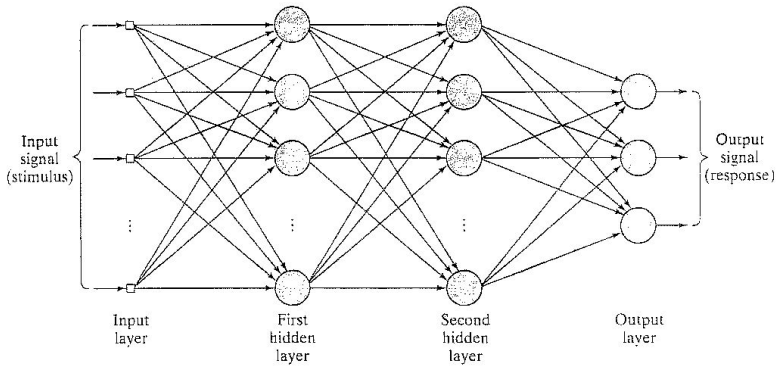
Mitmekihiline Tajur (Multilayer Perceptron)

Leopold Parts & Konstantin Tretjakov

27. märts 2003. a.

1 Sissejuhatus

Selles peatükis tuleb tegemist närvivõrkude olulise klassiga — mitmekihiliste närvivõrkudega. Reeglina koosneb närvivõrk kolmest osast. Esimeses kihis, mida kutsutakse sisendkihiks, asuvad sensorid (ehk sisendneuronid). Edasi tuleb üks või mitu peidetud neuronite kihti, mis täidavad arvutusülesandeid ning lõpuks üks kiht väljundneuroneid (mis samuti arvutusülesandeid täidavad). Sisendsignaal läbib närvivõrku kiht-kihilt alates sisendkihtidest kuni väljundini ühes suunas. Selliseid närvivõrke nimetatakse tavaliselt mitmekihilisteks tajuriteks (üldistamaks meile juba tuttavat ühekihilist tajurit).



Mitmekihilisi tajureid on edukalt kasutatud mitmete erinevate raskete ülesannete lahendamisel, treenides teda juhendamise all ning kasutades populaarset *vea tagastamise meetodit*. See algoritm põhineb kiirema languse meetodil. Seega võib öelda, et ta on ühtlasi sama populaarse kohaneva filtreerimise algoritmi üldistus: vähimruutude meetodi algoritm (LMS) moodustab erijuhu ühe lineaarse neuroni korral.

Vea tagastamisega õppimine koosneb kahest närvivõrku läbivast osast — edaspidisest ja äraspidisest. Edaspidisel võrgu läbimisel rakendatakse võrgu sisendneuronitele mingit sisendvektorit, mille mõju kandub läbi võrgu kiht-kihilt edasi. Väljundikiht moodustab väljundite hulga, mis on närvivõrgu tegelik vastus antud sisendile. Selle läbimise korral on sünaptilised kaalud fikseeritud. Äraspidisel võrgu läbimisel seevastu reguleeritakse sünaptilisi kaale vastavalt veaparandusreeglile. Leitakse väljundsignaali ja soovitud väljundi vaheline erinevus ning see kantakse veasignaalina läbi võrgu, sünaptiliste seostega vastupidises suunas. Sealjuures muudetakse sünaptilisi kaalusid, et vastust soovitud tulemusele lähemale nihutada.

Signaale, mis närvivõrku läbivad, on seega kahte sorti — funktsionaalsed signaalid, mis algavad sisendist ning kantakse läbi võrgu edasi väljundisse, ning veasignaalid,

mis saadakse oodatud tulemuse võrdlemisel saadud tulemusega ning kantakse alates väljundist tagasi igasse neuronisse.

Kordame üle ka fakti, et iga neuron peab suutma

- arvutada sisendsignaali(de) ja sünaptiliste kaalude põhjal mittelineaarse väljundsignaali
- arvutada mingi täpsusega gradientvektorit, mis määrab veaparanduse

Mitmekihilise tajuri kolm põhilist omadust on:

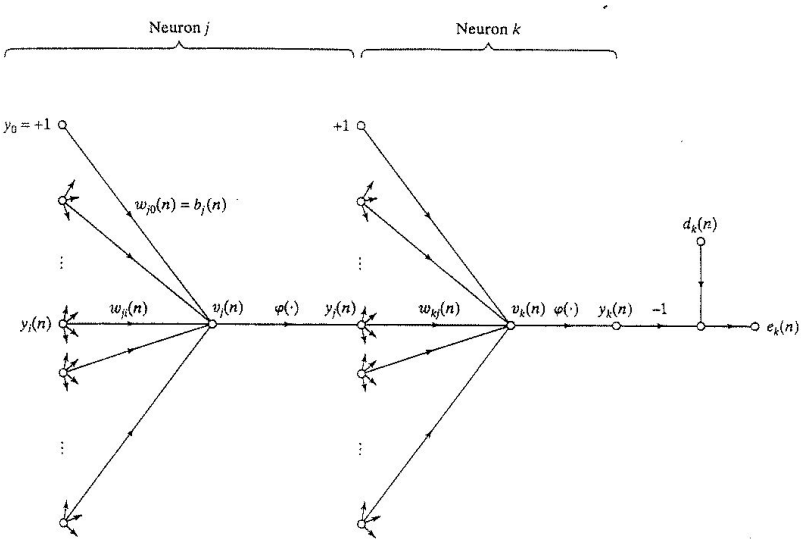
1. Igal võrgu neuronil on mittelineaarne väljundfunktsioon *(activation function)*. Tähtis on siinjuures tõik, et hoolimata mittelineaarsusest on see funktsioon kõikjal diferentseeruv. Kõige rohkem kasutatakse praktikas logistilist funktsiooni $y = \frac{1}{1+e^{-x}}$ või hüperboolset tangensit $y = \tanh(x)$. Mittelineaarsus on oluline seepärast, et vastasel korral oleks võrgu sisendi ja väljundi vaheline suhe lineaarne ning arvutatav vaid ühe neuronite kihi abil. Samas on eelpooltoodud logistiline funktsioon saadud loodusest jälgides tõeliste neuronite tegevust.

2. Võrgus on vähemalt üks kiht peidetud neuroneid, mis pole ei sisendkihi ega ka väljundkihi osad. Need võimaldavad närvivõrgul raskemaid ülesandeid lahendada, eraldades treeningu käigus sisendist üha oulisemaid andmeid.

