Neuroverkkojen matematiikka ja optimointialgoritmit

Jussi Mäki-Ikola

Pro gradu -tutkielma
Toukokuu 2023

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

MÄKI-IKOLA, JUSSI: Neuroverkkojen matematiikka ja optimointialgoritmit
Pro gradu -tutkielma, 52 s.
Matematiikka, Data-analytiikka
Toukokuu 2023

Tässä tutkielmassa esitellään neuroverkot ja niiden taustalla olevat matemaattiset mallit sekä havainnollistetaan neuroverkon toimintaa yksinkertaisen esimerkin avulla. Lisäksi tutkielmassa selitetään, mitä neuroverkon opettaminen tarkoittaa ja esitetään neuroverkkojen opettamisessa yleisimmin käytettyjä optimointialgoritmeja.

Tutkielmassa sovelletaan neuroverkkoja kahteen eri luokitteluongelmaan: käsinkirjoitettujen numeroiden tunnistamiseen ja hotellien varauksien peruuntumisen ennustamiseen. Molempiin aineistoihin sovitettiin neuroverkko tutkielmassa esitellyillä optimointialgoritmeilla. Havaittiin, että tietyillä algoritmeilla saatiin parempia tuloksia. Neuroverkolla saatiin kumpaankin aineistoon hyvä luokittelutarkkuus.

Tutkielmaa varten ohjelmoitiin neuroverkko Javascript-ohjelmointikielellä ja vertailtiin sitä lyhyesti Pythonin Keras-kirjaston kanssa. Javascript-ohjelman suorituskyky osoittautui kelvolliseksi.

Asiasanat: neuroverkko, vastavirta-algoritmi, koneoppiminen, optimointi

Sisällys

1 Johdanto

Nykyajan tietoyhteiskunnassa tekoälyn rooli on kasvanut merkittävästi. Dataa on saatavilla enemmän kuin koskaan aiemmin, ja sen onnistuneesta analysoinnista on merkittävää hyötyä. Neuroverkot ovat yksi käytetyimmistä koneoppimismenetelmistä. Vaikka niiden taustalla oleva matemaattinen malli on suhteellisen yksinkertainen, neuroverkkojen saavuttamat tulokset ovat olleet erinomaisia esimerkiksi kuvantunnistuksessa.

Warren McCulloch ja Walter Pitts julkaisivat ensimmäisen version neuronista vuonna 1943. Tällä neuronilla pystyttiin erottamaan kaksi luokkaa, mutta neuronin käyttäjän piti itse valita neuronille sopivat painot. Frank Rosenblatt julkaisi vuonna 1958 McCullochin ja Pittsin työn pohjalta neuronin, joka pystyi oppimaan neuronin painot. Neuroverkot eivät saavuttaneet pitkään aikaan suurta suosiota, sillä niiden opettaminen oli hankalaa. Kuitenkin aineistojen kokojen ja laskentatehon kasvun myötä neuroverkkojen saavuttamat tulokset ovat parantuneet ja niiden suosio on kasvanut. [1]

Tämän Pro gradu -tutkielman tavoitteena on esitellä neuroverkkojen matematiikkaa ja yleisimmät neuroverkkojen opetusalgoritmit. Neuroverkkojen matemaattiset taustat on hyvä tuntea, jotta niitä voidaan soveltaa paremmin käytännössä. Neuroverkkojen opetusalgoritmin valinnalla voi olla suuri merkitys lopputuloksen kannalta, joten myös niiden ymmärtäminen on tärkeää. Monet aineistot ovat nykyään valtavan suuria ja niiden oppiminen voi olla erittäin hidasta, joten nopeammilla opetusalgoritmilla voidaan säästää paljon aikaa ja laskenta-aikaa.

Tutkielman kaksi ensimmäistä lukua käsittelevät tarpeellisia esitietoja. Näissä luvuissa esitellään lyhyesti tutkielmassa käytetyt matemaattiset määritelmät ja käsitteet koneoppimisesta. Luvussa neljä esitellään neuroverkko ja neuroverkon opettaminen vastavirta-algoritmin avulla. Tässä luvussa myös johdetaan vastavirta-algoritmi. Tuloksia havainnollistetaan soveltamalla neuroverkkoa yksinkertaiseen XOR-ongelmaan. Luvussa viisi esitellään valikoidut algoritmit, joilla neuroverkkoja voidaan opettaa, ja luvussa kuusi sovelletaan näitä algoritmeja kahteen eri luokitteluongelmaan. Viimeisessä luvussa esitellään tutkielman johtopäätökset.

Tutkielman sovelluksissa käytetään Javascript-ohjelmointikielellä tehtyä neuroverkon toteutusta, ja sen tuloksia vertaillaan lyhyesti Python-kielen Keras-kirjaston kanssa.

2 Esitietoja

Tässä luvussa käydään läpi tärkeimpiä esitietoja tutkielman ymmärtämisen kannalta. Analyysin ja lineaarialgebran perustulokset oletetaan tunnetuiksi. Vektorilaskennan määritelmistä on tarkoituksella jätetty avaruuden $\mathbb{R}^n$ topologian yksityiskohtia pois. Tässä tutkielmassa määritelmiä tarvitaan rajoitteettoman optimoinnin tapauksessa, joten avoimien ja suljettujen joukkojen käsittely ei ole olennaista – onhan $\mathbb{R}^n$ sekä avoin että suljettu joukko. Tarkemmat määritelmät löytyvät esimerkiksi teoksista [2, 3].

Määritelmä 1 (Hadamardin tulo)**.** Olkoon $\boldsymbol{A}$ ja $\boldsymbol{B}$ $m \times n$ matriiseja. Matriisien *Hadamardin tulo* $\boldsymbol{A} \odot \boldsymbol{B}$ saadaan, kun kerrotaan matriisien elementit keskenään. Tällöin $(A \odot B)_{mn} = A_{mn} \cdot B_{mn}$. Tarkemmin [4]

$$\boldsymbol{A} \odot \boldsymbol{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \dots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \dots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \dots & a_{mn}b_{mn} \end{bmatrix}.$$

Lemma 1. Olkoon $\boldsymbol{D} = \operatorname{diag}(a_1, a_2, \dots, a_n)$ lävistäjämatriisi ja olkoon $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T$. Merkitään $\boldsymbol{d} = (a_1, a_2, \dots, a_n)^T$. Tällöin

$$\boldsymbol{D}\boldsymbol{x} = \boldsymbol{d} \odot \boldsymbol{x} = \boldsymbol{x} \odot \boldsymbol{d}.$$

Todistus. Seuraa suoraan matriisitulon ja Hadamardin tulon määritelmistä. Matriisin $\boldsymbol{D}$ lävistäjän ulkopuolella olevat alkiot ovat nollia eivätkä vaikuta matriisitulon tulokseen. □

Määritelmä 2 (Osittaisderivaatta)**.** Olkoon $f : \mathbb{R}^n \to \mathbb{R}$ ja $\boldsymbol{x} \in \mathbb{R}^n$. Jos raja-arvo

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x} + he_i) - f(\boldsymbol{x})}{h}$$

on olemassa, niin sitä kutsutaan funktion f osittaisderivaataksi pisteessä $\boldsymbol{x}$ muuttujan x_i suhteen. [3]

Määritelmä 3 (Gradientti)**.** Olkoon $f : \mathbb{R}^n \to \mathbb{R}$ funktio, jonka ensimmäisen kertaluvun osittaisderivaatat ovat olemassa kaikilla pisteillä $\boldsymbol{x} \in \mathbb{R}^n$. Tällöin funktion f *gradientti* pisteessä $\boldsymbol{x}$ on [3]

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} & \dfrac{\partial f}{\partial x_2} & \dots & \dfrac{\partial f}{\partial x_n} \end{bmatrix}^T.$$

Määritelmä 4 (Jacobin matriisi)**.** Olkoon $f : \mathbb{R}^n \to \mathbb{R}^m$ funktio, jonka ensimmäisen kertaluvun osittaisderivaatat ovat olemassa kaikilla pisteillä $\boldsymbol{x} \in \mathbb{R}^n$. Tällöin kuvauksen f *Jacobin matriisi* on [3]

$$J_f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^T f_1 \\ \vdots \\ \nabla^T f_m \end{bmatrix}.$$

Olkoon $\boldsymbol{y} = f(\boldsymbol{x})$. Jacobin matriisia pisteessä $\boldsymbol{x}$ voidaan merkitä myös

$$\frac{\partial(y_1, y_2, \ldots, y_m)}{\partial(x_1, x_2, \ldots, x_n)} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}.$$

Erikoistapauksessa, jossa $f : \mathbb{R}^n \to \mathbb{R}$, saadaan Jacobin matriisista helposti gradientti

$$\nabla f(\boldsymbol{x}) = J_f^T(\boldsymbol{x}).$$

Lause 1 (Ketjusääntö)**.** Olkoon $\boldsymbol{y} \in \mathbb{R}^{n_1}$ ja $\boldsymbol{x} \in \mathbb{R}^{n_2}$. Olkoon $g : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ differentioituva pisteessä $\boldsymbol{x}$ ja $f : \mathbb{R}^{n_2} \to \mathbb{R}^{n_3}$ differentioituva pisteessä $\boldsymbol{y} = g(\boldsymbol{x})$. Tällöin yhdistetty funktio $h = f \circ g : \mathbb{R}^{n_1} \to \mathbb{R}^{n_3}$ on differentioituva pisteessä x, ja sen Jacobin matriisi on

$$\begin{aligned} J_h(\boldsymbol{x}) &= J_f(g(\boldsymbol{x}))J_g(\boldsymbol{x}) = J_f(\boldsymbol{y})J_g(\boldsymbol{x}) \\ &= \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_{n_2}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n_3}}{\partial y_1} & \cdots & \frac{\partial f_{n_3}}{\partial y_{n_2}} \end{bmatrix} \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_{n_1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{n_2}}{\partial x_1} & \cdots & \frac{\partial g_{n_2}}{\partial x_{n_1}} \end{bmatrix}. \end{aligned}$$

Täten osittaisderivaatat funktiolle h saadaan kaavasta

$$\frac{\partial h_k}{\partial x_i} = \sum_{j=1}^{n_2} \frac{\partial f_k}{\partial y_j} \frac{\partial g_j}{\partial x_k}.$$

Jos $\boldsymbol{z} = h(\boldsymbol{x})$, saadaan yhtäpitävästi [3]

$$\begin{aligned} \frac{\partial(z_1, \ldots, z_{n_3})}{\partial(x_1, \ldots, x_{n_2})} &= \frac{\partial(z_1, \ldots, z_{n_3})}{\partial(y_1, \ldots, y_{n_1})} \frac{\partial(y_1, \ldots, y_{n_1})}{\partial(x_1, \ldots, x_{n_2})} \\ \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} &= \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}. \end{aligned}$$

Todistus. Sivuutetaan. Tulos on todistettu esimerkiksi teoksessa [2]. □

Määritelmä 5 (Konveksisuus)**.** Olkoon $f : \mathbb{R}^n \to \mathbb{R}$. Funktiota f sanotaan *konveksiksi*, jos kaikille $\lambda \in (0, 1)$ ja $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$

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

Funktio f on *aidosti konveksi*, jos epäyhtälö on voimassa aidosti kaikille $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$. Funktiota f sanotaan (aidosti) *konkaaviksi*, jos $-f$ on (aidosti) konveksi. [5]

3 Koneoppiminen

Koneoppiminen on tekoälyn osa-alue, jossa tietokoneita opetetaan ratkaisemaan ongelmia aineiston tai kokemuksen perusteella. Opetettu tietokonemalli voi olla luonteeltaan ennustava, jolloin mallia voi käyttää ennnusteiden tekemiseen. Toisaalta malli voi olla kuvaileva, jolloin aineistosta on mahdollista löytää uutta tietoa. [6]

Mitchell määrittelee oppimisen seuraavasti: *Tietokone-ohjelman sanotaan oppivan kokemuksesta E tehtävien T ja suorituskyvyn P suhteen, jos sen suorituskyky P tehtävissä T kasvaa, kun kokemus E kasvaa.* Esimerkiksi ohjelma voi parantaa suorituskykyään tammen pelaamisessa, jos sen voittoprosentti kasvaa sitä mukaa, kun ohjelma pelaa lisää pelejä. Koneoppiminen vaatii, että kokemus E, tehtävät T ja suorituskyvyn mittarit P ovat hyvin määriteltyjä. [7] Koneoppimista voi hyödyntää esimerkiksi seuraavanlaisiin ongelmiin [6, 7]:

- puheen- ja kuvantunnistus
- itseohjautuvat autot
- luottoluokitusten määrittäminen
- lääketieteellisten diagnoosien tekeminen
- autojen arvon määritys.

Koneoppiminen voidaan jakaa kolmeen eri osa-alueeseen opettajan roolin perusteella: ohjattuun oppimiseen, ohjaamattomaan oppimiseen ja vahvistusoppimiseen [6]. Tässä tutkielmassa käsitellään neuroverkkoja ohjatun oppimisen näkökulmasta.

3.1 Ohjattu oppiminen

Ohjatussa oppimisessa on tavoitteena löytää kuvaus $f : \mathbb{X} \to \mathbb{Y}$, kun syöte X ja ulostulo Y ovat tiedossa. Usein oletetaan, että f on tiettyä tunnettua muotoa oleva funktio, kuten astetta N oleva polynomi, jolla on parametrit θ. Tällöin ongelmana on löytää sellaiset parametrit θ, joilla funktio f kuvaa mahdollisimman hyvin kuvausta $\mathbb{X} \to \mathbb{Y}$. [6] Tätä kuvauksen hyvyyttä mitataan useimmiten virhefunktioilla, joista kerrotaan tarkemmin luvussa 3.2.

Ohjatun oppimisen ongelmat voidaan määrittää joko *regressio-* tai *luokitteluongelmiksi*. Regressio-ongelmassa syötteellä $\boldsymbol{x}$ on tarkoitus selittää jatkuvia muuttujia $\boldsymbol{y}$, kun taas luokitteluongelmassa selitettävät muuttujat $\boldsymbol{y}$ ovat diskreettejä. Esimerkiksi osakkeen arvon ennustaminen on regressio-ongelma, ja käsin piirrettyjen lukujen tulkitseminen on luokitteluongelma. [6]

Luokitteluongelmassa, jossa on N luokkaa, pisteen $\boldsymbol{x}$ tunnettu ulostulo $\boldsymbol{y}$ määritetään seuraavasti [6]:

$$y_i = \begin{cases} 1, & \text{jos piste } \boldsymbol{x} \text{ kuuluu luokkaan } i, \\ 0, & \text{jos piste } \boldsymbol{x} \text{ ei kuulu luokkaan } i. \end{cases}$$

Tällöin $\boldsymbol{y}$ on vektori, jossa on yksi 1 ja loput arvot nollia. Tämänlaista vektoria kutsutaan *one-hot -enkoodatuksi*. Olkoon nyt $\hat{\boldsymbol{y}}$ tunnetun ulostulon estimaattori. Tällöin tulos $\hat{\boldsymbol{y}}$ tulkitaan luokkaan i, jos

$$\hat{y}_i = \max\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N\}.$$

Erikoistapausta, jossa $N = 2$, kutsutaan *binääriseksi luokitteluksi*. Tällöin ulostulo y voidaan myös määritellä seuraavasti:

$$y = \begin{cases} 1, & \text{jos piste } \boldsymbol{x} \text{ kuuluu luokkaan 1,} \\ 0, & \text{jos piste } \boldsymbol{x} \text{ kuuluu luokkaan 2.} \end{cases}$$

Tällöin tulos $\hat{y}$ tulkitaan luokkaan 1, jos $\hat{y} \geq 0.5$, ja muutoin luokkaan 2. [6]

Yli- ja alisovitus
Useimmiten halutaan, että opetettu malli yleistyisi myös käytetyn aineiston ulkopuolelle. Ilmiötä, jossa malli oppii aineiston hyvin, muttei yleisty aineiston ulkopuolelle, kutsutaan *ylisovittamiseksi*. Ylisovittamisen välttämiseksi aineisto jaetaan yleensä opetus- ja testijoukoiksi, jolloin mallia testataan opetusalgoritmien iteraatioiden välissä testijoukkoa vasten. Opetus lopetetaan, jos testijoukon virhe alkaa kasvaa. Ylisovittamista voidaan välttää myös esimerkiksi aineiston kokoa kasvattamalla. Jos malli ei opi aineistoa eikä yleisty, sanotaan, että malli *alisovittaa*. [4]

3.2 Ohjaamaton oppiminen

Ohjaamattomassa oppimisessa ei ole tunnettuja ulostuloja, vaan pelkästään syötteitä. Tavoitteena on löytää rakenteita ja säännönmukaisuuksia syötteistä. Esimerkiksi klusterointi on ohjaamattoman oppimisen menetelmä, jossa tarkoituksena on ryhmitellä syötteitä samankaltaisuuksien perusteella. [6]

3.3 Vahvistusoppiminen

Vahvistusoppimisessa järjestelmälle annetaan palautetta toimintojen sarjan perusteella. Tällöin yksittäisellä toiminnolla ei ole merkitystä: toiminto on hyvä, jos se on osa hyvien toimintojen sarjaa. Vahvistusoppimista käytetään esimerkiksi pelien tekoälyissä ja robotiikassa. [6]

4 Neuroverkko

Neuroverkko (hermoverkko) on syvä yhdistetty funktio, joka jäljittelee ihmisen aivoja. Se muodostuu useista kerroksista: syötekerroksesta, ulostulokerroksesta ja niiden välissä olevista *piilokerroksista*. Neuroverkkojen kerrokset koostuvat *neuroneista*, jotka ovat yksinkertaisia laskentayksikoitä.

