmantoto) burtu i, j un k alfabētiskā secība atbilst slāņu secībai.

4. Neironu tīkla apmācīšanās

> **Apmācība** (*training*; *learning*) ir neirona tīkla svaru vērtību (un dažreiz arī citu parametru) uzstādīšana, balstoties uz apmācības paraugiem, kas reprezentē noteiktu problēmu.

Dažreiz rodas neskaidrība par to, ko īsti nozīmē **apmācība** (*training*) un ko **apmācīšanās** (*learning*). Patiesībā šie abi termini apzīmē apmēram vienu un to pašu darbību, tikai apmācīšanās ir no neironu tīkla viedokļa, bet apmācība – no ārēja procesa skatupunkta.

[Haykin, 1999] dota šāda apmācības definīcija:

> **Apmācīšanās** ir process, kurā notiek neironu tīkla brīvo parametru pielāgošana, kas tiek veikta ar simulāciju vidē, kurā neironu tīkls ir ievietots.

Apmācības algoritms ir ļoti svarīga neironu tīkla uzbūves sastāvdaļa, un katram neironu tīklu modelim tas ir atšķirīgs, tomēr var nosaukt vairākus pamatprincipus, kuri tiešāk vai netiešāk tiek pielietoti vairuma neironu tīklu apmācībā.

4.1. Apmācības process

Apmācības process nosaka šādu notikumu virkni (pēc [Haykin, 1999]):

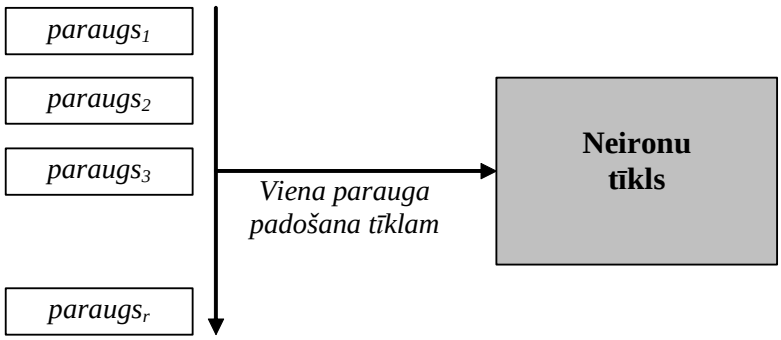
3. Neironu tīkls tiek darbināts noteiktā vidē;
4. Balstoties uz šo darbināšanu, neironu tīklā tiek veiktas izmaiņas brīvajos parametros;
5. Ņemot vērā veiktās izmaiņas neironu tīkla iekšējā struktūra, neironu tīkls reaģē uz vidi citā veidā.

> **Apmācības process** (*training process*) ir darbību kopums, kura laikā neironu tīklam noteiktā secībā vairākkārtīgi tiek padoti apmācības paraugi, pēc katra apmācības parauga vai paraugu kopuma padošanas katram neironam piemērojot apmācības algoritmu.

Apmācības procesa sākumā neironu tīkla svari noteiktā veidā (parasti – nejauši) tiek aizpildīti ar sākuma vērtībām.

Apmācības process parasti notiek pa epohām.

> **Epoha** (*epoch*) ir apmācības procesa daļa, kuras laikā neironu tīklam tieši vienu reizi tiek padoti visi apmācības paraugi.



Att. 4.1. Epoha

Tikai neliela daļa apmācības algoritmu dod iespēju apmācīt neironu tīklu vienas epohas laikā, t.i. vienā solī. Parasti apmācības process ir garāks, un katrs paraugs tā laikā tiek vairākas (un pat daudzas[4]) reizes parādīts neironu tīklam.

Ir divas metodes apmācības procesa organizēšanai:

- **vienlaidus** (*continuous*) metode (Alg. 4-1), kad svaru izmaiņa tiek veikta pēc katra parauga parādīšanas tīklam;
- **pakešu** (*batch*) metode (Alg. 4-2), kad svaru izmaiņa tiek veikta tikai epohas beigās (t.i., pēc visu paraugu parādīšanas).

Alg. 4-1. Vienlaidus apmācība (*continuous_training*).

```
Procedure continuous_training
Input configuration
    nnet – (neapmācīts) neironu tīkls
    Patterns – apmācības paraugu kopums
Output configuration
    nnet – (apmācīts) neironu tīkls
Using
    termination_criterion_met – apmācības procesa beigšanas pazīme
Begin
    Sagatavo nnet apmācībai;
    While Not termination_criterion_met
        Forall p in Patterns do
            Padod p ieejā tīklam nnet un darbina tīklu;
            Modificē nnet svarus
        Endforall
    Endwhile
End
```

Parasti apmācībai tiek izmantota tieši vienlaidus metode, tomēr atsevišķos gadījumos eksperimentāli noskaidrots, ka labākus rezultātus (ātrāka apmācīšanās, korektāka funkcijas aproksimācija) var iegūt ar pakešu metodi.

[4] Epohu skaits var būt mērāms pat tūkstošos.

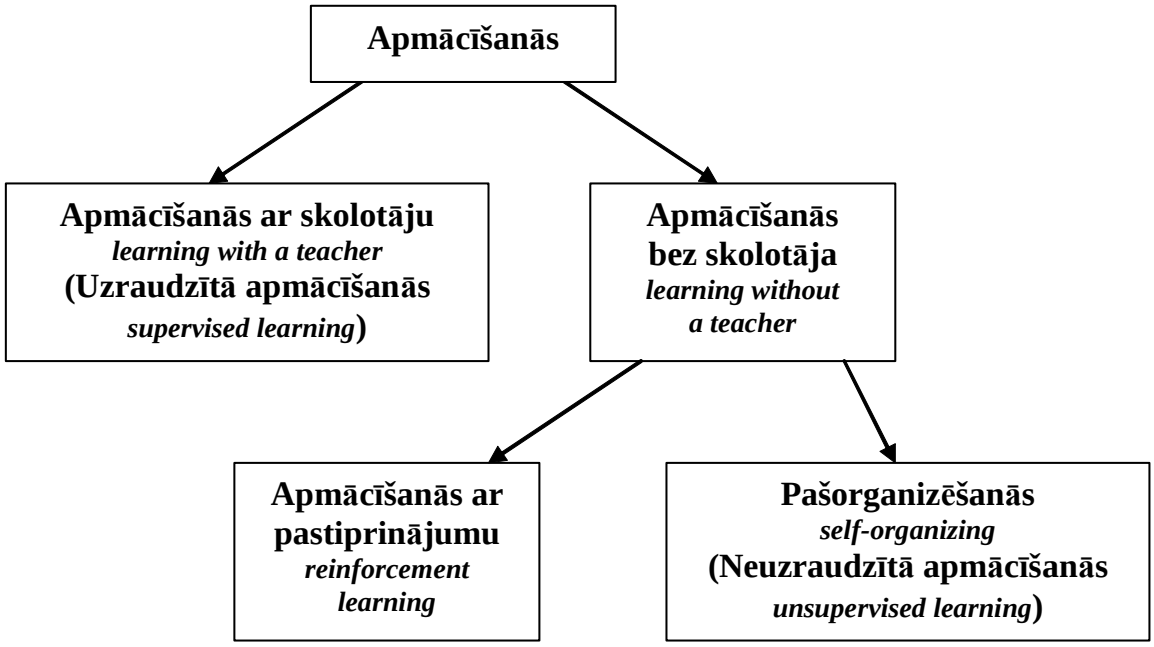
Tas, vai izmantot vienlaidus vai pakešu metodi, nav teorētiski pamatojams un ir atkarīgs no situācijas (neironu tīkla konfigurācija, problēmas tips un reprezentācija), tāpēc ir uzskatāms par heiristiku.

Alg. 4-2. Pakešu apmācība (*batch_training*).

Procedure *batch_training*
Input configuration
nnet – (neapmācīts) neironu tīkls
Patterns – apmācības paraugu kopums
Output configuration
nnet – (apmācīts) neironu tīkls
Using
termination_criterion_met – apmācības procesa beigšanas pazīme
Begin
Sagatavo *nnet* apmācībai;
While Not *termination_criterion_met*
Forall *p* **in** *Patterns* **do**
Padod *p* ieejā tīklam *nnet* un darbina tīklu
Endforall;
Modificē *nnet* svarus
Endwhile
End

4.2. Apmācīšanās paradigmas

Atkarībā no ārējās vides (apmācības paraugi, papildus informācija par tiem, algoritms) ietekmes uz neironu tīklu svaru un citu parametru izmaiņu, tiek izšķirtas vairākas apmācīšanās paradigmas jeb kategorijas (Att. 4.2).

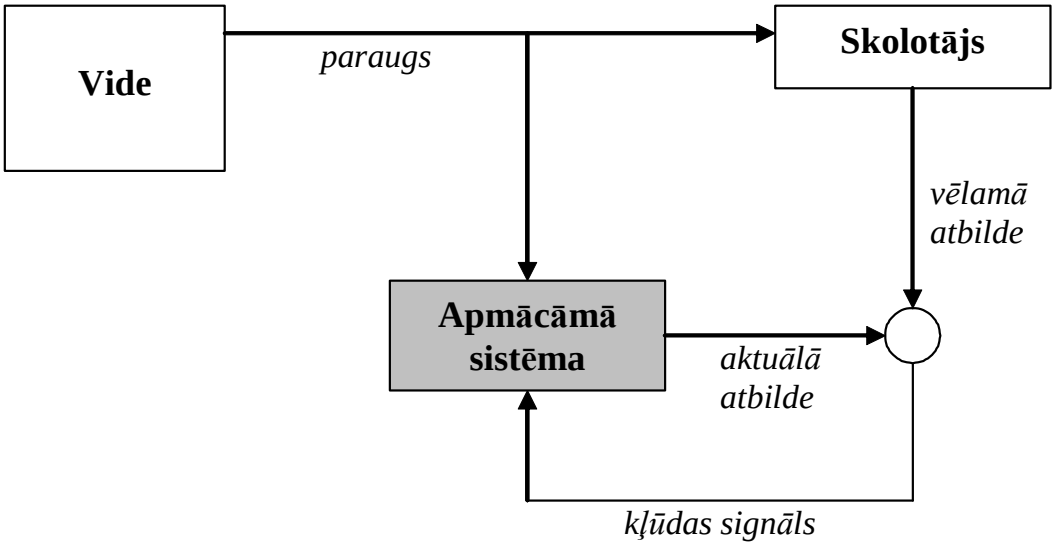


Att. 4.2. Neironu tīklu apmācības paradigmas

4.2.1. Apmācīšanās „ar skolotāju”

Par „skolotāju” var nosaukt mehānismu, kuram ir informācija par vidi pārīšu formā **<paraugs, vēlamā atbilde>** jeb, no neironu tīkla viedokļa, **<ieeja, izeja>**. „Vēlamā atbilde” reprezentē neironu tīkla optimālo reakciju uz doto paraugu. Darbinot neironu tīklu ar doto paraugu, tiek iegūta **aktuālā atbilde** (*actual response*). Neironu tīkla svari tiek modificēti, balstoties gan uz padoto ieejas paraugu, gan **kļūdas signālu** (*error signal*), kas tiek definēts kā starpība starp vēlamo atbildi un aktuālo atbildi. Tādējādi var teikt, ka apmācīšanās „ar skolotāju” tiek realizēta ar **kļūdas korekcijas** (*error correction*) metodi.

Apmācīšanas „ar skolotāju” (Att. 4.3) ir visplašāk pielietotā mašīnapmācīšanās (t.sk. neironu tīklu) paradigma. Tās populārākais pārstāvis ir vairākslāņu perceptrons.

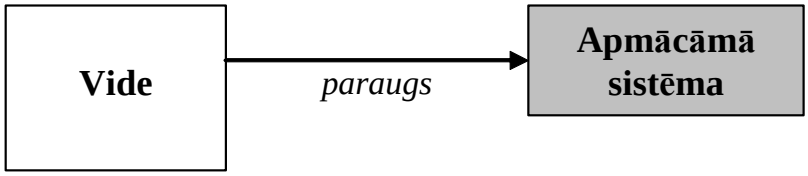


Att. 4.3. Apmācības „ar skolotāju" shēma (pēc [Haykin, 1999])

4.2.2. Neuzraudzītā apmācīšanās

Neuzraudzītās apmācīšanās (jeb pašorganizēšanās, Att. 4.4) gadījumā nav ārējā skolotāja vai cita mehānisma, kas uzraudzītu apmācības procesu. Pie šīs apmācīšanās paradigmas apmācāmā sistēma mēģina pati uztvert ieejas datu regularitāti un automātiski veido klases (resp., nosaka kā paraugi tiek klasificēti).

