Neuroverkon opettaminen vastavirta-algoritmilla

Reetta Kallio

Pro gradu -tutkielma
Toukokuu 2021

Tarkastajat:
Prof. Marko Mäkelä
Dos. Yury Nikulin

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

Reetta Kallio: Neuroverkon opettaminen vastavirta-algoritmilla
Pro gradu -tutkielma, 26 s.
Sovellettu matematiikka
Toukokuu 2021

Neuroverkot ovat tärkeä työkalu kuvan- ja puheentunnistuksessa, klusteroinnissa ja monessa muussa tehtävässä, joihin perinteinen tietokoneohjelma ei pysty. Työn tarkoitus on esitellä neuroverkkoja ja niiden opettamista vastavirta-algoritmin avulla. Vastavirta-algoritmi käyttää hyödykseen funktion differentoituvuutta ja minimoi neuroverkon virhefunktion gradienttimenetelmien avulla. Neuroverkoista esitellään perinteinen rakenne ja kaksi erilaista neuronimallia. Neuroverkkojen opettamisessa keskitytään vastavirta-algoritmin vahvuuksiin varsinkin sigmoid-neuroverkoissa.

Työ aloitetaan neuroverkon ja sen rakenteen ja opettamisen esittelyllä. Neuroverkko koostuu kerroksista, jotka sisältävät neuroneita. Neuroverkoissa on aina vähintään 2 kerrosta: syöte- ja ulostulokerros. Näiden kerrosten välillä voi olla yksi tai useampi piilokerros. Tarkastellaan myös mitä hyötyä neuroverkoista on eri aloilla. Työssä esitellään lisäksi erot ohjatun, ohjaamattoman ja osittain ohjatun oppimisen välillä. Neuroverkko opetetaan opetusaineiston avulla, joka voi sisältää pelkästään syötevektorin tai myös ulostulovektorin.

Neuroverkkojen jälkeen tutustutaan matemaattisiin käsitteisiin, joita hyödynnetään vastavirta-algoritmissa. Käsitteitä ovat osittaisderivaatta ja gradientti, konveksisuus ja ääriarvot. Tämän jälkeen siirrytään optimoinnin perusteisiin ja funktion ääriarvon etsimiseen gradienttimenetelmän avulla. Gradienttimenetelmissä käytetään yleensä yksiulotteista viivahakua oikean askelpituuden määrittämiseksi. Työssä esitetään myös ero perinteisten gradienttimenetelmien ja neuroverkon opettamisessa joskus käytettävän stokastisen gradienttimenetelmän välillä.

Matemaattisten käsitteiden jälkeen työssä esitellään vastavirta-algoritmi ja sen laskemistapa. Vastavirta-algoritmi aloittaa nimensä mukaisesti neuroverkon ulostulokerroksesta ja siirtyy aina edeltävään kerrokseen, kunnes saavuttaa syötekerroksen. Tavoitteena on löytää neuroverkolle optimaaliset muuttujat, jotta virhefunktion arvo on mahdollisimman pieni tai nolla. Työssä kerrotaan, miten vastavirta-algoritmia käytetään ja, miten sen laskukaavat toimivat. Lopuksi esitellään vastavirta-algoritmin käyttö numerontunnistuksesa ja sigmoid-neuroverkoissa.

Asiasanat: Neuroverkot, Vastavirta-algoritmi, Koneoppiminen, Optimointi

Sisällys

1 Johdanto

Maailman ja teknologian kehittyessä tietotekniikasta on tullut tärkeä osa-alue yhä useammalla alalla. Tietokoneilla suoritetaan vaativia laskutehtäviä, ja tekoäly opetetaan keskustelemaan ihmisen kanssa sekä pelaamaan esimerkiksi shakkia. Nämä ovat hyvin mekaanisia tehtäviä, joissa voidaan noudattaa samoja kaavoja uudelleen ja uudelleen. Tehtävät, joiden lopputulos ei ole ennakoitavissa, vaativat kuitenkin monimutkaisempia ratkaisumenetelmiä kuin toistuva kaava. Tällaisten tehtävien ratkaisussa voidaan hyödyntää neuroverkkoja.

Neuroverkot mukailevat ihmisten aivoja ratkaistessaan tehtäviä aikaisemmin oppimansa pohjalta. Ne muodostuvat kerroksista, jotka sisältävät yhden tai useamman neuronin. Neuroneista tunnetuimmat ovat perseptroni ja sigmoid-neuroni. Neuroverkkoon syötetään aina syötevektori eli aineisto, jota halutaan tutkia. Neuroverkon neuroneille määritellään aktivointifunktio, jonka avulla lasketaan syötevektorille uusi arvo jokaisessa neuroverkon kerroksessa. Neuroverkosta saadaan lopulta ulostulovektori, joka on tehtävän ratkaisu. Ulostulovektorin oikeellisuutta tarkastellaan opetusaineistossa virhefunktion avulla. Yleisin virhefunktio on keskineliövirhe. Virhefunktioon syötetään neuroverkon antama ulostulo sekä tavoiteulostulo.

Neuroverkkojen opettaminen vaatii tietokoneelta paljon laskemista. Opettaminen täytyy kuitenkin tehdä huolellisesti, jotta saavutetaan oikea lopputulos. Työssä käydään läpi neuroverkon opettamisessa käytettyjen oppimismenetelmien erot ohjatun, ohjaamattoman ja osittain ohjatun oppimisen välillä. Työssä esiteltävä neuroverkon opettamismenetelmä vastavirta-algoritmi edustaa ohjattua oppimista. Vastavirta-algoritmi pohjautuu vahvasti optimointiin ja siinä käytetään perinteisiä gradienttimenetelmiä neuroverkon antaman virheen minimointiin.

Suurta osaa neuroverkon matemaattisesta puolesta ei neuroverkon suunnittelijan tarvitse ymmärtää kokonaan erilaisten syväoppimisen kirjastojen ansiosta, mutta tässä työssä tutustutaan nimenomaan neuroverkkojen matemaattisiin perusteisiin.

Työssä esitellään neuroverkkojen rakenne ja opettaminen, muutama matemaattisen optimoinnin peruskäsite sekä vastavirta-algoritmi ja sen toiminta. Neuroverkon komponenteista esitellään erilaiset kerrokset sekä sigmoid-neuroni ja perseptroni. Matemaattisesta optimoinnista esitellään viivahaku ja gradienttimenetelmä. Työssä pohditaan mitä hyötyä neuroverkoista on ja käydään läpi niiden rakennetta sekä tarkastellaan myös mitä funktioita neuroverkko käyttää toimiessaan ja miten tuloksia tulee tulkita. Lisäksi selvitetään neuroverkon hyötyjä arjessa ja erilaisissa käyttökohteissa.

2 Neuroverkko

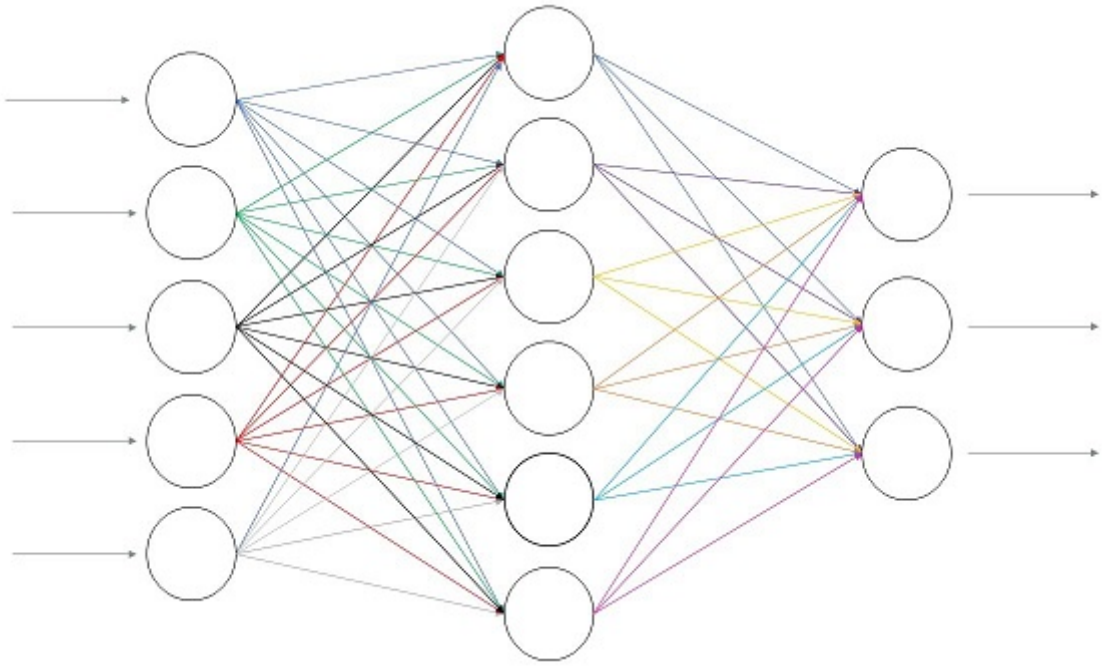
Neuroverkot lisäävät uusia mahdollisuuksia hyödyntää tietokoneita. Sen sijaan, että kerromme tietokoneelle tarkasti mitä haluamme sen tekevän tai muotoilemme laajan ongelman useammaksi pieneksi, annamme neuroverkon itse oppia opetusaineistosta ja muodostaa omat ratkaisunsa käsiteltävään ongelmaan. [18] Neuroverkot määritellään pehmolaskennaksi (engl. Soft computing) sumean logiikan ja geneettisten algoritmien ohella. [25] Neuroverkkojen avulla voidaan arvioida mitä tahansa jatkuvaa funktiota ja ratkaista useita erilaisia ongelmia kuten klusterointi, luokittelu, funktion arviointi, signaalinkäsittely ja kuvion- tai puheentunnistus. [26]

Neuroverkkoja voidaan käyttää tehtävissä, joihin perinteisen ohjelmoinnin tai symbolisen tekoälyn soveltaminen on hyvin haastavaa. Tällaisia tehtäviä ovat esimerkiksi postitoimistoissa käsinkirjoitettujen osoitteiden tunnistaminen tai lääketieteessä kasvainten löytäminen. [18] [20] Suurin hyöty neuroverkoista saadaan ilmiöiden monipuolisuuden ja jatkuvuuden mallintamisessa. Neuroverkkoja on mahdollista opettaa tunnistamaan hienovaraisia ja monisäikeisiä riippuvuussuhteita. [25] Neuroverkot voittivat jopa matemaattisia ohjelmistoja (Matlab ja Mathematica), kun oli kyse tarkkuudesta ja nopeudesta tehtäviä ratkoessa. Neuroverkon avulla onnistuttiin ratkaisemaan sellaisia tehtäviä, joihin nämä ohjelmistot eivät saaneet muodostettua ratkaisuja. [24]

Neuroverkot esiteltiin ensimmäisen kerran vuonna 1943, kun McCullough ja Pitts esittivät laskennallisen mallin, jonka inspiraationa toimivat ihmisen aivot. Tämä innosti alulle neuroverkkojen tutkimisen ja kehittämisen. [26] Neuroverkkoja on tutkittu enemmän 1980-luvun puolesta välistä lähtien, mutta vasta vuonna 2006 muodostui käsite syväoppimisesta ja tämä mahdollisti neuroverkkojen hyödyntämisen laajemmassa mittakaavassa. [18] [25]

Neuroverkoilla pyritään yhä jäljittelemään ihmisen aivoja. [17] Aivan kuten ihmisilläkin, neuroverkoilla on lähtötietona aiemmin ratkaistut ongelmat ja matemaattisissa ongelmissa se pyrkii tunnistamaan funktioita tai lausekkeita vertaamalla niitä aiemmin ratkaistuihin tehtäviin. [24] [25] Neuroverkkojen avulla voidaan kääntää tekstiä konteksti huomioiden, eikä vain sana sanalta, jolloin yksittäisten sanojen merkitys säilyy kyseisessä kontekstissa. Ne eivät kuitenkaan vielä ole ihmisaivojen vertaiset. Neuroverkoissa on murto-osa ihmisaivojen neuronimäärästä ja neuroverkkoja voidaan opettaa vain optimoimalla muuttujia. Ihmisaivot oppivat koko ajan ja monella eri tavalla. Jotta ongelmilta voidaan välttyä, neuroverkon kunnollinen opettaminen monipuolisella opetusaineistolla on erityisen tärkeää. [23]