Tässä työssä tarkastellaan neuroverkkoja ohjatun oppimisen näkökulmasta. Tavoitteena on löytää N kokoiselle opetusjoukolle $\mathcal{D} = \{\boldsymbol{x}^k, \boldsymbol{y}^k\}_{k=1}^N$ sellainen kuvaus $f : \mathbb{X} \to \mathbb{Y}$, joka yleistyy mahdollisimman hyvin testijoukolle.

Määritelmä 6 (Neuroni)**.** Olkoon $\boldsymbol{x} \in \mathbb{R}^n$ syöte ja $\boldsymbol{w} \in \mathbb{R}^n$ syötettä vastaavat *painot*. Neuronin ulostulo on tällöin [6]

$$z = \boldsymbol{w}^T \boldsymbol{x} + b = \sum_{j=1}^{n} w_j x_j + b,$$

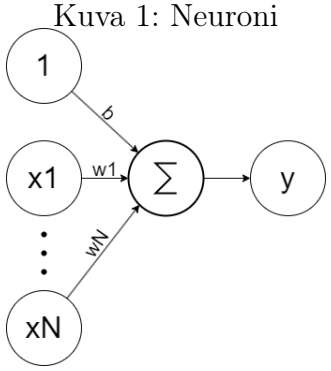
jossa $b \in \mathbb{R}$ on vakiotermi. Usein vakiotermi sisällytetään painovektoriin, jolloin $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^{n+1}$, jossa $w_0 = b$ ja $x_0 = 1$. Tällöin neuronin ulostulo on

$$z = \boldsymbol{w}^T \boldsymbol{x} = \sum_{j=0}^{n} w_j x_j.$$

Kaava on matriisimuodossa

$$z = \begin{bmatrix} b & w_1 & w_2 & \dots & w_n \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Kuva 1: Neuroni



Määritelmä 7 (Kerros)**.** Neuroverkon kerroksella tarkoitetaan kokoelmaa neuroneja, joilla on yhteinen syöte $\boldsymbol{a} \in \mathbb{R}^n$. Jos kerroksessa on m neuronia, niin neuronin i ulostulo on [6]

$$z_i = \boldsymbol{w}_i^T \boldsymbol{a} + b_i.$$

Olkoon $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ matriisi, jossa w_{ij} on syötteen a_j ja neuronin z_i välinen paino. Olkoon lisäksi $\boldsymbol{b} \in \mathbb{R}^n$ vektori, jossa b_i on neuronia i vastaava vakiotermi. Tällöin kerroksen ulostulo on

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{a} + \boldsymbol{b},$$

jossa $\boldsymbol{z} \in \mathbb{R}^m$. Jos vakiotermit sisällytetään painovektoreihin, yhtälö yksinkertaistuu muotoon

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{a},$$

jossa $w_{i0} = b_i$ ja $x_{i0} = 1$.

Neuroverkon kerros on matriisimuodossa

$$\boldsymbol{z} = \begin{bmatrix} b_1 & w_{11} & w_{12} & \dots & w_{1n} \\ b_2 & w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_m & w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}.$$

Neuroverkkoa sanotaan syväksi, jos siinä on lukuisia piilokerroksia. Tästä syntyykin nimitys *syväoppiminen*. Tarkastellaan nyt yleistä neuroverkkoa, jossa on L kerrosta. Sisääntulokerroksen indeksi on $l = 0$ ja ulostulokerroksen indeksi on $l = L$. Käytetään jatkossa seuraavia merkintöjä:

- n^l on kerroksessa l olevien neuronien lukumäärä
- w_{ij}^l on paino, joka on kerroksen $l-1$ neuronin j ja kerroksen l neuronin i välissä. Lisäksi w_{i0} on neuroniin i liittyvä vakiotermi.
- z_i^l on kerroksen l neuronin i ulostulo

$$z_i^l = \boldsymbol{w_i^l}\boldsymbol{a^{l-1}} = \sum_{j=0}^{n_l} w_{ij}^l a_j^{l-1}$$

- $\boldsymbol{a}^{l-1}$ on kerroksen l syöte, jossa $a_0 = 1$.

4.1 Aktivointifunktiot

Yllä esitellyllä neuroverkolla ei pysty esittämään epälineaarisia funktioita, sillä neuronit ovat lineaarikombinaatioita. Tämän vuoksi neuronien ulostulo z syötetään usein johonkin epälineaariseen funktioon, jota kutsutaan *aktivointifunktioksi*. [4]

Määritelmä 8 (Aktivointifunktio)**.** Olkoon $g_l : \mathbb{R}^m \to \mathbb{R}^m$. Neuroverkon kerroksen l aktivoitu ulostulo on

$$\boldsymbol{a}^l = \boldsymbol{g}_l(\boldsymbol{z}) = (g_{l,1}(\boldsymbol{z}), g_{l,2}(\boldsymbol{z}), \dots, g_{l,m}(\boldsymbol{z}))^T.$$

Jos vakiotermit sisällytetään painomatriisiin ja $\boldsymbol{a}$ ei ole verkon ulostulo, niin $\boldsymbol{a}^l \in \mathbb{R}^{m+1}$. Nyt $g_l : \mathbb{R}^m \to \mathbb{R}^{m+1}$ ja

$$\boldsymbol{a}^l = (1, g_{l,1}(\boldsymbol{z}), g_{l,2}(\boldsymbol{z}), \dots, g_{l,m}(\boldsymbol{z}))^T.$$

Tällöin

$$\boldsymbol{a}^l = \begin{cases} (1, g_{l,1}(\boldsymbol{z}), g_{l,2}(\boldsymbol{z}), \ldots, g_{l,m}(\boldsymbol{z}))^T, & \text{jos } l < L \\ (g_{l,1}(\boldsymbol{z}), g_{l,2}(\boldsymbol{z}), \ldots, g_{l,m}(\boldsymbol{z}))^T, & \text{jos } l = L. \end{cases}$$

Usein aktivointifunktiot esitetään kuvauksina $\mathbb{R} \to \mathbb{R}$. Tällöin kuitenkin tarkoitetaan ns. funktion vektoroitua muotoa, jossa jokaiselle kerroksen neuronille käytetään samaa aktivointifunktiota. Olkoon nyt $g_l : \mathbb{R} \to \mathbb{R}$. Tällöin

$$\boldsymbol{g}_l(\boldsymbol{z}) = \begin{cases} (1, g_l(z_1^l), g_l(z_2^l), \ldots, g_l(z_{n_l}^l))^T, & \text{jos } l < L \\ (g_l(z_1^l), g_l(z_2^l), \ldots, g_l(z_{n_l}^l))^T, & \text{jos } l = L. \end{cases}$$

Jatkossa merkinnällä $g(\boldsymbol{z})$ tarkoitetaan funktion vektoroitua muotoa, jos g on kuvaus $\mathbb{R} \to \mathbb{R}$.

Yllä oleva määritelmä on uudenlainen määritelmä aktivointifunktiolle, ja tätä määritelmää hyödynnetään seuraavan osion todistuksissa. Useissa lähteissä (esim. [1, 4]) aktivointifunktioita ei määritellä matemaattisesti kovinkaan tarkasti. Teoksessa [8] aktivointifunktio määritellään hieman vastaavasti funktiona $g_i(\boldsymbol{x})$, jossa $g_0(\boldsymbol{x}) = 1$. Tässäkään lähteessä ei toisaalta yleistetä aktivointifunktion määritelmää funktioille $R^m \to R^n$.

Esitellään seuraavaksi usein käytettyjä aktivointifunktioita. Aktivointifunktiot toteuttavat yleensä seuraavat ehdot [9]:

- funktio on differentioituva (melkein kaikkialla)
- funktio on helposti laskettavissa
- funktio on epälineaarinen.

Ulostulokerroksen aktivointifunktiot

Ulostulokerroksen aktivointifunktio valitaan ratkaistavan ongelman mukaan. Jos kyseessä on binäärinen luokittelu, niin aktivointifunktioksi valitaan *Sigmoid-funktio* [1]

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}.$$

Sigmoid-funktion ulostulo on aina välillä $(0, 1)$. Se siis soveltuu hyvin kuvaamaan todennäköisyyksiä.

Jos tehtävänä on luokitella syöte yhteen useammasta kategoriasta, niin ulostulokerroksen aktivointifunktioksi valitaan *Softmax-funktio*. Softmax-funktiota voidaan pitää Sigmoid-funktion yleistyksenä useammalle kategorialle. Funktion ulostulon jokainen komponentti kuuluu niin ikään välille $(0, 1)$. Softmax-funktion komponentin i yhtälö on [1]

$$\text{Softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$

Regressio-ongelmissa käytetään usein lineaarista aktivointifunktiota.
Olkoon $\alpha \in \mathbb{R}$. Tällöin ulostulokerroksen aktivointifunktio g on

$$g(x) = \alpha x.$$

Piilokerroksissa ei yleensä ole järkevää käyttää lineaarista aktivointia [9].

Piilokerroksien aktivointifunktiot
Piilokerroksien aktivointifunktion valintaan ei vielä ole tarkkoja ohjenuoria [1]. Useimmiten käytetään *ReLu-funktiota* (rectified linear unit), ja sitä suositellaan ensisijaiseksi vaihtoehdoksi [9, 1]. ReLu määritellään seuraavasti

$$\text{ReLu}(x) = \max\{0, x\}.$$

ReLu-funktion hyötyjä ovat sen arvojen ja sen derivaatan arvojen nopea laskeminen. ReLu-aktivoidut neuronit saattavat joskus päätyä tilanteeseen, jossa ne saavat aina arvoksi nolla. Näitä neuroneja sanotaan *kuolleiksi neuroneiksi*. Tämän ongelman ratkaisemiseksi käytetään *LeakyReLu-funktiota*

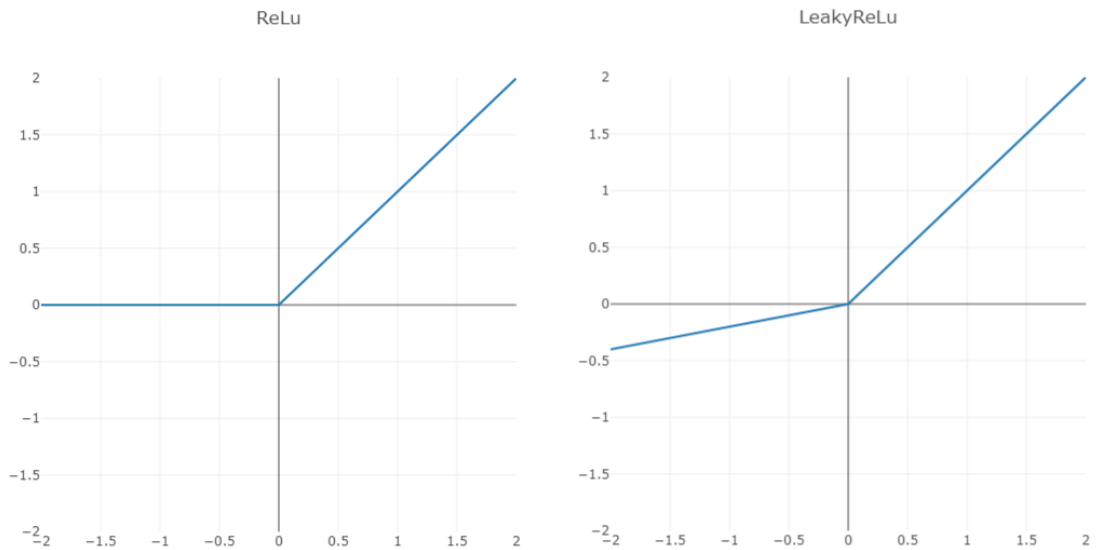
$$\text{LeakyReLu}(x) = \max\{\alpha x, x\},$$

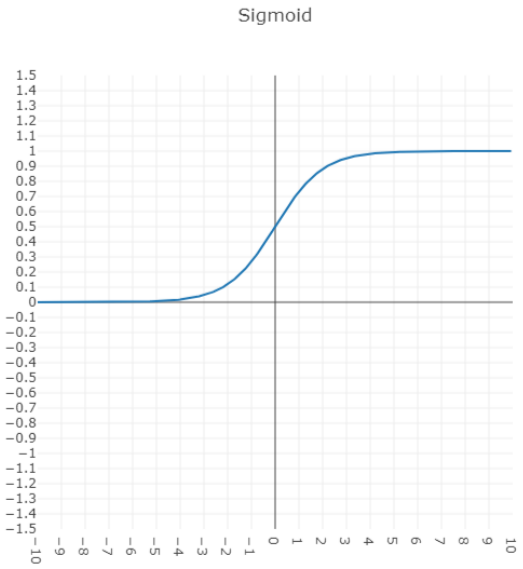
jossa $\alpha > 0$ on jokin pieni vakio kuten 0.01. [9]

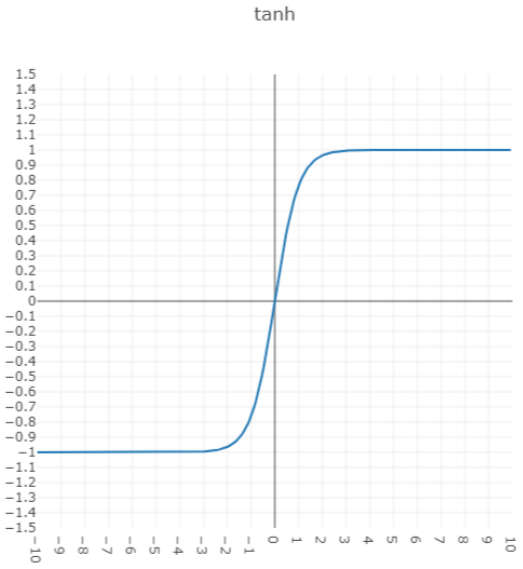
Sigmoid-funktiota voidaan käyttää piilokerroksien aktivointifunktiona, joka oli ennen ReLu-funktiota käytetyin aktivointifunktio [1]. Sigmoidin sijaan käytetään joskus *Hyperbolista tangenttia*

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Hyperbolisella tangentilla saadaan usein parempia tuloksia kuin Sigmoidilla [1].







4.2 Neuroverkon ulostulo

Nyt saadaan määriteltyä neuroverkon ulostulo $\boldsymbol{t} = \boldsymbol{a}^L$:

$$\begin{aligned}\boldsymbol{t} = f(\boldsymbol{x}) &:= g_L(\boldsymbol{W}^L \boldsymbol{a}^{L-1}) \\ &= g_L(\boldsymbol{W}^L g_{L-1}(\boldsymbol{W}^{L-1} \boldsymbol{a}^{L-2})) \\ &= g_L(\boldsymbol{W}^L g_{L-1}(\boldsymbol{W}^{L-1} \cdots g_1(\boldsymbol{W}^1 \boldsymbol{x}) \cdots)).\end{aligned}$$

Esimerkki 1 (XOR)**.** Tarkastellaan klassista luokitteluesimerkkiä, jossa luokitellaan totuusarvopareja XOR-säännön (exclusive or) mukaan. Syöte on $\boldsymbol{x} \in \{0,1\}^2$, ja ulostulo on $y = \{0,1\}$.