Viena no neuzraudzītās apmācīšanās realizācijas formām ir **apmācīšanās ar konkurenci** (*competitive learning*), kas tiek izmantota Kohonena modelī.



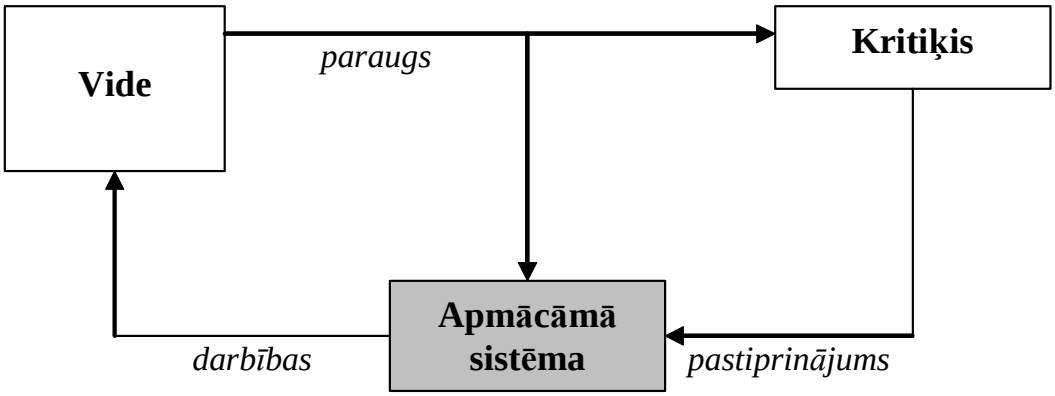
Att. 4.4. Neuzraudzītās apmācības shēma (pēc [Haykin, 1999])

4.2.3. Apmācīšanās ar pastiprinājumu

Apmācīšanās ar pastiprinājumu (*reinforcement learning*) ir salīdzinoši visgrūtāk uztveramā paradigma (daudzi to kļūdaini mēģina pieskaitīt neuzraudzītajai apmācībai), jo, piemēram, atšķirībā no apmācības ar skolotāju, nav tieši zināms katra ieejas parauga novērtējums, bet sasniedzamais mērķis ir globālāks, līdz ar to apmācīšanās notiek ar t.s. **aizturēto pastiprinājumu** (*delayed reinforcement*). Papildus faktors apmācīšanai ar pastiprinājumu ir tas, ka apmācītās sistēmas izeja tiek interpretēta kā darbība (*action*), nevis risinājums vai rezultāts.

Piemēri problēmām, kas atbilst *reinforcement learning* tipa problēmām, ir tādas spēles kā šahs vai dambrete, kur katra gājiena vērtība nav precīzi izvērtējama (rezultāts – uzvara, zaudējums vai neizšķirts – ir zināms tikai spēles beigās), bez tam katrā solī izdarāmais gājiens vispār nav izvērtējams atrauti no visas spēles kopumā (resp., gājiens pats par sevi nevar būt ne labs, ne slikts, laba vai slikta var būt tikai spēles maniere vai izvēlētā taktika).

Apmācība ar pastiprinājumu nav īpaši izplatīta neironu tīklu sfērā, atsevišķas tehnoloģijas neironu tīkliem būtu pieskaitāmas daļēji, vai arī tas atkarīgs no pieejas mašīnapmācības metožu klasifikācijā (piemēram, **dinamiskā programmēšana** (*dynamic programming*)).



Att. 4.5. Apmācības ar pastiprinājumu shēma

Apmācības ar pastiprinājumu shēma ir redzama Att. 4.5:

- Skolotāja vietā šeit stājas t.s. **kritiķis** (*critic*), kurš atšķiras no skolotāja tādējādi, ka tam nav zināma „pareizā atbilde" par ieejas paraugu, tai pat laikā zināms novērtējums (kaut arī ne tik tiešs un precīzs) ir pieejams.
- Otra un būtiska atšķirība ir, ka apmācāmā sistēma kā izejas rezultātu nosaka darbības, kas atstāj ietekmi uz vidi, no kuras nāk paraugi. Paraugu šāda tipa problēmu gadījumā parasti reprezentē vides stāvokli.

4.3. Apmācīšanās likumi

Šajā sadaļā tiks apskatīti daži svarīgi ar neironu tīkliem saistītie apmācīšanās likumi. Kāds no viņiem ir iebūvēts lielā daļā apmācīšanās algoritmu. Nodaļas izklāsts balstīts uz [Haykin, 1999].

4.3.1. Apmācīšanās ar kļūdu labošanu

Apmācīšanās ar kļūdu labošanu (*error-correction learning*) balstās uz to, ka katram ieejas signālam neironā ir zināma vēlamā atbilde, tādēļ, salīdzinot vēlamo atbildi ar neironu tīkla izdoto rezultātu, var izrēķināt kļūdu, kas ir par pamatu svaru izmaiņai tīklā.

Kļūda (jeb kļūdas signāls) laika solī t tiek definēta kā starpība starp vēlamo atbildi un aktuālo atbildi:

$$e_j(t) = d_j(t) - y_j(t), \tag{4.1}$$

kur y_j – neirona j dotā atbilde uz ieejas signālu x_j; d_j – vēlamā atbilde uz ieejas signālu x_j.

Svaru korekcija neironā tiek veikta ar mērķi, lai neirona dotā atbilde y_j būtu tuvāk vēlamajai atbildei d_j. Tas tiek panākts, minimizējot izpildes vērtības funkciju (*cost function of performance*) E, kas tiek definēta, izmantojot aprēķināto kļūdas signālu:

$$E(t) = \frac{1}{2} e_j^2(t) \tag{4.2}$$

Vērtības funkcijas minimizācija veido apmācīšanās likumu, kuru sauc arī par **deltas likumu** (*delta rule*) vai **Vidrova-Hofa** (*Widrow-Hoff*) likumu. Atbilstoši deltas likumam, svara izmaiņa Δw_{ji} laika solī t tiek definēta kā:

$$\Delta w_{ji}(t) = \eta e_j(t) x_i(t), \tag{4.3}$$

kur η – pozitīva konstante, apmācīšanās parametrs (*learning-rate parameter*).

4.3.2. Heba apmācīšanās

Heba apmācīšanās postulāts (*Hebb's postulate*) ir vecākais un pazīstamākais no visiem apmācīšanās likumiem, kas nosaukts par godu neirofiziologam D. Hebam (*Hebb*).

Heba apmācīšanās (*Hebbian learning*) ideja īsi būtu noformulējama šādi:

- Ja divi neironi, kas atrodas vienas sinapses (sinaptiskā svara) abos galos, aktivizējas vienlaicīgi (sinhroni), tad šī sinapse tiek pastiprināta, bet, ja asinhroni, tad pavājināta vai likvidēta.

Vienkāršākā Heba metodes forma parādīta sekojošajā formulā:

$$\Delta w_{ji}(t) = \eta y_j(t) x_i(t), \tag{4.4}$$

kur η – pozitīva konstante, apmācīšanās parametrs (*learning-rate parameter*).

4.3.3. Apmācīšanās ar konkurenci

Apmācīšanās ar konkurenci (*competitive learning*), kā jau rāda nosaukums, nozīmē to, ka izejošie neironi savā starpā konkurē, kurš kļūs aktīvs (dos izejas signālu).

Nosacīti var teikt, ka blakusesošos neironus savieno sinapses ar **kavējošu iedarbību** (*lateral inhibition*). Apmācīšanās ar konkurenci kontekstā parādās jēdziens **neirons-uzvarētājs** (*winning neuron*), kas tiek noteikts pēc neirona aktivitātes pakāpes noteiktā laika solī.

Kad starp neironiem ir noskaidrots uzvarētājs, tad atbilstoši standarta apmācīšanās ar konkurenci likumam, svaru izmaiņa tiek veikta šādi:

$$\Delta w_{ji}(t) = \begin{cases} \eta(x_i - w_{ji}); & \text{A} \\ 0; & \text{B} \end{cases}, \tag{4.5}$$

kur η – pozitīva konstante, apmācīšanās parametrs (*learning-rate parameter*); A – ja neirons *j* ir uzvarētājs; B – ja neirons *j* nav uzvarētājs.

Nedaudz sarežģītāks ir variants, kad apmācās ne tikai uzvarētājs, bet arī mazākā pakāpē arī uzvarētāja neirona kaimiņi.

5. Vienslāņa perceptrons

Vienslāņa perceptrons (*single-layer perceptron, SLP*) vai, vienkārši, perceptrons ir viens no atpazīstamākajiem neironu tīklu modeļiem, kuru jau 20.g. 60. gadu sākumā piedāvāja Frenks Rozenblats (*Frank Rosenblatt*). Savā klasiskajā formā tas praktiski netiek izmantots, jo nespēj risināt nelineāras klasifikācijas problēmas, tomēr ļoti daudzi neironu tīklu modeļi ir veidojušies tieši uz perceptrona bāzes, pazīstamākais no kuriem ir vairākslāņu perceptrons ar kļūdu atgriezeniskās izplatīšanās apmācības metodi.

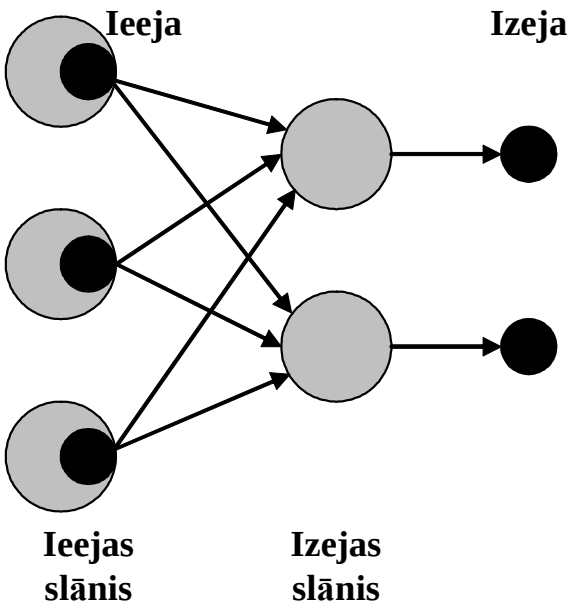
Perceptrona apmācības algoritms pārstāv t.s. uzraudzīto apmacīšanos (*supervised learning*) jeb apmācīšanos ar skolotāju (sk. nodaļu 4.2.1), kas ir visbiežāk pielietotā mašīnmācīšanās paradigma.

Perceptrons realizē paraugu klasifikāciju (atpazīšanu) vai, no matemātiskās puses skatoties, funkcijas modelēšanu (aproksimāciju). Daudzas reālās dzīves problēmas var tikt interpretētas kā **klasifikācijas** (*classification*) problēmas. Piemēram, bankas darbinieks kredītu daļā klasificē klientus vismaz divās grupās – tajos, kuriem var un tajos, kuriem nevar izsniegt kredītu. Bankas klientu klasifikācijas pamatā ir zināmais katra klienta riska līmenis, kuru var noteikt ienākumu līmenis, iepriekšējo kredītsaistību izpilde utt.

Balstoties uz vēsturiskajiem datiem par klientiem un par viņiem zināmo informāciju, var apmācīt neironu tīklu, kas spētu jauniem klientiem izrēķināt viņu uzticamības līmeni (modelētu uzticamības funkciju).

5.1. Perceptrona uzbūve

Vienslāņa perceptrons sastāv no viena vai vairākiem neironiem ar vienādu ieeju skaitu (Att. 5.1). Neironu skaits nosaka izeju skaitu visā neironu tīklā. Vienslāņa perceptronā visi neironi darbojas paralēli kā neatkarīgi klasifikatori.



Att. 5.1. Vienslāņa perceptrons ar 3 ieejām un 2 izejām

- Svari ir robežās [-*q*; +*q*], kur *q* ir jebkurš pozitīvs skaitlis, bet parasti ievietojas intervālā [-1; +1].
- Neironu izejošās vērtības ir intervālā [0; 1]. Tas attiecas arī uz ieejas slāni, tādējādi arī visa neironu tīkla ieejas vērtības ir padodamas šajā intervālā.
- Aktivitātes funkcija var būt gan lineāra, gan sliekšņveida, gan sigmoidāla.

5.2. Ar perceptronu risināmu problēmu piemēri

Problēmas parādītas ar tās reprezentējošo paraugu un tiem piesaistīto vēlamo atbilžu palīdzību. Perceptronam, kas apmācīts uz šādiem paraugiem, būtu jābūt spējīgam ne tikai apmācības laikā padotos paraugus, bet arī citus – līdzīgus.

5.2.1. Funkcija NOT (x_1 AND x_2)

Funkcijas aprakstu sk. arī Tab. 2.3.