Neuroverkon opettaminen tarkoittaa neuroverkon soveltamista käytettävään aineistoon. Se perustuu sopivan neuroverkkorakenteen etsintään ja neuroverkossa esiintyvien muuttujien muokkaamiseen. Tähän sisältyy mm. sopivan aktivointifunktion, neuronien määrän ja järjestyksen päättäminen. Muuttujien optimoinnissa kohdefunktio sisältää kaikki neuroverkon neuroneiden ja niiden välisten painojen arvot. Opettaminen on kriittinen vaihe neuroverkoissa, sillä mitä paremmin neuroverkon opettaa sitä paremmin se suoriutuu sille annetuista tehtävistä. Neuroverkon opettamisen onnistuminen riippuu arkkitehtuurista, opetusalgoritmista ja opetusympäristöstä. [26]



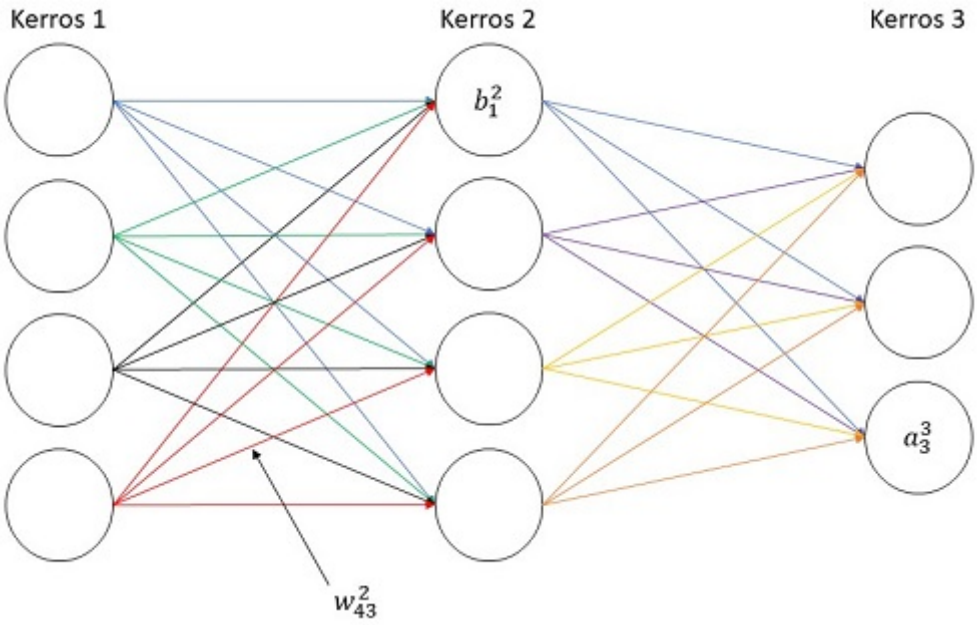
Kuva 1: Eteenpäin kytketty neuroverkko.

Kun neuroverkossa syöte jatkaa aina kerroksesta seuraavaan kerrokseen, eikä palaa taaksepäin tai saman kerroksen neuronit eivät ole liitettyjä toisiinsa, sitä kutsutaan eteenpäin kytketyksi neuroverkoksi (engl. Feedforward neural network), joka näkyy kuvassa 1. Neuroverkko voi myös olla takaisinkytketty neuroverkko (engl. Recurrent neural network), jossa syöte voi kulkea neuronista eteenpäin, taaksepäin tai saman kerroksen toiseen neuroniin. Tällöin aktivointifunktio on riippuvainen myös seuraavien kerroksien neuroneista. [18]

Esimerkki neuroverkon toiminnasta on Guillaume Lamplen ja François Chartonin vuonna 2019 muodostama neuroverkko, joka osasi laskea integraaleja ja ratkaista differentiaaliyhtälöitä. He muokkasivat matemaattisia ongelmia enemmän puumalliin. Puun oksiksi valittiin matemaattiset operaattorit, esimerkiksi jakolasku, vähennyslasku, potenssi ja trigonometriset funktiot. Lehtinä taas olivat numerot ja muuttujat. Tällä tekniikalla neuroverkot pilkkovat vaativia matemaattisia ongelmia pienemmiksi ongelmiksi, joita neuroverkko on ratkaissut aiemmin. [24]

2.1 Neuroverkon rakenne

Neuroverkko koostuu yleisesti solmujoukosta (ns. keinotekoiset hermosolut). Solmujoukko koostuu solmuista eli neuroneista. Niiden välillä on liitosjoukko (ns. keinotekoiset synapsit) eli painot. Solmut ja liitokset muodostavat neuroverkon kerrokset. Matalamman tason neuroverkossa on yleensä kolme kerrosta neuroneita, jotka käsittelevät ja muodostavat ulostulon. Tällaisissa neuroverkoissa on yksi piilokerros. Korkeamman tason neuroverkoissa on useampi piilokerros, joissa neuronit käsittelevät syötettä. Kummankin tason neuroverkot pystyvät ratkaisemaan monimutkaisia ongelmia, mutta yleisesti neuroverkot antavat tarkempia vastauksia mitä enemmän piilokerroksia niissä on. [17] [19] [25] Neuroverkon rakennetta havainnollistetaan ku-



Kuva 2: Neuroverkon komponentit graafisesti.

vassa 2.

Piilokerroksien nimi tulee siitä, ettei niitä varsinaisesti näe neuroverkossa. Niissä lasketaan syötteelle uusi arvo ja lisätään solmun sisältämä vakiotermi. Niillä ei ole tavoiteulostuloa, eikä niihin mene uusia syötteitä. Piilokerrokset ovat haastavia muodostaa kerralla oikein, koska verrattuna syöte- ja ulostulokerrosten muodostamiseen, niihin ei ole yleistä kaavaa. Jokainen piilokerros voi tunnistaa eri asian esimerkiksi, jos kuvan tunnistuksessa etsitään autoja ensimmäinen piilokerros etsii renkaita, toinen piilokerros tuulilasin ja ikkunat, ja loput piilokerrokset muita auton osia. Näiden oikeanlaiseen muodostamiseen on kehitetty useita heuristisia menetelmiä, joilla pyritään saavuttamaan haluttu lopputulos. [7] [18]

Neuroverkon syötekerroksessa ei vielä itsessään lasketa mitään, mutta syötekerrokseen annettu vektori jatkaa mahdollisten piilokerroksien kautta ulostulokerrokseen. Syötekerroksen neuroneiden määrä riippuu syötevektorin alkioiden määrästä. [17] [18]

Neuroverkon neuronia kerroksessa k merkitään:

$$a_i^k,$$

missä a kertoo kyseessä olevan neuroni, i kertoo kuinka mones neuroni on kerroksessaan ja k kuvastaa kuinka mones kerros on kyseessä. Neuroverkon termejä voidaan myös esittää vektoreina ja matriiseina. Neuronivektori kerrokselle k merkitään $\mathbf{a}^k = (a_1^k, \ldots, a_{r^k}^k)^T$. Neuronin a_i^k arvo saadaan [18]

$$a_i^k = g\left(\sum_j w_{ji}^k a_j^{k-1} + b_i^k\right) = g(z_j^k) = o_j^k.$$

Näin voidaan laskea koko kerroksen k neuroneiden arvot ja saadaan aktivointivektori $\mathbf{g}^k$. Neuronin arvon laskentakaavasta näkee selvästi, että neuroverkon seuraava kerros

riippuu aina edeltävästä kerroksesta. Aktivointivektorin avulla voidaan tarkastella neuroverkkoa laajemmassa näkökulmassa, eikä ainoastaan neuroni neuronilta. [18] Esimerkiksi kerrosten 1, 2 ja 3 neuronit saadaan

$$a_i^1 = x_{hi},\ a_i^2 = \sum_j w_{21} x_{hj} + b_i^2 \text{ ja } a_i^3 = g\big(\sum_j w_{21} a_j^2 + b_i^3\big).$$

Neuroneiden välillä on painot

$$w_{ji}^k,$$

missä w kertoo kyseessä olevan neuroneiden välinen paino, k kertoo mihin kerrokseen paino menee, i kuvaa kerroksessa k olevan neuronin paikkaa ja j kuvaa kerroksessa $k-1$ olevan neuronin paikkaa. Neuroverkon neuroneiden väliset painot voidaan esittää painomatriisina

$$\mathbf{W}^k = \begin{pmatrix} w_{11}^k & w_{12}^k & \dots & w_{1r^k}^k \\ w_{21}^k & w_{22}^k & \dots & w_{2r^k}^k \\ w_{31}^k & w_{32}^k & \dots & w_{3r^k}^k \\ \dots & \dots & \dots & \dots \\ w_{r^k1}^k & w_{r^k2}^k & \dots & w_{r^kr^k}^k \end{pmatrix}.$$

[18]

Jokaiseen neuroverkon neuroniin on liitetty vakiotermi

$$b_i^k,$$

missä b kertoo kyseessä olevan vakiotermi, i kertoo minkä neuronin vakiotermistä on kyse siinä kerroksessa ja k kuvastaa kuinka mones kerros on kyseessä. Kerroksen k vakiotermivektoria merkitään $\mathbf{b}^k = (b_1^k, \dots, b_{r^k}^k)^T$ [18]

Neuroneiden väliset painot ja neuroneihin liitetyt vakiotermit ovat neuroverkon muuttujia, joita muokataan neuroverkon opettamisen aikana. [17] [25] Neuroverkon kerroksen k neuroneiden määrää merkitään r^k.

Neuroverkon aineisto sisältää syötevektorin $\mathbf{x}_h = (x_{h1}, x_{h2}, \dots, x_{hn})$. Ohjatussa oppimisessa opetusaineisto sisältää myös syötevektoria vastaavan ulostulovektorin $\mathbf{y}_h = (y_{h1}, y_{h2}, \dots, y_{hm})$. Työssä käsiteltävä opetusaineisto muodostuu vektoripareista $(\mathbf{x}_h, \mathbf{y}_h)$. Näitä on N kappaletta ja niitä merkitään $\mathbf{X} = \big\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\big\}$. [12]

Neuroverkossa neuroneilla on aktivointifunktio g, jonka antama arvo jatkaa matkaansa seuraavaan kerrokseen. Aktivointifunktio voi olla mitä muotoa tahansa, mutta yleensä valitaan lineaarinen aktivointifunktio $g(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ [17], koska tällöin neuroverkon piilokerrosten lukumäärää ei ole rajoitettu. Lopullinen ulostulo on silloin vain lineaarikombinaatio alkuperäisestä syötteestä. Kuitenkin lineaaristen funktioiden ongelma on siinä, että neuroverkko ei tällöin pysty ratkaisemaan epälineaarisia ongelmia kuten XOR-logiikka. [21] Aktivointifunktio päätetään yleensä neuroverkon koon ja rakenteen mukaan. [18]

2.2 Neuroverkkomallit

Neuroverkoissa voidaan käyttää useampia erilaisia neuroverkkomalleja, joista tässä työssä esitellään kaksi tärkeintä: perseptroni- ja sigmoid-neuroverkko. Nämä ovat

käytetyimmät neuronimallit. Perseptroni on yksinkertaisin binäärinen neuronimalli, josta on jatkokehitelty sigmoid-neuroni. [18] Työssä esiteltävä vastavirta-algoritmi kehitettiin alunperin sigmoid-neuroverkkojen opettamiseen.