Syöte x_1	Syöte x_2	Ulostulo y
1	1	0
1	0	1
0	1	1
0	0	0

Muodostetaan tämän luokitteluongelman ratkaisemiseen neuroverkko, jossa on yksi kahden neuronin piilokerros. Nyt siis $L = 2, n^0 = 2$, ja $n^2 = 1$. Olkoon ensimmäisen kerroksen aktivointifunktio $g_1(z) = \text{ReLu}(z)$ ja toisen $g_2(z) = \text{Sigmoid}(z)$. Neuroverkon ulostulo on nyt

$$\begin{aligned}t &= g_2(\boldsymbol{W}^2 \boldsymbol{a}^1) \\ &= g_2(\boldsymbol{W}^2 g_1(\boldsymbol{W}^1 \boldsymbol{x})).\end{aligned}$$

Olkoon $\boldsymbol{x} = (1, 1)$ ja painojen alkuarvot

$$\begin{aligned}
\boldsymbol{W}^1 &= \begin{bmatrix} b_1^1 & w_{11}^1 & w_{12}^1 \\ b_2^1 & w_{21}^1 & w_{22}^1 \end{bmatrix} \\
&= \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ -0.5 & -0.5 & -0.5 \end{bmatrix}, \\
\boldsymbol{W}^2 &= \begin{bmatrix} b_1^2 & w_{11}^2 & w_{12}^2 \end{bmatrix} \\
&= \begin{bmatrix} -0.5 & 0.5 & 0.5 \end{bmatrix}.
\end{aligned}$$

Muodostetaan ensimmäisen kerroksen neuronien summat $\boldsymbol{z}^1$ ja niiden aktivoinnit $\boldsymbol{a}^1$:

$$\begin{aligned}
\boldsymbol{z}^1 &= \begin{bmatrix} b_1^1 & w_{11}^1 & w_{12}^1 \\ b_2^1 & w_{21}^1 & w_{22}^1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \\
&= \begin{bmatrix} b_1^1 + w_{11}^1 x_1 + w_{12}^1 x_2 \\ b_2^1 + w_{21}^1 x_1 + w_{22}^1 x_2 \end{bmatrix} \\
&= \begin{bmatrix} 0.5 + 0.5 + 0.5 \\ -0.5 + (-0.5) + (-0.5) \end{bmatrix} = \begin{bmatrix} 1.5 \\ -1.5 \end{bmatrix}, \\
\boldsymbol{a}^1 &= \mathrm{ReLu}(\boldsymbol{z^1}) = \begin{bmatrix} 1 \\ 1.5 \\ 0 \end{bmatrix}.
\end{aligned}$$

Nyt saadaan laskettua ulostulokerroksen arvot z^2 ja $t = a^2$

$$\begin{aligned}
z^2 &= \begin{bmatrix} b_1^2 & w_{11}^2 & w_{12}^2 \end{bmatrix} \begin{bmatrix} 1 \\ a_1^1 \\ a_2^1 \end{bmatrix} \\
&= b_1^2 + w_{11}^2 a_1^1 + w_{12}^2 a_2^1 \\
&= -0.5 + 0.5 \cdot 1.5 + 0.5 \cdot 0 = 0.25, \\
t &= \mathrm{Sigmoid}(0.25) \approx 0.562.
\end{aligned}$$

Vastaavalla prosessilla saadaan kaikille pisteille luokitukset

x_1	x_2	y	z^1	a^1	z^2	$t = \sigma(z^2)$
1	1	0	(1.5, -1.5)	(1, 1.5, 0)	0.25	0.562
1	0	1	(1, -1)	(1, 1, 0)	0	0.5
0	1	1	(1, -1)	(1, 1, 0)	0	0.5
0	0	0	(0.5, -0.5)	(1, 0.5, 0)	-0.25	0.438

Sigmoid-funktion arvoa tulkitaan todennäköisyytenä: jos $t \geq 0.5$, niin ennuste on 1 ja muutoin 0. Tuloksista huomataan, että verkko luokittelee pisteen $(1, 1)$ tulokseksi 1, joka on eri kuin haluttu arvo. Tulos ei ole yllättävä, sillä verkon painot alustettiin mielivaltaisesti. Muut pisteet verkko onnistui sattumalta luokittelemaan oikein. Tutkitaan seuraavaksi, miten verkon luokittelukykyä saadaan parannettua.

4.3 Virhefunktiot

Neuroverkon opettamisella tarkoitetaan prosessia, jossa neuroverkon parametreille $\boldsymbol{W}$ etsitään sellaiset arvot, joilla verkon luokitteluvirhe minimoituu. Useimmiten opettamisessa muodostetaan *virhefunktio*, joka minimoidaan opittavien parametrien suhteen. Mielletään $\boldsymbol{W}$ vektorina, jonka alkiot ovat w_{ij}^l. [6]

Olkoon $\boldsymbol{t} = f(\boldsymbol{x}; \boldsymbol{W})$ neuroverkon ennustama ulostulo ja olkoon $\boldsymbol{y}$ tunnettu ulostulo syötteelle $\boldsymbol{x}$. Neuroverkon antaman ennusteen tarkkuutta mitataan virhefunktiolla $E(\boldsymbol{t}, \boldsymbol{y})$. Regressiossa virhefunktio on useimmiten euklidisen etäisyyden neliö [6]

$$E(\boldsymbol{t}, \boldsymbol{y}) = \frac{1}{2} \parallel \boldsymbol{t} - \boldsymbol{y} \parallel^2 = \frac{1}{2} \sum_{k=1}^{d} (t_k - y_k)^2.$$

Luokittelussa virhefunktiona käytetään useimmiten *ristientropiaa* (*cross-entropy*) [6]

$$E(\boldsymbol{t}, \boldsymbol{y}) = -\boldsymbol{y} \cdot \ln(\boldsymbol{t}) = -\sum_{k=1}^{d} y_k \ln(t_k).$$

Binäärisen luokittelun erikoistapauksessa käytetään useimmiten *binääristä ristientropiaa* (*binary cross-entropy*). Tällöin verkon ulostulokerroksessa on yksi neuroni, jonka arvo on $t \in (0, 1)$. Tulkitaan tätä siten, että ensimmäisen kategorian todennäköisyys $P(y_1) = t$ ja toisen kategorian todennäköisyys $P(y_2) = 1 - t$. Nyt saadaan virhefunktioksi

$$\begin{aligned} E(t, \boldsymbol{y}) &= -(y_1 \ln(t) + y_2 \ln(1 - t)) \\ &= -(y \ln(t) + (1 - y) \ln(1 - t)). \end{aligned}$$

Kokoa N olevan opetusjoukon virhe on yksittäisten pisteiden virheiden keskiarvo [6]

$$\begin{aligned} C(\boldsymbol{W}) &= \frac{1}{N} \sum_{n=1}^{N} E(f(\boldsymbol{x}^{(n)}; \boldsymbol{W}), \boldsymbol{y}^{(n)}) \\ &= \frac{1}{N} \sum_{n=1}^{N} E(\boldsymbol{t}^{(n)}, \boldsymbol{y}^{(n)}). \end{aligned}$$

Täten saadaan opetusjoukon virhefunktioiksi

$$C_{MSE}(\boldsymbol{W}) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{k=1}^{d} (t_k^{(n)} - y_k^{(n)})^2 \qquad \text{ja}$$

$$C_{CE}(\boldsymbol{W}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{d} y_k^{(n)} \ln(t_k^{(n)}),$$

jossa C_{MSE} tarkoittaa *keskineliövirhettä* (*mean squared error*).

Jatkossa merkinnällä $E^{(n)}(\boldsymbol{W})$ tarkoitetaan arvojen $\boldsymbol{x}^{(n)}$ ja $\boldsymbol{y}^{(n)}$ vaikutusta virhefunktioon

$$E^{(n)}(\boldsymbol{W}) = E(f(\boldsymbol{x}^{(n)}; \boldsymbol{W}), \boldsymbol{y}^{(n)}).$$

Esimerkki 2. Jatketaan aiempaa esimerkkiä ja lasketaan pisteiden virheet käyttämällä virhefunktiona ristientropiaa. Pisteelle $\boldsymbol{x} = (1, 1)$ saadaan

$$\begin{aligned} E(t, y) &= -(0 \cdot \ln(0.562) + 1 \cdot \ln(1 - 0.562)) \\ &= -\ln(0.438) \\ &\approx 0.826. \end{aligned}$$

Muille pisteille saadaan laskettua seuraavat arvot:

x_1	x_2	y	$t = \sigma(z^2)$	$E(t, y)$
1	1	0	0.562	0.826
1	0	1	0.5	$-\ln(0.5) \approx 0.693$
0	1	1	0.5	0.693
0	0	0	0.438	0.576

Neuroverkon kokonaisvirheeksi saadaan lopulta

$$C_{CE}(\boldsymbol{W}) \approx \frac{0.826 + 0.693 + 0.693 + 0.576}{4} = 0.6970.$$

4.4 Vastavirta-algoritmi

Neuroverkon opettamisen tavoitteena on minimoida virhefunktio $C(\boldsymbol{W})$. Monet optimointimenetelmät edellyttävät tavoitefunktion gradientin laskemista. Neuroverkkojen virhefunktion gradientit saadaan hyödyntämällä derivoinnin ketjusääntöä rekursiivisesti. Tätä menetelmää kutsutaan *vastavirta-algoritmiksi* (*backpropagation*). Nimitys tulee siitä, että osittaisderivaattojen laskeminen aloitetaan ulostulokerroksesta, minkä jälkeen ketjusääntöä sovelletaan taaksepäin kunnes saadaan osittaisderivaatat ensimmäisen kerroksen parametrien $\boldsymbol{W}$ suhteen. [4]

Tavoitteena on siis saada laskettua virhefunktion gradientti kaikkien kerroksien painojen suhteen. Kokonaisvirheen gradientti saadaan yksittäisten pisteiden gradienttien keskiarvona [4]

$$\begin{aligned} \nabla C(\boldsymbol{W}) &= \nabla \left[\frac{1}{N} \sum_{n=1}^{N} E^{(n)}(\boldsymbol{W}) \right] \\ &= \frac{1}{N} \sum_{n=1}^{N} \nabla E^{(n)}(\boldsymbol{W}). \end{aligned}$$

Lähteessä [4] on esitelty tapa laskea virheen gradientti, jos aktivointifunktio on vektoroitu yhden muuttujan funktio. Esitetään aluksi siitä tuloksesta kaikille aktivointifunktioille yleistetty tulos. Sen jälkeen esitellään teoksessa [4] oleva erikoistapaus. Vastavirta-algoritmi on esitelty myös esimerkiksi teoksissa [6, 7].

Lause 2. Olkoon $\delta_i^l = \dfrac{\partial E}{\partial z_i^l}$ kerroksessa l olevan neuronin i osuus yksittäisen opetuspisteen virheestä eli

$$\boldsymbol{\delta}^l = \left(\frac{\partial E}{\partial \boldsymbol{z}^l} \right)^T.$$

Tällöin virheen osittaisderivaatat saadaan laskettua rekursiivisesti kaavoilla:

$$\begin{aligned}
\boldsymbol{\delta}^L &= \left(J_{g_l}(\boldsymbol{z}^L)\right)^T \nabla_a E \\
\boldsymbol{\delta}^l &= \left(J_{g_l}(\boldsymbol{z}^L)\right)^T (\boldsymbol{W}^{l+1})^T \boldsymbol{\delta}^{l+1} \\
\frac{\partial E}{\partial w_{ij}^l} &= \delta_i^l a_j^{l-1} \\
\frac{\partial E}{\partial b_i^l} &= \frac{\partial E}{\partial w_{i0}^l} a_0^{l-1} = \delta_i^l.
\end{aligned}$$

Merkinnällä $\nabla_a E$ tarkoitetaan vektoria, jonka arvot ovat virheen osittaisderivaatat ulostulokerroksen alkioiden suhteen

$$\nabla_a E = \left(\frac{\partial E}{\partial \boldsymbol{a}^L}\right)^T.$$

Todistus. Osoitetaan lause käyttämällä ketjusääntöä.
Kun $l = L$, saadaan

$$\begin{aligned}
(\boldsymbol{\delta}^L)^T &= \frac{\partial E}{\partial \boldsymbol{z}^L} \\
&= \frac{\partial E}{\partial \boldsymbol{a}^L} \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \\
&= (\nabla_a E)^T \frac{\partial g_L(\boldsymbol{z}^L)}{\partial \boldsymbol{z}^L} \\
&= (\nabla_a E)^T J_{g_l}(\boldsymbol{z}^L).
\end{aligned}$$

Täten

$$\boldsymbol{\delta}^L = \left(J_{g_l}(\boldsymbol{z}^L)\right)^T \nabla_a E.$$

Oletetaan seuraavaksi, että $\boldsymbol{\delta}^{l+1}$ on tiedossa. Nyt

$$\begin{aligned}
(\boldsymbol{\delta}^l)^T &= \frac{\partial E}{\partial \boldsymbol{z}^l} \\
&= \frac{\partial E}{\partial \boldsymbol{z}^{l+1}} \frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{z}^l} \\
&= (\boldsymbol{\delta}^{l+1})^T \frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{z}^l} \\
&= (\boldsymbol{\delta}^{l+1})^T \frac{\partial (\boldsymbol{W}^{l+1} \boldsymbol{a}^l)}{\partial \boldsymbol{z}^l} \\
&= (\boldsymbol{\delta}^{l+1})^T \boldsymbol{W}^{l+1} \frac{\partial g_l(\boldsymbol{z}^l)}{\partial \boldsymbol{z}^l} \\
&= (\boldsymbol{\delta}^{l+1})^T \boldsymbol{W}^{l+1} J_{g_l}(\boldsymbol{z}^l).
\end{aligned}$$

Siis

$$\begin{aligned}
\boldsymbol{\delta}^l &= \left(J_{g_l}(\boldsymbol{z}^L)\right)^T \left((\boldsymbol{\delta}^{l+1})^T \boldsymbol{W}^{l+1}\right)^T \\
&= \left(J_{g_l}(\boldsymbol{z}^L)\right)^T (\boldsymbol{W}^{l+1})^T \boldsymbol{\delta}^{l+1}.
\end{aligned}$$

Osoitetaan vielä kolmas kaava. Ketjusäännön nojalla

$$\frac{\partial E}{\partial \boldsymbol{W}^l} = \frac{\partial E}{\partial \boldsymbol{z}^l}\frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{W}^l} = \boldsymbol{\delta}^l \frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{W}^l}.$$

Mielivaltainen Jacobin matriisin $\dfrac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{W}^l}$ alkio on

$$\frac{\partial z_k^l}{\partial w_{ij}^l} = \frac{\partial(\sum_{n=0}^{n_l} w_{kn}^l a_n^{l-1})}{\partial w_{ij}^l} = \begin{cases} a_j^{l-1}, & \text{jos } k = i, \\ 0, & \text{jos } k \neq i. \end{cases}$$

Täten saadaan lopulta

$$\frac{\partial E}{\partial w_{ij}^l} = \sum_{k=1}^{n_l} \delta_k^l \frac{\partial z_k^l}{\partial w_{ij}^l} = \delta_i^l a_j^{l-1}.$$

□

Lause 3. Olkoon $\boldsymbol{W}_1$ painomatriisi $\boldsymbol{W}$, josta on poistettu vakiotermisarake. Jos kerroksen l aktivointifunktiona on vektoroitu yhden muuttujan funktio, niin vektori $\boldsymbol{\delta}^l$ saadaan kaavoilla

$$\boldsymbol{\delta}^l = \begin{cases} \nabla_a E \odot g_L'(\boldsymbol{z}^L) & \text{jos } l = L, \\ ((\boldsymbol{W}_1^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot g_l'(\boldsymbol{z}^l) & \text{jos l} < \text{L}, \end{cases}$$

jossa

$$g_l'(\boldsymbol{z}^l) = (g_l'(z_1^l), g_l(z_2^l), \ldots, g_l'(z_{n_l}^l))^T.$$

Todistus. Lähteessä [4] kaavat on osoitettu käyttämällä ketjusääntöä virheisiin δ_i^l. Osoitetaan tässä kaavat hieman eri tavalla käyttämällä Jacobin matriiseja. Merkitään $\boldsymbol{D^l} = \operatorname{diag}(g_l'(z_1^l), \ldots, g_l'(z_{n_l}^l))$.