3. Võrk on väga sidus - sünapseid (seoseid) on palju.

Tänu nende omaduste kooslusele ning võimele õppida kogemustest evib mitmekihiline tajur suurt arvutusvõimsust. Needsamad tegurid on aga kahjuks ka põhjused, miks meie praegused teadmised sellistest võrkudest on piiratud. Tänu mittelineaarsusele ja sidususele on teoreetilist analüüsi raske läbi viia. Peidetud kihtide kasutamine teeb õppimisprotsessi visualiseerimise raskemaks. Mõnes mõttes peab õppimisprotsess otsustama, milliseid mustri osasid peaksid peidetud neuronid esindama. Seega peab nende valikuks olema suur erinevate funktsioonide ruum ning võrgu treenimisel tuleb tähelepanu pöörata ka sisendi formaadile.

Vea tagastamise algoritm on tähtis veel seepärast, et tema abiga on nüüd olemas arvutuslikult tõhus viis mitmekihiliste võrkude treenimiseks. Kuigi ei ole võimalik väita, et ta annab kõikide lahendatavate ülesannete korral optimaalse lahendi, on ta siiski suutnud hajutada Minsky ja Paperti poolt algatatud pessimismi.



2 Vea tagastamise algoritm (Back-propagation algorithm)

2.1 Õppimisreegel

Nagu juba öeldud, vea tagastamise algoritm on iteratiivne algoritm, mis minimiseerib nn. veaenergia funktsiooni *(total error energy)*:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n)$$

kus C on väljundneuronite hulk. Selleks kasutame kiirema languse meetodit. Siin ja edaspidi tähistame nagu tavaliselt:

- $y_j(n) = \phi_j(v_j)$ — neuroni j väljundsignaal treeningnäite n esitamisel (ehk algoritmi n-dal iteratsioonisammul). [1]
- $d_j(n)$ — oodatud õige väärtus neuronis j, ehk see väljund mida me tahame saada.
- $e_j(n) = d_j(n) - y_j(n)$ — neuroni j veasignaal.
- $\phi_j(\cdot)$ — neuroni j väljundfunktsioon.
- $v_j(n) = \sum_i w_{ji}(n)y_i(n)$ — neuroni j kohalik väli *(induced local field)*, ehk kõikide sisendite kaalutud summa.
- $w_{ji}(n)$ — sünaptiline kaal neuroni i väljundi ja neuroni j sisendi vahel. $w_{j0}(n) = b_j(n)$ on neuroni j nihe (ning vastavaks neuroni sisendiks on alati $+1$).
- $\Delta w_{ji}(n) = w_{ji}(n+1) - w_{ji}(n)$ — parandus tehtud vastavale kaalule iteratsiooni sammul n.
- η — õpitegur.

Vastavalt kiirema languse meetodi põhimõttele, muudetakse võrgu kaalud igal sammul veafunktsiooni gradiendi vastassuunas, ehk $\Delta w_{ji} = -\eta\frac{\partial \mathcal{E}}{\partial w_{ji}}$:

$$\Delta w_{ji} = -\eta\frac{\partial \mathcal{E}}{\partial v_j}\frac{\partial v_j}{\partial w_{ji}} = -\eta\frac{\partial \mathcal{E}}{\partial v_j}y_j = \eta\delta_j y_j \tag{1}$$

Kus $\delta_j = -\frac{\partial \mathcal{E}}{\partial v_j}$ on neuroni "lokaalne gradient" *(local gradient)*. [2] Arvutame δ_j eraldi juhul, kui j on väljundneuron ja kui ta on peidetud neuron.

1. Väljundneuroni korral:

$$\delta_j = -\frac{\partial \mathcal{E}}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial v_j} = -(e_j)(-1)(\phi_j'(v_j)) = \phi_j'(v_j)e_j \tag{2}$$

[1] Indeks n siin kõikjal viitab ainult sellele, et tegu on väärtustega iteratsioonisammul n. Suurt kasu sellest ei ole, seega me lubame endale vabadust jätta neid mõnikord kirjutamata

[2] Võrrelge valemit (1) LMS õppimisreegliga adaptiivse lineaarse neuroni jaoks

2. Peidetud kihis L_s oleva neuroni korral:

$$\delta_j = -\frac{\partial \mathcal{E}}{\partial v_j} = -\sum_{k \in L_{s+1}} \frac{\partial \mathcal{E}}{\partial v_k}\frac{\partial v_k}{\partial v_j} = \sum_{k \in L_{s+1}} \delta_k \frac{\partial v_k}{\partial v_j}$$

(kus summa võetakse üle kõigi järgmises kihis olevate neuronite)

$$\frac{\partial v_k}{\partial v_j} = \frac{\partial}{\partial v_j}\left(\sum_{i \in L_s} w_{ki} y_i\right) = \sum_{i \in L_s} w_{ki}\frac{\partial y_i}{\partial v_j} = w_{kj}\frac{\partial \phi_j(v_j)}{\partial v_j} = w_{kj}\phi_j'(v_j)$$

seega

$$\delta_j = \sum_{k \in L_{s+1}} \delta_k w_{kj}\phi_j'(v_j) = \phi_j'(v_j)\sum_{k \in L_{s+1}} \delta_k w_{kj} \tag{3}$$

Saadud valemid (1), (2) ja (3) annavad kokku õppimisalgoritmi ühe sammu ühe treeningnäite jaoks. On näha ka see, et kaalude ühekordseks parandamiseks tuleb teostada kaks võrgu läbimist — edaspidine ja äraspidine *(forward and backward passes)*. Edaspidise läbimise käigus esitatakse võrgu sisenditele vastav treeningnäide, ning arvutatakse alustades esimesest kihist kõigi neuronite kohalikud väljad v_j ning väljundid y_j. Pärast võrreldakse saadud väljundvektorit oodatava tulemusega ning arvutatakse neuronite lokaalsed gradiendid — seda juba alustades viimasest kihist ning liikudes esimese kihi poole. Kui on leitud kõigi neuronite väljundid ja lokaalsed gradiendid, teostatakse kaalude parandus.

2.2 Järjestikuse ja paketipõhise treenimise moodused

Närvivõrgu õppimine toimub mingi määratud treeningnäidete komplekti võrgule mitmekordsel esitamisel ning antud protseduuri kordamisel. Terve treeningnäidete komplekti ühekordset esitamist nimetatakse epohhiks *(epoch)*. Õppimine toimub epohhide kaupa seni kuni sünaptilised kaalud ja nihked ei stabiliseeru ning vea ruutkeskmine üle terve komplekti ei koondu mingiks miinimumiks. Eristatakse kaks treenimisviisi: järjestikune *(sequential)* ja paketipõhine *(batch)*.