Perseptroni-neuroverkko sisältää yleensä vain syöte- ja ulostulokerrokset. Tällainen neuroverkko ei osaa kuitenkaan ratkaista epälineaarisia tehtäviä. Yksinkertaiseen perseptroni-neuroverkkoon voidaan lisätä yksi tai useampi piilokerros, jolloin myös epälineaaristen tehtävien ratkaiseminen onnistuu. [26]

Perseptronit muodostavat monesta binäärisestä syötteestä yhden binäärisen tuloksen. Yksittäisen syötteen $\mathbf{x}_h$ tärkeyttä voidaan painottaa painon $\mathbf{w}^k$ koolla. Jos $\sum_i w^k_{ji} x_{hi}$ ylittää tietyn kynnyksen p, syötteen arvo on 1 ja muuten 0 [18]

$$\text{tulos} \;=\; \begin{cases} 0, \text{ jos } \sum_i w^k_{ji} x_{hi} < p \;\Leftrightarrow\; \sum_i w^k_{ji} x_{hi} - p < 0 \\ 1, \text{ jos } \sum_i w^k_{ji} x_{hi} \geq p \;\Leftrightarrow\; \sum_i w^k_{ji} x_{hi} - p \geq 0. \end{cases}$$

Perseptroni on yksinkertainen neuroni, jonka aktivointifunktio on porrasfunktio. Tämä on varhaisimpia malleja neurolaskennassa ja perseptroneiden painojen määrittäminen aineistosta onnistuu perseptroni-algoritmin avulla. Algoritmissa neuronille syötetään yksi syöte opetusaineistosta ja päivitetään neuroverkon painoja, jos saatu tulos ei vastaa opetusaineistossa olevaa tulosta. [9]

Perseptroneita sisältävä neuroverkko on altis muutoksille, koska sen ulostulona on aina 0 tai 1. Pieni muutos neuroverkon painoissa voi muokata neuroverkon antamia ulostuloja merkittävästi. Neuroverkon painoja ei voida muuttaa niin, että yksi opetusaineiston syöte saa oikean lopputuloksen ilman, että riskeerataan kaikkien muiden opetusaineiston syötteiden lopputulosten oikeellisuutta.[18]

Sigmoid-neuroverkko kehitettiin perseptroni-neuroverkon jatkona. Vaikka monikerroksinen perseptroni-neuroverkko osasi ratkaista epälineaarisiakin ongelmia, haluttiin muodostaa yleistetympi estimoija. Sigmoid-neuronit ovat samankaltaisia kuin perseptronit, mutta sen sijaan, että sigmoid-neuroni saisi tuloksen 0 tai 1, se voi saada tuloksen välillä $[0, 1]$. Tällöin pieni muutos sigmoid-neuronien yksittäisessä painossa aiheuttaa vain pienen muutoksen tuloksissa. Käyttämällä sigmoid-neuronia perseptronin sijaan, voidaan vähentää neuroverkon herkkyyttä muutokselle. [18]

Kolmogorovin teoreema vuonna 1957 todisti, että eteenpäin kytketty sigmoid-neuroverkko, joka sisältää edes yhden piilokerroksen, osaa estimoida mitä tahansa jatkuvaa funktiota. Tällöin piilokerroksen neuronimäärän tulisi olla äärellinen ja neuroverkon aktivointifunktion tulisi olla sigmoid-funktio. [26] Sigmoid-neuronin kohdalla perseptroneissa käytetty kynnys p korvataan yleensä neuroniin liitetyllä vakiotermilllä $b = -p$. Sigmoid-neuronin antamaa ulostuloa kuvataan sigmoid-funktiolla [18]

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$

Vaikka sigmoid-funktion avulla saadaan arvot välillä $[0, 1]$, se ei aina kerro haluttua tietoa. Joskus tarvitaan kyllä/ei-tyyppinen tai totuusarvoinen vastaus. Tällöin sigmoid-neuroverkkoa voidaan kuitenkin modifioida perseptroni-neuroverkon kaltaiseksi niin, että yli 0.5 arvoista annetaan tulos 1 ja alle 0.5 arvoista tulos 0. Sigmoid-funktiota käytetään yleensä neuroverkon aktivointifunktiona, koska sitä on yksin-

kertaista derivoida. [18] Varsinkin gradienttimenetelmät hyödyntävät usein sigmoid-funktiota, koska sillä on helppo derivaatta

$$\frac{d}{dz}\sigma(z) = \frac{e^z}{(e^z+1)^2}.$$

Aktivointivektori $\mathbf{g}^k$ voidaan kirjoittaa sigmoid-funktiolle muodossa [18]

$$\mathbf{g}^k = \sigma\left(\mathbf{w}^k\mathbf{a}^{k-1} + \mathbf{b}^k\right) = \frac{1}{1+e^{-\sum_i w_{ji}^k a_i^k - b_i^k}}.$$

2.3 Neuroverkon opettaminen

Neuroverkon kunnollinen opettaminen opetusaineiston avulla on tärkeintä luotettavan mallin muodostamisessa. Opetusaineiston tulee olla laaja, jotta opettaminen on tehokasta. [3] Opettamisen kautta neuroverkko yleistää oppimiaan esimerkkejä perusperiaatteiksi ihmisaivojen tapaan. [25] Neuroverkon opettamisessa muokataan neuroverkon muuttujia siten, että ensin opetusaineistolla saadaan neuroverkosta oikea tulos ja sitten tarkistetaan testiaineiston avulla neuroverkon soveltuvuus uusien samankaltaisten tehtävien ratkaisemiseen. Kun opetus- ja testiaineiston syötteet muodostavat haluttuja ulostuloja, voidaan neuroverkko yleistää vastaaviin aineistoihin. [26] Neuroverkon tietty rakenne ja opettamisen kautta muokatut muuttujat soveltuvat yleensä tietynlaiseen sovellukseen. Numerontunnistukseen tarkoitettua neuroverkkoa ei siis voi hyödyntää tekstin kääntämisessä.

Neuroverkon opettaminen voi olla tahattomasti puolueellista saatavilla olevan opetusaineiston takia. Google Kääntäjä on kehitetty kääntämään tekstiä kontekstin perusteella, mutta käytetty opetusaineisto on voinut olla puolueellista kielissä, joissa on feminiininen ja maskuliininen hän-pronomini. Kääntäessä sukupuolineutraaleista kielistä virkkeitä, joissa esiintyy hän-pronomini, Google Kääntäjä käänsi tekniikan ja luonnontieteiden alan ammattinimikkeitä sisältäviä virkkeitä englanniksi käyttäen maskuliinista hän-pronominia. [22]

Neuroverkkojen opettamiseen on esitetty lukuisia algoritmeja ja tekniikoita. Neuroverkot käyttävät todennäköisyyslaskentaa hyödykseen, kun opettelevat uutta ja antavat vastauksia opetusaineiston avulla oppimansa pohjalta. Aikaisemmin on keskitytty hyvin paljon gradienttipohjaisiin menetelmiin, mutta niiden rajoittunuuden takia on esitelty myös metaheuristisia optimointimenetelmiä. Gradienttimenetelmät saattavat myös päätyä helpommin lokaaliin minimiin globaalin minimin sijaan. [18] [24] [25] [26] Tässä työssä esitellään gradienttipohjainen vastavirta-algoritmi menetelmänä opettaa neuroverkko.

Neuroverkon opettamisessa käytetään yleensä virhefunktiota $C(\mathbf{X}, \theta)$. Virhefunktio määrittää halutun ulostulon $\mathbf{y}_h$ ja neuroverkon antaman ulostulon $\hat{\mathbf{y}}_h$ välisen erotuksen syötteelle $\mathbf{x}_h$. Virhefunktion termi θ pitää sisällään kaikki neuroverkon muuttujat eli painot w_{ji} ja vakiotermit b_i^k. Virhefunktiona käytetään yleensä keskineliövirhettä [12]

$$C(\mathbf{X}, \theta) = \frac{1}{2N}\sum_h ||\hat{\mathbf{y}}_h - \mathbf{y}_h||^2. \tag{1}$$

Tämä on yleisin virhefunktio, koska keskineliövirhe ei ole koskaan negatiivinen. Sen differentointi ja ääriarvon etsiminen on yksinkertaisempaa, koska sen kaikki komponentit ovat positiivisia tai korotettu toiseen potenssiin. Tavoitteena neuroverkon opettamisessa on, että $\hat{\mathbf{y}} = \mathbf{y}$, joten etsitään optimaaliset painot w_{ji}^k ja vakiotermit b_i^k virhefunktion minimointiin. [18] Toinen mahdollinen virhefunktio voisi olla suurimman uskottavuuden estimointi (engl. Maximum Likelihood Estimate), jonka tarkkuus kasvaa opetusaineiston syötteiden määrän kasvaessa. [5]

Neuroverkon voi opettaa myös ilman mitään menetelmiä tai virhefunktiota, mutta se toimii vain hyvin pienille neuroverkoille. Tällöin oikeita muuttujien arvoja voidaan etsiä kokeilemalla. Suuremmissa neuroverkoissa on monta eri muuttujaa, eivätkä muuttujat välttämättä muodosta sileää funktiota, jolloin niiden käsin ratkaiseminen ei ole kannattavaa. Korkeamman tason neuroverkoissa myöskään pienet muutokset eivät vaikuta neuroverkon lopputulokseen välttämättä mitenkään. Gradienttimenetelmien avulla voidaan huomioida kaikki eri neuroverkon muuttujat kerralla, jolloin ne suoriutuvat neuroverkon opettamisesta paremmin kuin perinteinen derivointi. [18][19]

Neuroverkkojen opettamisessa on kolme pääsuuntaa. Pääsuuntien sisällä on useita eri menetelmiä opettaa neuroverkkoja, mutta ne voidaan aina sijoittaa jonkin näistä alle. Neuroverkko voidaan opettaa ohjatusti, ohjaamattomasti tai osittain ohjatusti. Neuroverkon opetusaineisto sisältää ohjaamattomassa oppimisessa vain syötevektorit $\mathbf{x}_h$. Ohjatussa oppimisessa opetusaineisto sisältää myös syötevektoria vastaavan ulostulovektorin $\mathbf{y}_h$. Työssä käsiteltävä opetusaineisto muodostuu vektoripareista $(\mathbf{x}_h, \mathbf{y}_h)$. [12] Neuroverkko tarvitsee toimiakseen aina aineiston eli syötteet. Opettamisen aikana käytetään yleensä opetusaineistoa, jossa syötteelle tiedetään myös oikea ulostulo. Opetusaineiston avulla pyritään löytämään sellaiset painot w_{ji}^k ja vakiotermit b_i^k, että virhefunktio minimoituu. Osa opetusaineistosta on yleensä testiaineisto, jolla sitten tarkistetaan neuroverkon tarkkuutta ja kykyä antaa oikea ulostulo.

Ohjattua oppimista käytetään silloin, kun on saatavilla opetusaineisto, joka sisältää myös ulostulovektorin. Ohjatusti opetettu neuroverkko osaa toimia jatkossa samojen ennakkokäsitysten perusteella myös eri aineiston kanssa ja tuottaa oikean vastauksen, esimerkkinä käsinkirjoitetun tekstin tunnistaminen tai kuvien luokittelu aihepiirin perusteella. Ohjattua oppimista voidaan käyttää hyvinkin monimutkaisiin tehtäviin. Siinä käytetään lokeroitua aineistoa. Lokeroidussa aineistossa alkiolle on ennalta määritelty erilaisia lokeroita. Lokerot voivat olla esimerkiksi "puu", "kasvain", "auto"tai "3". Lokeroidun aineiston avulla neuroverkot voivat tunnistaa itse tuottavatko ne oikean lopputuloksen. [11] Ohjatun oppimisen menetelmä on esimerkiksi vastavirta-algoritmi, joka esitellään tässä työssä tarkemmin myöhemmin. Toinen ohjatun oppimisen menetelmä on tukivektorikone, joka ratkaisee luokitteluongelmia kolmikerroksisen neuroverkon avulla. [26]

Aineiston lokeroinnin haastavuus riippuu hyvin paljon aineistosta itsestään. Valokuvissa ihmisen on useimmiten helppo nähdä mitä siinä näkyy, mutta esimerkiksi röntgenkuvien tulkitsemiseen tai kasvaimien tunnistamiseen tarvitaan ammattilainen. Lokeroidun aineiston saaminen on paljon kalliimpaa ja hitaampaa kuin lokeroimattoman. [2] [11] [20]

Ohjaamaton oppiminen tapahtuu puolestaan lokeroimattoman aineiston avulla.

Lokeroimattomassa aineistossa neuroverkko ei saa ennalta tietää aineistosta mitään lokeroiden avulla. Lokeroimatonta aineistoa saa helposti keräämällä erilaisten kyselyiden ja näytteiden kautta ja sitä on tarjolla huomattavasti enemmän kuin lokeroitua. [11] Ohjaamatonta oppimista voidaan hyödyntää suurten aineistomäärien esikäsittelyssä, esimerkiksi klusteroinnissa tai muissa vastaavissa tehtävissä. [11] Mahdollisuudet ohjaamattoman oppimisen kanssa ovat kuitenkin rajalliset. [2] Se toimii paremmin, jos algoritmia ei tarvitse yleistää muihin aineistoihin. Esimerkki ohjaamattoman oppimisen menetelmästä on Kohosen itseorganisoituva kartta, jossa on kaksikerroksinen neuroverkko. [26]

Osittain ohjatussa oppimisessa algoritmia opetetaan lokeroidun ja lokeroimattoman opetusaineiston avulla. Siinä ohjelmoija klusteroi ensin samantyyppiset aineistot käyttämällä jotakin ohjaamattoman oppimisen algoritmia ja lokeroidun aineiston avulla käsittelee lopunkin aineiston. [2] Osittain ohjatun oppimisen algoritmit vaativat aineistolta seuraavia ominaisuuksia.