Problēmu reprezentē 4 paraugi *P*(4), kam katram ir pievienota vēlamā atbilde *D*(4).

$$P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \quad D = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Vienosimies par tādu paraugu apzīmēšanas formu, kad paraugi (un attiecīgi arī to vēlamās atbildes) izvietotas virzienā no augšas uz leju, līdz ar to, piemēram, $P_1 = (0, 0)$, $D_1 = 1$.

Šī problēma ir ļoti vienkārša tādā nozīmē, ka apmācībai ir pieejami un aptverami pilnīgi visi iespējamie varianti, bet visu problēmu gadījumā situācija nav tik pateicība, un tieši tajos gadījumos neironu tīklu pielietošana būtu vispiemērotākā.

5.2.2. Ciparu atpazīšana

Problēmu reprezentē 20 paraugi $P(20)$, kas apzīmē 10 dažādus ciparus, katrs paraugs ir bitu karte ar izmēru 32 × 40, tātad ieeju skaits neironu tīklā = 32 × 40 = 1280.

(T (transponētā) – jo paraugi izvietoti horizontāli, nevis vertikāli.)

Vēlamā atbilde var tikt kodēta dažādos veidos, un ir atkarīga arī no izvēlētās neironu tīkla arhitektūras.

Pirmais variants ir neironu tīkls ar vienu neironu (1280 ieejas), tātad ar vienu izeju. Visas 10 vērtības (kas atbilst cipariem) jānokodē vienā skaitlī, turklāt ar ierobežojumiem: ja lietotā aktivitātes funkcija ir loģistikā, tad iespējamais intervāls ir [0, 1]. Šajā piemērā normalizēsim ciparu vērtības kā 0.2 + *c* * 0.05 (nekur nav teikts, ka būtu jādara tieši tā!), kur *c* ir attiecīgā cipara skaitliskā vērtība. Šeit iegūstam šādu vēlamo atbilžu virkni (tā kā cipari atkārtojas, daļa atbilžu ir izlaista):

$$D_1(20) = (0.2 \quad 0.25 \quad 0.3 \quad 0.35 \quad 0.4 \quad 0.45 \quad 0.5 \quad 0.55 \quad 0.6 \quad 0.65 \quad 0.2 \quad \ldots \quad 0.65)^T$$

Otrais variants ir neironu tīkls ar 4 neironiem, kur katram no viņiem izejā būs 0 vai 1 (bināra vērtība), līdz ar to katru ciparu kodēs 4 bināri skaitļi:

$$D_2(20,4) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \ldots & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \ldots & 1 \end{pmatrix}^T$$

5.2.3. Normalizācija un denormalizācija

Darbinot neironu tīklu, mēs parasti pieņemam, ka dati, kas tiek padoti neironu tīklam, ir sagatavoti vajadzīgajā formātā, un ka rezultāts, ko dos neironu tīkls, būs saprotams mums.

> **Normalizācija** ir reālās pasaules objekta pārveidošana neironu tīklam saprotamā formā.

Iepriekšējā piemērā ar ciparu atpazīšanu (sadaļa 5.2.2) normalizācija nozīmē konkrēta cipara attēla pārveidošanu par skaitļu (piemēram 0 un 1) virknīti. Bez tam normalizācija ir cipara vērtības pārveidošana par attiecīgu kodu. Piemēram, cipara 2 pārveidošana par 0.3 vienā gadījumā un (0 0 1 0) otrā gadījumā.

Tieši tāpat būtu jāveic pretējā darbība – interpretēt neirona sniegto rezultātu:

> **Denormalizācija** ir neironu tīkla izejas signāla pārveidošana ārējai programmai atbilstošā vai cilvēkam saprotamā formā .

Iepriekšējā piemērā ar ciparu atpazīšanu (otrajā variantā ar 4 izejām) mēs padodam cipara 6 bildi ieejā neironu tīklam. Neironu tīkls izdod rezultātu (0 1 1 0). Denormalizācija nozīmē interpretēt šo 0 un 1 virkni par ciparu 6.

5.3. Perceptrona darbināšana

Perceptrona **summēšanas funkcija** ir summa no ieejas signālu un attiecīgo svaru reizinājumiem:

$$NET_j = \sum_{i=1}^{m} w_{ji} x_i + b_j \tag{5.1}$$

Papildus svara b iesaistīšanos summēšanas funkcijā var pierakstīt arī kā w_0, papildus pieņemot, ka x_0=1:

$$NET_j = \sum_{i=0}^{m} w_{ji} x_i \tag{5.2}$$

Perceptrona darbināšanā var izmantot gan lineāru, gan sliekšņveida, gan sigmoidālu **aktivitātes funkciju**.

Lineārā aktivitātes funkcija:

$$y_j = \mathrm{f}_{\mathrm{lin}}(NET_j) = NET_j \tag{5.3}$$

Aprēķinu ērtībai lineāras aktivitātes funkcijas vietā var izmantot pseido-lineāro:

$$y_j = \mathrm{f}_{\mathrm{pseudo\_lin}}(NET_j) = \begin{cases} 1; & NET_j > 1 \\ 0; & NET_j < 0 \\ NET_j; & else \end{cases} \tag{5.4}$$

Sliekšņveida aktivitātes funkcija, kuru noteiktā abstrakcijas līmenī arī var iedomāties, kā lineāru funkciju:

$$y_j = \mathrm{f}_{\Theta}(NET_j) = \begin{cases} B; & NET_j > \Theta \\ A; & NET_j \le \Theta \end{cases}, \tag{5.5}$$

kur Θ – slieksnis (*threshold*).
Loģistiskā aktivitātes funkcija:

$$y_j = \mathrm{f}_{\log}(NET_j) = \frac{1}{1 + e^{-\frac{1}{g} NET}}, \tag{5.6}$$

kur g – līknes slīpuma koeficients (*gain*).

Alg. 5-1. Vienslāņa perceptrona darbināšana (*run_SLP*).

```
Function run_SLP (x) Returns y
Input configuration
    w(n,m) – neironu tīkla svari
    b(n) – neironu tīkla papildus svari (bias)
    x(m) – ieejas signāls
    m – ieejas signāla lielums (ieeju skaits)
    n – izejas signāla lielums, arī neironu skaits tīklā
Output configuration
    y(n) – izejas signāls
Using
    f_act – izvēlētā aktivitātes funkcija, (5.3), (5.5) vai (5.6)
Begin
    For j:=1 To n Do
        NET := b(j);
        For i:=1 To m Do
            NET := NET + w(j,i) * x(i)
        Endfor;
        y(j) := f_act (NET)
    Endfor
End
```

6. Vienslāņa perceptrona apmācīšanās

6.1. Kvadrātiskās kļūdas metode

6.1.1. Algoritma apraksts

Perceptrona apmācīšanās viena no formām ir kvadrātisko kļūdu minimizācijas metode, kur tiek izmantots tāds mehānisms kā gradients, kas saistīts ar aktivitātes funkcijas atvasinājumu. Tādējādi, pielietojot šo apmācības metodi, aktivitātes funkcijai ir obligāti jābūt atvasināmai, un starp trim sadaļā 5.3 apskatītajām aktivitātes funkcijām, sliekšņveida funkcija šim nolūkam diemžēl ir nederīga.

Lineārās aktivitātes funkcijas (5.3) (var pieņemt, ka arī (5.4)) atvasinājums:

$$\mathrm{f}'_{\mathrm{lin}} = \mathrm{f}'_{\mathrm{pseudo\_lin}} = 1 \tag{6.1}$$

Loģistiskās aktivitātes funkcijas (5.6) atvasinājums:

$$\mathrm{f}'_{\mathrm{log}} = \frac{1}{g}\mathrm{f}_{\mathrm{log}}(1-\mathrm{f}_{\mathrm{log}}) = \frac{1}{g}y(1-y) \tag{6.2}$$

kur g – līknes slīpuma koeficients (*gain*).

Apmācīšanās balstās uz kļūdas korekciju. Kļūdu izrēķina kā starpību starp vēlamo atbildi d un aktuālo atbildi y uz paraugu s:

$$e_j = d_j^{(s)} - y_j^{(s)} \tag{6.3}$$

Svara korekcija apmācīšanās ar kļūdu labošanu gadījumā ir proporcionāla kļūdai, kā arī ieejai un aktivitātes funkcijas atvasinājumam:

$$\Delta w_{ji} = \eta x_i e_j \mathrm{f}'_{\mathrm{act}}, \tag{6.4}$$

kur η – apmācīšanās parametrs.

Jaunā svara vērtība izsakāma šādi:

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t+1), \tag{6.5}$$

kur t – apmācīšanās procesa solis.

Dažreiz svaru modificēšanā tiek izmantota t.s. **inerces heiristika**, kad svara izmaiņu daļēji ietekmē iepriekšējā solī izdarītā svara izmaiņa:

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t+1) + \alpha \Delta w_{ji}(t), \tag{6.6}$$

kur α – inerces (*momentum*) faktors.

Svara korekcija (6.4) lineārai aktivitātes funkcijai ir precizējama šādi, jo lineārās funkcijas atvasinājums ir 1:

$$\Delta w_{ji} = \eta x_i e_j \tag{6.7}$$

Savukārt svara korekcija (6.4) sigmoidālai aktivitātes funkcijai ir precizējama šādi (sk. loģistiskās funkcijas atvasinājumu (6.2)):

$$\Delta w_{ji} = \frac{\eta}{g} x_i e_j y_j (1 - y_j) \tag{6.8}$$

Pseidokodā Alg. 6-1 dots vienslāņa perceptrona apmācīšanās process. Tagad un turpmāk algoritmi tiks doti pēc vienlaidus (*continuous*) metodikas (Alg. 4-1).

Kritērijs apmācīšanās procesa beigšanai ir neironu tīkla kvadrātiskās kļūdas samazināšanās zem noteikta līmeņa. Kvadrātisko kļūdu epohai aprēķina šādi:

$$E = \frac{1}{2} \sum_{s=1}^{r} \sum_{j=1}^{n} e_{sj}^2 , \tag{6.9}$$

kur r – kopējais paraugu skaits, n – neironu skaits, ½ – koeficients bez saturīgas, bet tikai ar tehnisku nozīmi.

Apmācīšanās process notiek tik ilgi, kamēr kļūda ir pietiekoši liela un pārsniedz noteiktu maksimālo kļūdas līmeni ε un kamēr nav izsmelts maksimālais pieļaujamo epohu skaits:

$$E > \varepsilon \wedge epoch \leq max\_epochs \tag{6.10}$$

Tādējādi ir iespējami divi varianti:

- Apmācības process ir beidzies, pateicoties mazai kvadrātiskajai kļūdai ($E \leq \varepsilon$). Šajā gadījumā tiek uzskatīts, ka neirons ir veiksmīgi apmācījies.
- Apmācības process ir beidzies, jo tika izsmelts maksimāli pieļaujamais epohu skaits (*epoch* > *max_epochs*). Šajā gadījumā neironu tīkls nav apmācījies.

Ņemot vērā to, ka apmācīšanās process katru reizi sākas no cita svaru komplekta (svari tiek inicializēti gadījumu veidā), dažādi apmācības procesi pie vienādiem paraugiem un vienādas neironu tīkla arhitektūras teorētiski var novest pie dažādiem rezultātiem (dažādi apmācītiem neironu tīkliem), vienkāršākajā gadījumā – apmācīšanās noritējusi veiksmīgi vai neveiksmīgi.

Neironu tīkla neapmācīšanās katrā atsevišķā gadījumā nav uzskatāma par lielu neveiksmi, jo gradientu metodes (kas ir pamatā šai apmācībai) viens no trūkumiem ir potenciāla iespēja iekļūt t.s. lokālajā minimumā.

Iemesli, kādēļ neironu tīkls varētu būt apmācījies neveiksmīgi:

- Pārāk mazs pieļaujamo epohu skaits[5];
- Neveiksmīgi izvēlēts apmācīšanās parametrs η[6];
- Pārāk maza iepriekš noteiktā maksimālā pieļaujamā kļūda ε (it sevišķi komplektā ar pārāk lielu apmācīšanas parametru η);
- Problēma ir nelineāra[7];

[5] Atsevišķiem uzdevumiem var būt nepieciešami pat tūkstoši epohu.

[6] Parasti par lielu. Tipiskākais lielums varētu būt ar kārtu 0.01. Tai pat laikā dažās specifiskās konfigurācijās apmācīšanās ir iespējama ar daudz lielāku apmācīšanās parametru, un tādēļ notikt ļoti ātri.

[7] Vienslāņa perceptrons spēj atrisināt tikai lineāras problēmas, pat ja aktivitātes funkcija ir nelineāra.