1. *Järjestikune treenimine* seisneb selles, et võrgu kaalude parandamine toimub pärast igat treeningnäite esitamist (eespool tuletatud valemid sobivad just selle juhu jaoks).

2. *Paketipõhises treenimises* toimub kaalude muutmine alles pärast *kõigi* näidete esitamist ning üritatakse minimiseerida keskmist veaenergiat $\mathcal{E}_{av}$:

$$\mathcal{E}_{av} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j \in C} e_j^2(n)$$

kus N on treeningnäidete komplekti suurus. Kaalude muutmine arvutatakse peaaegu samamoodi nagu eespool, ainult et seekord esineb valemites ka keskmine:

$$\Delta w_{ji} = -\eta\frac{\partial \mathcal{E}_{av}}{\partial w_{ji}} = -\frac{\eta}{N}\sum_{n=1}^{N} e_j(n)\frac{\partial e_j(n)}{\partial w_{ji}} = \cdots$$

Mõlemal viisil on oma voorused: paketipõhine treenimine annab parema lähenemise gradientvektorile (siis minimiseeritakse ju otseselt vea ruudu keskmist), ta on teoreetiliselt lihtsamini käsitletav ning annab hea arvutuste paralleliseerimise võimaluse. Samas nõuab järjestikune õppimine vähem mäluresurssi, on lihtsamini realiseeritav ning, tingimusel et näidete esitamise järjekord on iga epohhi jaoks juhuslik, omab juhuslikku iseloomu. Stohhastiline otsing kaalude ruumis aga vähendab võimalust, et algoritm takerdub lokaalses miinimumis. Kui andmestik on liiaga — s.t. treeningnäidete komplektis on korduvaid, siis erinevalt paketipõhisest õppimisest suudab järjestikune õppimine seda ära kasutada (kuna siis korratakse algoritmi ühe ja sama näite peal mitu korda ning võrk hakkab tema peal rohkem "kontsentreeruma", ehk minimiseeritakse just selle näite esitamise viga). Nendel põhjustel kasutatakse järjestikust õppimismoodust üldiselt rohkem.

2.3 Algoritmi peatamiskriteerium

Üldjuhul pole seda, et vea tagastamise algoritm koondub, võimalik näidata. Selle pärast ei leidu täpset kriteeriumi algoritmi peatamise jaoks, küll on olemas mõned mõistlikud tingimused, igaüks nendest oma vooruste ja puudustega:

- Kuna veafunktsiooni miinimumis (nii lokaalses kui globaalses) peab tema gradient $\nabla \mathcal{E}_{av}(\mathbf{w})$ võrduma nulliga, siis üks mõistlik koondumise kriteerium on järgmine (Kramer ja Sangiovanni-Vincentelli, 1989): *loeme, et algoritm on koondunud niipea kui gradientvektori norm saab väiksemaks etteantud piisavalt väikest arvust* ϵ. Selle kriteeriumi puudused on selles, et mõnikord võtab õppimine palju aega, ning et on vaja arvutada gradientvektorit.

- Üks miinimumi omadus seisneb selles, et veafunktsioon $\mathcal{E}_{av}(\mathbf{w})$ on seal statsionaarne. Sellest saame niisuguse tingimuse: *loeme, et algoritm on koondunud, kui keskmise vea ruudu muut iga epohhiga on piisavalt väike*

- Ja lõpuks veel üks kasulik kriteerium: *pärast iga õppimise iteratsiooni testitakse võrgu üldistamise võimet. Algoritm peatatakse, kui on saavutatud piisavalt head tulemused*

2.4 Väljundfunktsioon

Mitmekihilise tajuri neuronite üks tähtis parameeter on nende sigmoidaalne mittelineaarne väljundfunktsioon $\phi(\cdot)$. Tavaliselt kasutatakse väljundfunktsioonina kas logistilist funktsiooni või hüperboolset tangensit. Vaatleme kuidas näeb välja õppimisreegel mõlemal juhul:

1. *Logistiline funktsioon.*

$$\phi_j(v_j) = \frac{1}{1 + \exp(-av_j)} \qquad a > 0 \quad \text{and} \quad -\infty < v_j < \infty \tag{4}$$

   Tema tuletis on

$$\phi_j'(v_j) = \frac{a \exp(-av_j)}{(1 + \exp(-av_j))^2} = a\phi_j(v_j)(1 - \phi_j(v_j))$$

Seega lokaalne gradient võrdub väljundneuroni korral

$$\delta_j = e_j \phi_j'(v_j) = a(d_j - y_j) y_j (1 - y_j)$$

ning peidetud neuroni korral

$$\delta_j = \phi_j'(v_j) \sum_k \delta_k w_{kj} = a y_j (1 - y_j) \sum_k \delta_k w_{kj}$$

2. *Hüperboolne tangens*[3]

$$\phi_j(v_j) = a \tanh(b v_j), \qquad a > 0, b > 0 \tag{5}$$

$$\phi_j'(v_j) = \frac{b}{a}(a - \phi_j(v_j))(a + \phi_j(v_j))$$

Ning vastavad valemid lokaalse gradiendi arvutamiseks on järgmised:

$$\delta_j = \frac{b}{a}(d_j - y_j)(a - y_j)(a + y_j), \qquad \text{väljundneuroni korral}$$

$$\delta_j = \frac{b}{a}(a - y_j)(a + y_j) \sum_k \delta_k w_{kj}, \qquad \text{peidetud neuroni korral}$$

Märkime, et mõlemal juhul lokaalse gradiendi arvutamiseks ei ole vaja funktsiooni $\phi_j(\cdot)$ arvutada.