1. **Jatkuvuusoletus** Algoritmi olettaa, että lähellä toisiaan olevat pisteet ovat samankaltaisia ja voidaan lokeroida samaan lokeroon

2. **Klusterointioletus** Aineisto voidaan jakaa diskreetteihin osiin ja samassa osassa olevat pisteet ovat todennäköisemmin samassa lokerossa

3. **Monisto-oletus** Aineisto on upotettu suurempaan ulottuvuuteen kuin mitä sen todellinen ulottuvuus on. Esimerkiksi pöydällä olevan lasilevyn päälle asetettujen esineiden sijainti voidaan kertoa xy-koordinaateilla. Jos lasilevy nostetaan pystyyn pöytää vasten, tarvitaan myös z-koordinaatti, vaikka esineet ovat edelleen samalla lasilevyllä. Tällöin voidaan käyttää hyödyksi pienemmässä ulotuuvuudessa määriteltyjä etäisyyksiä.

Osittain ohjattua oppimista käytetään hyödyksi esimerkiksi puheentunnistuksessa ja Internetin sisällön arvioinnissa. Myös Googlen hakukone opetetaan osittain ohjatulla oppimisella arvioimaan tuloksena tulleen nettisivun relevanssia. [2]

Teoreettisesti seuraavilla neljällä keinolla on mahdollista löytää globaali minimi: [26]

1. Neuroverkon muuttujat on alussa määritelty satunnaisesti kolmikerroksiselle neuroverkolle. Neuroverkon piilokerroksen neuronien määrää ei ole rajoitettu. Gradienttimenetelmällä voidaan tässä tilanteessa välttyä lokaaliin minimiin jumittuminen.

2. Neuroverkon opetusaineisto on lineaarisesti separoituvaa ja neuroverkon rakenne on pyramidimainen. Tällöin virhealueella ei ole lokaalia minimiä,

3. Kolmikerroksisessa neuroverkossa on N opetussyötettä, jotka on valittu jonkin kriteerin perusteella, ja $N - 1$ sigmoid-neuronia piilokerroksessa. Piilokerroksen viimeinen neuroni on ns. Dummy-neuroni, jonka tarkoitus on pienentää virhettä. Tällöin virhealueella ei ole lokaalia minimiä.

4. Opetusalgoritmia voidaan parantaa globaalin laskeutumisen algoritmilla. Globaali laskeutumisen algoritmi kehitettiin korvaamaan gradienttimenetelmät,

mutta sitä ei vielä ole tutkittu tarpeeksi, jotta voitaisiin varmuudella sanoa sen olevan gradienttimenetelmiä tehokkaampi keino. Tätä menetelmää ei käydä sen tarkemmin läpi, koska se ei ole oleellinen tämän työn kannalta. [4]

Nämä neljä keinoa ovat kuitenkin riippuvaisia piilokerroksen neuroneiden määrästä, opetusaineiston koosta, ulostulokerroksen neuroneiden määrästä ja siitä, ettei piilokerroksen neuronien lukumäärä saa olla pienempi kuin opetusaineiston syötteiden lukumäärä. Nämäkään keinot eivät yleensä takaa globaalin minimin löytymistä, koska todellinen maailma poikkeaa hyvin paljon teoreettisesta maailmasta. [26]

Vaikka neuroverkon opettamiseen on kehitetty monta erilaista algoritmia, algoritmeilla on hyvin samankaltaiset perusperiaatteet. Neuroverkon opettamisessa on yksinkertaisuudessaan viisi eri vaihetta: [19]

Vaihe 1 Neuroneiden välille asetetaan alkupainot w_{ji}^k ja vakiotermit b_i^k.

Vaihe 2 Opetusaineiston syöte kulkee neuroverkon läpi ja saadaan ulostulo.

Vaihe 3 Koska toimitaan opetusaineiston kanssa, tiedetään mikä oikean ulostulon kuuluisi olla. Muodostetaan virhefunktio, joka määrittää erotuksen oikean vastauksen ja neuroverkon antaman ulostulon välillä.

Vaihe 4 Lasketaan neuroverkon ulostulon ja halutun ulostulon välinen virhe ja sopivalla menetelmällä muokataan neuroneiden muuttujia sopivammiksi. Neuroverkon opettamiseen on useampia menetelmiä, joista tässä työssä esitellään vastavirta-algoritmi.

Vaihe 5 Päivitetään uudet painot ja vakiotermit neuroneiden välille.

Vaiheita 2-5 voidaan toistaa niin monta kertaa, että neuroverkon ulostulo vastaa tavoiteulostuloa. [19] Vaiheet voidaan kuvata hyvinkin yksinkertaisesti, mutta ne saattavat olla hyvin työläitä. Vaiheiden läpikäymisen jälkeen neuroverkkoa voidaan testata testiaineistolla. Tämä kertoo opettamisen onnistumisesta.

3 Matemaattisia määritelmiä

Neuroverkko rakennetaan aina funktioiden avulla ja on tärkeää ymmärtää niiden toimintaperiaatteet. Neuroverkon opettaminen vaatii differentiaalilaskentaa, jotta virhefunktio saadaan minimoitua. Tässä osiossa esitellään osittaisderivaatta ja gradientti, konveksisuus ja ääriarvot tukemaan myöhemmin esiteltävää vastavirta-algoritmia. Näiden lisäksi käydään lyhyesti läpi optimoinnin perusteet ja gradienttimenetelmät. Gradienttimenetelmissä käytettävät viivahakumenetelmät käyttävät hyödykseen funktion kvasikonveksisuutta tai pseudokonveksisuutta.

3.1 Osittaisderivaatta ja gradientti

Funktion osittaisderivaatta on funktio derivoituna yhden sen muuttujan suhteen. Usean muuttujan differentoituvalla funktiolla on derivaatta sen jokaiselle muuttujalle. Olkoon $f : \mathbb{R}^n \to \mathbb{R}$. Derivoimalla jokainen muuttuja erikseen saadaan funktion osittaisderivaatta, joka merkitään

$$\frac{\partial f(x_1, \ldots, x_n)}{\partial x_i}.$$

Näitä osittaisderivaattoja voidaan merkitä gradientilla $\nabla f(x_1, \ldots, x_n)$.

Esimerkki 3.1. Olkoon $f(x_1, x_2, x_3) = x_1^3 + 2x_2x_1 - 3x_3^2$. Tällöin sen gradientti on

$$\nabla f(x_1, x_2, x_3) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{pmatrix} = \begin{pmatrix} 3x_1^2 + 2x_2 \\ 2x_1 \\ -6x_3 \end{pmatrix}.$$

3.2 Konveksisuus

Funktion ollessa konveksi, sen tangentit eivät leikkaa käyrän missään kohdassa. Olkoon funktio $f : S \to \mathbb{R}$ ja $S \neq \emptyset$ on konveksi avaruuden $\mathbb{R}$ osajoukko. Funktio f on konveksi, jos [14]

$$f(\lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda) f(\mathbf{x}_2)\ \forall \mathbf{x}_1,\ \mathbf{x}_2 \in S,\ \lambda \in (0, 1).$$

Vastaavasti funktio f on aidosti konveksi, jos

$$f(\lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) < \lambda f(\mathbf{x}_1) + (1 - \lambda) f(_2)\ \forall \mathbf{x}_1,\ \mathbf{x}_2 \in S,\ \mathbf{x}_1 \neq \mathbf{x}_2,\ \lambda \in (0, 1).$$

Funktio f on (aidosti) konkaavi, jos $-f$ on (aidosti) konveksi. [14]

Lause 3.2. Jos funktio on konveksi ja konkaavi samaan aikaan, se on lineaarinen tai tarkemmin affiini.

Todistus. Konveksille funktiolle pätee

$$f((\lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \leq \lambda f(\mathbf{x}_1) + (1 - \lambda) f(\mathbf{x}_2).$$

Vastaavasti konkaaville funktiolle pätee

$$f((\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \geq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2).$$

Tästä seuraa, että nämä yhdistämällä saadaan

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) = \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

eli funktio on affiini. □

Kvasikonveksin funktion arvot kahden pisteen välisellä janalla eivät ylitä sen arvoja päätepisteissä. Olkoon edelleen funktio $f : S \to \mathbb{R}$ ja $S \neq \emptyset$ on konveksi avaruuden $\mathbb{R}$ osajoukko. Tällöin funktio f on kvasikonveksi, jos [14]

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \max\{f(\mathbf{x}_1), f(\mathbf{x}_2)\} \; \forall \; \mathbf{x}_1, \; \mathbf{x}_2 \in S, \; \lambda \in (0,1).$$

Vastaavasti funktio f on aidosti kvasikonveksi, jos

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) < \max\{f(\mathbf{x}_1), f(\mathbf{x}_2)\} \; \forall \; \mathbf{x}_1, \; \mathbf{x}_2 \in S, \; \mathbf{x}_1 \neq \mathbf{x}_2, \; \lambda \in (0,1).$$

Funktio f on (aidosti) kvasikonkaavi, jos funktio $-f$ on (aidosti) kvasikonveksi. Konveksit funktiot ovat aina kvasikonvekseja ja aidosti kvasikonvekseja. Kvasikonveksi funktio ei ole välttämättä jatkuva, eikä kvasikonveksien funktioiden summa ole välttämättä kvasikonveksi. Aidosti kvasikonveksin funktion lokaalit minimit ovat globaaleja minimejä. [14] [16]

Funktion pseudokonveksisuus edellyttää differentioituvuutta. Pseudokonveksi funktio käyttäytyy samoin kuin konveksi funktio lokaalin minimin löytymisen suhteen. Olkoon funktio $f : S \to \mathbb{R}$ ja $S \neq \emptyset$ on konveksi avaruuden $\mathbb{R}$ osajoukko. Funktio f on pseudokonveksi, jos

$$\nabla f(\mathbf{x}_1)^T(\mathbf{x}_2 - \mathbf{x}_1) \geq 0 \Longrightarrow f(\mathbf{x}_2) \geq f(\mathbf{x}_1), \; \forall \; \mathbf{x}_1, \mathbf{x}_2 \in S$$

tai

$$f(\mathbf{x}_2) < f(\mathbf{x}_1) \Longrightarrow \nabla f(\mathbf{x}_1)^T(\mathbf{x}_2 - \mathbf{x}_1) < 0, \; \forall \; \mathbf{x}_1, \mathbf{x}_2 \in S.$$

Aidosti pseudokonveksille funktiolle pätee

$$\nabla f(\mathbf{x}_1)^T(\mathbf{x}_2 - \mathbf{x}_1) \geq 0 \Longrightarrow f(\mathbf{x}_2) > f(\mathbf{x}_1), \; \forall \; \mathbf{x}_1, \mathbf{x}_2 \in S, \; \mathbf{x}_1 \neq \mathbf{x}_2$$

tai

$$f(\mathbf{x}_2) \leq f(\mathbf{x}_1) \Longrightarrow \nabla f(\mathbf{x}_1)^T(\mathbf{x}_2 - \mathbf{x}_1) < 0, \; \forall \; \mathbf{x}_1, \mathbf{x}_2 \in S, \; \mathbf{x}_1 \neq \mathbf{x}_2.$$

Pseudokonveksi funktio on aina kvasikonveksi ja aidosti kvasikonveksi. Jokainen konveksi funktio on aina pseudokonveksi, mutta tämä ei välttämättä päde toisinpäin. [14]

3.3 Ääriarvot

Funktiolla $f : \mathbb{R}^n \to \mathbb{R}$ on lokaali ääriarvo pisteessä $\mathbf{x}_0$, jos vektorin $\mathbf{x}_0$ läheisyydessä pätee

$$f(\mathbf{x}_0) \left\{ \begin{matrix} \leq \\ \geq \end{matrix} \right\} f(\mathbf{x}) \; \forall \; \mathbf{x} \; \neq \; \mathbf{x}_0.$$