Kun $l = L$, saadaan

$$\begin{aligned} \frac{\partial g_L(\boldsymbol{z}^L)}{\partial \boldsymbol{z}^L} &= \begin{bmatrix} \dfrac{\partial g_L(z_1^L)}{\partial z_1^L} & \cdots & \dfrac{\partial g_L(z_1^L)}{\partial z_{n_L}^L} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial g_L(z_{n_L}^L)}{\partial z_1^L} & \cdots & \dfrac{\partial g_L(z_{n_L}^L)}{\partial z_{n_L}^L} \end{bmatrix} \\ &= \begin{bmatrix} g_L'(z_1^L) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & g_L'(z_{n_L}^L) \end{bmatrix} \\ &= \operatorname{diag}(g_L'(z_1^L), \ldots, g_L'(z_{n_L}^L)). \end{aligned}$$

Edellisen lauseen ja lemman 1 nojalla

$$\begin{aligned}\boldsymbol{\delta}^L &= \left(J_{g_l}(\boldsymbol{z}^L)\right)^T \nabla_a E \\ &= (\boldsymbol{D}^L)^T \nabla_a E \\ &= \boldsymbol{D}^L \nabla_a E \\ &= g_L'(\boldsymbol{z}^L) \odot \nabla_a E \\ &= \nabla_a E \odot g_L'(\boldsymbol{z}^L).\end{aligned}$$

Kun $l < L$, saadaan

$$\begin{aligned}\frac{\partial g_L(\boldsymbol{z}^l)}{\partial \boldsymbol{z}^l} &= \begin{bmatrix} \frac{\partial 1}{\partial z_1^l} & \cdots & \frac{\partial 1}{\partial z_1^l} \\ \frac{\partial g_l(z_1^l)}{\partial z_1^l} & \cdots & \frac{\partial g_l(z_1^l)}{\partial z_{n_l}^l} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_l(z_{n_L}^L)}{\partial z_1^L} & \cdots & \frac{\partial g_l(z_{n_L}^L)}{\partial z_{n_L}^L} \end{bmatrix} \\ &= \begin{bmatrix} 0 & \dots & 0 \\ g_l'(z_1^l) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & g_l'(z_{n_l}^l) \end{bmatrix}.\end{aligned}$$

Edellisen lauseen perusteella

$$(\boldsymbol{\delta}^l)^T = (\boldsymbol{\delta}^{l+1})^T \boldsymbol{W}^{l+1} J_{g_l}(\boldsymbol{z}^l).$$

Huomataan, että Jacobin matriisin $J_{g_l}(\boldsymbol{z}^l)$ ensimmäinen rivi on nollarivi. Tästä seuraa, että painomatriisin ensimmäinen sarake ja Jacobin matriisin ensimmäinen rivi ovat matriisitulossa $\boldsymbol{W}^{l+1} J_{g_l}(\boldsymbol{z}^l)$ merkityksettömiä. Tällöin

$$\begin{aligned}\boldsymbol{W}^{l+1} J_{g_l}(\boldsymbol{z}^l) &= \boldsymbol{W}_1^{l+1} \mathrm{diag}(g_l'(z_1^l), \dots, g_l'(z_{n_l}^l)) \\ &= \boldsymbol{W}_1^{l+1} \boldsymbol{D}^l.\end{aligned}$$

Nyt

$$\begin{aligned}\boldsymbol{\delta}^l &= \left((\boldsymbol{\delta}^{l+1})^T \boldsymbol{W}_1^{l+1} \boldsymbol{D}^l\right)^T \\ &= (\boldsymbol{D}^l)^T ((\boldsymbol{\delta}^{l+1})^T \boldsymbol{W}_1^{l+1})^T \\ &= \boldsymbol{D}^l (\boldsymbol{W}_1^{l+1})^T \boldsymbol{\delta}^{l+1} \\ &= g_L'(\boldsymbol{z}^L) \odot ((\boldsymbol{W}_1^{l+1})^T \boldsymbol{\delta}^{l+1}) \\ &= ((\boldsymbol{W}_1^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot g_L'(\boldsymbol{z}^L).\end{aligned}$$

□

Jälkimmäinen tapa laskea vektorit $\boldsymbol{\delta}$ on huomattavasti tehokkaampi ja sitä kannattaa käyttää implementaatioissa, jos mahdollista. Tehokkaammaan siitä tekevät seuraavat ominaisuudet:

1. Vältetään Jacobin matriisin transponointi ($\mathcal{O}(m^2)$).
2. Vektorin $g_l'(\boldsymbol{z}^l)$ luominen ($\mathcal{O}(m)$) on nopeampaa kuin matriisin luominen ($\mathcal{O}(m^2)$).
3. Hadamardin tulo ($\mathcal{O}(m^2)$) on nopeampi laskea kuin matriisitulo lävistäjämatriisin kanssa ($\mathcal{O}(m^3)$).

Tavallisimmin käytetyistä aktivointifunktioista kaikki paitsi Softmax-funktio tukevat lauseen 3 tapaa.

Algoritmi 1 (Vastavirta-algoritmi)**.** Olkoon kokonaisvirheen gradientti ∇C aluksi nollavektori.
Toistetaan kaikille opetusjoukon pisteille n:

Askel 1: Lasketaan kaikille kerroksille $l = 1, 2, \ldots, L$ neuronien aktivoidut summat

$$\begin{aligned} \boldsymbol{z}^{n,l} &= \boldsymbol{W}^l \boldsymbol{a}^{n,l-1} \\ \boldsymbol{a}^{n,l} &= g_l(\boldsymbol{z}^{\boldsymbol{n,l}}). \end{aligned}$$

Askel 2: Lasketaan ulostulokerroksen virheet

$$\boldsymbol{\delta}^{n,L} = \nabla_a E^{(n)} \odot g_L'(\boldsymbol{z}^{n,L}).$$

Askel 3: Lasketaan kaikille kerroksille $L-1, L-2, \ldots, 1$ virheet

$$\boldsymbol{\delta}^{n,l} = ((\boldsymbol{W}_1^{l+1})^T \boldsymbol{\delta}^{n,l+1}) \odot g_l'(\boldsymbol{z}^{n,l}).$$

Askel 4: Muodostetaan gradientti saaduista virheistä kaavoilla

$$\begin{aligned} \frac{\partial E^{(n)}}{\partial w_{ij}^l} &= \delta_i^{n,l} a_j^{n,l-1}, \text{ ja} \\ \frac{\partial E^{(n)}}{\partial b_i^l} &= \delta_i^{n,l}. \end{aligned}$$

Askel 5: Päivitetään kokonaisvirheen gradientti

$$\nabla C \leftarrow \nabla C + \frac{1}{n} \nabla E^{(n)}.$$

Huomionarvoista algoritmin implementoinnissa on se, että matriisit $(\boldsymbol{W}_1^l)^T$ kannattaa luoda koko opetusjoukolle kerran, eikä jokaiselle pisteelle erikseen. Täten vältetään hidas operaatio yksittäisten opetuspisteiden gradienttien laskemisessa.

Esimerkki 3. Lasketaan edellisen esimerkin virhefunktion gradientti käyttämällä vastavirta-algoritmia. Askel 1 on suoritettu jo aiemmin, ja sen tulokset ovat seuraavassa taulukossa.

x_1	x_2	y	z^1	a^1	z^2	$t = \sigma(z^2)$
1	1	0	(1.5, -1.5)	(1, 1.5, 0)	0.25	0.562
1	0	1	(1, -1)	(1, 1, 0)	0	0.5
0	1	1	(1, -1)	(1, 1, 0)	0	0.5
0	0	0	(0.5, -0.5)	(1, 0.5, 0)	-0.25	0.438

Askel 2: Lasketaan aluksi $g'_L(\boldsymbol{z}^{n,L}) = \sigma'(\boldsymbol{z}^{n,2})$. Sigmoid-funktion derivaatta on tunnetusti

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

Täten saadaan

$$\sigma'(\boldsymbol{z^{n,2}}) = \left[\sigma(z^{n,2})(1 - \sigma(z^{n,2}))\right] = \sigma(z^{n,2})(1 - \sigma(z^{n,2})).$$

Lasketaan seuraavaksi $\nabla_a C$. Jätetään opetuspisteen $(x^{(n)}, y^{(n)})$ indeksit n merkitsemättä selkeyden vuoksi. Merkitään $a^2 = t$. Saadaan

$$\begin{aligned}
\nabla_a E &= \left[\frac{\partial E}{\partial t}\right] \\
&= \frac{d}{dt}\left[-(y\ln(t) + (1-y)\ln(1-t))\right] \\
&= -\frac{y}{t} + \frac{1-y}{-(1-t)} \\
&= \frac{t-y}{t-t^2}.
\end{aligned}$$

Nyt

$$\begin{aligned}
\delta^2 &= \nabla_a E \odot g'_2(\boldsymbol{z}^{(2)}) \\
&= \frac{t-y}{t-t^2}\sigma(z^{(2)})(1 - \sigma(z^{(2)})) \\
&= \frac{t-y}{t-t^2}t(1-t) \\
&= t - y.
\end{aligned}$$

Muille pisteille saadaan laskettua vastaavat tulokset.

x_1	x_2	y	t	δ^2
1	1	0	0.562	0.562
1	0	1	0.5	-0.5
0	1	1	0.5	-0.5
0	0	0	0.438	0.438

Askel 3: Lasketaan virheet $\boldsymbol{\delta}^l$ kerroksille $l = 1$. Saadaan laskettua [1]

$$\begin{aligned}\boldsymbol{\delta}^1 &= ((\boldsymbol{W}_1^2)^T \delta^2) \odot \mathrm{ReLu}'(\boldsymbol{z}^1) \\ &= \left(\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \cdot \delta^2 \right) \odot \begin{bmatrix} \mathrm{ReLu}'(z_1^1) \\ \mathrm{ReLu}'(z_2^1) \end{bmatrix} \\ &= 0.5\delta^2 \begin{bmatrix} \mathrm{ReLu}'(z_1^1) \\ \mathrm{ReLu}'(z_2^1) \end{bmatrix}.\end{aligned}$$

Ensimmäiselle pisteelle saadaan

$$\begin{aligned}\delta^{1,1} &= 0.5 \cdot 0.562 \begin{bmatrix} \mathrm{ReLu}'(1.5) \\ \mathrm{ReLu}'(-1.5) \end{bmatrix} \\ &= 0.281 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0.281 \\ 0 \end{bmatrix},\end{aligned}$$

ja vastaavasti muille pisteille

x_1	x_2	δ^1
1	1	(0.281, 0)
1	0	(-0.25, 0)
0	1	(-0.25, 0)
0	0	(0.219, 0)

.

Askel 4: Muodostetaan gradientit saaduista virheistä. Gradientti on yksittäiselle pisteelle

$$\begin{aligned}\nabla E &= \begin{bmatrix} \frac{\partial E}{\partial b_1^1} & \frac{\partial E}{\partial w_{11}^1} & \frac{\partial E}{\partial w_{12}^1} & \frac{\partial E}{\partial b_2^1} & \frac{\partial E}{\partial w_{21}^1} & \frac{\partial E}{\partial w_{22}^1} & \frac{\partial E}{\partial b_1^2} & \frac{\partial E}{\partial w_{11}^2} & \frac{\partial E}{\partial w_{12}^2} \end{bmatrix}^T \\ &= \begin{bmatrix} \delta_1^1 & \delta_1^1 a_1^0 & \delta_1^1 a_2^0 & \delta_2^1 & \delta_2^1 a_1^0 & \delta_2^1 a_2^0 & \delta_1^2 & \delta_1^2 a_1^1 & \delta_1^2 a_2^1 \end{bmatrix}^T.\end{aligned}$$

Seuraavassa taulukossa esitetään jokaisen opetuspisteen virheiden gradientit $\nabla E^{(n)}$ ja kokonaisvirheen gradientti ∇C.

∇E	b_1^1	w_{11}^1	w_{12}^1	b_2^1	w_{21}^1	w_{22}^1	b_1^2	w_{11}^2	w_{12}^2
	0.281	0.281	0.281	0	0	0	0.562	0.843	0
	-0.25	-0.25	0	0	0	0	-0.5	-0.5	0
	-0.25	-0.25	0	0	0	0	-0.5	-0.5	0
	0.219	0	0	0	0	0	0.438	0.219	0
∇C	0	0.008	0.008	0	0	0	0	0.016	0

[1] ReLu-funktion derivaatta ei ole määritelty nollassa, mutta useimmiten käytännössä käytetään arvoa $\mathrm{ReLu}'(0) = 0$. Tällä ei pitäisi olla suurta merkitystä lopputulokseen, sillä käytetyt luvut ovat todellisuudessa liukulukuja ja siten approksimaatioita. Tällöin valinta $\mathrm{ReLu}'(\approx 0) = 0$ on järkevä. [1]

Huomataan, että suurin osa termeistä on nollia, joten osa neuroneista on kokonaan merkityksettömiä tuloksen kannalta. Neuronit ovat siis kuolleita, joten tällä neuroverkolla tulee olemaan suuria haasteita oppia haluttu tehtävä.

Esimerkki 4. Johdetaan kaava vektorille $\boldsymbol{\delta}^L$ tapauksessa, jossa ulostulokerroksen aktivointifuntiona on Softmax-funktio ja virhefunktiona ristientropia. Oletetaan, että odotettu ulostulo $\boldsymbol{y}$ on one-hot -enkoodattu, jolloin $\sum_{i=1}^{n} y_i = 1$. Lasketaan aluksi Softmax-funktion Jacobin matriisin alkiot. Merkitään $\text{Softmax}(\boldsymbol{z}) = S(\boldsymbol{z})$. Jacobin matriisin mielivaltainen alkio on

$$\begin{aligned}\frac{\partial \mathrm{S}_i(\boldsymbol{z})}{\partial z_j} &= \frac{\partial \left(\dfrac{e^{z_i}}{\sum_k e^{z_k}} \right)}{\partial z_j} \\ &= \frac{\left(\dfrac{\partial e^{z_i}}{\partial z_j} \sum_k e^{z_k} \right) - \left(e^{z_i} \dfrac{\partial}{\partial z_j} \sum_k e^{z_k} \right)}{\left(\sum_k e^{z_k} \right)^2}.\end{aligned}$$

Jos $i = j$, saadaan

$$\begin{aligned}\frac{\partial \mathrm{S}_i(\boldsymbol{z})}{\partial z_i} &= \frac{(e^{z_i} \sum_k e^{z_k}) - (e^{z_i} e^{z_i})}{\left(\sum_k e^{z_k} \right)^2} \\ &= \frac{e^{z_i}}{\sum_k e^{z_k}} - \frac{(e^{z_i})^2}{\left(\sum_k e^{z_k} \right)^2} \\ &= S_i(\boldsymbol{z}) - (S_i(\boldsymbol{z}))^2 \\ &= S_i(\boldsymbol{z})(1 - S_i(\boldsymbol{z})).\end{aligned}$$

Jos $i \neq j$, saadaan

$$\begin{aligned}\frac{\partial \mathrm{S}_i(\boldsymbol{z})}{\partial z_i} &= \frac{(0 \cdot \sum_k e^{z_k}) - (e^{z_i} e^{z_j})}{\left(\sum_k e^{z_k} \right)^2} \\ &= -\frac{e^{z_i} e^{z_j}}{\sum_k e^{z_k} \sum_k e^{z_k}} \\ &= -S_i(\boldsymbol{z}) S_j(\boldsymbol{z}).\end{aligned}$$

Edellisen esimerkin tavoin saadaan laskettua virheen osittaisderivaatat ulostulokerroksen suhteen

$$\frac{\partial E}{\partial t_i} = -\frac{y_i}{t_i} = -\frac{y_i}{S_i(\boldsymbol{z})}.$$

Merkitään yksinkertaisuuden vuoksi $S_i(\boldsymbol{z}) = S_i$. Nyt lauseen 2 nojalla

$$\begin{aligned}(\boldsymbol{\delta}^L)^T &= (\nabla_a E)^T \left(J_S(\boldsymbol{z}) \right) \\ &= \begin{bmatrix} -\dfrac{y_1}{S_1} & \cdots & -\dfrac{y_n}{S_n} \end{bmatrix} \begin{bmatrix} S_1(1 - S_1) & \cdots & -S_1 S_n \\ -S_2 S_1 & \cdots & -S_2 S_n \\ \vdots & \ddots & \vdots \\ -S_n S_1 & \cdots & S_n(1 - S_n) \end{bmatrix}.\end{aligned}$$

Mielivaltaiselle indeksille i saadaan

$$\begin{aligned}\delta_i^L &= y_1 S_i + y_2 S_i + \cdots - y_i(1 - S_i) \cdots + y_n S_i \\ &= -y_i + S_i \sum_{j=1}^{n} y_j \\ &= S_i - y_i \\ &= t_i - y_i.\end{aligned}$$

Neuroverkon toteutuksessa voidaan käyttää tätä tulosta suoraan. Tällöin ei tarvitse laskea Jacobin matriisin ja vektorin $\nabla_a E$ laskennallisesti raskasta tuloa.

5 Virhefunktion optimointi

Tässä luvussa käydään läpi erilaisia algoritmeja, joilla voidaan minimoida virhefunktio $C(\boldsymbol{W})$. Merkitään jatkossa $\boldsymbol{d_k} = \nabla C(\boldsymbol{W_k})$. Optimointiongelma on formaalisti

$$\begin{aligned} \min \quad & C(\boldsymbol{W}) \\ \text{s.t} \quad & w_{ij}^{l} \in \mathbb{R}. \end{aligned}$$

Kyseessä on siis moniuloitteinen ja rajoitteeton optimointiongelma. Tällä minimointiongelmalla on selvästi ratkaisu, sillä C on alhaalta päin rajoitettu: $C \geq 0$.

5.1 Gradienttimenetelmä

Gradienttimenetelmä tai *nopeimman laskeutumisen menetelmä* (*gradient descent/ steepest descent*) on Cauchyn vuonna 1847 kehittämä iteratiivinen optimointimenetelmä. Tavallista gradienttimenetelmää kutsutaan myös nimellä *satsigradienttimenetelmä* (*batch gradient descent*) [1]. Gradienttimenetelmässä iteraation seuraava piste $\boldsymbol{w_{k+1}}$ saadaan siirtymällä *askelpituuden* $\epsilon_k > 0$ verran negatiivisen gradientin suuntaan [5]

$$\boldsymbol{W_{k+1}} = \boldsymbol{W_k} - \epsilon_k \nabla C(\boldsymbol{W_k}) = \boldsymbol{W_k} - \epsilon_k \boldsymbol{d_k}.$$

Jos tavoitefunktio on alhaalta päin rajoitettu, gradienttimenetelmä konvergoi aina lokaaliin minimiin. Jos funktio on lisäksi konveksi, niin gradienttimenetelmä löytää globaalin minimin. [5]

Koneoppimisessa askelpituudesta käytetään usein nimitystä *oppimisnopeus* [1]. Askelpituuden määräämisessä voidaan käyttää useita erilaisia tapoja. Optimaalinen askelpituus saadaan minimoimalla funktio

$$\phi(\epsilon_k) = C(\boldsymbol{w_k} + \epsilon_k \boldsymbol{d_k}), \text{ jossa } \epsilon_k > 0.$$

Tätä menetelmää kutsutaan *viivahauksi* [5]. Useimmiten tarkka viivahaku on koneoppimisessa liian hidasta, minkä vuoksi oppimisnopeus voidaan määrätä vakioksi ϵ. Viivahaku voidaan tehdä myös epätarkasti käyttämällä esimerkiksi Armijon sääntöä [5]. Lisäksi yleinen käytäntö on määrätä *oppimisnopeuden aikataulu* (*learning rate schedule*), jossa iteraation k oppimisnopeus ϵ_k saadaan vähenevästä lukujonosta [1].