2.5 Inertsitegur (Momentum Term)

Vea tagastamise algoritm leiab mingi trajektoori kaalude ruumis, mis lähendab kiirema languse meetodi poolt pakutud trajektori. Õpitegur η määrab lähendamistäpsust — mida väiksem on η, seda siledam on trajektoor, ning seda aeglasemini toimub õppimine. Kui η on suur, siis õppimine toimub kiiremini, aga suureneb oht, et kaalude muutmised saavad liiga suurteks ning protsess muutub ebastabiilseks. Lihtne viis õppimist kiirendada ilma ebastabiilsuse ohtu on modifitseerida "delta reeglit" (valem (1)) niimoodi:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \tag{6}$$

kus α on tavaliselt positiivne konstant, mida nimetatakse inertsiteguriks *(momentum term)*. Valemi (6) nimetatakse *üldistatud deltareegliks (generalized delta rule)*, ning peaaegu alati kasutatakse vea tagastamise algoritmis just seda reeglit.
Kirjutame valemit lahti:

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ji}} \tag{7}$$

Näeme, et $\Delta w_{ji}(n)$ esineb eksponentsiaalselt kaalutud rea osasummana. Et rida koonduks peab kehtima $0 \leq |\alpha| < 1$. Kui $\alpha = 0$, töötab algoritm ilma inertsita. α saab olla ka negatiivne, kuid negatiivset α praktikas ei kasutata. Olgu siis $\alpha > 0$. Märkame, et

- kui osatuletiste $\frac{\partial \mathcal{E}(t)}{\partial w_{ji}}$ väärtused on järjest mitmel iteratsioonil sama märgiga, siis kaalutud summa (7) suureneb absoluutväärtuse poolest. See tähendab, et inerts kiirendab langust ühtlaselt langevates suundades.

[3] Tegelikult hüperboolne tangens on seesama logistiline funktsioon, ainult nihutatud ja skaleeritud

- kui osatuletised $\frac{\partial \mathcal{E}(t)}{\partial w_{ji}}$ on eri märkidega järjestikustes iteratsioonidel, siis summa (7) absoluutväärtus väheneb. Seega inerts stabiliseerub langust suundade jaoks, mis võnguvad märgi suhtes.

Seega tõepoolest, α toimib nagu inerts. Ta stabiliseerib ja kiirendab langust, ning mõnikord hoiab ära õppimisprotsessi peatamise lokaalses miinimumis veatasandil.

2.6 Heuristikad vea tagastamise algoritmi jõudluse parandamiseks

Ei saa öelda, et vea tagastamise algoritm töötab alati väga hästi. Räägitakse isegi, et vea tagastamisega närvivõrgu disainimine on pigem kunst kui teadus. Vaatamata sellele on mõned meetodid, mis märgatavalt parandavad algoritmi jõudlust.

Järjestikune vs. pakettipõhihe õppimine Nagu juba öeldud, on järjestikune õppimine arvutuslikult kiirem kui paketipõhine. Eriti kui treeningandmete hulk on suur ning liiaga. Järjestikuse õppimise juures on aga tähtis, et iga epohhi jaoks on näited juhuslikus järjekorras.

Informatsiooni mahu maksimiseerimine Üldiselt on hea, kui iga õppimise käigus tajurile esitatud treeningnäide kannaks võimalikult palju informatsiooni antud ülesande kohta (LeCun, 1993). Näiteks on soovitatav kasutada näiteid, mis annavad õppimise käigus suurema vea, või niisugusi näiteid, mis radikaalselt erinevad teistest. Samas peab siin olema ettevaatlik, sest

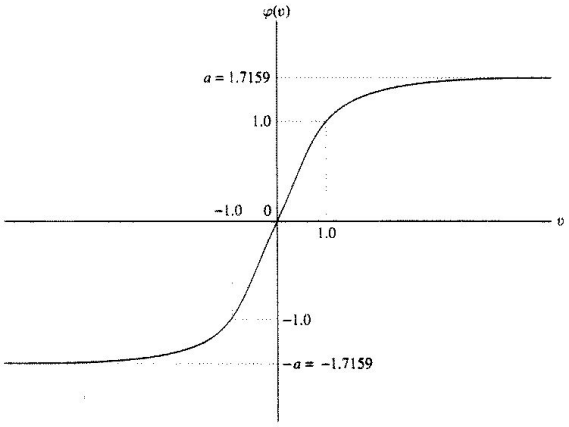
- niisugune tegevus muudab näidete loomuliku jaotust
- erindi olemasolu näidete komplektis võib õppimisprotsessi jaoks omada katastroofilisi tagajärgi. Erindite õppimine võib kõvasti häirida võrgu üldistamisvõimet.

Väljundfunktsioon Tavaliselt õppimine toimub kiiremini, kui neuronite väljundfunktsioon on paaritu. Sobiv funktsioon on muidugi hüperboolne tangens (5). Parameetrite a ja b sobivad väärtused on (LeCun, 1989, 1993):

$$a = 1.7159 \qquad b = \frac{2}{3} \tag{8}$$

Niimoodi defineeritud väljundfunktsioon $\phi(\cdot)$ omab niisugusi kasulikke omadusi:

- $\phi(1) = 1 \quad \text{and} \quad \phi(-1) = -1$
- $\phi'(0) = 1.1424$ on lähene ühele.
- $\phi''(v)$ saavutab maksimumi $v = 1$ korral.



Oodatud väärtused On tähtis, et tajuri oodatud väljundi väärtused oleksid valitud tema sigmoidaalse väljundfunktsiooni väärtuste piirkonnast. Täpsemalt, väljundneuroni j oodatud väljund d_j peab erinema mingi ϵ võrra tema väljundfunktsiooni piiritlevatest väärtustest. Vastasel juhul võrgu kaalud tavaliselt kasvavad kiiresti õppimise esimeste iteratsioonide käigus, ning neuronid küllastuvad. See tähendab, et neuronite kohalikud väljad saavutavad suuri väärtusi, mille korral nende väljundfunktsioonide tuletised on lähedased nullile. Edasi toimuvad kaalude muutmised juba väga aeglaselt. Näiteks, kui väljundfunktsioonina on valitud hüperboolne tangens ennem mainitud parameetrite a ja b väärtustega, siis tema piiritlevad väärtused on a ja $-a$, ning on saab võtta $\epsilon = 0.7159$. Siis oodatud väljund võib olla mugavalt valitud $-a + \epsilon = -1$ ja $a - \epsilon = 1$ vahel.

Sisendite normaliseerimine Iga treeningnäidete komplekti iga komponent (mis vastab võrgu mingile sisendile) peab olema *eeltöödeldud* nii, et tema keskväärtus oleks lähedane nullile või väike võrreldes tema standardhälvega. Põhjendamiseks vaatleme juhtu, kus sisendid on kooskõlaliselt positiivsed. Siis esimese peidetud kihi kõigi neuronite kaalud saavad ainult kas kasvada või kahaneda koos. Seega nende neuronite kaaluvektorid saavad liikuda mööda vea pinda ainult siksakkidega, millega kaasneb aeglane koondumine, ning mida tuleks vältida.