Ääriarvo voidaan selvittää esimerkiksi derivoimalla differentoituva funktio. Yksiulotteisessa tapauksessa ääriarvo löytyy yleensä, kun $\frac{d}{dx} f(x) = 0$. Jos derivaatan $f'(x)$ arvo on ääriarvon vasemmalla puolella positiivinen ja oikealla negatiivinen, kyseessä on lokaali maksimi. Jos taas derivaatan $f'(x)$ arvo on vasemmalla negatiivinen ja oikealla positiivinen, kyseessä on lokaali minimi. [13]

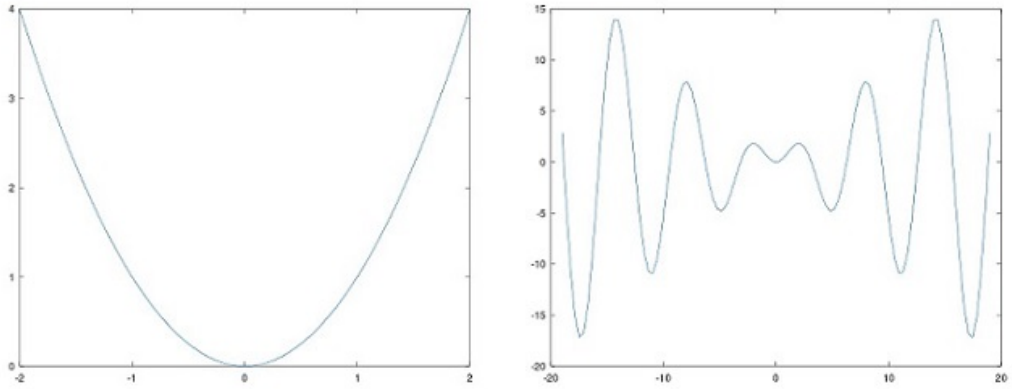
Jos lokaali minimi tai maksimi on sama koko funktion määrittelyjoukossa, kyseessä on tällöin globaali minimi tai maksimi. Funktiolla ei kuitenkaan aina ole ääriarvoja, esim. $f(x) = x$ tai $f(x) = x^3$. Jos $\lim_{x \to \pm\infty} f(x) = \pm\infty$, ei funktiolla ole globaaleja ääriarvoja. Funktiolla voi kuitenkin myös olla vain toinen globaali ääriarvo. Kuvassa 3 vasemmalla on funktion $f(x) = x^2$ globaali minimi. Tämän globaali minimarvo on $f(0) = 0$. Funktiolla $f(x) = x^2$ ei kuitenkaan ole globaalia maksimia. Oikealla kuvassa 3 on esimerkkinä useammasta lokaalista ääriarvosta funktio $f(x) = \sin(x) \cdot x$. Tällä funktiolla ei ole lainkaan globaaleja ääriarvoja, koska funktio hajaantuu.

Differentoituvalle usean muuttujan funktiolle voidaan löytää ääriarvo, kun sen gradientti on nollavektori

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right) = \mathbf{0}.$$

Gradientti pisteessä $\mathbf{x}$ voi olla nollavektori myös satulapisteessä tai lokaalissa ääriarvossa, eikä vain globaalissa ääriarvossa. Jos differentoituva funktio on lisäksi konveksi tai pseudokonveksi, gradientin nollakohta on funktion globaali minimi. Usean muuttujan funktion konveksisuus voidaan selvittää Hessen matriisin avulla, jos funktio on vähintään kahdesti differentoituva. Hessen matriisi muodostuu funktion toisista osittaisderivaatoista [14]

$$H(\mathbf{x}) \; = \; \nabla^2 f(\mathbf{x}).$$



Kuva 3: Globaali ääriarvo (vasemmalla) ja lokaalit ääriarvot(oikealla).

Kahden muuttujan funktiolle $f(x_1, x_2)$ tämä olisi

$$H(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix}.$$

Hessen matriisin täytyy olla positiivisemidefiniitti, jotta funktio olisi konveksi eli [14]

$$\det(H(\mathbf{x})) \geq 0.$$

Epäkonveksin funktion ääriarvojen löytäminen ei ole yhtä yksinkertaista. Sen ääriarvoja voidaan yrittää etsiä myöhemmin esiteltävällä gradienttimenetelmällä, mutta gradientin nollakohdan löytyminen ei vielä takaa globaalin ääriarvon löytymistä.

3.4 Optimointi

Matemaattisen optimoinnin perusajatus on hyvin yksinkertainen: halutaan minimoida tai maksimoida kohdefunktiota. Hyviä esimerkkejä optimoinnista ovat kauppalistan kirjoittaminen tuotteiden osastojen mukaan; suunnitelma, missä järjestyksessä hoitaa aamutoimet tai nopein työmatkan reitti.

Klassinen optimointiesimerkki on kauppamatkustajaongelma, jossa kauppiaan tulee kulkea tiettyjen kaupunkien läpi ja muodostaa reitti näiden kaupunkien välille. Kauppias palaa alkupisteeseen ja tavoite on minimoida kuljettu matka.

Kauppamatkustajaongelma voidaan kuvata verkkona, joka koostuu solmuista ja kaarista. Solmut ovat joukon alkioita, jotka kuvaavat esimerkiksi eri kaupunkeja. Solmuja yhdistää kaaret, joilla voidaan kuvata esimerkiksi välimatkoja. Kaarilla on aina paino. Solmut ja kaaret virittävät yhdessä verkon, kun taas joukko sisältää vain solmut. Verkko voi olla suunnattu, jolloin kaarten paino on positiivinen yhteen suuntaan ja toiseen negatiivinen tai sitten suuntaamaton, jolloin kaarten painot ovat aina positiivisia riippumatta siitä mihin suuntaan niitä kuljetaan.

Tavoitteena on muodostaa solmujen välille reitti, joka sisältää tarvittavat solmut ja täyttää tehtävänannon. Reitin kokonaispainon tulisi olla mahdollisimman suuri tai mahdollisimman pieni. Suurta painoa tavoitellaan esimerkiksi liikevoitoissa ja pientä taas esimerkiksi polttoaineen kulutuksessa.

Epälineaarinen optimointitehtävä voidaan kirjoittaa muodossa [15]

$$\begin{Bmatrix} \max \\ \min \end{Bmatrix} f(x_1, \ldots, x_n)$$

$$s.t.\ q_i(x_1, \ldots, x_n) \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} z_i,\ i = 1, \ldots, m.$$

Tässä kohdefunktio f ja rajoitefunktiot q_i ovat muuttujien x_j funktioita ja vakiot z_i ovat annettuja parametreja. Funktioista f ja q_i ainakin yhden tulee olla epälineaarinen.

Rajoitteinen optimointi voidaan myös nähdä Lagrangen funktion minimointitehtävänä. Lagrangen funktiossa alkuperäisestä kohdefunktiosta vähennetään rajoitusten ja niitä vastaavien Lagrangen kertoimien muodostamat termit. Lagrangen

funktiolla on satulapiste alkuperäisen tehtävän ratkaisupisteessä. Yleensä Lagrangen funktio on tosin vain teoreettinen väline, jotta duaalinen kohdefunktio saadaan derivoitua. [1]

Lagrangen funktio voidaan kirjoittaa muodossa

$$L(x_1, x_2, \ldots, x_n, \lambda) = f(x_1, x_2, \ldots, x_n) + \sum_{i=1}^{m} \lambda_i q_i(x_1, x_2, \ldots, x_n),$$

missä vakiot $\lambda_i$ovat Lagrangen kertoimia. Jos sekä kohdefunktio f, että rajoitefunktiot q_i ovat differentoituvia, niin Lagrangen funktion satulapiste $(x_1^*, x_2^*, \ldots, x_n^*)$ löytyy sen gradientin nollakohdasta

$$\nabla L(x_1^*, x_2^*, \ldots, x_n^*, \lambda) = \mathbf{0}.$$

[14]

3.5 Gradienttimenetelmät

Gradienttimenetelmissä käytetään hyödyksi funktion osittaisderivaattoja. Tavoitteena on löytää minimi kohdefunktiolle. Tässä työssä esitellään moniulotteinen gradienttimenetelmä ja kaksi vaihtoehtoa siinä käytettävälle viivahaku-menetelmälle.

Gradienttimenetelmissä etsitään funktion minimi tai maksimi käyttäen funktion derivaattaa apuna. Funktion tulee olla jatkuvasti derivoituva, jotta näitä menetelmiä voidaan käyttää. Funktion muuttujien määrää ei ole mitenkään rajattu. Gradienttimenetelmiä hyödynnetään yleensä esimerkiksi neuroverkkojen opettamisessa. Gradienttimenetelmien avulla voidaan muokata neuroverkon parametreja siten, että neuroverkko tuottaa vähemmän virheitä.[18]

Nopeimman laskeutumisen menetelmässä etsitään funktion f minimiä nimensä mukaisesti suunnasta, johon funktion arvo vähenevät kaikkein nopeimmin. Tässä menetelmässä tavoitteena on muodostaa jono, jossa $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$, kun $k = 1, 2, \ldots$. [27]

Askel 1 Valitaan ensin jokin arvo $\epsilon > 0$ ja lähtöpiste $\mathbf{x}_1$. Jos normi $||\nabla f(\mathbf{x}_1)|| < \epsilon$ voidaan lopettaa. Jos ei, siirrytään askeleeseen 2.

Askel 2 Valitaan suunnaksi

$$\mathbf{d}_k = -\frac{\nabla f(\mathbf{x}_k)}{||\nabla f(\mathbf{x}_k)||}$$

ja määritetään esim. viivahaulla funktion $\phi(\lambda) = f(\mathbf{x}_k + \lambda \mathbf{d}_k)$ minimi, kun $\lambda > 0$. Käytetyllä menetelmällä saadaan tulos λ_k. Muuttujan λ_k avulla saadaan uusi piste $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$. Palataan askeleeseen 1 nyt arvolla $\mathbf{x}_{k+1}$.

Samaa algoritmia voi myös käyttää funktion maksimin löytämiseen, mutta tällöin gradientin suunta $\mathbf{d}_k = \frac{\nabla f(\mathbf{x}_k)}{||\nabla f(\mathbf{x}_k)||}$ on positiivinen. [16] Gradientin normin käyttäminen nopeimman laskeutumisen menetelmässä tekee siitä stabiilimman, mutta se ei ole välttämätöntä.

Tästä selkeästi nähdään, että usean muuttujan gradienttimenetelmässä lasketaan ∇f monta kertaa ja liikutaan päinvastaiseen suuntaan. Tällöin päästään putoamaan funktion laaksoon ja saavutetaan globaali minimi. [18] Optimointitehtävän kohdefunktiona voi myös olla virhefunktio, esimerkiksi mallintamisessa ja neuroverkon opettamisessa tavoitteena on minimoida mallin tuomaa virhettä.

3.5.1 Viivahaku

Viivahaku on tärkeä osa moniulotteista optimointia. Viivahaussa kohdefunktiona on [16]

$$\min_{\lambda} \ \phi(\mathbf{x}_k + \lambda \mathbf{d}_k).$$

Funktion minimiä voitaisiin etsiä sen derivaatan nollakohdasta. Useampi ulotteisella funktiolla tällöin kuitenkin tarvittaisiin gradienttia ja funktion ollessa epälineaarinen minimin löytäminen derivoinnin avulla voi olla haastavaa. Funktion $\phi(\lambda)$ minimille määritellään epävarmuusväli $a \leq \lambda \leq b$. Epävarmuusvälin päätepisteiksi valitaan pisteet, joiden derivaatat ovat eriarvoiset. Tällöin funktiolla on ääriarvo välin sisällä. Funktion minimi etsitään lyhentämällä epävarmuusväliä jakopisteiden avulla. Jakopiste on piste epävarmuusvälin sisällä. Näitä lasketaan yksi tai kaksi jokaiselle iteraatiokierroksella riippuen menetelmästä ja muodostetaan uusi epävarmuusväli saatujen jakopisteen/-pisteiden avulla.[16]

Seuraavaksi esitellään kaksi erilaista eliminointimenetelmää eli menetelmiä, joiden tavoite on pienentää epävarmuusväliä, jolla funktion minimi sijaitsee.