- Neveiksmīgi izvēlēts līknes slīpuma koeficients g (loģistiskajai aktivitātes funkcijai)[8];
- Vienkārši neveiksmīga sākotnējo svaru konfigurācija (tiek parasti tiek uzstādīti kā gadījuma vērtības), tādēļ apmācības procesa rezultātā iestājies lokālais minimums.

Alg. 6-1. Vienslāņa perceptrona apmācīšanās (*train_SLP*).

Procedure *train_SLP*
Input configuration
neapmācīts neironu tīkls
w(n,m) – neironu tīkla svari
p(r) – apmācības paraugi
d(r) – apmācības paraugu attiecīgās vēlamās atbildes
r – apmācības paraugu skaits
m – parauga lielums, arī neironu tīkla ieeju skaits
n – neironu skaits tīklā
η – apmācīšanās parametrs
max_epochs – maksimālais epohu skaits
ε – maksimālā pieļaujamā epohas kļūda
Output configuration
apmācīts neironu tīkls
Using
f_act – izvēlētā aktivitātes funkcija
f_act' – izvēlētās aktivitātes funkcijas atvasinājums
run_SLP – neironu tīkla darbināšana (Alg. 5-1)
Begin
Inicializē neironu tīkla svarus;
epoch := 1;
Do
E := 0; {epohas kļūda}
For *s*:=1 **To** *r* **Do** {pa visiem paraugiem}
{darbina neironu tīklu ar doto paraugu:}
x := *p(s)*;
y := ***run_SLP*** (*x*);
{apmācīšanās algoritms:}
For *j*:=1 **To** *n* **Do** (pa visiem neironiem}
{kļūdas aprēķins:}
e(j) : = *d(j)* - *y(j)*;
E : = *E* + 0.5**e*^{2}*(j)*;
{svaru korekcija:}
For *i*:=0 **To** *m* **Do**
w(j,i) := *w(j,i)* + η**x(i)***e(j)** ***f_act'*** (*j*)
Endfor
Endfor
Endfor;
epoch := *epoch* + 1
While *epoch* $\leq$ *max_epochs AND E* > ε
End

[8] Problēmas var rasties, ja aktivitātes funkcijas slīpums ir „nepareizajā vietā". Koeficientam *g* būtu jābūt tādam, lai pie tipiskām NET vērtībām nestātos situācija, kad pie lielām *NET* izmaiņām būtu tikai mazas *y* izmaiņas vai otrādi – *NET* un *y* vērtībām tipiskajā definīcijas apgabalā jābūt līdzsvarotām.

Komentāri pie Alg. 6-1:

- Nav precizēta aktivitātes funkcija;
- Nav iekļauta inerces heiristika;
- Neironā papildus svars (*bias*) tiek kodēts kā 0-tais svars;
- Sākotnējā svaru inicializācija nav precizēta, tomēr parasti tā notiek uzstādot tiem gadījuma vērtības no intervāla [-*q*, *q*], kur $0 \le q \le 1$, piemēram, [-0.3, 0.3].

6.1.2. Dinamiska apmācīšanās parametra izmantošana

Apmācīšanās parametrs η iepriekš parādītajās apmācības formulās ir konstante, tomēr tiek pielietotas heiristikas, kad tas apmācības procesa gaitā mainās – laika gaitā pakāpeniski samazinoties. Šeit piedāvātās apmācīšanās parametra izmaiņas funkcijas ņemtas no [Haykin, 1999].

Viena no vienkāršākajām apmācības koeficienta izmaiņas funkcijām ir t.s. stohastiskās aproksimācijas grafiks (*stochastic approximation schedule*):

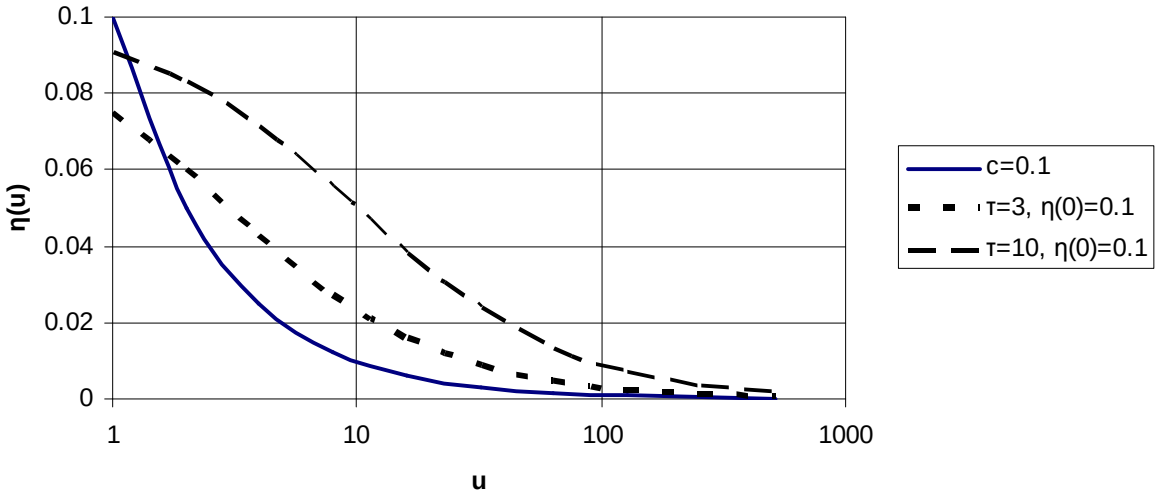
$$\eta(u) = \frac{c}{u}, \tag{6.11}$$

kur *c* – brīvi izvēlēta pozitīva konstante, bet *u* – iterācijas (epohas) numurs 1, 2, 3...

Otrs ir "Vispirms atrast, tad precizēt" grafiks (*search-then-converge*):

$$\eta(u) = \frac{\eta_0}{1 + \frac{u}{\tau}}, \tag{6.12}$$

kur η_0 – parametra sākuma vērtība; τ – izmaiņas ātrums.



Att. 6.1. Apmācīšanās parametra η izmaiņas grafiku piemēri pie dažādiem koeficientiem: η_0, τ, c (pēc formulām (6.11), (6.12)).

6.1.3. Apmācīšanās piemērs

Problēma: Funkcija NOT (x_1 AND x_2) (sk. 5.2.1)

Neironu tīkla arhitektūra: 1 neirons ar 2 ieejām (svari w_1, w_2 un b).

Darbināšana tiek veikta ar sliekšņveida funkciju (kas nav gluži korekti, bet tā tiek lietota kā lineāras funkcijas abstrakcija konkrētajam piemēram):

$$NET = x_1 w_1 + x_2 w_2 + b$$

$$y = \begin{cases} 1; & NET > 0.5 \\ 0; & NET \le 0.5 \end{cases}$$

Apmācīšanās algoritms veidots, pieņemot, ka aktivitātes funkcija ir lineāra (y' = 1). Apmācīšanās parametrs η=1.0, kas nav tipiska tā vērtība. Tiek pieņemts, ka $x_0 = 1$ un $w_0 = b$.

$$e(t) = d(t) - y(t)$$

$$w_i(t+1) = w_i(t) + x_i(t)e(t)$$

$$E = \frac{1}{2}\sum_t e^2(t)$$ (kļūda pa visiem soļiem t epohas ietvaros)

Tab. 6.1. Funkcijas NOT (x_1 AND x_2) apmācīšanās ar mazāko kvadrātu metodi.

epoha	t	x_1	x_2	w_1	w_2	b	NET	y	d	e	E
1	1	0	0	0	0	0	0	0	1	1	
	2	0	1	0	0	1	1	1	1	0	
	3	1	0	0	0	1	1	1	1	0	
	4	1	1	0	0	1	1	1	0	-1	0.707
2	5	0	0	-1	-1	0	0	0	1	1	

	6	0	1	-1	-1	1	0	0	1	1	
	7	1	0	-1	0	2	1	1	1	0	
	8	1	1	-1	0	2	1	1	0	-1	0.866
3	9	0	0	-2	-1	1	1	1	1	0	
	10	0	1	-2	-1	1	0	0	1	1	
	11	1	0	-2	0	2	0	0	1	1	
	12	1	1	-1	0	3	2	1	0	-1	0.866
4	13	0	0	-2	-1	2	2	1	1	0	
	14	0	1	-2	-1	2	1	1	1	0	
	15	1	0	-2	-1	2	0	0	1	1	
	16	1	1	-1	-1	3	1	1	0	-1	0.707
5	17	0	0	-2	-2	2	2	1	1	0	
	18	0	1	-2	-2	2	0	0	1	1	
	19	1	0	-2	-1	3	1	1	1	0	
	20	1	1	-2	-1	3	0	0	0	0	0.500
6	21	0	0	-2	-1	3	3	1	1	0	
	22	0	1	-2	-1	3	2	1	1	0	
	23	1	0	-2	-1	3	1	1	1	0	
	24	1	1	-2	-1	3	0	0	0	0	0.000

Apmācības process beidzies veiksmīgi pēc epohas #6, jo epohas kvadrātiskā kļūda samazinājās līdz 0. Parasti tik vienkārši un skaisti nekad nav – kļūda samazinās, bet tomēr paliek zināmā līmenī.

6.1.4. Teorētiskā bāze

Aprakstītā metode ar kļūdu labošana balstās uz gradienta metodes pielietojumu kļūdas funkcijai.

Par skalāra lauka *u* **gradientu** fiksētā punktā M_0 sauc vektoru, kas nosaka virzienu, kādā skalārā lieluma vērtība palielinās visātrāk. Gradienta koordinātas aprēķina, izmantojot funkcijas *u* parciālos atvasinājumus punktā M_0.

Ja $u = u(x;y;z)$, tad *u* gradients ir:

$$\nabla u = \left[\frac{\partial u}{\partial x}; \frac{\partial u}{\partial y}; \frac{\partial u}{\partial z} \right]$$

Skalāra lauka gradients jebkurā apgabala punktā ir perpendikulārs līmeņlīnijas/līmeņvirsmas pieskarei (resp., iet pa normāli).

Pielietojot gradientu metodi perceptrona apmācībā:

- Par skalārā lauka funkciju ņemsim kļūdas funkciju (neironam *j*):

$$E_j = \frac{1}{2} e_j^2 \tag{6.13}$$

  kur *e* – kļūda, kas dotajam paraugam nosaka vēlamās un aktuālās neironu tīkla rekacijas starpību (6.3), bet par argumentiem ņemot neironu tīkla svarus *W*.

- Tā kā mums interesē virziens, kurā skalārā lieluma (kļūdas funkcijas) vērtība samazinās, nevis palielinās, gradientu jāņem ar mīnus zīmi

Tādējādi neirona *j* svaru izmaiņu (fiksētam apmācības piemēram) mēs varam noformulēt šādi:

$$\Delta W_j = -\nabla E_j(W_j) = -\left[\frac{\partial E_j}{\partial w_{j1}}; \frac{\partial E_j}{\partial w_{j2}}; ...; \frac{E_j}{\partial w_{jm}}\right]$$

Viena svara izmaiņa izskatās šādi:

$\Delta w_{ji} \equiv -\dfrac{\partial E_j}{\partial w_{ji}}$ jeb

$$\Delta w_{ji} = -\eta \frac{\partial E_j}{\partial w_{ji}} \tag{6.14}$$

Aprēķinu vienkāršošanai izteiksim kļūdas funkcijas parciālo atvasinājumu izvērstā formā:

$$\frac{\partial E_j}{\partial w_{ji}} = \frac{\partial E_j}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial NET_j}\frac{\partial NET_j}{\partial w_{ji}} \tag{6.15}$$

Atbilstoši formulām (5.2), (6.3), (6.13), bet neprecizējot formulu konkrētai aktivitātes funkcijai izteiksmes atsevišķās komponentes var izteikt šādi:

$$\frac{\partial E_j}{\partial e_j} = \frac{\partial(\frac{1}{2}e_j^2)}{\partial e_j} = e$$

$$\frac{\partial e_j}{\partial y_j} = \frac{\partial(d_j - y_j)}{\partial y_j} = -1$$

$$\frac{\partial y_j}{\partial NET_j} = \frac{\partial(\mathrm{f}_{\mathrm{act}}(NET_j))}{\partial NET_j} = \mathrm{f}'_{\mathrm{act}} = y'$$

$$\frac{\partial NET_j}{\partial w_{ji}} = \frac{\partial(\sum_k w_{jk}x_k)}{\partial w_{ji}} = \frac{\partial(w_{ji}x_i)}{\partial w_{ji}} = x_i$$

Tādējādi kļūdas funkcijas parciālais atvasinājums (6.15) ir izsakāms šādi:

$$\frac{\partial E_j}{\partial w_{ji}} = -x_i e y', \tag{6.16}$$

bet svara korekcija (6.14) šādi (sk. formulu (6.4)):

$$\Delta w_{ji} = -\eta \frac{\partial E_j}{\partial w_{ji}} = \eta x_i e y' \tag{6.17}$$

6.2. Perceptrona apmācīšanās ar pseido-apgrieztās matricas metodi

6.2.1. Algoritma apraksts

Doti *s* paraugi garumā *m*. Tādējādi paraugu kopums *P* un vēlamo atbilžu kopums *D* ir pierakstāmi kā matricas šādi:

$$P = \begin{pmatrix} p_{11} & p_{12} & \dots & p_{1m} \\ p_{21} & p_{22} & \dots & p_{2m} \\ \dots & \dots & \dots & \dots \\ p_{s1} & p_{s2} & \dots & p_{sm} \end{pmatrix} \quad D = \begin{pmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \dots & \dots & \dots & \dots \\ d_{s1} & d_{s2} & \dots & d_{sn} \end{pmatrix}$$

Tiks apmācīts neironu tīkls no *n* neironiem, kur katrā ir *m* svari. Neironu tīkla svaru kopums pierakstāms kā:

$$W = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{pmatrix}$$

Matricu notācijā pseido-apgrieztās matricas metodi (*pseudo-inverse*) svaru *W* izrēķināšanai ir ļoti viegli pierakstīt:

$$W = P^{+} D \,, \tag{6.18}$$

kur ($^{+}$) – pseido-apgriešanas (*pseudo-inverse*) operators:

$$X^{+} = (X^{T} X)^{-1} X^{T} \tag{6.19}$$

Protams, ietilpīgākā darbība šeit ir matricas apgriešana – ($^{-1}$).