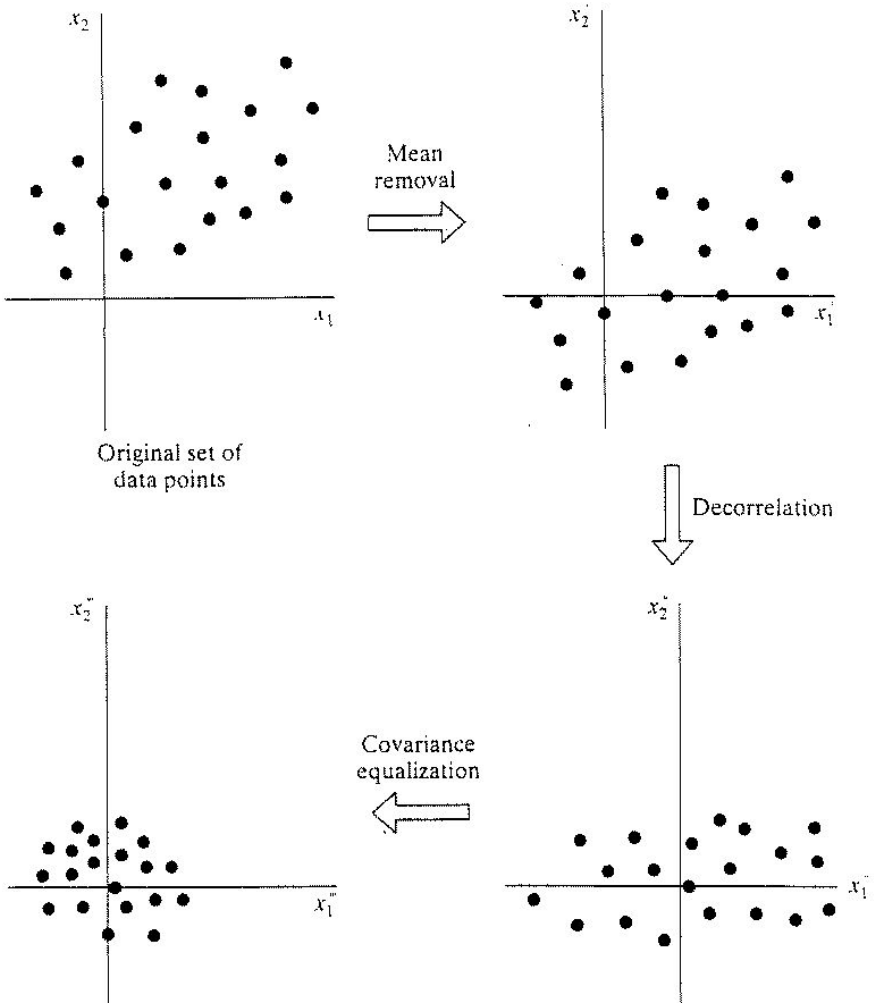
Tegelikult sisendite normaliseerimine õppimisprotsessi kiirendamiseks peab sisaldama veel kaks sammu (LeCun, 1993):

- Sisendid peavad olema *mittekorreleeritud*
- Dekorreleeritud sisendid peavad olema skaleeritud nii, et nende kovariatsioonid on peaaegu võrdsed. Sellega saavutatakse see, et erinevad sünaptilised kaalud õpivad umbes sama kiirusega.

Kaalude initsialiseerimine Sünaptiliste kaalude ja nihete hea valik võib võrgu kavandamisel suuresti abiks olla. Küsimus on selles, milline valik on hea? Kui kaaludele omistada suured algväärtused, siis on suur tõenäosus, et võrgu neuronid küllastuvad, samas kui võtta algväärtused liiga väikesteks, on võimalik, et algoritm opereerib väga siledal veapinna keskpunktile lähedal alal. Seega on kaalude algväärtuste õige valik kuskil vahepeal.

Vaatleme konkreetset näidet. Oletame, et kõigi neuronite sisendid on keskväärtusega null ning ühik-dispersiooniga:

$$\forall i \quad \mu_y = E[y_i] = 0, \qquad \sigma_y^2 = E[(y_i - \mu_i)^2] = E[y_i^2] = 1$$



Sisendite normaliseerimine

Oletame ka, et sisendid on mittekorrelleeritud, s.t.

$$E[y_i y_k] = \begin{cases} 1 & \text{kui } k = i \\ 0 & \text{kui } k \neq i \end{cases}$$

ning kaalud valitakse ühtlasest jaotusest keskväärtusega 0 ja dispersiooniga σ_w^2. Meil on vaja leida σ_w^2 niimoodi, et neuronite kohalike väljade standardhälve σ_v asetseks selles piirkonnas, mis on nende väljundfunktsioonide lineaarse ja küllastunud osade vahel. Kuna

$$\mu_v = E[v_j] = E\left[\sum_{i=1}^{m} w_{ji} y_i\right] = \sum_{i=1}^{m} E[w_{ji}]E[y_i] = 0$$

$$\sigma_v^2 = E[(v_j - \mu_v)^2] = E[v_j^2] = E\left[\sum_{i=1}^{m}\sum_{k=1}^{m} w_{ji} w_{jk} y_i y_i\right] = \sum_{i=1}^{m}\sum_{k=1}^{m} E[w_{ji} w_{jk}]E[y_i y_i]$$

$$= \sum_{i=1}^{m} E[w_{ji}^2] = m\sigma_w^2$$

(kus m on neuroni sisendite arv), siis valides näiteks $\sigma_w^2 = m^{-1/2}$, saame, et $\sigma_v = 1$, mis on just sobiv väärtus juhul kui me kasutame neuroni väljundfunktsioonina hüperboolset tangensit parameetritega a ja b määratud valemitega (8).

Õppimine vihjetest (Learning from hints) Kui võrk ehitatakse selleks, et lähendada mingi funktsiooni $f(\cdot)$, siis on alati kasulik kasutada ära kõikvõimalikku informatsiooni, mis meil sellest funktsioonist olemas on. Igasugused funktsiooni omadused nagu sümmeetriad või invariandid võivad aidata kaasa otsimise kiirendamisele ning lõpliku hinnangu täpsuse parandamisele.