Viivahaussa voidaan käyttää suoria menetelmiä, jolloin gradienttia ei tarvita. Tehokkain suora menetelmä on Fibonaccin menetelmä, jossa toinen epävarmuusvälin jakopiste jatkaa aina seuraavalle kierrokselle. Jokaisella kierroksella tarvitsee siis laskea vain yksi uusi arvo. Fibonaccin menetelmä käyttää hyödykseen Fibonaccin lukuja F_n

$$F_1 = 1, \ F_2 = 1,$$

$$F_{n+1} = F_n + F_{n-1}, \ n = 1, 2, \ldots$$

Fibonaccin menetelmässä valitaan ensin lopullisen epävarmuusvälin pituus $L > 0$ ja vakio $\epsilon > 0$ siten, että funktion arvot ovat vielä erotettavissa tällä epävarmuusvälillä ja vakiolla. Valitaan luku n siten, että $F_n > \frac{1}{L}(b_1 - a_1)$. Aloitusepävarmuusväli on $[a_1, b_1]$ ja ensimmäiset jakopisteet lasketaan kaavoilla

$$\lambda_1 = a_1 + \frac{F_{n-2}}{F_n}(b_1 - a_1)$$

$$\mu_1 = a_1 + \frac{F_{n-1}}{F_n}(b_1 - a_1).$$

Tämän jälkeen lasketaan arvot $\phi(\lambda_1)$ ja $\phi(\mu_1)$. [16]

Jos iteraatiokierroksella k $\phi(\lambda_k) > \phi(\mu_k)$, voidaan aidon kvasikonveksisuuden nojalla [16] valita uudeksi epävarmuusväliksi $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$ ja $\lambda_{k+1} = \mu_k$. Jos $k = n-2$ siirrytään viimeiseen vaiheeseen ja määritellään lopullinen epävarmuusväli seuraavanlaisesti

$$\lambda_{n-1} = \mu_{n-2} \text{ ja } \mu_{n-1} = \mu_{n-2} + \epsilon.$$

Jos

$$\phi(\lambda_{n-1}) > \phi(\mu_{n-1}), \text{ niin } a_n = \lambda_{n-1} \text{ ja } b_n = b_{n-1}.$$

Muussa tapauksessa lasketaan

$$\mu_{k+1} = a_{k+1} + \frac{F_{n-k-1}}{F_{n-k}}(b_{k+1} - a_{k+1})$$

ja tämän avulla lasketaan $\phi(\mu_{k+1})$. [16]

Toisin, jos $\phi(\lambda_k) \leq \phi(\mu_k)$ valitaan uudeksi epävarmuusväliksi $a_{k+1} = a_k$, $b_{k+1} = \mu_k$ ja $\mu_{k+1} = \lambda_k$. Jos $k = n - 2$ siirrytään viimeiseen vaiheeseen ja määritellään lopullinen epävarmuusväli seuraavanlaisesti [16]

$$\lambda_{n-1} = \lambda_{n-2} \text{ ja } \mu_{n-1} = \lambda_{n-2} + \epsilon.$$

Jos

$$\phi(\lambda_{n-1}) \leq \phi(\mu_{n-1}), \text{ niin } a_n = a_{n-1} \text{ ja } b_n = \mu_{n-1}.$$

Muussa tapauksessa lasketaan

$$\lambda_{k+1} = a_{k+1} + \frac{F_{n-k-2}}{F_{n-k}}(b_{k+1} - a_{k+1})$$

ja tämän avulla lasketaan $\phi(\lambda_{k+1})$.

Uusien arvojen λ_{k+1} ja μ_{k+1} laskemisen jälkeen, asetetaan $k \leftarrow k+1$ ja palataan edelliseen vaiheeseen. Kun ollaan saavutettu iteraatiokierros $k = n - 2$, saadaan lopullinen epävarmuusväli $[a_n, b_n]$ seuraavasti: [16]

$$\text{Jos } \phi(\lambda_{n-1}) \leq \phi(\mu_{n-1}), \text{ niin } a_n = \lambda_{n-1} \text{ ja } b_n = b_{n-1}.$$

$$\text{Jos } \phi(\mu_{n-1}) \leq \phi(\lambda_{n-1}), \text{ niin } a_n = a_{n-1} \text{ ja } b_n = \mu_{n-1}.$$

Viivahaussa voidaan myös käyttää hyödyksi derivaattaa. Bisektiomenetelmä etsii funktion ϕ minimin suljetulta välillä. Funktion tulee olla pseudokonveksi ja siis differentoituva. Bisektiomenetelmässä aloitetaan epävarmuusvälillä $[a_k, b_k]$ ja lasketaan funktion ϕ derivaatan arvo pisteessä λ_k. Pisteeksi λ_k kannattaa valita $\lambda_k = \frac{1}{2}(a_k + b_k)$.

Jos $\phi'(\lambda_k) = 0$, kyseessä on globaali minimipiste.

Muulloin valitaan uusi epävarmuusväli ja lasketaan funktion ϕ derivaatan arvo uuden epävarmuusvälin pisteessä. Uusi epävarmuusväli on $[a_k, \lambda_k$, jos $\phi'(\lambda_k) > 0$.

Jos taas $\phi'(\lambda_k) < 0$ uusi epävarmuusväli on $[\lambda_k, b_k]$. Tätä voidaan iteroida niin kauan, kunnes epävarmuusväli on tarpeeksi pieni. [16]

3.6 Stokastinen gradienttimenetelmä

Neuroverkon opettamisessa ratkaistaan laajoja epäkonvekseja optimointitehtäviä. Optimointitehtävissä on yleensä riskinä saavuttaa vain lokaali ääriarvo, eikä globaalia. Neuroverkon ohjaamaton opettaminen ei tarjoa välttämättä tarpeeksi hyvää pohjaa monimutkaisempien ongelmien ratkaisemiseen. Nykyään kuitenkin neuroverkoille saadaan hyvin pieni virhe monimutkaisissa tehtävissä, vaikka opettamiseen olisi käytetty vain stokastisia gradienttimenetelmiä. [10]

Stokastinen gradienttimenetelmä käyttää vain osaa q opetusaineiston syötteistä laskiessaan virhefunktion C gradienttia. Näin saadaan muodostettua nopeammin estimaatti ∇C_q koko opetusaineiston todelliselle gradientille ∇C, joka nopeuttaa gradienttimenetelmän suorittamista ja samalla neuroverkon oppimista. Esimerkiksi 50 000 syötteen opetusaineistosta voidaan valita $q = 10$ ja laskenta nopeutuu 5000-kertaiseksi. Näin pienellä otoskoolla ei saada kovinkaan tarkkaa estimaattia, mutta estimaatin avulla saadaan suunta, jonne virhefunktion arvo pienenee. [18]

Stokastinen gradienttimenetelmä sisältää enemmän heilumista, mutta saattaa päätyä nopeammin lokaaliin minimiin. Gradienttimenetelmässä taas koko neuroverkon arvot ovat muistissa, joka kuormittaa konetta huomattavasti. Neuroverkon oppiminen saattaa olla hitaampaa gradienttimenetelmällä. [8]

4 Vastavirta-algoritmi

Vastavirta-algoritmi on neuroverkon opettamisessa käytettävä menetelmä, joka hyödyntää differentiaalilaskentaa neuroverkon painojen optimoinnissa. Sen perusajatus on hyvin yksinkertainen. Vastavirta-algoritmi käyttää aiemmin esiteltyjä gradienttimenetelmiä etsiäkseen optimiarvot neuroverkon neuroneiden välisille painoille. [19]

4.1 Toimintaperiaate

Vastavirta-algoritmin suorittamisessa käytetään hyväksi yleisiä vektoreiden laskusääntöjä, kuten yhteenlasku ja vektorin kertominen matriisilla. Menetelmässä saadaan selville myös miten virhefunktio C käyttäytyy neuroverkon painojen tai vakiotermien muutoksen suhteen. Vastavirta-algoritmissa neuroverkon painoja tai vakiotermejä parannetaan osittaisderivaattojen

$$\frac{\partial C}{\partial w} \text{ tai } \frac{\partial C}{\partial b}$$

avulla. [18] Vastavirta-algoritmi ei siis ole vain nopeampi tapa opettaa neuroverkkoja vaan se kertoo myös paljon neuroverkon reagoimisesta muutokseen.

Vastavirta-algoritmin nimi tulee siitä, että virheen laskeminen ja uusien painojen päivittäminen neuroverkkoon aloitetaan ulostulokerroksesta ja edetään neuroverkon alkuun syötekerrokseen. [12] Vastavirta-algoritmin avulla neuroverkon painot saadaan muokattua siten, että ulostulokerroksen antama ratkaisu on mahdollisimman lähellä haluttua ratkaisua. [19] Takaisin alkuun iteroivan laskennan avulla voidaan tarkastella jokaisen kerroksen gradientteja tehokkaasti verrattuna siihen, että ne laskettaisiin kaikki erikseen huomioimatta neuroverkon kerroksien riippuvuussuhteita. [12]

Vastavirta-algoritmi esiteltiin ensimmäisen kerran jo 1970-luvulla, mutta vasta vuonna 1986 ymmärrettiin sen hyödyntämismahdollisuudet. Tällöin David Rumelhart, Geoffrey Hinton ja Ronald Williams esittelivät artikkelin "Learning Represetations by Back-Propagation Errors", jossa esiteltiin vastavirta-algoritmi yksinkertaisena menetelmänä neuroverkon opettamisessa. Vastavirta-algoritmi käyttää artikkelin mukaan vain ensimmäisiä derivaattoja, eikä siksi konvergoi yhtä nopeasti kuin menetelmät, joissa hyödynnetään myös toista derivaattaa. Se on kuitenkin huomattavasti helpompi laskennallisesti ja sen tehokkuutta voidaan parantaa gradienttimenetelmillä. Vastavirta-algoritmi oli merkittävästi nopeampi tapa opettaa neuroverkot kuin aikaisemmat menetelmät ja sen avulla voitiin ratkaista täysin uudenlaisia tehtäviä. [4] [6] [12] [18] Vastavirta-algoritmilla on mahdollista saada hyviä tuloksia kustannustehokkaasti.

Vastavirta-algoritmi on ohjatun oppimisen menetelmä, koska opetusaineistosta tiedetään tavoiteulostulo. Jos tavoiteulostuloa ei tiedetä, vastavirta-algoritmi ei ole toimiva menetelmä, koska tällöin virhefunktion arvoa ei voida laskea. Vastavirta-algoritmin käyttämista varten tulee tietää myös neuroverkon rakenne sekä määritellä käytettävä virhefunktio. [12] Virhefunktio voi vaihdella, mutta klassisessa vastavirta-algoritmissa se on keskineliövirhe. Neuroverkon rakenteen tulee olla eteenpäin kytketty, jotta virheiden riippuvuussuhde säilyy oikeanlaisena vastavirta-algoritmia varten.