6.2.2. Apmācīšanās piemērs

Apmācīšanās piemēru ņemsim tādu pašu kā 6.1.3 (Funkcija NOT (x_1 AND x_2)).

Pseido-apgrieztās matricas metodē papildus svars un tam atbilstošā ieeja ar vērtību 1 parādīsies tiešā veidā, līdz ar to apmācības paraugu matricā parādās viena lieka kolonna no vieniniekiem (salīdzināt ar 5.2.1):

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad D = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Pēc formulas (6.19) aprēķini veicami šādi (matricas apgriešana šajā materiālā netiks detalizēta):

$$P^T P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix}$$

$$\left(P^T P\right)^{-1} = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \frac{3}{4} & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}$$

$$P^+ = \left(P^T P\right)^{-1} P^T = \begin{pmatrix} \frac{3}{4} & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$$W = P^+ D = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$

Tab. 6.2. Iegūtā neironu tīkla pārbaude.

x_0	x_1	x_2	w_0	w_1	w_2	$NET = \sum_i x_i w_i$	$y = \begin{cases} 1; & NET > 0.5 \\ 0; & NET \le 0.5 \end{cases}$	d
1	0	0	0.25	0.5	0.5	0.25	1	1
1	0	1				0.75	1	1
1	1	0				0.75	1	1
1	1	1				1.25	0	0

7. Vairākslāņu perceptrons

Vairākslāņu perceptrons (*multi-layer perceptron*, *MLP*) ir vienslāņa perceptrona paplašinājums, kurā bez izejas slāņa ir arī vēl vismaz viens "īstais" slānis. Vairākslāņu perceptrona domājamais pielietojums, neatšķiras no vienslāņa perceptrona domājamā, atšķirība ir tikai spējā izpildīt uzstādīto uzdevumu – vairākslāņu perceptrons ir no skaitļošanas viedokļa daudz spēcīgāks, jo spēj risināt nelineāras klasifikācijas problēmas. Vairākslāņu perceptrons bija zināms jau 20. gs. 60.-jos, gados, tikai tajā laikā un vēl ilgi pēc tam nebija atklāts tā apmācības algoritms. Par "Backpropagation" algoritma atklājējiem uzskata Rummelhartu, Hintonu un Viljamsu (*Rummelhart*, *Hinton*, *Williams*), kas par to paziņoja 1986. gadā.

XOR (izslēdzošā VAI) problēma ir populārākā no triviālajām nelineārajām problēmām, tātad, kuru nespēj risināt vienslāņa perceptrons, bet vairākslāņu perceptrons jau spēj.

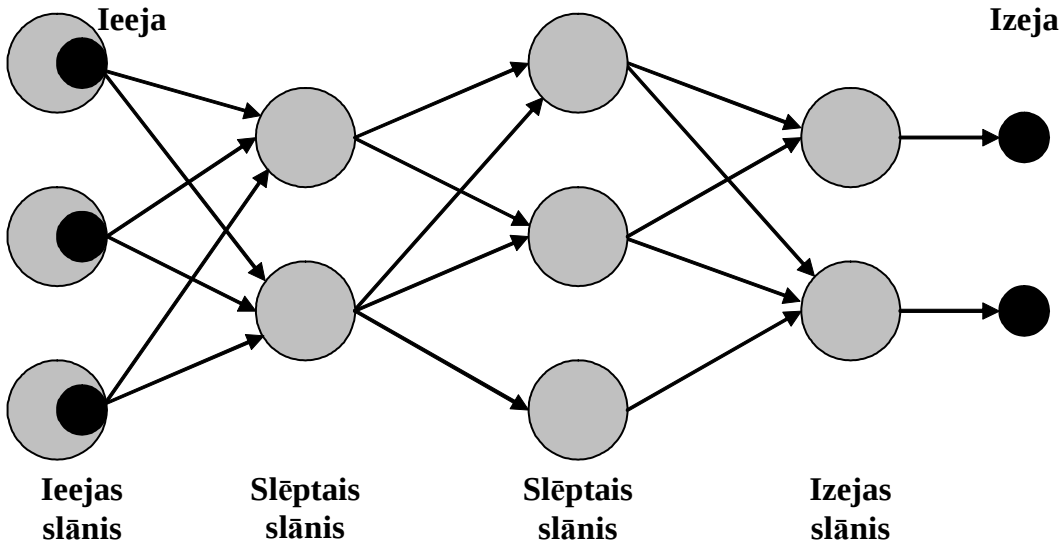
Tab. 7.1. Funkcija „Izslēdzošais VAI" (x_1 XOR x_2) – vienkāršākā nelineārā problēma.

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

7.1. Vairākslāņu perceptrona uzbūve

Vairākslāņu perceptronam (atšķirībā no vienslāņa perceptrona) ir vismaz viens slēptais slānis.

- Visi slāņi ievietoti virknē un arī tiek darbināti viens aiz otra, katrs no viņiem – tāpat kā vienslāņa perceptrons, tikai slāņa n izejas kalpo par ieejām slānī n+1.
- Katrā slānī aktivitātes funkcijas tips var atšķirties (parasti gan tā nav, un visur lieto sigmoidālo funkciju), tomēr, lai ar vairākslāņu perceptrona palīdzību varētu risināt nelineāras problēmas (cita iemesla to lietot vienslāņa perceptrona vietā nemaz nav!), vismaz slēptajā slānī (slēptajos slāņos) aktivitātes funkcijai jābūt nelineārai.
- Blakus slāņu neironi ir savienoti katrs ar katru, tāpēc neironu skaits slānī n atbilst svaru skaitam slāņa n+1 neironos.
- Teorētiski, lai atrisinātu (modelētu) jebkuru (nelineāru) problēmu, pietiek ar vienu nelineāru slēpto slāni (un tā parasti arī dara), tomēr atsevišķos gadījumos kā heiristiku mēdz pielietot arī vairāku slēpto slāņu arhitektūru.



Att. 7.1. Vairākslāņu perceptrons ar diviem slēptajiem slāņiem

7.2. Vairākslāņu perceptrona darbināšana

Vairākslāņu perceptrona darbināšana notiek, izmantojot tās pašas formulas, kuras izmanto vienslāņa perceptrons (sk. sadaļu 5.3), vienīgi tas tiek izvērsts pa vairākiem slāņiem (Alg. 7-1).

Alg. 7-1. Vairākslāņu perceptrona darbināšana (*run_MLP*).

```
Function run_MLP (x) Returns y
Input configuration
    layers(lcount) – neironu tīkla slāņi (neskaitot ieejas slāni)
    x() – ieejas signāls
    w() – neironu tīkla svari
    b() – neironu tīkla papildus svari
Output configuration
    y() – izejas signāls
Begin
    For k:=1 To count(layers) Do  {pa visiem slāņiem secībā no sākuma uz beigām}
        For j:=1 To n Do
            NET := b(k,j);
            For i:=1 To m Do
                NET := NET + w(k,j,i) * x(i)
            Endfor;
            y(j) := f_act (NET)
        Endfor;
        x := y  {kārtējā slāņa izeja kļūst par nākošā slāņa ieeju}
    Endfor
End
```

7.3. Vairākslāņu perceptrona apmācīšanās ar kļūdu atgriezeniskās izplatīšanās metodi

Kļūdu atgriezeniskās izplatīšanās metode (*error backpropagation*) ir populārākā vairākslāņu perceptrona apmācīšanās metode. Jau nosaukums rāda, ka „kaut kas" šeit notiek pretējā virzienā:

- Apmācīšanās notiek, sākot ar izejas slāni, un tad atpakaļ līdz pirmajam – pretējā virzienā nekā notika darbināšana.
- Apmācīšanās atšķirīgi notiek izejas slānim, jo tam ir pieejama vēlamā atbilde, pārējiem slāņiem nav, tā vietā tiek veikta kļūdu signāla apkopošana nākošajā slānī un izmantošana vēlamās atbildes vietā.

Ņemot vērā to, ka apmācīšanās notiek atšķirīgi izejas slānī un pārējos slāņos, pieraksta ērtības labad tiek ieviests jēdziens **lokālais gradients** δ, kas uzskatāms par tādu kā kļūdas funkcijas vispārinājumu.

Izejas slāņa apmācīšanās notiek tāpat kā vienslāņa perceptronam:

$$\Delta w_{ji} = \eta x_i \, \mathrm{f}'_{\mathrm{act}} \, e_j \tag{7.1}$$

Tomēr, ieviešot lokālā gradienta jēdzienu, tas būtu pārrakstāms šādi:

$$\Delta w_{ji} = \eta x_i \delta_j \,, \tag{7.2}$$

kur δ_j – lokālais gradients neironam j, kuru **izejas slānim** aprēķina šādi:

$$\delta_j^{(out)} = \mathrm{f}'_{\mathrm{act}} \, e_j \tag{7.3}$$

Slēptajiem slāņiem lokālā gradienta aprēķināšana ir sarežģītāka, jo tā izmanto nākošajā slānī (kas apmācīts pirms kārtējā!) aprēķinātais tā neironu lokālais gradients:

$$\delta_j^{(hidden)} = \mathrm{f}'_{\mathrm{act}} \sum_{k=1}^{n} \delta_k w_{kj} \,, \tag{7.4}$$

kur n – neironu skaits nākošajā slānī; δ_j – nākošā slāņa k-tā neirona lokālais gradients; w_{kj} – nākošā slāņa k-tā neirona j-tais svars.

Tā kā parasti par aktivitātes funkciju tiek izmantota loģistiskā funkcija, tad lokālā gradienta aprēķins būtu precizējams šādi:

$$\delta_j^{(out)} = \frac{1}{g} y_j (1 - y_j) e_j \tag{7.5}$$

$$\delta_j^{(hidden)} = \frac{1}{g} y_j (1 - y_j) \sum_{k=1}^{n} \delta_k w_{kj} \,, \tag{7.6}$$

Vairākslāņu perceptrona viena slāņa apmācīšana notiek tāpat kā vienslāņa perceptronam, ar to atšķirību, ka izejas slānis tiek apmācīts citādāk nekā slēptie (Alg. 7-2).

Vairākslāņu perceptrona apmācīšanās ar kļūdu atgriezeniskās izplatīšanās metodi notiek pa slāņiem pretējā virzienā – sākot ar izejas slāni un atpakaļ.

Alg. 7-2. Vairākslāņu perceptrona apmācīšanās (*train_MLP*).