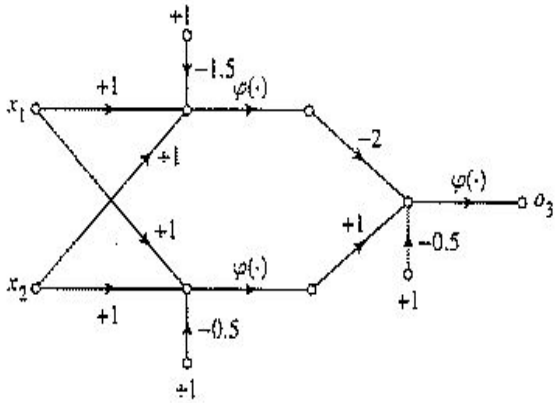
Õppimiskiirused Ideaalis peavad kõik neuronid võrgus õppima sama kiirusega. Viimastes kihtides on aga tavaliselt suuremad lokaalsed gradiendid kui esikihtidel. Seega õpitegur η peab viimastes kihtides olevate neuronite jaoks väiksem olema. Neuronitel paljude sisenditega peab õpitegur veel olema väiksem kui neuronitel vähema sisendite arvuga. LeCun (1993) pakkus, et õpitegur peab olema pöördelises sõltuvuses neuroniga seotud sünaptiliste seoste arvu ruutjuurest.

3 XOR ülesanne

Üks Minsky ja Paperti pessimistlikumaid tulemusi näitas, et lihtne ühekihiline tajur ei suuda teatud juhul eristada isegi ühikruudu tippe. Vaatleme ühikruutu tippudega $(0,0)$, $(0,1)$, $(1,0)$ ja $(1,1)$. Värvime need tipud "üle ühe"mustaks ja valgeks - mustad on siis tipud $(0,0)$ ja $(1,1)$ ning valged $(0,1)$ ja $(1,0)$. Värvi saame lihtsalt kätte ka XOR operatsiooni kasutades - kui antud tipu koordinaatide XOR on 1, on tegu valget, muidu aga musta värvi tipuga. Proovime nüüd rakendada ühte lineaarset neuronit eristamaks musti ja valgeid tippe. Kuna vastus tuleb lineaarne, saame me joone, mis eristab kahte sisendite klassi. Ühele poole joont jäävad punktid klassifitseeritakse valgeteks ja ülejäänud mustadeks. Joone asukoha ja suuna tasandil määravad neuroni sisendite kaalud ning tema nihe. On selge, et ühe sirge joonega me sisendit vajalikul viisil eristada ei saa. Küll aga on see võimalik kahest kihist koosneva võrguga.

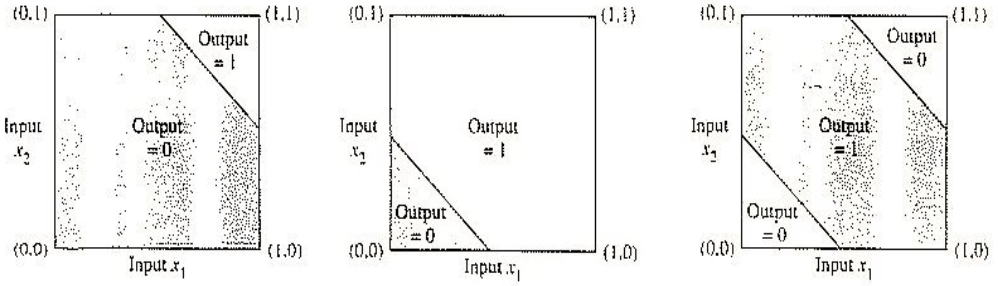
Võtame kolm neuronit, neist kaks ühte kihti ja kolmanda väljundisse. Iga neuroni väljundfunktsiooni defineerime kui $\phi(x) = 1$, kui $x > 0$, muidu 0.

Järgnevalt fikseerime iga neuroni jaoks tema kaalud ja nihke.



- Esimese kihi esimene neuron: $w_{11} = w_{12} = +1 \qquad b_1 = -\frac{3}{2}$
  Neuroni väljundiks on pooltasand $x_1 + x_2 - \frac{3}{2} > 0$

- Esimese kihi teine neuron: $w_{21} = w_{22} = +1 \qquad b_2 = -\frac{1}{2}$ Neuroni väljundiks on seega pooltasand $x_1 + x_2 - \frac{1}{2} > 0$
- Teise kihi neuron: $w_{31} = -2 \qquad w_{32} = +1 \qquad b_2 = -\frac{1}{2}$



Selle neuroni väljundiks on esimese kihi neuronite poolt moodustatud pooltasandite kombinatsioon. Teisel neuronil on väljundile positiivne (ergutav) efekt, samas kui esimesel neuronil on tugev negatiivne (pärssiv) mõju. Kui tegu on sisendiga $(0, 0)$ siis mõlemad esimese kihi neuronid annavad väljundiks nulli ning seega kogu võrgu tulemuse määrab viimase neuroni nihe - tulemus on 0. Kui sisendiks anda (1,1), siis mõlemast esimese kihi neuronist antakse edasi signaal 1 ning teise kihi neuron võtab suure negatiivse kaaluga arvesse esimese neuroni vastust. Kokku tuleb kohalik väli $-2 + 1 - \frac{1}{2} = -\frac{5}{2} < 0$ ja võrgu väljundiks 0. Kui nüüd sisend on $(1, 0)$ või $(0, 1)$, siis esimene neuron annab tulemuseks nulli ning seega kolmas (ehk väljund-) neuron väljastab tänu teise neuroni positiivsele tulemusele vastuseks ühe. Seega antud võrk eraldab tõepoolest meie klasse ning lahendab XOR ülesande.

XOR ülesannet võib analoogiliselt laiendada ka rohkematesse mõõtmetesse. Ka näiteks ühikkuubi tipud on niimoodi üksteisest eraldtavad. Veelgi enam, tuleb välja, et mitmekihilise tajuri abiga on võimalik väljendada suvalist predikaati. Küll aga suureneb siis peidetud kihtide neuronite arv sisendi kasvades eksponentsiaalselt.

On tõestatud üks veelgi suurem tulemus — nimelt on näidatud (Hornik, Stinchcombe & White, 1989; Funahashi, 1989; Cybenko, 1989; Hartman, Keeler & Kowalski, 1990) et ühest peidetud kihist mittelineaarsetest neuronitest piisab selleks, et lähendada suvalist lõpliku arvu katkevuspunktidega funktsiooni *(the universal approximation theorem).*

4 Omaduste äratundmine

Vea tagastamisega õppimisel on peidetud neuronitel mitmekihilise tajuri töös täita tähtis omaduste äratundja roll. Õppimisprotsessi arenedes hakkavad peidetud neuronid järkjärgult "avastama" treeningandmeid iseloomustavaid jooni. See toimub sisendandmete mittelineaarsete teisenduste käigus peidetud ruumi *(hidden space, feature space).* Selles uues ruumis võib sisendandmete klassifitseerimine olla lihtsam kui algselt.