Gradienttimenetelmä on neuroverkkojen optimoinnissa hidas menetelmä. Algoritmi hidastuu sitä mukaa, kun opetusjoukko kasvaa. Useimmiten optimoinnissa käytetään stokastisia algoritmeja, joissa virhefunktio muodostetaan jokaisella iteraatiolla satunnaisesta opetusjoukon osajoukosta. Tätä osajoukkoa kutsutaan *minisatsiksi* (*minibatch*). Minisatsien käyttämisen etu on, että virhefunktion gradientin laskemiseen käytetty aika ei kasva, vaikka opetusjoukon alkioiden määrä kasvaisi. [1]

Esimerkki 5. Optimoidaan XOR-esimerkin virhe käyttämällä gradienttimenetelmää. Aiemmin huomattiin, että suurin osa neuroneista oli kuolleita. Korjataan tämä siten, että vaihdetaan ensimmäisen kerroksen aktivointifunktio Sigmoid-funktioksi.

Valitaan oppisnopeudeksi vakio $\epsilon = 1$. Nyt ensimmäisen iteraation gradientiksi saadaan

$$\begin{aligned}\boldsymbol{d}_1 &= \begin{bmatrix}\begin{bmatrix}\frac{\partial C}{\partial b_1^1} & \frac{\partial C}{\partial w_{11}^1} & \frac{\partial C}{\partial w_{12}^1}\\ \frac{\partial C}{\partial b_2^1} & \frac{\partial C}{\partial w_{21}^1} & \frac{\partial C}{\partial w_{22}^1}\end{bmatrix} & \begin{bmatrix}\frac{\partial C}{\partial b_1^2} & \frac{\partial C}{\partial w_{11}^2} & \frac{\partial C}{\partial w_{12}^2}\end{bmatrix}\end{bmatrix}\\ &= \begin{bmatrix}\begin{bmatrix}-0.0197 & -0.0241 & -0.0241\\ -0.0197 & -0.0241 & -0.0241\end{bmatrix} & \begin{bmatrix}-0.122 & -0.108 & 0.108\end{bmatrix}\end{bmatrix}.\end{aligned}$$

Lasketaan painomatriisien päivitykset $\boldsymbol{W}_2 = \boldsymbol{W}_1 - 1 \cdot \boldsymbol{d}_1$. Saadaan

$$\begin{aligned}\boldsymbol{W}_2^1 &= \begin{bmatrix}0.5197 & 0.5241 & 0.5241\\ -0.4803 & -0.4759 & -0.4759\end{bmatrix}\\ \boldsymbol{W}_2^2 &= \begin{bmatrix}-0.3775 & 0.6080 & 0.3920\end{bmatrix}.\end{aligned}$$

Näillä painoilla verkon virhe on 0.724 ja luokittelutarkkuus 50 %. Algoritmia jatkamalla saadaan iteraatiolla $k = 78$ luokittelutarkkuus 100 %. Tällöin verkon virhe on 0.348. Lopullisiksi painoiksi saadaan

$$\begin{aligned}\boldsymbol{W}_{78}^1 &= \begin{bmatrix}-0.7610 & 2.557 & 2.557\\ 0.7551 & -0.7304 & -0.7304\end{bmatrix}\\ \boldsymbol{W}_{78}^2 &= \begin{bmatrix}-1.358 & 2.543 & 1.948\end{bmatrix}.\end{aligned}$$

5.2 Ongelmat opettamisessa

Neuroverkkojen virhefunktioita ei voida pitää konvekseina, minkä vuoksi minimointiongelmalla voi olla useita ratkaisuja. On todennäköistä, että neuroverkkojen virheillä on erittäin monta lokaalia minimiä, mikä tekee globaalin optimin löytämisestä haasteellista. [1] On kuitenkin osoittautunut, että suurille neuroverkoille lokaalit minimit ovat usein riittävän hyviä ratkaisuja minimointiongelmaan [10].

Lokaaleja minimejä merkittävämpi ongelma on satulapisteet eli pisteet, joissa Hessen matriisin ominaisarvot saavat positiivisia ja negatiivisia arvoja. Ensimmäisen kertaluvun menetelmät eivät useimmiten juutu satulapisteisiin, mutta toisen kertaluvun menetelmille satulapisteet ovat haaste. [1]

Gradienttipohjaisissa menetelmissä gradientit voivat saada hyvin suuria arvoja, jolloin seuraava askel saattaa mennä todella kauaksi sen hetkisestä tilanteesta. Tällaista ilmiötä kutsutaan *räjähtäviksi gradienteiksi* (*exploding gradients*). Räjähtävät gradientit voivat hidastaa oppimista merkittävästi tai estää oppimisen kokonaan. Tämä ongelma voidaan ratkaista *leikkaamalla* gradientti eli määräämällä askeleelle maksimipituus. Olkoon $c > 0$ gradientin maksimipituus. Ennen painojen päivitystä asetetaan gradientti $\boldsymbol{d}_k$ [1]

$$\boldsymbol{d}_k \leftarrow \begin{cases}\boldsymbol{d}_k & \|\boldsymbol{d}_k\| < c,\\ \dfrac{c\boldsymbol{d}_k}{\|\boldsymbol{d}_k\|} & \|\boldsymbol{d}_k\| \geq c.\end{cases}$$

Toisaalta gradientit saattavat saada hyvin pieniä arvoja, jolloin neuroverkko oppii hitaasti. Tällöin sanotaan, että *gradientit katoavat* (*vanishing gradients*). Katoavia gradientteja esiintyy erityisesti neuroverkoissa, joissa piilokerroksien aktivointina käytetään Sigmoid-funktion tapaisia funktioita. Siksi ReLu-funktio on useimmiten parempi vaihtoehto piilokerroksen aktivointifuntioksi. [1]

5.3 Painojen alustus

Painojen alustuksella voi olla suuri vaikutus optimointialgoritmin suppenemiseen. Huonosti tehty painojen alustus voi esimerkiksi hidastaa oppimista ja aiheuttaa räjähtäviä tai katoavia gradientteja. [1]

Painot
Painot generoidaan yleensä satunnaisesti joko normaalijakaumasta tai tasajakaumasta. Kerrokselle l, jolla on m syötettä ja n neuronia, voidaan generoida painot tasajakaumasta [1]

$$\boldsymbol{W}^l \sim \mathbb{U}\left(-\frac{1}{m}, \frac{1}{m}\right).$$

Usein käytetään myös normalisoitua jakaumaa [11]

$$\boldsymbol{W}^l \sim \mathbb{U}\left(-\frac{1}{\sqrt{m+n}}, \frac{1}{\sqrt{m+n}}\right).$$

Tätä kutsutaan joko Glorot- tai Xavier-alustukseksi.

Vakiotermit
Useimmiten kaikki vakiotermit b_i^l asetetaan pieneksi vakioksi $c \in \mathbb{R}$. Yleinen käytäntö on asettaa kaikkien vakiotermien arvo nollaksi. [1]

5.4 Algoritmien lopetusehdoista

Useimmiten neuroverkon optimoinnissa optimoidaan verkon virhettä opetusjoukon suhteen ja toivotaan, että virhe pienenee sitä mukaa myös testijoukossa. Pienin mahdollinen opetusjoukon virhe ei välttämättä takaa parasta yleistymistä testijoukolle, sillä algoritmi saattaa ylisovittaa. Tämän vuoksi opettaminen on useimmiten järkevää lopettaa siinä vaiheessa, kun testijoukon virhe ei enää pienene. [4]

Kokonaisvirheen laskeminen on hidasta koko opetus- tai testijoukolle, joten sitä ei kannata tehdä jokaisen iteraation jälkeen. Yleinen käytäntö on laskea kokonaisvirhe joukoille aina, kun algoritmi on iteroinut opetusjoukon kertaalleen läpi. Tähän kuluvaa iteraatioiden lukumäärää kutsutaan *epookiksi* (*epoch*). Esimerkiksi, jos opetusjoukon koko on N ja minisatsien koko m, niin yhdessä epookissa on $\lceil \frac{N}{m} \rceil$ iteraatiota. Yleensä valitaan jokin tietty määrä epookkeja, jonka jälkeen opettaminen viimeistään lopetetaan. [4]

Lopettamisehdoiksi voidaan siis valita esimerkiksi:

1. Lopetetaan opettaminen T-epookin jälkeen.
2. Lopetetaan opettaminen, kun algoritmi alkaa ylisovittaa.
3. Lopetetaan opettaminen, kun virhe alittaa jonkin tietyn raja-arvon $\xi > 0$.

5.5 Stokastinen gradienttimenetelmä

Stokastinen gradienttimenetelmä eli *minisatsigradienttimenetelmä* (*Stochastic gradient descent / SGD*) on gradienttimenetelmän variaatio, jossa virhefunktio muodostetaan minisatsista. Tällöin virhefunktiosta saatu gradientti on estimaattori oikeasta gradientista, jolloin gradientissa on kohinaa. Algoritmi ei tämän vuoksi välttämättä koskaan löydä minimiä, vaan jää herkästi oskilloimaan minimin ympärille. Siksi stokastisessa gradienttimenetelmässä kannattaa käyttää vähenevää oppimisnopeutta. [1]

Riittävät ehdot stokastisen gradienttimenetelmän suppenemiselle ovat [1]

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \text{ ja}$$
$$\sum_{k=1}^{\infty} (\epsilon_k)^2 < \infty.$$

Usein oppimisnopeus määritetään lineaarisesti väheneväksi iteraatioon t asti, jonka jälkeen oppimisnopeus pidetään vakiona. Olkoon $k, t \in \mathbb{N}$, $\epsilon_t < \epsilon_0 < 0$ ja $\alpha = \frac{k}{t}$. Nyt oppimisnopeus iteraatiolla k saadaan kaavasta [1]

$$\epsilon_k = \begin{cases} (1-\alpha)\epsilon_0 + \alpha\epsilon_t & k \leq t, \\ \epsilon_t & k > t. \end{cases}$$

Algoritmi 2 (Stokastinen gradienttimenetelmä)**.** Olkoon (ϵ_k) oppimisnopeuden aikataulu ja olkoon $\boldsymbol{W}$ painojen alkuarvot. Olkoon m minisatsin koko.

Askel 1: Muodostetaan opetusjoukosta satunnainen minisatsi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Olkoon $\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}\}$ näitä vastaavat tunnetut ulostulot.

Askel 2: Lasketaan gradientin estimaatti käyttämällä vastavirta-algoritmia. Saadaan

$$\boldsymbol{d} = \frac{1}{m} \sum_{i=1}^{m} \nabla E^{(i)}(\boldsymbol{W}).$$

Askel 3: Lasketaan uudet painot

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \epsilon_k \boldsymbol{d}.$$

Askel 4: Lopetetaan algoritmi, jos lopetusehto on saavutettu. Muutoin asetetaan $k \leftarrow k + 1$ ja siirrytään takaisin askeleeseen 1.

Stokastinen gradienttimenetelmä voi olla hidas konvergoimaan minimiin. Käydään seuraavissa luvuissa läpi nopeampia menetelmiä stokastisten virhefunktioiden optimointiin.

5.6 Momentti

Momentti (*momentum*) on stokastisen gradienttimenetelmän muunnelma, jossa seuraavan iteraation gradientin laskemisessa hyödynnetään aikaisempien iteraatioiden gradientteja. Seuraavaan gradienttiin lisätään ekpsonentiaalisesti hajoava liukuva keskiarvo aikaisemmista gradienteista, jolloin gradientissa olevaa kohinaa saadaan vähennettyä. Algoritmin nimi kuvaa hyvin sen taustalla olevaa ajatusta: algoritmilla on ikään kuin liikemäärä, joka pitää algoritmin lähempänä alkuperäistä suuntaa. Tällöin aikaisemmasta poikkeava suunta ei vie algoritmia niin kauas optimista kuin stokastisessa gradienttimenetelmässä. [1]

Algoritmi 3 (Momentti)**.** Olkoon (ϵ_k) oppimisnopeus ja $\boldsymbol{W}$ painojen alkuarvot. Olkoon m minisatsin koko, $\alpha \in [0, 1)$ momentin kerroin ja $\boldsymbol{v}$ alkunopeus.

Askel 1: Muodostetaan opetusjoukosta satunnainen minisatsi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Olkoon $\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}\}$ näitä vastaavat tunnetut ulostulot.

Askel 2: Lasketaan gradientin estimaatti käyttämällä vastavirta-algoritmia. Saadaan

$$\boldsymbol{d} = \frac{1}{m} \sum_{i=1}^{m} \nabla E^{(i)}(\boldsymbol{W}).$$

Askel 3: Lasketaan uusi nopeus

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon_k \boldsymbol{d}.$$

Askel 4: Lasketaan uudet painot

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{v}.$$

Askel 5: Lopetetaan algoritmi, jos lopetusehto on saavutettu. Muutoin asetetaan $k \leftarrow k + 1$ ja siirrytään takaisin askeleeseen 1.

Yleisiä arvoja parametrille α ovat 0.5, 0.9 ja 0.99. Parametria voidaan kasvattaa iteraatioiden myötä, mutta se ei ole yhtä tärkeää kuin oppimisnopeuden pienentäminen. [1] Algoritmi tuottaa arvolla $\alpha = 0$ saman lopputuloksen kuin stokastinen gradienttimenetelmä.

5.7 Nesterovin momentti

Nesterovin momentti on momentin variaatio, jossa nopeus lisätään painomatriisiin ennen gradienttien laskemista. Tämän tarkoituksena on korjata tavallisessa momentissa nopeustermistä aiheutuvaa hyppäystä, joka saattaa olla liian suuri. [12] Nesterovin momentti ei kuitenkaan nopeuta suppenemista stokastisten gradienttien tapauksessa [1].

Algoritmi 4 (Nesterovin momentti)**.** Olkoon (ϵ_k) oppimisnopeus ja $\boldsymbol{W}$ painojen alkuarvot. Olkoon m minisatsin koko, $\alpha \in [0, 1)$ momentin kerroin ja $\boldsymbol{v}$ alkunopeus.

Askel 1: Muodostetaan opetusjoukosta satunnainen minisatsi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Olkoon $\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}\}$ näitä vastaavat tunnetut ulostulot.

Askel 2: Lasketaan korjattu painomatriisi

$$\tilde{\boldsymbol{W}} \leftarrow \boldsymbol{W} + \alpha \boldsymbol{v}.$$

Askel 3: Lasketaan gradientin estimaatti käyttämällä vastavirta-algoritmia. Saadaan

$$\boldsymbol{d} = \frac{1}{m} \sum_{i=1}^{m} \nabla E^{(i)}(\tilde{\boldsymbol{W}}).$$

Askel 4: Lasketaan uusi nopeus

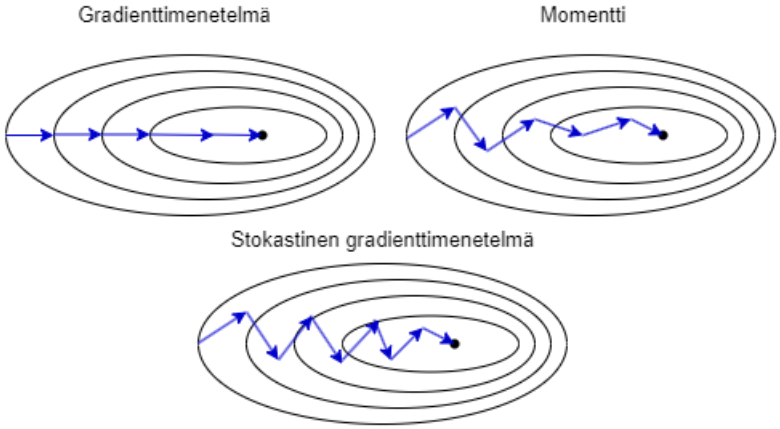
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon_k \boldsymbol{d}.$$

Askel 5: Lasketaan uudet painot

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{v}.$$

Askel 6: Lopetetaan algoritmi, jos lopetusehto on saavutettu. Muutoin asetetaan $k \leftarrow k + 1$ ja siirrytään takaisin askeleeseen 1.

Kuva 2: Gradienttimenetelmiä havainnollistava kuva. Momentti-algoritmi löytää iteraatioiden edetessä parempia suuntia kuin stokastinen gradienttimenetelmä.