Procedure *train_MLP*
Input configuration
neapmācīts neironu tīkls
layers(lcount) – neironu tīkla slāņi (neskaitot ieejas slāni)
n() – neironu skaits slānī
m() – svaru skaits neironā
w() – neironu tīkla svari
p() – apmācības paraugi
d() – apmācības paraugu attiecīgās vēlamās atbildes
max_epochs – maksimālais epohu skaits
ε – maksimālā pieļaujamā epohas kļūda
η – apmācīšanās parametrs
Output configuration
apmācīts neironu tīkls
Using
f_act – izvēlētā aktivitātes funkcija
f_act' – izvēlētās aktivitātes funkcijas atvasinājums
run_MLP – vairākslāņu percpetrona darbināšana (Alg. 7-1)
Begin
Inicializē neironu tīkla svarus;
epoch := 1;
Do
E := 0; {epohas kļūda}
For *s*:=1 **To** *r* **Do** {pa visiem paraugiem}
{darbina neironu tīklu ar doto paraugu:}
x := *p(s)*;
y := ***run_MLP*** (*x*);
{pa visiem slāņiem, sākot ar pēdējo, un uz sākumu:}
For *h*:=*layer_count* **Downto** 1 **Do**
For *j*:=1 **To** *n(h)* **Do** (pa visiem neironiem kārtējā slānī}
{lokālā gradienta aprēķins:}
If *h*=*layer_count* {ja izejas slānis}
e(h,j) := *d(j)* - *y(j)*;
$E := E + 0.5*e^2(h,j)$;
neuronerror(h,j) : = *e(h,j)*
Else {ja slēptais slānis}
{izrēķina kļūdu no nākošā slāņa neironiem:}
neuronerror(h,j) := 0;
For *k*:=1 **To** *n(h*+1*)* **Do**
neuronerror(h,j) := *neuronerror(h,j)* +
w(h+1,*k,j)***δ(h*+1,*k)*
Endfor
Endif;
δ(h,j) : = ***f_act'*** (*h,j*)**neuronerror(h,j)*;
{svaru korekcija:}
For *i*:=0 **To** *m(h,j)* **Do**
w(h,j,i) := *w(h,j,i)* + η**x(h,i)***δ(h,j)*
Endfor
Endfor
Endfor

```
            Endfor;
            epoch := epoch + 1
        While epoch ≤ max_epochs AND E > ε
    End
```

Apmācība ar kļūdu atgriezeniskās izplatīšanās metodi (*error backpropagation*) uzskatāma par standarta algoritmu daudzslāņu perceptrona apmācīšanā, kas tiek uzskatīts par etalonu, pret kuru salīdzina citus algoritmus. Algoritms savu nosaukumu ieguvis no tā, ka kļūdas funkcijas parciālie atvasinājumi tiek noteikti, veicot izejas neironu kļūdas signāla izplatīšanu apgrieztā virzienā no slāņa uz slāni (*back-propagating the error signals*).

Kļūdu atgriezeniskās izplatīšanās metodes paātrināšana.

Šeit ir uzskaitītas dažas heiristikas, kas var tikt izmantotas apmācības algoritma konverģences paātrināšanā (pēc [Haykin, 1999]).

1. Katram maināmam neironu tīkla parametram (svaram) būtu jābūt savam individuālam apmācības koeficientam.
2. Katram apmācības koeficientam būtu jānodrošina iespēja tikt izmainītam apmācības procesa laikā.
3. Ja potenciālā svara izmaiņa kārtējā solī ir ar to pašu (algebrisko) zīmi kā iepriekšējā solī vai vairākos iepriekšējos soļos, tad apmācības koeficients dotajam svaram jāpalielina.
4. Ja potenciālā svara izmaiņa kārtējā solī ir ar pretēju (algebrisko) zīmi nekā iepriekšējā solī vai vairākos iepriekšējos soļos, tad apmācības koeficients dotajam svaram jāsamazina.
5. Neironu tīkla slāņos, kas ir vairāk uz sākuma galu, apmācības koeficientam jābūt lielākam.

8. Kohonena tīkls

Kohonena tīkls (*Kohonen network*) pieder t.s. pašorganizējošo neironu tīklu modeļiem. Cits tā nosaukums ir Pašorganizējošā karte (*self-organizing map, SOM*). Šo modeli 1982. gadā piedāvāja somu profesors Teivo Kohonens (*Teuvo Kohonen*). Kohonena tīklu apmāca ar nolūku, lai tas varētu veikt paraugu klāsterizāciju.

Kohonena algoritms pārstāv t.s. neuzraudzīto apmācīšanos (*unspervised learning*, sk. nodaļu 4.2.2), tā realizācijai izmantojot apmācīšanās ar konkurenci (*competitive learning*) formu.

No pirmā acu uzmetiena klasiskais Kohonena algoritms varētu likties triviāls un „bez lielām cerībām uz nopietnu problēmu risināšanu", bet ja arī tā liekas, tad priekšstats ir maldīgs, jo Kohonena algoritms ir efektīvs līdzeklis ar nopietnu matemātisko bāzi.

8.1. Paraugu klāsterizācija un Kohonena tīkls

Paraugu **klāsterizācija** (*clustering*) no klasifikācijas atšķiras ar to, ka iepriekš nav zināmas grupas, kurās paraugi būtu jāsaklasificē. Paraugu sadalījums pa grupām tiek noteikts pašā klāsterizācijas procesā, zināms ir tikai grupu skaits, kurās. Taču nav teikts, ka visās grupās būtu jāieklasificē kaut cik līdzīgs skaits paraugu – sadalījums pa grupām var būt ļoti nevienmērīgs, vairākas grupas atstājot vispār bez paraugiem. Vienā grupā atrodošies paraugi ir savā ziņā līdzīgi viens otram, respektīvi tuvu viens otram pēc noteikta attāluma mēra. Par attāluma mēru starp paraugiem bieži lieto Eiklīda attālumu vai tā kvadrātu, kas ir paraugu atbilstošo elementu starpību kvadrātu summa.

Klāsterizāciju parasti nevar tik tiešā veidā pielietot dažādu problēmu risināšanā kā klasifikāciju. Parasti ir nepieciešams papildus darbs – vairākkārtīga eksperimentēšana (dažādu sadalījumu klāsteros veidošana), datu analīze. Ja informācijas daudzums ir pietiekams, ērtāk ir izmantot klasifikāciju. Tomēr ne vienmēr pirms klasifikācijas veikšanas ir zināms, kādās tieši grupās būtu jāklasificē, un klāsterizācija varētu būt viens no veidiem, kā šādas grupas definēt.

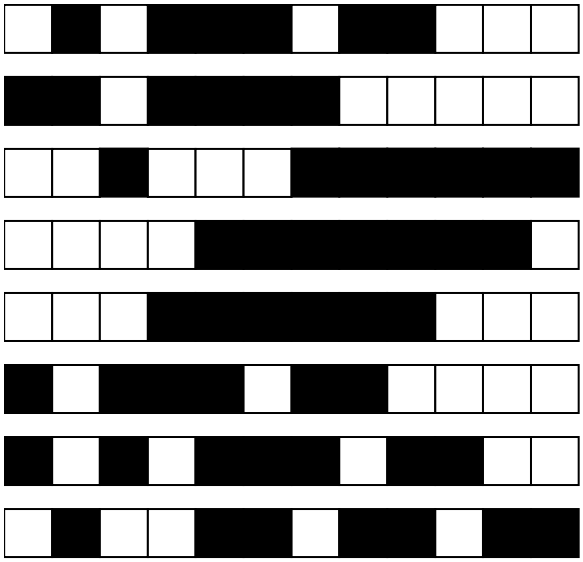
Viens no piemēriem klāsterizācijas veikšanai ir aizdomīgo bankas darījumu identificēšana. Problēma, kādēļ šeit ne vienmēr noderēs klasifikācija kā uzdevuma risināšana metode, ir tāda, ka ne vienmēr iespējams atlasīt apmācīšanas vajadzībām nepieciešamos pozitīvos un negatīvos piemēros. Tas, kas ir zināms, ir vien informācijas par dažiem darījumiem kā noziedzīgiem. Ja noteikts darījumu kopums (ieskaitot kādus, par kuriem ir skaidri zināms, ka tie ir „sliktie") tiek noteiktā veidā sadalīts klāsteros un izrādās, ka kādā no klāsteriem tie ir sastopami pietiekoši lielā daudzumā vai proporcijā, būtu īpaši jāpievērš uzmanība arī pārējiem darījumiem šajā klāsterī.

Vēl viens interesants izmantošanas piemērs klāsterizācijai ir attēlu saspiešana. Šeit neironu tīklam (vai citai apmācošai sistēmai) tiek padoti attēla pikseļi, kas tiek sagrupēti pēc līdzības. Pēc tam katras grupas pikseļus attēlā aizstāj ar vidusmēra pikseli no šīs grupas – ieguvums ir tāds, ka dažādu pikseļu krāsu pēc pārveidošanas ir tik, cik klāsteros tikusi veikta grupēšana, līdz ar to viena pikseļa kodēšanai nepieciešams mazāk atmiņas. Daudzos attēlos pikseļi ir viendabīgi, un šāda veida saspiešana neradīs lielu kvalitātes pasliktinājumu. Piemēram, ja mēs sagrupējam pikseļus 32 grupās un attiecīgi veicam pārveidošanu, tad viena pikseļa kodēšanai pietiek ar 5 bitiem (ierasto 24 bitu vietā krāsainiem attēliem).

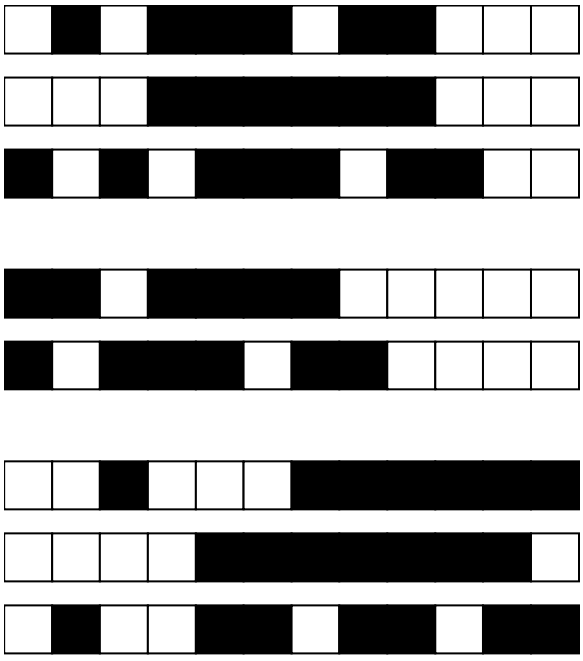
Plašs klāsterizācijas metožu pielietojums ir bioinformātikā, kur tiek analizēts gan cilvēku, gan citu dzīvu būtņu genoms, konkrēti DNS struktūra.

8.2. Ar Kohonena tīklu risināmas problēmas piemērs

Problēmu reprezentē 8 paraugi $P(8)$, kas apzīmē 8 dažādas bitu virknes garumā 12.



Gadījums #1. Klāsterizācija 3 klasēs – C1, C2, C3 (vizuāli: melnie laukumi koncentrējušies attiecīgi vidū, pa kreisi vai pa labi). Jāatceras, ka šis nav vienīgais pareizais variants, kā būtu jāgrupē dotie paraugi – tas var būt atkarīgs gan no objektīviem faktoriem, ka definētā topoloģija, gan arī pie fiksētas topoloģijas vairākkārt laižot algoritmu, var iegūt dažādus rezultātus (jo arī Kohonena tīklam, tāpat kā perceptronam, sākotnējā svaru uzstādīšana notiek gadījumu veidā, un sākotnējā svaru konfigurācija ietekmē dalīšanu klāsteros).

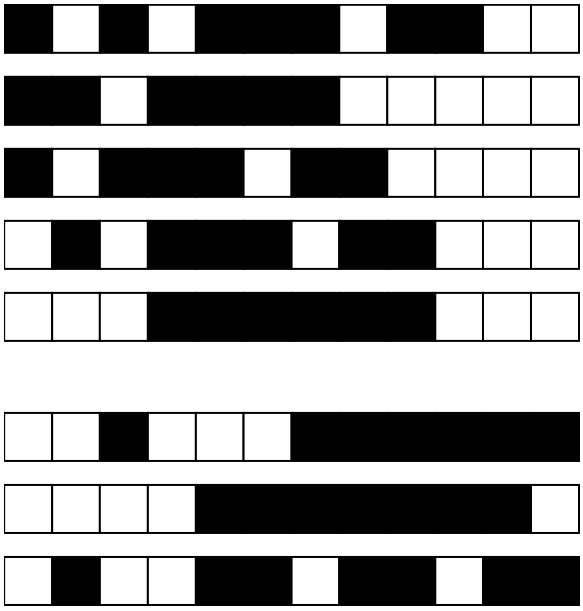


C1

C2

C3

Gadījums #2. Klāsterizācija 2 klasēs – C1, C2 (vizuāli: melnie laukumi koncentrējušies attiecīgi pa kreisi vai pa labi).