5 Jacobi maatriks

Olgu W mitmekihilise tajuri kõigi vabade parameetrite (sünaptiliste kaalude ja nihete) arv. Siis me saame moodustada kaalude vektori $\mathbf{w}$, kus kaalud ja nihked võetakse loomulikus järjekorras kihtide kaupa alates sisenditest, seejärel neuronite kaupa kihis ning sünapsi kaupa neuronis. Siis võrk arvutab põhimõtteliselt mingit funktsiooni $F(\mathbf{w}, \mathbf{x}(n))$,

kus $\mathbf{x}(n)$ on n-s treeningandmete komplekt. Selle põhjal on võimalik leida hulk W-st osatuletisest mingi treeningnäite $\mathbf{x}(n)$ suhtes. Korrates sarnast arvutust $n = 1, 2, \ldots, N$ jaoks saame me lõpuks $N \times W$ osatuletiste maatriksi. Seda maatriksit nimetatakse ka antud mitmekihilise tajuri Jacobi maatriksiks (jakobiaaniks).

Katselised tulemused näitavad, et paljud närvivõrgu ülesanded on sisemiselt valesti püstitatud, mille tulemuseks on väikese astakuga osatuletiste maatriks. Siin astakut nimetame väikeseks, kui ta on väiksem nii veergude kui ka ridade arvust. Astaku puudujääk tähendab aga automaatselt seda, et vea tagastamise algoritm saab võimalikest otsingusuundadest ainult osalist infot ning treeningajad kasvavad hüppeliselt.

6 Hessi maatriks

Vea hinnangufunktsiooni $\mathcal{E}_{av}(w)$ Hessi maatriksiks (hessiaaniks) H nimetatakse $\mathcal{E}_{av}(\mathbf{w})$ teist tuletist kaalude vektori $\mathbf{w}$ suhtes.

$$H = \frac{\partial^2 \mathcal{E}_{av}(\mathbf{w})}{\partial \mathbf{w}^2}$$

Hessi maatriksil närvivõrkude uurimisel tähtis osa:

- Hessi maatriksi omaväärtused mõjutavad vea tagastamise abil õppimise dünaamikat
- Hessi maatriksi pöördmaatriks annab aluse ebaoluliste sünaptiliste kaalude kustutamiseks
- Hessi maatriks on aluseks vea tagastamisega õppimise alternatiividele.

Kolmandas peatükis veendusime, et Hessi maatriksi struktuurist sõltub suuresti LMS algoritmi koonduvus. Nii on see ka vea tagastamise algoritmi juures, aga seos on keerukam. Tavaliselt on vea tagastamisega õppimise kaudu treenitud mitmekihilise tajuri Hessi maatriksi omaväärtused sellises komplektis (LeCun et al., 1991; LeCun, 1993);

- veidi väikesi omaväärtusi
- palju keskmisi omaväärtusi
- veidi suuri omaväärtusi

Sellist grupeerimist mõjutavad faktorid on:

- Sisendid, mille keskmine ei ole nullilähedane
- Korrelatsioonid võrgu sisendväärtuste vahel ja korrelatsioonid võrgu väljundväärtuste vahel
- vea hinnangufunktsiooni teiste tuletiste suur variatsioon järjestikuste kihtide vahel. Teised tuletised on tihti madalamates kihtides väiksemad, kuna sünaptilised kaalud esimestes kihtides õpivad aeglaselt ning viimastes kihtides kiiremini.

Kolmandas peatükis nägime, et LMS algoritmi õppimisaeg on tundlik väärtuse $\lambda_{max}/\lambda_{min}$ suhtes, kus λ_{max} ja λ_{min} on Hessi maatriksi vastavalt suurim omaväärtus ja vähim nullist erinev omaväärtus. Katsed näitavad, et analoogiline tulemus kehtib ka vea tagastamise algoritmi jaoks (mis on loogiline, sest on see ju LMS algoritmi üldistus).

7 Üldistamine

Vea tagastamisega õppimise käigus alustame me tavaliselt komplekti sisenditega ning kasutame sünaptiliste kaalude arvutamiseks vea tagastamise algoritmi. Õppimises kasutatakse võimalikult palju treeningnäiteid eesmärgiga võrku üldistada. Närvivõrku nimetame üldistatavaks siis, kui tema sisendite põhjal arvutatud väljundifunktsioon on täpne (või piisavalt lähedal) ka testandmete jaoks, mida treeningu käigus ei kasutatud. Seega võib närvivõrgu treenimist vaadelda funktsiooni sobitamise ülesandena.

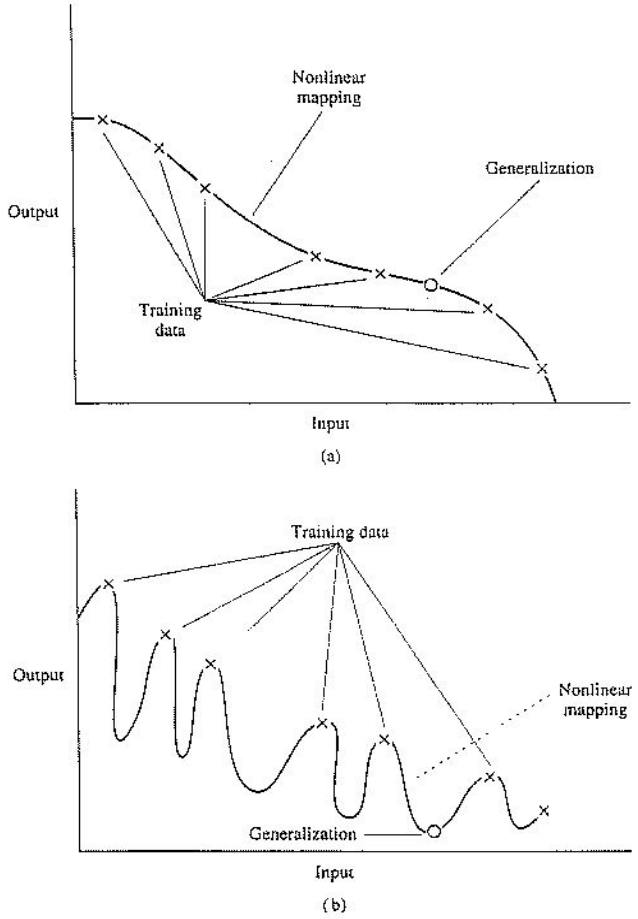


FIGURE 4.19 (a) Properly fitted data (good generalization) (b) Overfitted data (poor generalization).