Vastavirta-algoritmi on edelleen neuroverkkojen opettamisen selkäranka, koska sen avulla virhefunktion gradientti voidaan laskea tehokkaasti. [18] Gradientin avulla voidaan minimoida virhefunktio, mikä on neuroverkkojen opettamisen tavoite. [19]

Vastavirta-algoritmi on suoraan käytettävissä erilaisissa syväoppimisen kirjastoissa, eikä käyttäjän tällöin tarvitse itse laskea osittaisderivaattoja tai uusia painoja. On kuitenkin tärkeää ymmärtää vastavirta-algoritmin toimintaperiaatteet, jotta osaa käyttää sitä oikeissa tilanteissa. [19]

Vastavirta-algoritmin käyttö ei aina takaa globaalin minimin löytymistä. Ensinnäkin konvergenssi voi olla niin hidasta, ettei globaalia minimiä saavuteta äärellisessä ajassa. Toisena ongelmana voi olla lokaaliin minimiin juuttuminen. Vastavirta-algoritmi ei osaa vältellä lokaalin minimin laaksoja, joten löydetyt ääriarvot eivät välttämättä ole globaaleja. Vastavirta-algoritmi toimii hyvin yksinkertaisilla konvekseilla funktioilla, joilla on vain yksi minimi, mutta saattaa päätyä lokaaliin minimiin epäkonvekseilla tai monimutkaisemmilla funktioilla. [4]

Vastavirta-algoritmin tavoitteena on löytää oikeat painot w_{ji}^k ja vakiotermit b_i^k neuroverkon piilokerroksille. Näitä parametreja on haastava määritellä kerralla oikein, koska piilokerroksien neuroneilla ei ole mitään ennalta määriteltyjä tavoiteulostuloja. Neuronin virhe on aina riippuvainen edeltävien kerroksien neuroneiden virheistä ja vaikuttaa seuraavien kerroksien neuroneiden virheeseen. Näiden ratkaiseminen voi olla hyvinkin vaativaa, jos neuroverkossa on useampi piilokerros. [12]

Vastavirta-algoritmissa käytetään yleensä virhefunktiona C keskineliövirhettä, joka on esitetty kaavassa (1). [12] Virhefunktion minimointiin käytetään gradienttimenetelmiä, joita esiteltiin aiemmin kappaleessa 3.6. Tavoitteena on saada virhefunktion C arvo mahdollisimman lähelle nollaa eli löytää sen globaali minimi, jotta virhefunktio (1) $C(\mathbf{X},\ \theta) \to 0$. [18]

Vastavirta-algoritmin virheiden laskeminen on hyvin samankaltaista kuin neuroverkon eteenpäin menevä ulostulon laskeminen. Ennen vastavirta-algoritmin käyttämistä pitää laskea kaikki neuroneiden a_i^k ja neuroverkon ulostulojen o_i^k arvot. Nämä lasketaan jokaiselle iteraatiokierroksella uudestaan ennen vastavirta-algoritmin hyödyntämistä. Neuroverkko tulee siis kulkea läpi aina ensin etuperin, jotta sen voi kulkea läpi takaperin. [12]

Kun vastavirta-algoritmilla ollaan saavutettu alkusolmu ja kaikki osittaisderivaatat ovat selvillä, voidaan neuroverkon painot päivittää uusiksi gradienttimenetelmän avulla. Seuraavaksi kuljetaan neuroverkko eteenpäin ja saadaan uusi ulostulo. Tästä lasketaan taas virhetermit ja iteroidaan vastavirta-algoritmilla alkuun. Tätä toistetaan, kunnes löydetään globaali minimi tai konvergenssiehto täyttyy eli virhe on tarpeeksi pieni. [12]

Neuroverkon opettaminen gradienttimenetelmillä vaatii virhefunktion gradientin laskemista painojen ja vakiotermien suhteen. Oppimisvauhti α vaikuttaa siihen, miten painot muuttuvat jokaisen iteraation jälkeen. Se vastaa askelpituutta λ, joka voidaan määrittää luvussa 3.5.1 kuvatuilla viivahakumenetelmillä. Parametrit muuttuvat seuraavasti jokaisella iteraatiokierroksella [12]

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial C(\mathbf{X},\ \theta^t)}{\partial \theta},$$

jossa θ^t tarkoittaa iteraatiokierroksessa t käytettyjä parametreja. [12]

4.2 Yksi syöte-ulostulo -pari neuroverkossa

Vastavirta-algoritmin derivointi on hyvin yksinkertaista ketjusäännön ja tulosäännön avulla. Näitä sääntöjä voidaan hyödyntää vain jos aktivointifunktio on differentoituva. [12]

Yksinkertaistetaan esitystä vielä hieman lisää ja lasketaan vakiotermi b_i^k mukaan painoihin w_{0i}^k ja määritetään ulostulo $o_0^{k-1} = 1$ neuronille a_i^k kerroksessa $k-1$. Tästä seuraa, että $w_{0i}^k = b_i^k$. Tämä on yhtenevä alkuperäiseen muotoiluun, koska [12]

$$a_i^k = b_i^k + \sum_{j=1}^{r^{k-1}} w_{ji}^k o_j^{k-1} = w_{0i}^k + \sum_{j=1}^{r^{k-1}} w_{ji}^k o_j^{k-1} = \sum_{j=0}^{r^{k-1}} w_{ji}^k o_j^{k-1}.$$

Aikaisemmin mainitun keskineliövirhefunktion yleisyyden perusteella minimoidaan virhefunktiota

$$C(\mathbf{X}, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2$$

neuroverkon painojen suhteen. Jokaiselle painolle w_{ji}^k lasketaan $\frac{\partial C}{\partial w_{ji}^k}$. Virhefunktion avulla jokainen virhetermi voidaan ketjusäännön nojalla laskea jokaiselle syöte-ulostulo -parille erikseen. Osittaisderivaatta lasketaan jokaiselle virhetermille erikseen ja sitten summataan virhetermit yhteen lopuksi. Tällöin ylläolevien merkintöjen mukaisesti saadaan [12]

$$\frac{\partial C(\mathbf{X}, \theta)}{\partial w_{ji}^k} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial w_{ji}^k} \Big(\frac{1}{2} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \Big) = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial C_d}{\partial w_{ji}^k}.$$

Osittaisderivoinnin helpottamisen takia keskitytään yhteen syöte-ulostulo -pariin kerrallaan. Kun tämä on kerran derivoitu, voidaan muodostaa yleinen kaava kaikille syöte-ulostulo -pareille joukossa $\mathbf{X}$. Yksinkertaistetuksi virhefunktioksi saadaan $C = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^2$. [12]

4.3 Yleistetty vastavirta-algoritmin virhefunktion derivointi

Aloitetaan vastavirta-algoritmin virhefunktion C derivointi soveltamalla ketjusääntöä virhefunktion osittaisderivaattaan [12]

$$\frac{\partial C}{\partial w_{ji}^k} = \frac{\partial C}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ji}^k},$$

missä a_j^k on kerroksen k neuroni j ennen aktivointifunktion käyttöä neuronin ulostulon laskemiseen. Tästä nähdään, että muutos virhefunktiossa C painon w_{ji}^k suhteen on sama asia kuin muutos virhefunktiossa C neuronin a_j^k suhteen kerrottuna neuronin a_j^k arvon muutoksella painon w_{ji}^k suhteen. [12]

Ensimmäistä termiä merkitään yleensä virheenä $\delta_j^k = \frac{\partial C}{\partial a_j^k}$ ja toinen termi voidaan laskea [12]

$$\frac{\partial a_j^k}{\partial w_{ji}^k} = \frac{\partial}{\partial w_{ji}^k} \left(\sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1} \right) = \left(\sum_{j=0}^{r_{k-1}} \frac{\partial w_{ji}^k}{\partial w_{ji}^k} o_j^{k-1} \right) = o_j^{k-1}.$$

Tästä seuraa, että

$$\frac{\partial C}{\partial w_{ji}^k} = \delta_j^k o_j^{k-1}.$$

Nämä ovat yleiset kaavat riippumatta aktivointi- tai virhefunktiosta. Kuitenkin virhetermi δ_j^k on riippuvainen virhefunktiosta, joten se täytyy laskea aina uudestaan. Vastavirta-algoritmin aktivointifunktiona käytetään yleensä sigmoid-funktiota sen differentoituvuuden helppouden takia. [12]

4.4 Neuroverkon kerroksen k virhetermi

Merkitään neuroverkon kerroksen k neuronin i aiheuttamaa virhettä δ_i^k. Vastavirta-algoritmissa virhe on riippuvainen seuraavan kerroksen $k+1$ virheestä δ_i^{k+1}. Toisin sanoen virheet vaikuttavat takaperin viimeisestä kerroksesta ensimmäiseen. Ensimmäinen virhe lasketaan neuroverkon ulostulon ja halutun ulostulon erotuksena. Edeltävän kerroksen $k-1$ virheet lasketaan painojen w_{ji}^k painotettuna tulona ja se skaalataan aktivointifunktion derivaatalla $g'(a_i^k)$. Tätä jatketaan kunnes myös syötekerroksen virhetermit on laskettu. [12] Virhetermin suuruudesta näkee, kuinka paljon yksittäinen neuroni on vastuussa neuroverkon kokonaisvirheestä. [3]

Määritellään ensin ulostulokerroksen virhetermi sellaiselle neuroverkolle, jonka ulostulokerroksessa on vain yksi neuroni. Vastavirta-algoritmin tavoitteena on määritellä virhetermin δ_1^k arvo. Valitaan $j = 1$ ja olkoon $g(a_1^k)$ aktivointifunktio neuroverkon piilokerroksille ja $g_o(a_1^k)$ aktivointifunktio neuroverkon ulostulokerrokselle. Virhefunktiolle saadaan [12]

$$C(\mathbf{X}, \theta) = \frac{1}{2}(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 = \frac{1}{2}(g_o(a_1^k) - \mathbf{y}_i)^2.$$

js ketjusäännön nojalla saadaan virhetermille δ_1^k [12]

$$\delta_1^k = (g_o(a_1^k) - \mathbf{y}_i)^2 g_o'(a_1^k) = (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 g_o'(a_1^k).$$

Kun edellä mainitut yhdistetään saadaan virhefunktion C osittaisderivaataksi painon w_{i1}^k suhteen [12]

$$\frac{\partial C}{\partial w_{i1}^k} = \delta_1^k o_i^{(k-1)} = (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 g_o'(a_1^k) o_i^{(k-1)}.$$

Ulostulokerroksen virhetermin laskemisen jälkeen voidaan vastavirta-algoritmin nimen mukaisesti siirtyä taaksepäin neuroverkossa ja katsoa piilokerrosten virhetermejä. Niiden laskeminen on hieman monimutkaisempaa kuin ulostulokerroksen, koska piilokerroksessa voi olla useampi neuroni. Ketjusäännön avulla se on kuitenkin laskettavissa. [12]

Virhetermille δ_j^k, jossa $1 \leq k < m$ ja jossa m on neuroverkon kerrosten määrä, saadaan

$$\delta_j^k = \frac{\partial C}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial C}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k},$$

missä l käy läpi kerroksen $k+1$ neuronit $1, \ldots, r^{k+1}$. Vakiotermi o_0^k joka liittyy painoon w_{0j}^{k+1} on kiinnitetty, joten se ei riipu edeltävien kerroksien ulostuloista. Tällöin l ei saa koskaan arvoa 0. [12]

Koska $\delta_j^k = \frac{\partial C}{\partial a_j^k}$, niin ylläoleva funktio voidaan esittää muodossa [12]

$$\delta_j^k = \frac{\partial C}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial C}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k} = \sum_{j=1}^{r^{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}.$$

Tästä neuronin a_l^{k+1} määritelmän nojalla saadaan

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} g'(a_j^k) = g'(a_j^k) \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1}.$$

Tätä kutsutaan vastavirta-algoritmin kaavaksi. Kun kaikki tämä yhdistetään, saadaan virhefunktiolle C osittaisderivaatta painon w_{ji}^k, $1 \leq k < m$ suhteen [12]

$$\frac{\partial C}{\partial w_{ji}^k} = \delta_j^k o_i^{(k-1)} = g'(a_j^k) o_i^{(k-1)} \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1}.$$

Nyt neuroverkolle voidaan päivittää uudet painot yhtälön

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial C(\mathbf{X}, \theta^t)}{\partial \theta}$$

mukaisesti eli esimerkiksi painolle w_{23}^4 tämä olisi

$$(w_{23}^4)^2 = (w_{23}^4)^1 - \alpha \frac{\partial C(\mathbf{X}, (w_{23}^4)^1)}{\partial (w_{23}^4)}.$$

Tämä voidaan toistaa kaikille neuroverkon painoille, jonka jälkeen neuroverkolle lasketaan samoilla syötteillä uusi ulostulo ja toistetaan vastavirta-algoritmi.

4.5 Vastavirta-algoritmi numerontunnistuksessa

Käsinkirjoitettujen numeroiden tunnistamisessa ongelma voidaan jakaa kahteen pienempään alatehtävään. Ensin kuva, jossa on useampi numero jaetaan useampaan kuvaan siten, että jokaisessa kuvassa on vain yksi numero. Tämä on hyvin helppoa ihmisille, mutta haastavampaa tietokoneelle. [18] Segmentointi onnistuu esimerkiksi luokittelemalla jokainen kuvan pikseli tiettyyn segmenttiin tai jakamalla kuva suoraan tasakokoisiin segmentteihin.