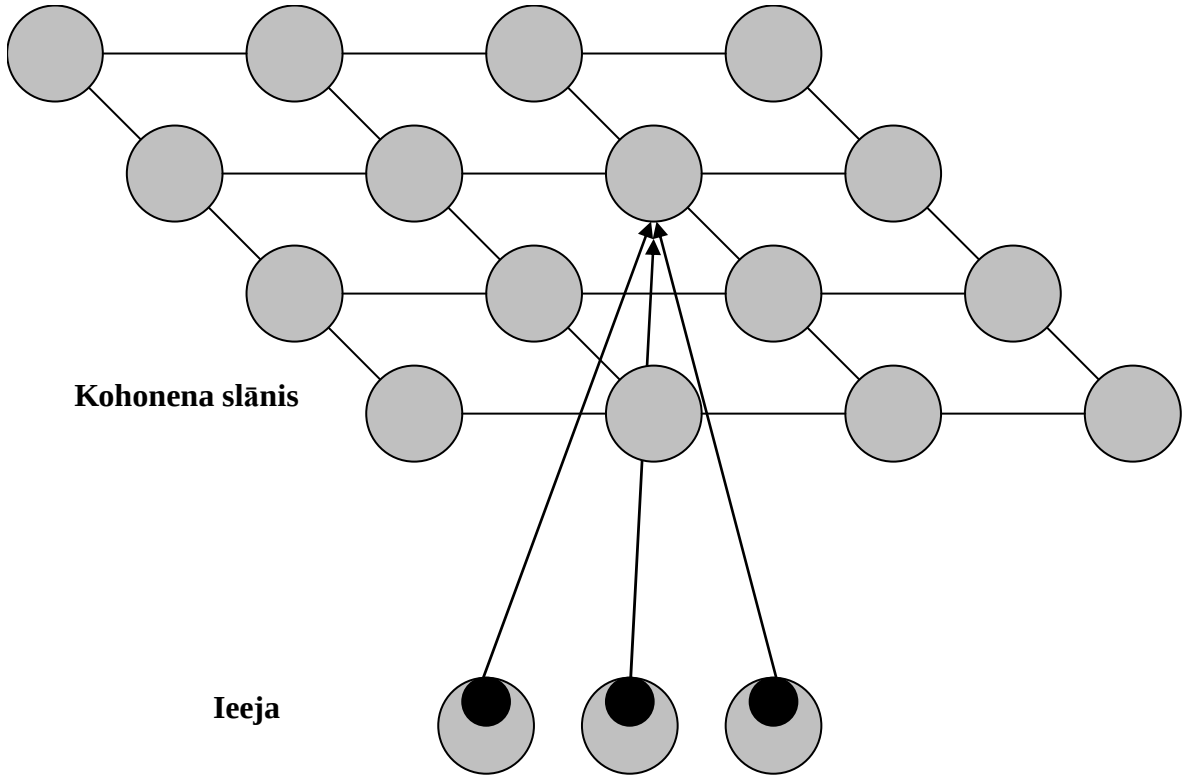
Abi gadījumi parāda klāsterizācijas problēmas risināšanas būtību, tomēr ir pārāk vienkārši un tendenciozi, lai ļautu pilnībā spriest par klāsterizācijas procesa raksturu. Svarīgākais, ko vajadzētu vēlreiz akcentēt, ir ka klāsterizācijas procesa paraugi netiek sagrupēti vienādās vai līdzīgās daļās – ļoti tipisks gadījums ir tāds, ka vairākām klasēm netiek pieklasificēts neviens paraugs, un ne vienmēr šāds gadījums būtu uztverams par neveiksmīgu.

Konkrēts dalījums klasēs klasterācijas procesa rezultātā būtu uztverams nevis kā likums vai noteikums (kā to dažreiz varētu uztvert perceptrona un vispār uzraudzītās apmācīšanās gadījumā), bet gan kā priekšlikums vai ideja, ka „arī šādā veidā paraugi ir klasificējami". Šāda nenoteiktība no vienas puses it kā padara klāsterizācijas izmantošanu sarežģītāku nekā tā ir klasifikācijas gadījumā, tomēr te jautājums drīzāk ir nevis par neērtāku metodi, bet par to, ka ir problēmas, kuru risināšanai ir derīga arī šāda pieeja.

8.3. Kohonena tīkla uzbūves un darbības principi

Līdzīgi kā vienslāņa percpetronam, arī Kohonena tīklam ir viens vienīgs (īstais) slānis – Kohonena slānis. Būtiskākā atšķirība ir tāda, ka Kohonena slāni ir būtisks neironu savstarpējais izvietojums jeb topoloģija, jo apmācīšanās algoritms ir atkarīgs no tā, cik tālu ir divi neironi viens no otra.

Ja percpetrona gadījumā tiek doti apmācības paraugi, lai uz tiem perceptrons tiktu apmācīts, bet pēc tam parasti darbotos ar citiem (sk. vispārināšanas īpašība), tad Kohonena tīkla uzdevums parasti ir klāsterizēt pašus apmācības paraugus (neizslēdzot gan iespēju to pēc tam darīt ar citiem).



Att. 8.1. Kohonena tīkls ar 3 ieejām un 16 (4 × 4) neironiem Kohonena slānī. Saites parādītas tikai uz vienu no 16 Kohonena slāņa neironiem. Neironu izvietojums Kohonena slānī – 2-dimensiju, kvadrātisks.

Dažas Kohonena tīkla īpašības:

- Neironu svari standarta variantā ir robežās [0; +1];
- Kohonena tīkla sniegtais rezultāts uz ieejas paraugu izpaužas kā neirona numurs, kuram ieejas paraugs visvairāk atbilst, kas reprezentā klāsteri, pie kā dotais paraugs ir ticis pievienots;
- Atšķirībā no perceptrona – katram paraugam nav dota vēlamā vērtība;
- Pirms apmacīšanas izvēloties neironu skaitu Kohonena slānī, automātiski tiek noteikts, cik klāsteros tiks mēģināts grupēt ieejas paraugus.

Kohonena slāņa topoloģija var būt ne tikai **taisnstūrveida** vai **kvadrātiska** – Kohonena algoritms ne mazākā mērā to nenosaka. Diezgan populāra ir arī **heksagonālā** topoloģija (t.i., sešstūrainā „medus šūnu" topoloģija).

Kaut arī topoloģijas, kuras lieto Kohonena slānī, tiek nosauktas pēc vizuālā izskata – no Kohonena algoritma viedokļa **definēt topoloģiju nozīmē definēt attālumu** $d(\cdot,\cdot)$ starp jebkuriem diviem neironiem slānī. Piemēram, svarīgāk par „smuko bildīti" ir zināt, vai kvadrātiskā topoloģijā tiks uzskatīts, ka starp diviem neironiem viena kvadrāta diagonālēs attālums būs 2 vai tikai kvadrātsakne no 2 (resp., Eiklīda attālums) – tas nav Kohonena algoritma kompetencē un jādefinē Kohonena tīkla konstruētājam.

8.4. Kohonena tīkla darbināšana

Apmācīts Kohonena tīkls piekārto tam padoto paraugu vienai no *n* klasēm, kur *n* ir neironu skaits Kohonena tīklā.

Ideja ir pavisam vienkārša – ieejas paraugs noteiktā veidā (piemēram, pēc Eiklīda mēra) tiek salīdzināts ar katra neirona svariem, un tas neirons, kurš ir vistuvāk, tiek pasludināts par uzvarētāju, un tā reprezentējošajai klasei paraugs arī tiek piekārtots.

Summēšanas (izplatīšanās) **funkcijas** mērķis ir atrast starpību starp ievades paraugu un neirona svariem. Vispopulārākās metodes šim nolūkam ir Eiklīda attālums (8.1) vai tā kvadrāts (8.2), bet var izmantot arī citus mērus.

$$NET_j = \left\| \mathbf{x} - \mathbf{w}_j \right\| = \sqrt{\sum_{i=1}^{m} \left(x_i - w_{ji} \right)^2} \tag{8.1}$$

$$NET_j = \left\| \mathbf{x} - \mathbf{w}_j \right\|^2 = \sum_{i=1}^{m} \left(x_i - w_{ji} \right)^2 \tag{8.2}$$

Atšķirībā no perceptrona, rēķinot aktivizācijas funkciju, ir svarīga ne tikai paša neirona summēšanas vērtība, bet arī visu pārējo neironu vērtības – tikai neirons-uzvarētājs izejā dod nenulles vērtību.

Neirons-uzvarētājs ir tas, kurš bijis vistuvāk ieejas paraugam, un tāds var būt tikai viens. Formula (8.3) parāda, kā tiek iegūts neirona-uzvarētāja numurs.

$$j_{win} = \text{index} \left(\min \left(\mathbf{NET} \right) \right) \tag{8.3}$$

kur **NET** – visa Kohonena slāņa summēšanas vērtību kopums.
Neirons-uzvarētājs izejā dabū 1, visi pārējie neironi 0:

$$y_j = \begin{cases} 1; & j = j_{win} \\ 0; & j \neq j_{win} \end{cases} \tag{8.4}$$

kur j_{win} – uzvarētāja neirona kārtas numurs.
Tehniski Kohonena tīkla darbināšanas rezultāts uz konkrētu paraugu ir ciparu virkne, kur ir tieši viens viennieks, bet pārējās nulles, bet saturīgi – viennieka pozīcija norāda klasi, kam pieklasificēts ieejas paraugs.

Kohonena tīkla darbināšanas algoritms vienam paraugam parādīts Alg. 8-1.

Alg. 8-1. Kohonena tīkla darbināšana (*run_Kohonen*).

Function *run_Kohonen* (*x*) **Returns** *y*
Input configuration
    w(n,m) – neironu tīkla svari
    x(m) – ieejas signāls
Output configuration
    y(n) – izejas signāls
Using
    ***sqr*()** – kvadrāts
    ***sqrt*()** – kvadrātsakne
Begin

```
            {Eiklīda attāluma izrēķināšana:}
        For j:=1 To n Do
            sum := 0;
            For i:=1 To m Do
                sum := sum+ sqr(x(i) - w(j,i))
            Endfor;
            NET(j) := sqrt(sum);
            y(j) := 0
        Endfor;
            {uzvarētāja neirona noskaidrošana:}
        minvalue := NET(1);
        minindex := 1;
        For j:=2 To n Do
            If NET(j) < minvalue Then
                minvalue:= NET(j);
                minindex := j
            Endif
        Endfor
        y(minindex) := 1; {neironam uzvarētājam izejā 1, pārējiem 0}
End
```

8.5. Kohonena tīkla apmācīšanās

8.5.1. Apmācīšanās algoritms

Kohonena apmācības triviālā gadījuma pamatprincips ir uzvarētāja neirona pietuvināšanās ieejas paraugam (formula (8.5)). Netriviālajā gadījumā ieejas paraugam attiecīgā mērā pietuvojas arī uzvarētāja neirona kaimiņi.

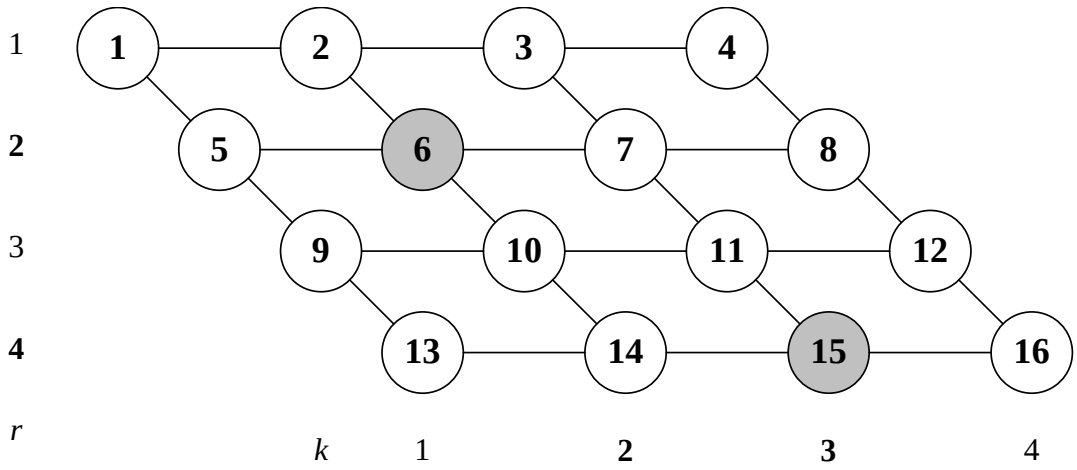
$$\Delta w_{ji} = \eta\left(x_i - w_{ji}\right)\mathrm{D}\left(\mathrm{d}\left(j,\, j_{win}\right)\right) \tag{8.5}$$

kur $\mathrm{d}(\cdot,\cdot)$ – Kohonena slāņa topoloģijas definētais divu neironu attālums; $\mathrm{D}(\cdot)$ – kaimiņa funkcija, kas, ņemot vērā dotā neirona topoloģisko attālumu līdz uzvarētājam neironam, nosaka svaru maiņas pakāpi.