Võrgu üldistamisvõime sõltub peamiselt kolmest tegurist:

- Õppimisalgoritm ja iteratsioonide arv.
- Treeningnäidete hulk.
- Peidetud neuronite arv.

Selleks et hinnata võrgu üldistamisvõimet, peame kõigepealt defineerima sobiva mõõdu vea jaoks. Selle võtame kui õppimisprotsessi keskmise vea

$$\mathcal{E}_{learn} = \frac{1}{N} \sum_{c=1}^{N} \mathcal{E}(n)$$

kus $\mathcal{E}(n)$ on n-nda treeningnäite oodatud ja saavutatud väljundi erinevus:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j\in C}(d_j(n) - y_j(n))^2$$

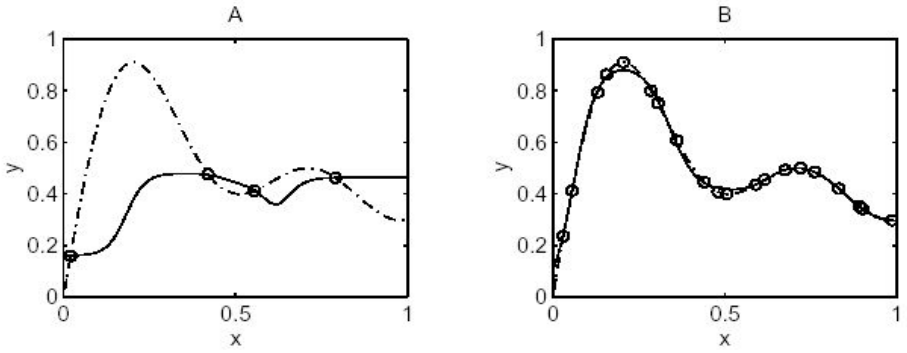
See on õppimisprotsessi käigus mõõdetav viga.

Ilmselt erineb võrgu õppimisviga fikseeritud testandmetel võrgu veast kõikidel võimalikel testandmetel. Seega tuleks võrgu üldistamisvõime määramise juures arvesse võtta mitte ainult treeningnäiteid, vaid kogu olemasolevat testimisruumi. Defineerime siin testandmete keskmise vea

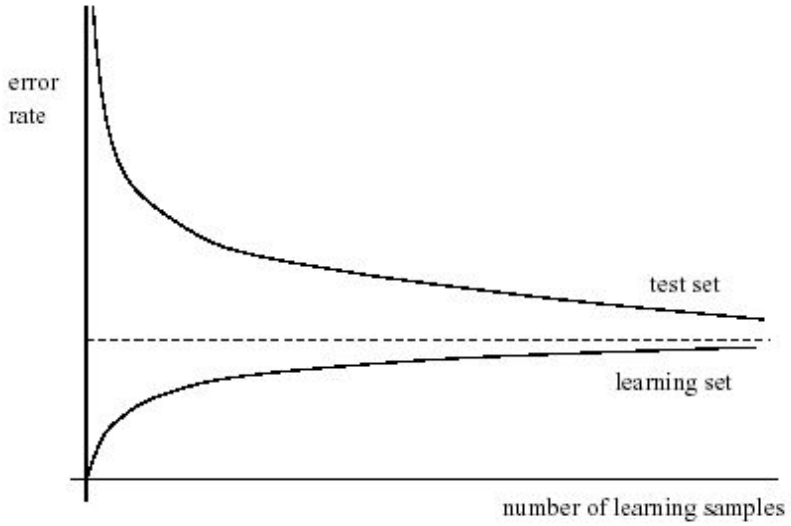
$$\mathcal{E}_{test} = \frac{1}{P_{test}}\sum_{n=1}^{P_{test}}\mathcal{E}(n)$$

kus P_{test} on kõikvõimalike testandmete arv. Vaatleme nüüd, kuidas erinevad vea suurused sõltuvad treeningandmete hulgast ja peidetud neuronite arvust.

Treeningnäidete hulk Vaatleme lihtsat ülesannet - funktsiooni $y = f(x)$ lähendamist mitmekihilise tajuriga. Vaatleme kõigepealt 5 sigmoidaalse peidetud neuroniga võrku. Oletame et treeningnäiteid on vähe ning võrk on nende peal treenitud. Algne funktsioon on joonisel kujutatud katkendliku joonega. Näeme, et $\mathcal{E}_{learn}$ on väike - võrgu poolt antud väljend kattub peaaegu täislikult treeningnäidete oodatud tulemusega. Samas aga on $\mathcal{E}_{test}$ suur, sest teistel kohtadel on võrgu viga suur. Kui sama eksperimenti korrata 20 treeningnäitega, saavutame hea funktsiooni lähenduse — $\mathcal{E}_{test}$ on väike, kuid peame lõivu maksma veidi suurenenud $\mathcal{E}_{learn}$ arvelt.



Järgneval joonisel on antud vigade sõltuvus treeningandmete hulgast — õppimisviga suureneb rohkemate andmetega, kuid üldistamisviga väheneb.



Seega väike õppimisviga väheste sisendite korral ei garanteeri veel võrgu üldiselt head taset. Treeningnäidete arvu suurenemisel koonduvad mõlemad väärtused samaks arvuks. See sõltub antud võrgu väljendusvõimsusest — kui hea on lähend optimaalsete kaalude korral. Kui vead mõnikord praktikas ei koondu samaks arvuks, siis tähendab see seda, et õppimisprotsess ei suutnud leida globaalset vea miinimumi.

Peidetud neuronite arv Mida rohkem on peidetud neuroneid, seda täpsem peaks olema üldistus. Tõepoolest $\mathcal{E}_{learn}$ väheneb alati peidetud neuronite lisamisel. Kahjuks ei saa sama öelda üldistamisvõime kohta - alates teatud hetkest muudab peidetud neuronite lisamine võrgu üldistamisvõimet halvemaks. Seda nähtust kutsutakse ka ületreeninguks (overtraining). Eriti hästi paistab see silma juhtudel, kus andmed sisaldavad müra (seega kõikides reaalse elu ülesannetes). Sel juhul võrk treenib ennast ka müra peal selle asemel, et moodustada sujuv lähend. Võrk on küll treenitud antud treeningandmete peal hästi töötama, kuid ei üldistu korralikult kõigile andmetele.