Edellä mainittujen algoritmien haasteena on hyvän oppimisnopeuden löytäminen. Tämän ongelman ratkaisuun on kehitetty menetelmiä, joissa oppimisnopeutta sopeutetaan automaattisesti oppimisen edetessä.

5.8 AdaGrad

AdaGrad-algoritmissa (*Adaptive Gradient Algorithm*) [13] jokaiselle opittavalle parametrille on oma oppimisnopeutensa. Näitä oppimisnopeuksia suhteutetaan aiempien gradienttien neliösumman neliöjuureen. Tällöin suuren osittaisderivaatan omaavien parametrien oppimisnopeutta pienennetään nopeasti. Pienen osittaisderivaatan omaavien parametrien nopeutta pienennetään hitaasti. [1]

Algoritmi 5 (AdaGrad)**.** Olkoon ϵ oppimisnopeus ja olkoon $\boldsymbol{W}$ painojen alkuarvot. Olkoon m minisatsin koko ja $\boldsymbol{r} = \boldsymbol{0}$ gradienttien neliösumman alkuarvo. Valitaan nollalla jakamisen välttämiseksi $\lambda > 0$ (usein 10^{-7} tai 10^{-8}).

Askel 1: Muodostetaan opetusjoukosta satunnainen minisatsi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Olkoon $\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}\}$ näitä vastaavat tunnetut ulostulot.

Askel 2: Lasketaan gradientin estimaatti käyttämällä vastavirta-algoritmia. Saadaan

$$\boldsymbol{d} = \frac{1}{m} \sum_{i=1}^{m} \nabla E^{(i)}(\boldsymbol{W}).$$

Askel 3: Lasketaan gradientin neliö ja lisätään se neliösummaan

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{d} \odot \boldsymbol{d}.$$

Askel 4: Muodostetaan muutosvektori

$$\Delta \boldsymbol{W} = -\frac{\epsilon}{\sqrt{\boldsymbol{r} + \lambda}} \odot \boldsymbol{d},$$

jossa $\frac{\epsilon}{\sqrt{\boldsymbol{r} + \lambda}}$ tarkoittaa vektoria, jonka alkiot ovat $\frac{\epsilon}{\sqrt{r_i + \lambda}}$.

Askel 5: Lasketaan uudet painot

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}.$$

Askel 6: Lopetetaan algoritmi, jos lopetusehto on saavutettu. Muutoin asetetaan $k \leftarrow k + 1$ ja siirrytään takaisin askeleeseen 1.

5.9 AdaDelta

AdaDelta-algoritmi [14] pyrkii ratkaisemaan seuraavat AdaGrad-algoritmin ongelmat:

1. parametrien oppimisnopeudet vähenevät jatkuvasti, jolloin oppiminen voi hidastua liikaa tai pysähtyä kokonaan,

2. tarve valita globaali oppimisnopeus.

AdaGrad-algoritmissa kaikkien iteraatioiden gradienttien neliöt pidetään muistissa. AdaDelta-algoritmissa sen sijaan huomioidaan gradientit pelkästään liukuvalta aikaväliltä T. Olkoon $\alpha \in [0, 1)$. Vektorin $\boldsymbol{v}$ neliöille lasketaan eksponentiaalisesti hajoava keskiarvo, joka iteraatiolla k on

$$E(\boldsymbol{v}^2)_k = \alpha E(\boldsymbol{v}^2)_{k-1} + (1-\alpha)\boldsymbol{v}_k \odot \boldsymbol{v}_k.$$

Vektorille $\boldsymbol{v}_k$ saadaan *neliöllinen keskiarvo* (*root mean square*)

$$\mathrm{RMS}(\boldsymbol{v}_k) = \sqrt{E(\boldsymbol{v}^2)_k + \lambda} = \left(\sqrt{E(v^2)_{k1} + \lambda}, \ldots, \sqrt{E(v^2)_{kn} + \lambda}\right)^T.$$

Nyt parametrien muutosvektori on

$$\Delta \boldsymbol{W}_k = -\frac{\mathrm{RMS}(\Delta \boldsymbol{W})_{k-1}}{\mathrm{RMS}(\boldsymbol{d})_k}\boldsymbol{d}_k.$$

Tällä notaatiolla tarkoitetaan vektoria, jonka alkiot ovat

$$\Delta W_{k,i} = -\frac{\mathrm{RMS}(\Delta W)_{k-1,i}}{\mathrm{RMS}(d)_{k,i}}d_{k,i}.$$

Algoritmi 6 (AdaDelta)**.** Olkoon $\boldsymbol{W}$ painojen alkuarvot ja m minisatsin koko. Olkoon $\alpha \in [0, 1)$, $E(\boldsymbol{d}^2) = \boldsymbol{0}$ ja $E(\Delta \boldsymbol{W}^2) = \boldsymbol{0}$. Valitaan nollalla jakamisen välttämiseksi $\lambda > 0$.

Askel 1: Muodostetaan opetusjoukosta satunnainen minisatsi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Olkoon $\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}\}$ näitä vastaavat tunnetut ulostulot.

Askel 2: Lasketaan gradientin estimaatti käyttämällä vastavirta-algoritmia. Saadaan

$$\boldsymbol{d} = \frac{1}{m}\sum_{i=1}^{m} \nabla E^{(i)}(\boldsymbol{W}).$$

Askel 3: Lasketaan gradientin liukuvan keskiarvon seuraava arvo

$$E(\boldsymbol{d}^2) \leftarrow \alpha E(\boldsymbol{d}^2) + (1-\alpha)\boldsymbol{d} \odot \boldsymbol{d}.$$

Askel 4: Muodostetaan muutosvektori $\Delta \boldsymbol{W}$

$$\Delta \boldsymbol{W} = -\frac{\mathrm{RMS}(\Delta \boldsymbol{W})}{\mathrm{RMS}(\boldsymbol{d})}\boldsymbol{d}.$$

Askel 5: Lasketaan muutoksien liukuvan keskiarvon seuraava arvo

$$E(\Delta \boldsymbol{W}^2) \leftarrow \alpha E(\Delta \boldsymbol{W}^2) + (1-\alpha)\Delta \boldsymbol{W}^2 \odot \Delta \boldsymbol{W}^2.$$

Askel 6: Lasketaan uudet painot

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}.$$

Askel 7: Lopetetaan algoritmi, jos lopetusehto on saavutettu. Muutoin asetetaan $k \leftarrow k + 1$ ja siirrytään takaisin askeleeseen 1.

5.10 RMSProp

RMSProp-algoritmi [15] on samankaltainen algoritmi kuin AdaDelta. RMSProp-algoritmissa painojen muutosvektori saadaan kaavasta [16]

$$\Delta \boldsymbol{W}_k = -\frac{\epsilon}{\mathrm{RMS}(\boldsymbol{d})_k}\boldsymbol{d}_k,$$

johon usein valitaan oppimisnopeus $\epsilon = 0.001$ ja $\alpha = 0.9$.

Algoritmi 7 (RMSProp)**.** Olkoon ϵ oppimisnopeus ja $\boldsymbol{W}$ painojen alkuarvot. Olkoon m minisatsin koko, $\alpha \in [0, 1)$ ja $E(\boldsymbol{d}^2) = \boldsymbol{0}$. Valitaan nollalla jakamisen välttämiseksi $\lambda > 0$.

Askel 1: Muodostetaan opetusjoukosta satunnainen minisatsi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Olkoon $\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}\}$ näitä vastaavat tunnetut ulostulot.

Askel 2: Lasketaan gradientin estimaatti käyttämällä vastavirta-algoritmia. Saadaan

$$\boldsymbol{d} = \frac{1}{m}\sum_{i=1}^{m} \nabla E^{(i)}(\boldsymbol{W}).$$

Askel 3: Lasketaan gradientin liukuvan keskiarvon seuraava arvo

$$E(\boldsymbol{d}^2) \leftarrow \alpha E(\boldsymbol{d}^2) + (1 - \alpha)\boldsymbol{d} \odot \boldsymbol{d}.$$

Askel 4: Muodostetaan muutosvektori $\Delta \boldsymbol{W}$

$$\Delta \boldsymbol{W} = -\frac{\epsilon}{\mathrm{RMS}(\boldsymbol{d})}\boldsymbol{d}.$$

Askel 5: Lasketaan uudet painot

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}.$$

Askel 6: Lopetetaan algoritmi, jos lopetusehto on saavutettu. Muutoin asetetaan $k \leftarrow k + 1$ ja siirrytään takaisin askeleeseen 1.

5.11 Adam

Adam-algoritmia (*adaptive moments*) [17] voidaan pitää RMSProp-algoritmin ja momentin yhdistelmänä [1]. Adam-algoritmissa pidetään RMSPropin ja AdaDeltan tavoin kirjaa gradienttien neliösummista, ja momentin tavoin Adam-algoritmissa pidetään kirjaa edellisistä gradienteista. Näissä aiemmin esitellyissä algoritmeissa vektorien alkuarvot alustetaan nollavektoreiksi, mikä aiheuttaa sen, että kyseiset vektorit ovat harhaisia nollaan päin. Adam-algoritmissa pyritään korjaamaan tätä harhaa. [17]

Algoritmi 8 (Adam)**.** Olkoon ϵ oppimisnopeus ja $\boldsymbol{W}$ painojen alkuarvot. Olkoon m minisatsin koko, $\beta_1, \beta_2 \in [0, 1)$, $\boldsymbol{m} = \boldsymbol{0}$ ja $\boldsymbol{r} = \boldsymbol{0}$. Valitaan nollalla jakamisen välttämiseksi $\lambda > 0$.

Askel 1: Muodostetaan opetusjoukosta satunnainen minisatsi $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Olkoon $\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}\}$ näitä vastaavat tunnetut ulostulot.

Askel 2: Lasketaan gradientin estimaatti käyttämällä vastavirta-algoritmia. Saadaan

$$\boldsymbol{d} = \frac{1}{m} \sum_{i=1}^{m} \nabla E^{(i)}(\boldsymbol{W}).$$

Askel 3: Lasketaan seuraavat arvot momentin ja gradientin neliön liukuville keskiarvoille

$$\begin{aligned} \boldsymbol{m} &\leftarrow \beta_1 \boldsymbol{m} + (1 - \beta_1)\boldsymbol{d}, \\ \boldsymbol{r} &\leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\boldsymbol{d} \odot \boldsymbol{d}. \end{aligned}$$

Askel 4: Lasketaan harhakorjatut estimaatit (tässä k on eksponentti)

$$\begin{aligned} \hat{\boldsymbol{m}} &\leftarrow \frac{\boldsymbol{m}}{1 - \beta_1^k}, \\ \hat{\boldsymbol{r}} &\leftarrow \frac{\boldsymbol{r}}{1 - \beta_2^k}. \end{aligned}$$

Askel 5: Lasketaan uudet painot

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \frac{\epsilon \cdot \hat{\boldsymbol{m}}}{\sqrt{\hat{\boldsymbol{m}}} + \lambda}.$$

Askel 6: Lopetetaan algoritmi, jos lopetusehto on saavutettu. Muutoin asetetaan $k \leftarrow k + 1$ ja siirrytään takaisin askeleeseen 1.

Algoritmin kehittäjät suosittelevat oletusarvoiksi $\beta_1 = 0.9$, $\beta_1 = 0.999$, $\lambda = 10^{-8}$ ja $\epsilon = 0.001$.

6 Sovellukset

Vertaillaan edellä esitettyjä optimointialgoritmeja kahden eri luokitteluongelman ratkaisemisessa:

1. Tunnistetaan kuvista käsinkirjoitettuja numeroita.

2. Ennustetaan, kuinka todennäköisesti asiakas peruuttaa hotellivarauksen.

Vertailtavia suureita ovat saavutettu tarkkuus ja suorituskyky.

Kyseiset aineistot valittiin vertailuun, koska ne ovat riittävän suuria neuroverkon opettamiseen, eivätkä ne vaadi juurikaan esikäsittelyä. Ensimmäistä aineistoa käytetään usein koneoppimisalgoritmien vertailussa, joten siihen sovelletusta neuroverkosta saadaan hyvä lähtökohta algoritmien vertailuun. Toinen aineisto valittiin, sillä se on riittävän erilainen verrattuna ensimmäiseen aineistoon. Aineistossa on vain kaksi luokkaa ja merkittävästi vähemmän muuttujia kuin ensimmäisessä. Muuttujista osa on lisäksi kategorisia. Täten on mahdollista, että aineistojen välillä havaitaan eroja eri algoritmien suorituskyvyissä.

Tehtäviä varten ohjelmoitiin neuroverkko alusta asti ilman valmiita kirjastoja. Ohjelmointikieleksi valittiin Javascript, koska Javascriptia ei perinteisesti ole käytetty data-analytiikassa. Sitä kuitenkin voidaan suorittaa selaimessa, joten tätä toteutusta olisi mahdollista hyödyntää helposti esimerkiksi verkkosivustoilla. Vertaillaan myös lyhyesti tätä Javascript-toteutusta Keras-kirjaston antamiin tuloksiin.

6.1 Neuroverkon toteutuksesta

Neuroverkko-ohjelmiston koodin rakennetta on havainnollistettu kuvassa 3. Esitellään seuraavaksi toteutuksen tärkeimmät luokat. Ohjelmiston koodit ovat saatavilla Github-sivustolla. [2]

Matrix
Matrix-luokka on toteutus matriisista. Luokka sisältää toteutukset esimerkiksi matriisitulosta, Hadamardin tulosta ja skalaarilla kertomisesta. Matriisi on toteutettu yksiulotteisena listana, jolloin matriisin $M \in \mathbb{R}^{M \times N}$ alkio m_{ij} saadaan listan indeksistä $i \cdot N + j$. Yksiulotteisesta listasta on nopeampaa hakea alkioita kuin kaksiulotteisesta ja lisäksi yksiulotteinen lista varaa vähemmän muistia.

Dataset
Dataset-luokka on yksinkertainen toteutus neuroverkkoon syötetylle aineistolle. Luokka sisältää listan syöte-ulostulo -pareja ja metodin, jonka avulla voidaan luoda satunnaisia minisatseja.

ActivationFunction
ActivationFunction-luokka sisältää aktivointifunktioiden toteutukset. Aktivointifunktiot on toteutettu määritelmän 9 mukaisina kuvauksina $\mathbb{R}^m \to \mathbb{R}^{m+1}$. Myös vastavirta-algoritmissa käytetty g' on toteutettu jokaiselle aktivointifunktiolle. Tämän hetki-

[2] https://github.com/jussimi/neural/tree/main

sessä toteutuksessa on mahdollista käyttää ReLu-, Sigmoid-, Tanh- tai Softmax-aktivointeja. Softmax-funktiota voidaan käyttää vain ulostulokerroksessa ristientropian kanssa, ja se on toteutettu kuten esimerkissä 4.

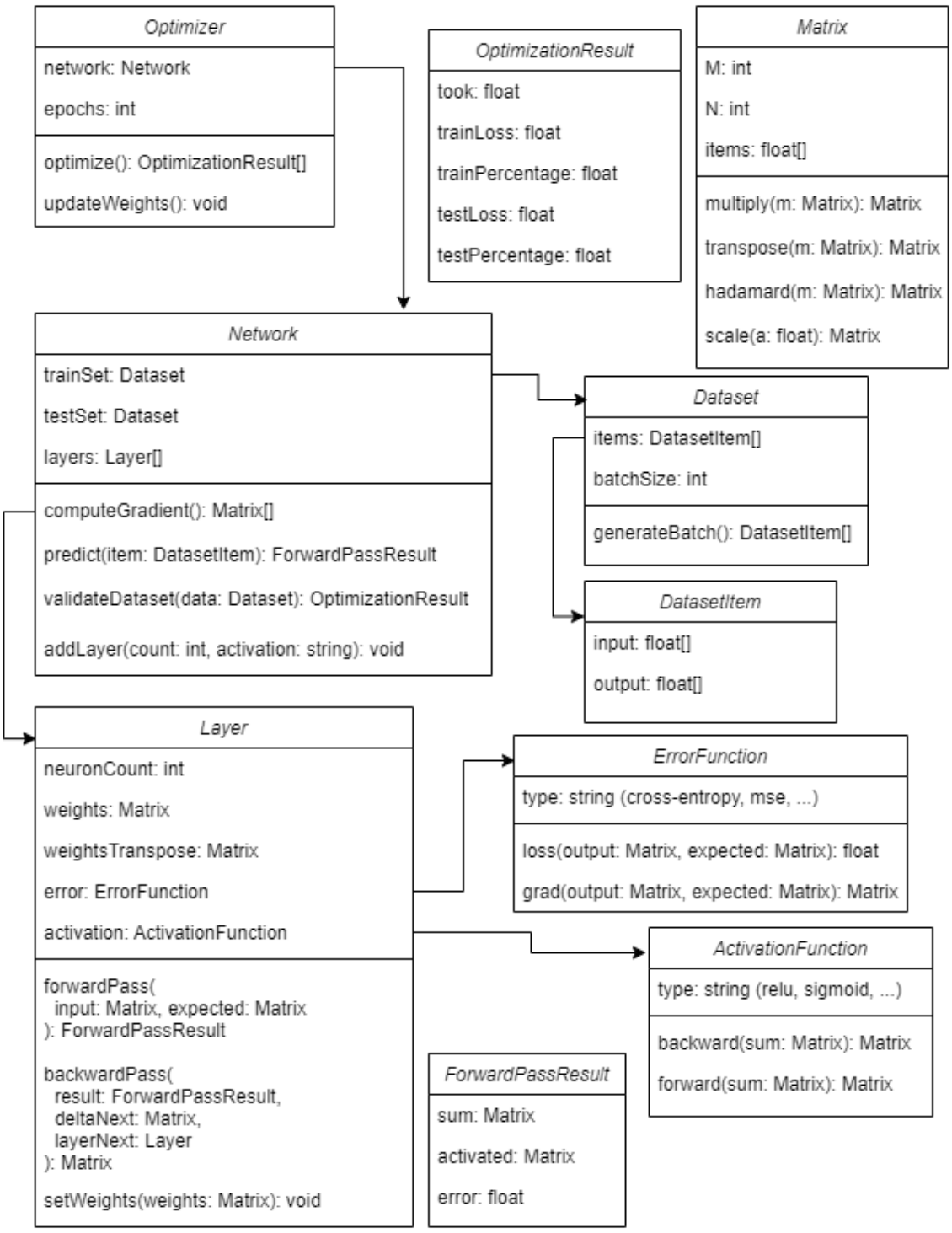
ErrorFunction
ErrorFunction-luokka sisältää virhefunktiot ja niiden gradientit. Toteutuksessa voidaan käyttää ristientropiaa, binääristä ristientropiaa tai keskineliövirhettä.