Kvadrātiskas (arī taisnstūrveida) topoloģijas gadījumā vienkāršākais attāluma mērs starp neironiem ir Eiklīda attālums (formula (8.6), Att. 8.2).

$$\mathrm{d}\left(j_1,\, j_2\right) = \sqrt{\left(j_1^{(k)} - j_2^{(k)}\right)^2 + \left(j_1^{(r)} - j_2^{(r)}\right)^2} \tag{8.6}$$

kur $j^{(k)}$ – neirona j kolonnas numurs; $j^{(r)}$ – neirona j rindas numurs.



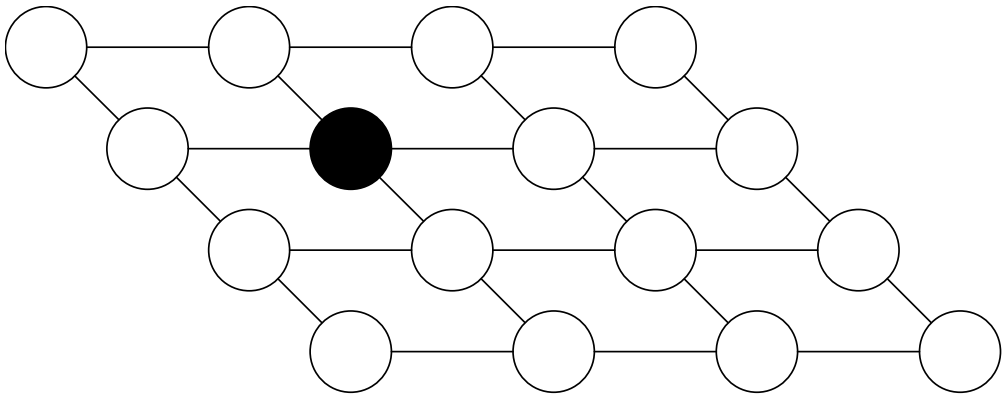
$$\mathrm{d}(6,15) = \sqrt{(3-2)^2 + (4-2)^2} = \sqrt{5} \approx 2.236$$

Att. 8.2. Attālumi starp neironiem kvadrātiskas topoloģijas gadījumā.

Triviālākais kaimiņa funkcijas variants ir, ka uzvarētājs neirons apmācās pilnā mērā, bet pārējie neironi vispār nemaz (formula (8.7), Att. 8.3).

$$\mathrm{D}_0(z) = \begin{cases} 1; & z = 0 \\ 0; & z \neq 0 \end{cases} \tag{8.7}$$

kur z – attālums starp uzvarētāju neironu un doto neironu, kuru nosaka $\mathrm{d}(\cdot,\cdot)$ (tikai uzvarētājam tas ir 0).

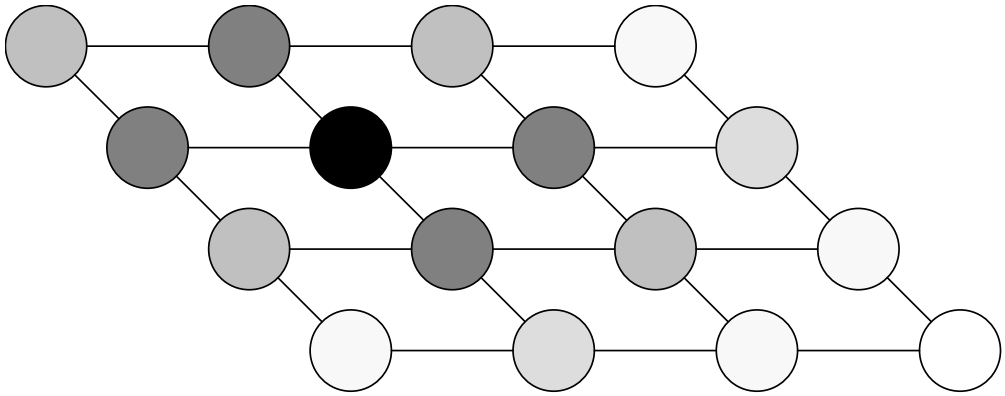


Att. 8.3. Triviālās kaimiņa funkcijas ietekme uz apmācīšanās intensitāti.

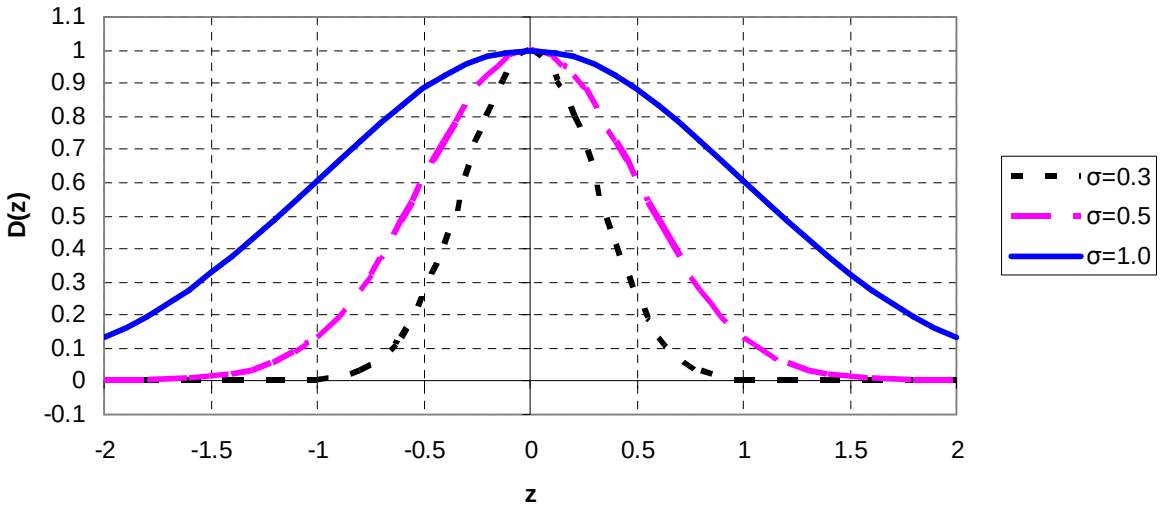
Gausa funkcija kaimiņa funkcijā dod iespēju apmācīties arī citiem neironiem, un šāda veidā rodas pamats iegūt netriviālus rezultātus.

$$\mathrm{D}_{Gauss}(z) = e^{-\frac{z^2}{2\sigma^2}} \tag{8.8}$$

kur σ – līknes slīpuma faktors (kurš var tikt mainīt arī apmācības procesa laikā, sk. zemāk).



Att. 8.4. Gausa kaimiņa funkcijas ietekme uz apmācīšanās intensitāti.



Att. 8.5. Gausa kaimiņa funkcijas grafiks.

Alg. 8-2. Kohonena tīkla apmācīšanās (*train_Kohonen*).

Procedure *train_Kohonen*
Input configuration
neapmācīts neironu tīkls
w(n,m) – neironu tīkla svari
p(r) – apmācības paraugi
η – apmācīšanās parametrs
max_epochs – maksimālais epohu skaits
Output configuration
apmācīts neironu tīkls
Using
run_Kohonen – neironu tīkla darbināšana (Alg. 8-1)
d(·,·) – attālums starp diviem neironiem, ko nosaka slāņa topoloģija (piemēram, (8.6))
D(·) – kaimiņa funkcija ((8.7) vai (8.8))
Begin
Inicializē neironu tīkla svarus;
epoch := 1;

```
        Do
            For s:=1 To r Do  {pa visiem paraugiem}
                {darbina neironu tīklu ar doto paraugu:}
                x := p(s);
                y := run_Kohonen (x);
                {uzvarētājs neirona numurs – numurs neironam, kam izejā 1:}
                j_winner := index (y(j); where y(j) = 1);
                {apmācīšanās algoritms:}
                For j:=1 To n Do  (pa visiem neironiem}
                    {svaru korekcija:}
                    For i:=0 To m Do
                        w(j,i) := w(j,i) + η * (x(i) - w(j,i)) * D (d (j, j_winner))
                    Endfor
                Endfor
            Endfor;
            epoch := epoch + 1
        While epoch ≤ max_epochs
    End
```

Komentārs pie Alg. 8-2:

- Sākotnējā svaru inicializācija nav precizēta, tomēr parasti tā notiek uzstādot tiem gadījuma vērtības no intervāla [0, 1].

8.5.2. Apmācīšanās procesa parametru kontrole

Kohonena tīkla apmācības procesu varētu iedalīt 2 fāzēs [Haykin, 1999]:

1. Pašorganizēšanās (kārtošanas) fāze;
2. Konverģences fāze.

Pašorganizēšanās fāzes laikā notiek topoloģiskā sakārtošana un neironu svaru vektori nonāk aptuveni vajadzīgajās vietās. Pašorganizēšanās fāze varētu prasīt apmēram 1000 vai vairāk iterāciju (epohu).

Konverģences fāze nodrošina t.s. īpašību kartes (*feature map*) noprecizēšanu, lai nodrošinātu ieejas paraugu telpas pareizu statistisko kvantizāciju. Konverģences fāze ilgst vismaz 500n iterāciju, kur n ir neironu skaits tīklā.

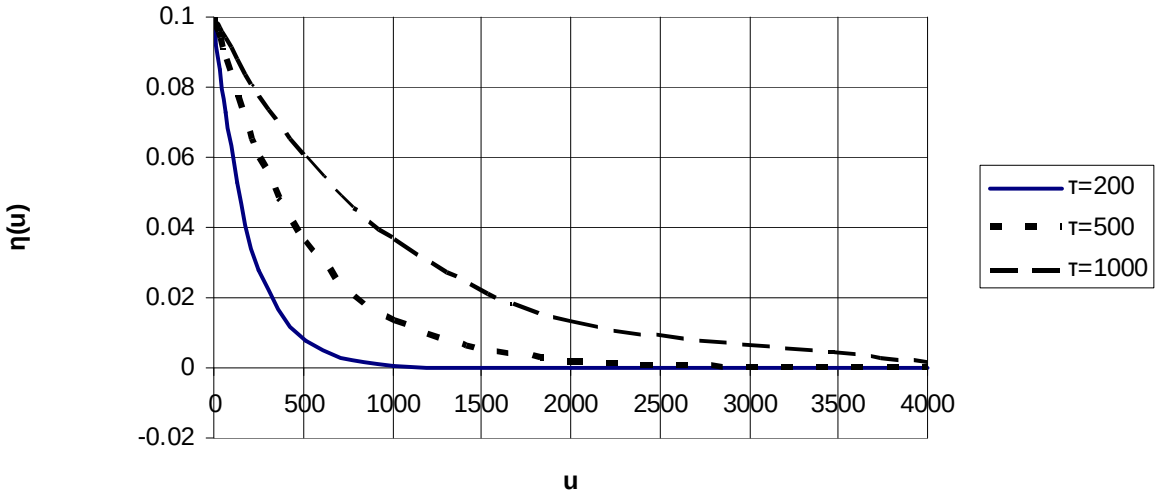
Kohonena tīkla apmācīšanās procesā piedalās gan apmācīšanās parametrs η, gan līknes slīpuma koeficients σ (ja tiek lietota Gausa kaimiņu funkcija). Apmācīšanās procesa laikā abus parametrus var dinamiski samazināt.

Apmācīšanās parametra izmaiņa apmācīšanās laikā.

Apmācīšanās parametra η izmaiņu laikā var veikt pēc formulas (8.9)

$$\eta(u) = \eta_0 e^{\left(-\frac{u}{\tau_\eta}\right)} \tag{8.9}$$

kur η_0 – parametra sākuma vērtība; τ_η – izmaiņas ātrums; u – epohas numurs.



Att. 8.6. Apmācīšanās parametra izmaiņas grafiks atbilstoši (8.9) pie η_0 = 0.1.

Tipiska sākuma koeficientu izvēle formulai (8.9) ir η_0 = 0.1, τ=1000.

Līknes slīpuma parametra izmaiņa apmācīšanās laikā.

Līknes slīpuma parametra σ izmaiņu laikā notiek tāpat kā apmācīšanās parametram

$$\sigma(u) = \sigma_0 e^{\left(-\frac{u}{\tau_\sigma}\right)} \tag{8.10}$$

kur σ_0 – parametra sākuma vērtība; τ_σ – izmaiņas ātrums; u – epohas numurs.

Atšķirīgi tiek noteiktas koeficientu vērtības:

$\sigma_0 = \frac{\sqrt{n}-1}{2}$, kur n ir neironu skaits slānī;

$\tau_\sigma = \frac{1000}{\log(\sigma_0)}$.

Izvēloties šādu līknes slīpuma izmaiņu, apmācības procesa beigās kaimiņa funkcijai būtu jāsatur tikai daži paši tuvākie kaimiņi vai pat neviens kaimiņš.