Yksittäisten lukujen tunnistaminen onnistuu neuroverkolla, jossa on vain yksi piilokerros. Syötekerroksen neuroneiden lukumäärä r^1 vastaa yhden segmentin pikseleitä. Esimerkiksi, jos yhden segmentin koko on 10×10 pikseliä, syötekerroksessa on 100 neuronia. Syötekerroksen neuronit antavat tuloksen 1, jos pikseli on musta. ja tuloksen 0, jos pikseli on valkoinen. [18]

Piilokerroksen neuroneiden optimaalinen määrä r^2 selviää kokeilemalla. Ensin piilokerroksessa on 15 neuronia. Ulostulokerroksessa taas on 10 neuronia. Jokainen

ulostulokerroksen neuroni vastaa lukua $(0, 1, 2, \ldots, 9)$. Vastaukseksi neuroverkko antaa sen luvun, jota vastaavan neuronin arvo on suurin. Esimerkiksi, jos ulostulovektori on $(0.234, 0.456, 0.637, 0.768, 0.753, 0.783,\ 0.795, 0.583, 0.432, 0.678)$, tämän segmentin sisältämä käsinkirjoitettu luku on 6 neuroverkon mukaan. [18]

Nyt neuroverkon malli on suunniteltu ja voidaan keskittyä neuroverkon opettamiseen. Tähän tarvitaan ensimmäisenä opetusaineisto. MNIST-aineisto on saatavilla ilmaiseksi ja se sisältää 60 000 opetussyötettä ja 10 000 testisyötettä. Opetusaineiston käsialanäytteet on kerätty 250 ihmiseltä, joista puolet oli US Census Bureaun työntekijöitä ja puolet lukio-oppilaita. Kuvien koko on 28×28 pikseliä, joten $r^1 = 784$. Testiaineisto on kerätty samalla jakaumalla, mutta 250 eri henkilöltä kuin opetusaineisto. [18]

Opetusaineiston syöteparia merkittiin $(\mathbf{x}_i, \mathbf{y}_i)$, jossa $\mathbf{x}_i$ sisältää 784 alkiota arvoltaan 0 tai 1 ja $\mathbf{y}_i$ sisältää 10 alkiota väliltä [0, 1]. Virhefunktiona oli funktio (1) eli keskineliövirhe. Jokin toinen virhefunktio saattaisi antaa eri painot ja vakiotermit, mutta keskineliövirhe on todettu tehokkaaksi virheen minimoinnissa. [18]

Neuroverkossa on paljon lopputulokseen vaikuttavia ominaisuuksia, esimerkiksi painot, vakiotermit, aktivointifunktio, virhefunktio, neuroneiden määrä, piilokerrosten määrä ja opetusaineisto. Neuroverkon opettamisessa pääsee kuitenkin hyvin pitkälle, kun keskittyy virhefunktion minimointiin. Derivaattojen laskeminen jokaiselle muuttujalle erikseen uudestaan ja uudestaan minimin löytämiseksi ei ole tehokasta, joten käytetään gradienttimenetelmää, joka on esitelty luvussa 4.2. [18]

Gradienttimenetelmän avulla löydetään ne painot ja vakiotermit, joilla virhefunktio minimoituu. Painoille saadaan uudet arvot

$$w_{ji}^k \to w_{ji}^{k\prime} = w_{ji}^k - \alpha \frac{\partial C}{\partial w_{ji}^k}.$$

Samoin vakiotermien arvot voidaan päivittää

$$b_i^k \to b_i^{k\prime} = b_i^k - \alpha \frac{\partial C}{\partial b_i^k}.$$

Tätä iteroimalla virhefunktion arvo pienenee ja neuroverkko oppii tunnistamaan numeroita. Stokastinen gradienttimenetelmä nopeuttaa tätä prosessia, muuten jokaiselle opetusaineistolle tarvitsee laskea oma gradientti ja ottaa näistä keskiarvo. [18]

4.6 Vastavirta-algoritmi sigmoid-neuroverkoissa

Vastavirta-algoritmia voidaan käyttää niissä neuroverkoissa, joiden aktivointifunktio on differentoituva. Neuroverkko, joka sisältää perseptroneja, ei ole toimiva, koska sen aktivointifunktio on porrasfunktio, jonka derivaatta on aina 0. Tällöin neuroverkon muuttujia on mahdotonta muokata gradientin avulla. [21]

Vastavirta-algoritmi on alunperin suunniteltu käytettäväksi sigmoid-neuroverkoissa. Tässä tilanteessa merkitään piilokerrosten aktivointifunktiota $g(x) = \sigma(x)$ ja ulostulokerroksen aktivointifunktio on identiteettifunktio $g(x) = x$. [12]

Vastavirta-algoritmin käyttöä sigmoid-neuroverkoissa puoltaa myös se, että sigmoid-aktivointifunktion derivaatta on hyvin kätevä [12]

$$g'(x) = \frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)).$$

Sigmoid-funktion derivaatta on aina positiivinen, jolloin sitä on helppo seurata globaaliin minimiin. Vastaavasti ulostulokerroksen aktivointifunktion derivaatta on kätevästi vakiofunktio 1. [12]

Sigmoid-neuroverkoissa ei tarvitse laskea ja muistaa kaikkia neuroneiden ja neuroneiden ulostulojen arvoja, mikä vähentää huomattavasti muistin tarvetta. Niissä ei tarvitse laskea aktivointifunktion derivaatan arvoa jokaisella kerralla, koska aktivointifunktion derivaatta käyttää sigmoid-funktion arvoa, joka saadaan ilman aktivointifunktion arvojen muistamista. [12]

Sigmoid-neuroverkossa päästään myös eroon lineaarisuuden ongelmasta. Kun aktivointifunktio on sigmoid-funktio, ulostulo on sigmoid-funktio, jonka muuttujana on pistetulo syötteen ja painojen välillä, johon on lisätty vakiotermit. Tätä ei voi esittää lineaarikombinaationa syötteelle. Neuroverkon epälineaarisella aktivointifunktiolla voi ratkaista huomattavasti monimutkaisempia ongelmia kuin lineaarisella aktivointifunktiolla. [21]

5 Loppupäätelmät

Työssä huomattiin, että neuroverkkoihin liittyy paljon matematiikkaa. Neuroverkkojen opettaminen vastavirta-algoritmin avulla vaatii differentiaalilaskennan hallitsemista.

Neuroverkoista on paljon hyötyä työelämässä esimerkiksi käsinkirjoitettujen tekstien tunnistamisessa. Neuroverkkoja pyritään luomaan ihmisten aivojen kaltaisiksi, ja kuten ihmisetkin ne käyttävät oppimiaan asioita hyödyksi ja muodostavat yleistyksiä, joita käytetään jatkossa tuottamaan vastauksia. Mitä laajempi opetusaineisto, sen paremmin neuroverkko toimii jatkossa.

Neuroverkon opettaminen vastavirta-algoritmin avulla vie paljon muistia ja saattaa vaatia useamman iterointikierroksen. Vastavirta-algoritmi toimii nimensä mukaisesti ulostulokerroksesta syötekerrokseen ja laskee virhetermit lopusta alkuun. Virhetermit ovat riippuvaisia toisistaan. Vastavirta-algoritmin tavoite on ratkaista jollakin gradienttimenetelmällä lokaali minimi tai saavuttaa konvergenssiehto eli tarpeeksi pieni arvo virhefunktiolle.

Vastavirta-algoritmi on ohjatun oppimisen menetelmä eli se vaatii valmiiksi luokitellun aineiston toimiakseen. Menetelmä muokkaa neuroverkon painoja jokaisella iterointikierroksella ja tavoite on minimoida virhefunktiota, jona käytetään yleensä keskineliövirhettä.

Vastavirta-algoritmi on toimiva menetelmä opettaa neuroverkkoja silloin, kun neuroverkon aktivointifunktio on differentoituva kuten sigmoid-funktio. Vastavirta-algoritmia voi käyttää esimerkiksi kuvien ja tekstin tunnistamisen opettamiseen. Vastavirta-algoritmi on tehokas menetelmä, koska jokaiselle painolle voidaan käyttää samaa funktiota, jolloin neuroverkon painoja ei tarvitse korjata yksi kerrallaan. Tämä nopeuttaa huomattavasti suuren neuroverkon opettamista.

Vastavirta-algoritmi ei kuitenkaan ole täydellinen menetelmä neuroverkon opettamiseen. Vastavirta-algoritmissa käytettävä gradienttimenetelmä saattaa päätyä lokaaliin minimiin globaalin minimin sijaan tai sen konvergoiminen minimiä kohti voi niin hidasta, ettei se ole mahdollista äärellisessä ajassa. Vastavirta-algoritmia ei voida käyttää kaikille mahdollisille aktivointi- tai virhefunktioille, koska funktioiden tulee olla differentoituvia.

Viitteet

[1] Alex J. Simola, Bernhard Schölkopf: *A tutorial on support vector regression*, Australian National University, heinäkuu 2002

[2] AlindGupta: *https://www.geeksforgeeks.org/ml-semi-supervised-learning/*, haettu 5.3.2020

[3] Anas Al-Masri: *How Does Back-Propagation in Artificial Neural Networks Work?* towardsdatascience.com haettu 2.12.2020

[4] Bedri C. Cetin, Joel W. Burdick, Jacob Barhen: *Global Descent Replaces Gradient Descent to Avoid Local Minima Problem in Learning with Artificial Neural Networks* California Institute of Technology, 1993

[5] Cristopher M. Bishop: *Neural Networks for Pattern Recognition* p. 39, 1995

[6] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams: *Learning represetations by back-progating errors* Nature 323, 533–536, 1986

[7] DeepAI: The front page of A.I.: *https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning* haettu 24.2.2021

[8] Heli Tuominen: *Johdatus tekoälyn taustalla olevaan matematiikkaan* https://tim.jyu.fi/view/143092DKUvbnUuGytQ, haettu 27.02.2021

[9] Helsingin yliopisto, Elements of AI, *https://www.elementsofai.com/fi/*, haettu 23.7.2020

[10] Ian J. Goodfellow, Oriol Vinyals, Andrew M. Saxe: *Qualitatively Characterizing Neural Network Optimization Problems*, ICLR 2015

[11] Iryna Sydorenko *https://labelyourdata.com/articles/introduction-to-labeled-data-what-why-and-how/* haettu 19.1.2021

[12] John Mcgonagle, George Shalkouski, Christopher Williams, Andrew Hsu, Jlmlm Khlm, Aaron Miller: *Backpropagation* Brilliant.org haettu 9.12.2020

[13] Kaija Häkkinen: *Matematiikan propedeuttinen kurssi.* Jyväskylän yliopisto 2006

[14] Marko M. Mäkelä: *Konveksi analyysi ja optimointi.* Turun yliopisto, syksy 2014

[15] Marko M. Mäkelä: *Matemaattinen optimointi I.* Turun yliopisto, kevät 2015

[16] Marko M. Mäkelä: *Optimointialgoritmit.* Turun yliopisto, syksy 2016

[17] Martti Lehto, Pekka Neittaanmäki, Riku Nyrhinen, Anniina Ojalainen, Ilkka Pölönen, Ilkka Rautiainen, Toni Ruohonen, Heli Tuominen, Petri Vähäkainu, Sami Äyrämö, Sanna-Mari Äyrämö: *Tekoälyn perusteita ja sovelluksia*, 2018

[18] Michael A. Nielsen: *Neural Networks and Deep Learning.* Determination Press, 2015

[19] Missinglink AI *Neural network Guide*, haettu 2.12.2020

[20] Muhammad W, Hart GR, Nartowt B, Farrell JJ, Johung K, Liang Y, Deng J: *Pancreatic Cancer Prediction Through an Artificial Neural Network*, Front. Artif. Intell. 2:2. doi: 10.3389/frai.2019.00002, 2019

[21] Nahua Kang: *https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f* haettu 18.01.2021

[22] Prates, M.O.R., Avelar, P.H., Lamb, L.C.: *Assessing gender bias in machine translation: a case study with Google Translate* Neural Comput & Applic 32, 6363–6381, 2020 https://doi.org/10.1007/s00521-019-04144-6

[23] Saumya: *What is the difference between artificial neural network and the human brain?* https://www.verzeo.com/blog-artificial-neural-network-vs-human-brain, haettu 2.5.2021

[24] Stephen Ornes: *Symbolic mathematics finally yields to Neural Networks* https://www.quantamagazine.org/symbolic-mathematics-finally-yields-to-neural-networks-20200520/, haettu 22.6.2020

[25] Timo Honkela: *Neuroverkot: johdatus moderniin tekoälyyn*, Teknillinen korkeakoulu, 1996

[26] Varun Kumar Ojha, Ajith Abraham, Václav Snášel: *Metaheuristic design of feedforward neural networks: A review of two decades of research* Engineering Applications of Artificial Intelligence 60, p. 97-116, 2017

[27] Wenqing Hu: *Nonlinear Optimization in Machine Learning* Missouri S&T