Layer
Layer-luokka kuvastaa neuroverkon kerrosta. Kerrokselle on määritelty neuronien lukumäärä, painomatriisi ja kerroksessa käytetty aktivointifuntkio. Kerroksilla on myös painomatriisin transpoosi, joka lasketaan silloin, kun painoja muutetaan. Layer-luokalla on metodit, joilla lasketaan kerroksen ulostulot ja vastavirta-algoritmin virheet $\boldsymbol{\delta}$ ja $\nabla_a E$.

Network
Network-luokka kuvastaa varsinaista neuroverkkoa. Verkon attribuutteja ovat testi- ja opetusjoukot sekä kerrokset. Network-luokka sisältää lausetta 3 vastaavan toteutuksen vastavirta-algoritmista ja metodin, joka laskee kokonaisvirheen ja luokittelutarkkuuden testi- tai opetusjoukolle.

Optimizer
Optimizer-luokka on abstrakti luokka, jonka vastuulla on minimoida verkon virhe. Jokaista iteraatiota ennen luodaan uusi minisatsi, ja jokaisen epookin jälkeen lasketaan virheet testi- ja opetusjoukolle. Erilaiset optimointialgoritmit on tehty perimällä Optimizer-luokka. Perivät luokat toteuttavat painojen päivityksen kyseisten algoritmien mukaan.

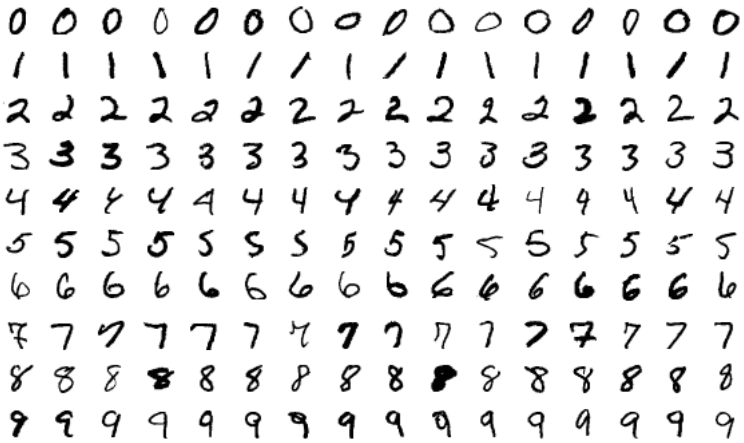
Kuva 3: Suuntaa antava kaavio neuroverkon koodin rakenteesta



6.2 Käsinkirjoitettujen numeroiden tunnistaminen

Sovelletaan yllä olevia algoritmeja kuvantunnistusongelmaan, jossa on tarkoituksena tunnistaa käsin kirjoitettuja numeroita. Aineistona käytetään Mnist-aineistoa. [3] Siinä olevat kuvat ovat mustavalkoisia kuvia, joissa on 28x28 pikseliä ja tulokset ovat numeroita $0 - 9$. Mahdollisia luokkia on siis yhteensä 10. Aineisto on jaettu opetus- ja testijoukkoihin. Opetusjoukossa on 60000 kuvaa ja testijoukossa 10000 kuvaa. Jokainen luokka kattaa noin 10 % aineistosta eli aineiston luokat ovat tasapainoisia.

Kuva 4: Esimerkkejä Mnist-aineistosta. [18]



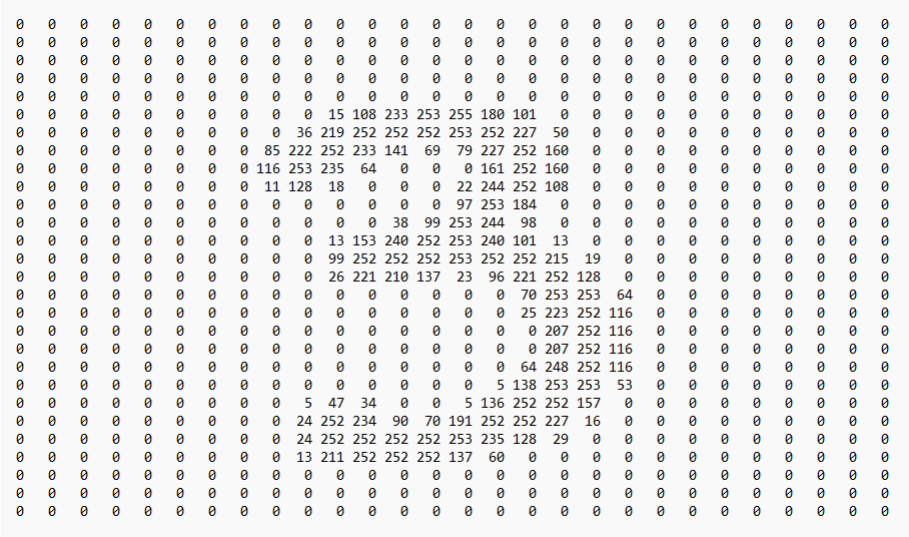
Yksittäinen aineiston piste on vektori, jossa on $1 + 28 \cdot 28 = 785$ alkiota. Vektorin ensimmäinen arvo kertoo tunnetun ulostulon ja loput arvot kertovat pikseleiden kirkkauden. Pikselien arvot sijoittuvat välille $[0, 255]$. Arvo 0 esittää täysin valkoista ja 255 täysin mustaa pikseliä. Aineistoa voidaan käyttää opettamiseen sellaisenaan, mutta syötteet kannattaa normalisoida välille $[0, 1]$, jotta vältetään gradienttien arvojen räjähtäminen. Täten jaetaan jokaisen syötteen kaikki alkiot luvulla 255.

Koska kyseessä on luokittelutehtävä, jossa on enemmän kuin kaksi luokkaa, ulostulokerroksessa käytetään Softmax-aktivointia ja virhefunktiona ristientropiaa. Ulostulot tulee siis one-hot -enkoodata. Yksinkertaisin tapa kyseisen aineiston ulostulojen enkoodaamiseen on asettaa ulostulovektorin $\boldsymbol{y}$ indeksille $n + 1$ arvo 1. Tässä n on syötteen tunnettu luokka. Esimerkiksi luvulle 3 saadaan
enkoodaus $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$.

Vertaillaan aluksi erilaisia neuroverkon rakenteita. Valitaan neuroverkkoon yksi piilokerros ja tutkitaan millaisia tuloksia saadaan, kun vaihdellaan kerroksen aktivointifunktiota ja neuronien lukumäärää. Käytetään opettamiseen aluksi stokastista gradienttimenetelmää tasaisesti vähenevällä oppimisnopeudella. Jatketaan opettamista 20 epookin ajan, ja valitaan minisatsien kooksi 200. Seuraavassa taulukossa esitetään vertailun tulokset. Taulukossa notaatiolla "Sigmoid-128" tarkoitetaan 128 neuronin kerrosta, jonka aktivointifunktiona on Sigmoid-funktio.

[3] https://www.kaggle.com/datasets/oddrationale/mnist-in-csv

Kuva 5: Esimerkki syötevektorista



Kerros	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus
Sigmoid-32	0.2828	0.9209	0.2744	0.9245
Sigmoid-64	0.2734	0.9223	0.2659	0.9254
Sigmoid-128	0.2819	0.9185	0.2749	0.9219
ReLu-32	0.1605	0.9545	0.1667	0.9486
ReLu-64	0.1282	0.9643	0.1377	0.9587
ReLu-128	0.1135	0.9684	0.1256	0.9631
Tanh-32	0.1708	0.9521	0.1786	0.9476
Tanh-64	0.1511	0.9575	0.1563	0.9554
Tanh-128	0.1514	0.9579	0.1590	0.9547

Havaitaan, että Sigmoid-aktivointia käyttävissä verkoissa oppiminen on merkittävästi huonompaa kuin muissa. Tämä johtuu siitä, että virheen gradientit katoavat. Kerros, jossa on ReLu-aktivointi ja 128 neuronia, vaikuttaa parhaalta, joten vertaillaan eri optimointialgoritmeja tämänlaiseen neuroverkkoon. Verkon rakennetta on havainnollistettu alla olevassa taulukossa.

Kerros	Neuronien lkm.	Aktivointi	Opittavien parametrien lkm.
Sisääntulo	784	-	-
1	128	ReLu	$784 \cdot 128 + 128 = 100480$
Ulostulo	10	Softmax	$128 \cdot 10 + 10 = 1290$

Tutkitaan, miten opetusjoukon ja testijoukon virheet käyttäytyvät eri optimointialgoritmien kanssa. Suoritetaan opetus jokaiselle algoritmille kolme kertaa ja raportoidaan eri ajojen keskiarvot. Jatketaan opetusta jälleen 20 epookin ajan, ja pidetään minisatsien koko samana.

Algoritmien käyttämät parametrit valitaan seuraavasti.

- Stokastinen gradienttimenetelmä: tasaisesti vähenevä oppimisnopeus $1 \to 0.05$, jonka jälkeen nopeus pidetään vakiona.

- Momentti: momentin arvona käytetään $\alpha = 0.5$ ja oppimisnopeus määrätään samoin kuin stokastisessa gradienttimenetelmässä.
- Nesterovin momentti: samat parametrit kuin momentissa.
- AdaGrad: oppimisnopeus on vakio $\epsilon = 0.01$.
- AdaDelta: momentin arvo $\alpha = 0.9$.
- RMSProp: oppimisnopeus $\epsilon = 0.001$ ja momentti $\alpha = 0.9$.
- Adam: oppimisnopeus $\epsilon = 0.001$ ja momentit $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Taulukossa on esitetty tulokset.

Algoritmi	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus
SGM	0.1173	0.9676	0.1278	0.9621
Momentti	0.0700	0.9811	0.0910	0.9721
Nesterov	0.0691	0.9814	0.0912	0.9728
AdaGrad	0.0620	0.9833	0.0895	0.9727
AdaDelta	0.0244	0.9945	0.0684	0.9790
RMSProp	0.0115	0.9975	0.0748	0.9783
Adam	0.0128	0.9971	0.0842	0.9762

Tuloksista havaitaan, että stokastinen gradienttimenetelmä ei minimoi opetusjoukon virhettä yhtä hyvin kuin muut algoritmit. Lisäksi huomataan, että RMSProp- ja Adam-algoritmit pääsevät muita algoritmeja pienempään virheeseen. Momentti-algoritmien välillä ei havaita eroa.

Adam- ja RMSProp-algoritmien pieni opetusvirhe ei enää vaikuta välittyvän testijoukon virheeseen, vaan algoritmit ylisovittavat hieman. Seuraavassa taulukossa on algoritmien parhaimpien testivirheiden keskiarvot ja epookit, joilla paras testivirhe on keskimäärin saavutettu.

Algoritmi	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus	Epookki
RMSProp	0.0221	0.9946	0.0691	0.9791	14
Adam	0.0181	0.9960	0.0729	0.9778	15.67

Algoritmien kestoissa ei ollut merkittävää eroa. Yhden epookin opettaminen kesti kaikilla algoritmeilla noin $21 - 24$ sekuntia. Osion lopussa olevissa kuvaajissa esitetään opetuksen tulokset eri epookeilla.

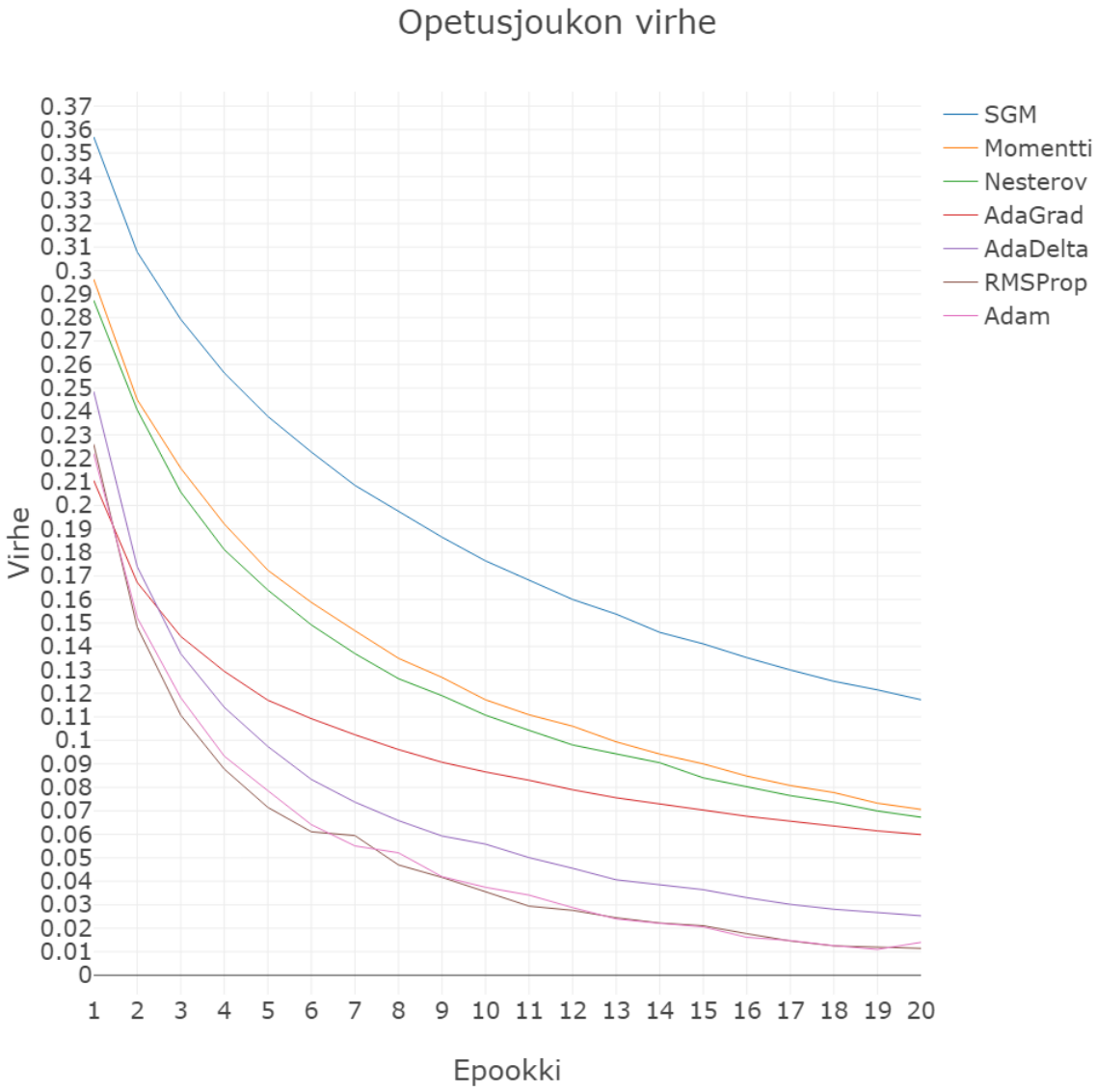
Vertailu Keras-kirjaston kanssa

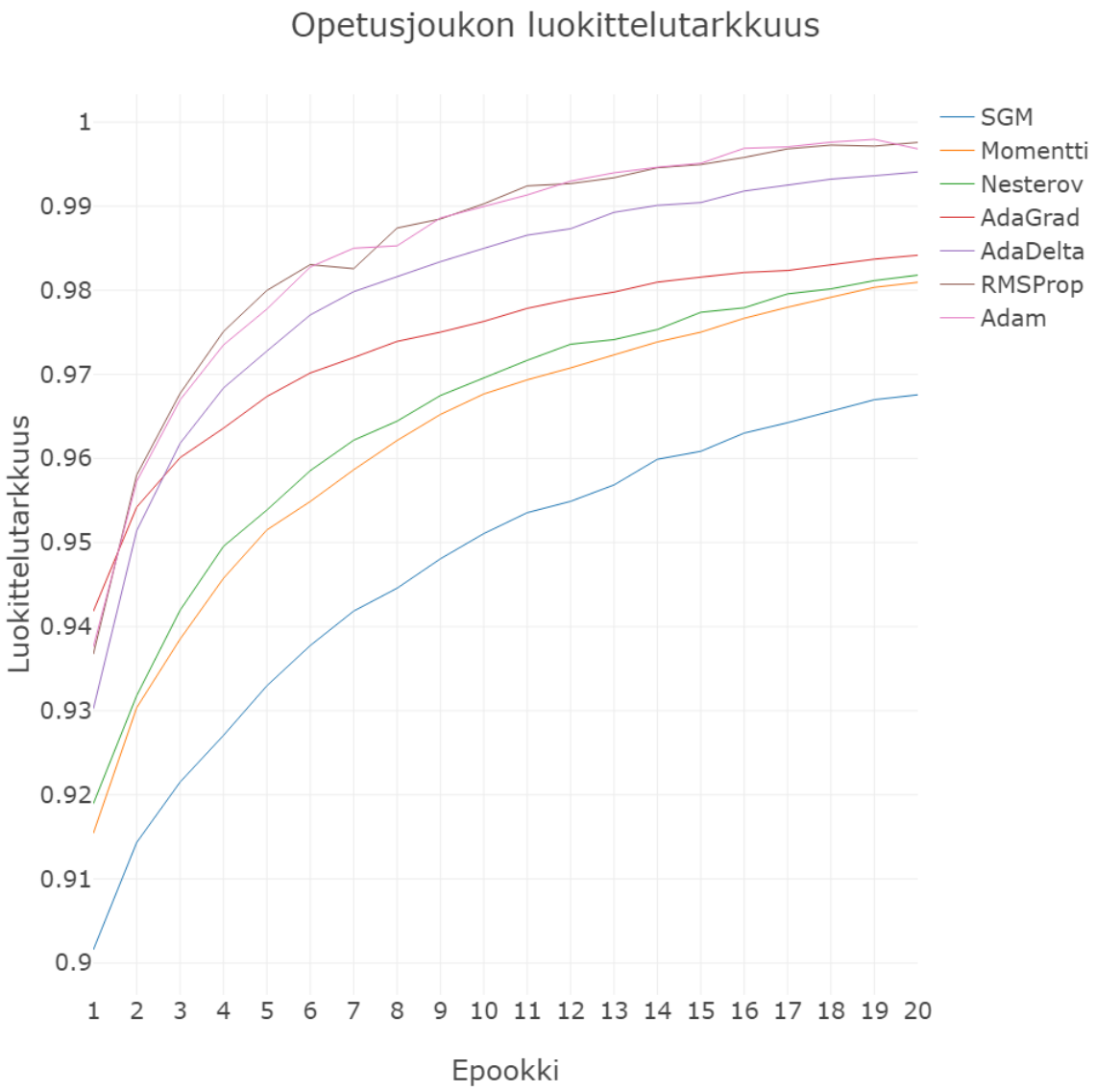
Opetetaan samanlainen neuroverkko Keras-kirjaston avulla. Optimointialgoritmeissa käytetään samoja parametreja kuin aiemmin.

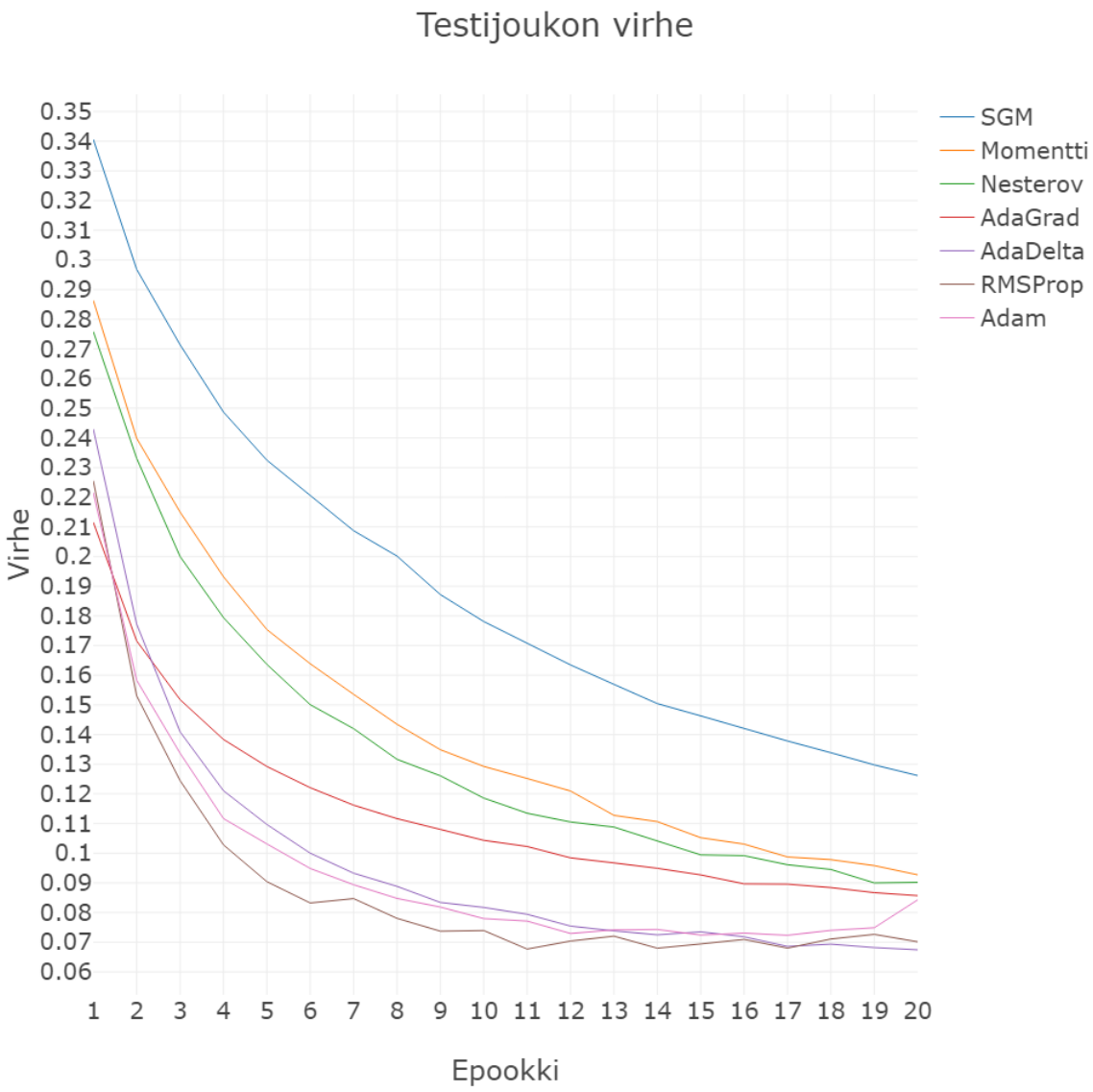
Algoritmi	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus
SGM	0.1176	0.9678	0.1259	0.9623
Momentti	0.0696	0.9813	0.0908	0.9730
Nesterov	0.0716	0.9806	0.0922	0.9730
AdaGrad	0.1721	0.9526	0.1708	0.9521
AdaDelta	0.0238	0.9948	0.0681	0.9790
RMSProp	0.0081	0.9989	0.0691	0.9805
Adam	0.0104	0.9983	0.0835	0.9759

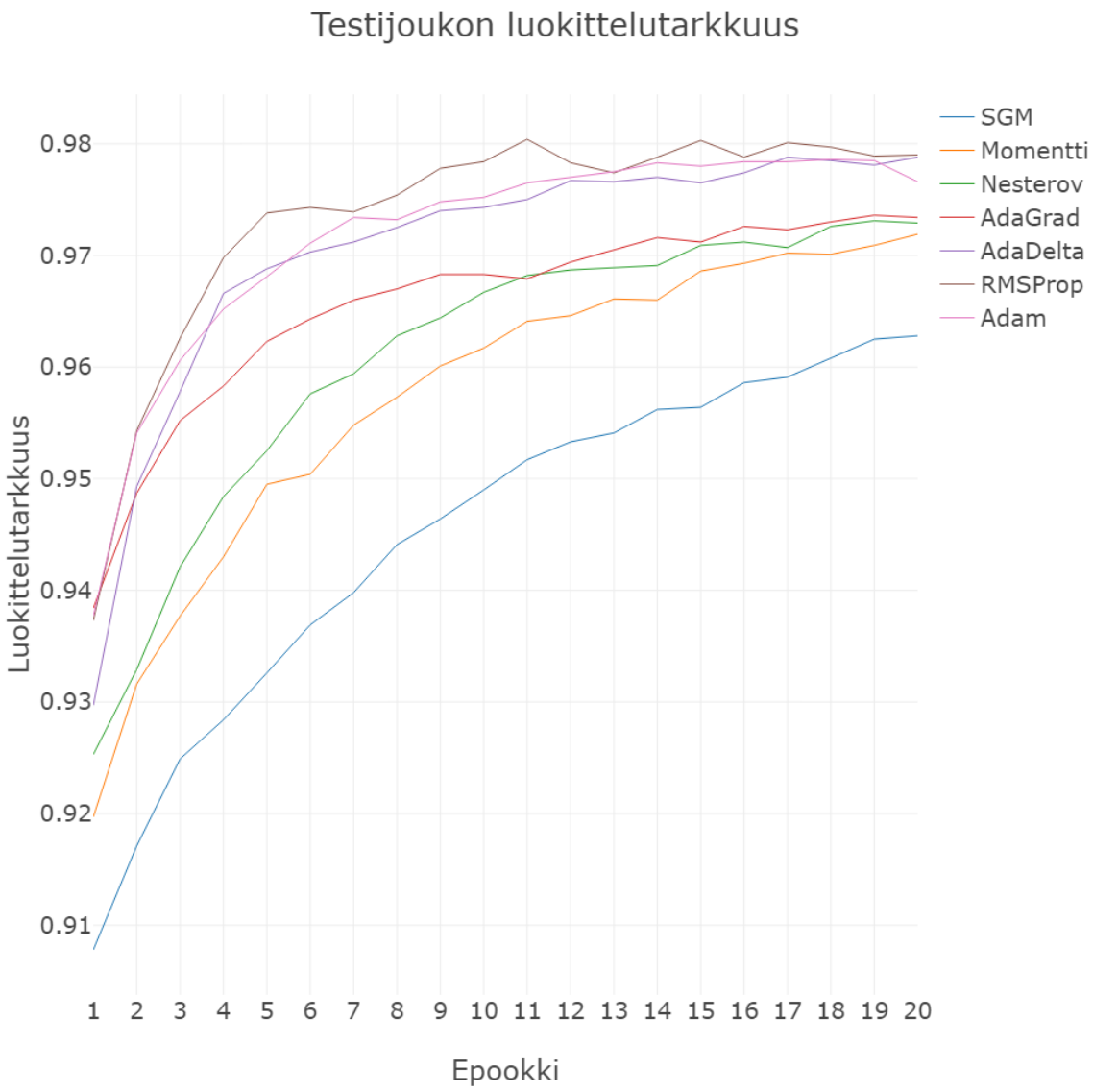
Havaitaan, että Keras-kirjastolla saadut tulokset ovat hyvin lähellä Javascript-toteutuksen tuloksia. Keras-kirjaston AdaGrad-algoritmilla saatiin selvästi heikompi tulos. Sen parametreja pitäisi vielä optimoida paremman tuloksen saamiseksi.

Keras-kirjaston algoritmien suoritusajat olivat huomattavasti pienempiä. Yhden epookin opettaminen kesti kaikilla algoritmeilla hieman alle kaksi sekuntia. Javascript-toteutuksen suorituskyky oli tähän verrattuna kuitenkin yllättävän hyvä.




Opetusjoukon luokittelutarkkuus
Luokittelutarkkuus
1
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.9
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Epookki
SGM
Momentti
Nesterov
AdaGrad
AdaDelta
RMSProp
Adam


Testijoukon virhe
Virhe
Epookki
SGM
Momentti
Nesterov
AdaGrad
AdaDelta
RMSProp
Adam
0.35
0.34
0.33
0.32
0.31
0.3
0.29
0.28
0.27
0.26
0.25
0.24
0.23
0.22
0.21
0.2
0.19
0.18
0.17
0.16
0.15
0.14
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Testijoukon luokittelutarkkuus
Luokittelutarkkuus
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Epookki
SGM
Momentti
Nesterov
AdaGrad
AdaDelta
RMSProp
Adam

Koodiesimerkki neuroverkon luomisesta ja opettamisesta Javascript-toteutuksen kanssa.

```
import fs from "fs";

import { Network } from "../Network";
import { AdamOptimizer } from "../Optimizer";
import { Dataset } from "../Dataset";
import { oneHotEncode } from "../utils";

const LABEL_SIZE = 10;
const BATCH_SIZE = 200;
const EPOCHS = 20;

// Parse CSV-file into a Dataset-object.
const readDataSet = (
  path: string,
  batchSize: number | undefined = undefined
): Dataset => {
  const items = fs
    .readFileSync(path)
    .toString()
    .split("\n")
    .map((line: string) => {
      const items = line.split(",").map((i) => parseInt(i));
      const output = oneHotEncode(items.shift(), LABEL_SIZE);
      // Scale values to 0-1.
      const input = items.map((i) => i / 255);
      return { input, output };
    });
  return new Dataset(items, batchSize);
};

const trainData = readDataSet("./mnist_train.csv", BATCH_SIZE);
const testData = readDataSet("./mnist_test.csv");

const network = new Network(trainData, testData, "cross-entropy");
network.add("relu", 128);
network.add("softmax", LABEL_SIZE);

const optimizer = new AdamOptimizer(network, {
  epochs: EPOCHS,
  momentDecay: 0.9,
  gradientDecay: 0.999,
  learningRate: 0.001,
});

const { results, took } = optimizer.optimize();
console.log("Took %s", took);
console.log("Accuracy: ", results[results.length - 1]);
```

Koodiesimerkki Mnist-aineiston opettamisesta Keras-kirjastolla. Mnist-aineisto on tässä kirjastossa valmiiksi saatavilla.

```
import numpy as np
import time
from tensorflow import keras
from tensorflow.keras import layers

label_size = 10

(x_train, y_train),
(x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

y_train = keras.utils.to_categorical(y_train, label_size)
y_test = keras.utils.to_categorical(y_test, label_size)

batch_size = 200
epochs = 20

model = keras.Sequential([
      layers.Flatten(input_shape=(28, 28, 1)),
      layers.Dense(128, activation='relu'),
      layers.Dense(num_classes, activation="softmax")])

model.compile(
     loss="categorical_crossentropy",
     optimizer=keras.optimizers.Adam(learning_rate=0.001),
     metrics=["accuracy"])

model.fit(
     x_train,
     y_train,
     batch_size=batch_size,
     epochs=epochs,
     validation_data=(x_test, y_test))

score_train = model.evaluate(x_train, y_train, verbose=0)
score_test = model.evaluate(x_test, y_test, verbose=0)

print("Train", score_train)
print("Test", score_test)
```

6.3 Hotellivarauksen peruuntumisen ennustaminen

Sovelletaan neuroverkkoa seuraavaksi aineistoon [4] , jossa on luokiteltu hotelliin tehtyjä varauksia sen mukaan peruuntuiko varaus vai ei. Aineistossa on 19 muuttujaa ja 36725 jäsentä. Aineiston kategorisina muuttujina ovat

- `Booking_ID`: Varauksen yksilöivä tekstiarvoinen tunniste. Tätä arvoa ei hyödynnetä luokittelussa.
- `type_of_meal_plan`: Asiakkaan valitseman ruokailun tyyppi. Erilaisia tyyppejä on neljä kappaletta.
- `required_car_parking_space`: Tarvitseeko asiakas autopaikkaa.
- `room_type_reserved`: Huoneen tyyppi. Näitä on seitsemän erilaista.
- `market_segment_type`: Kategorinen muuttuja, joka kertoo, mitä kautta varaus on tehty.
- `repeated_guest`: Muuttuja, joka kertoo, onko asiakas asioinut hotellissa ennen.
- `booking_status`: Selitettävä muuttuja, joka kertoo peruttiinko varaus (1) vai ei (0). Peruttuja varauksia on 32.4 % kaikista varauksista.

Aineiston numeeriset muuttujat ovat

- `no_of_adults`: Varauksessa olevien aikuisten lukumäärä.
- `no_of_children`: Varauksessa olevien lasten lukumäärä.
- `no_of_weekend_nights`: Viikonloppuna yövyttyjen öiden lukumäärä.
- `no_of_week_nights`: Arkena yövyttyjen öiden lukumäärä.
- `no_of_previous_cancellations`: Aiempien peruutuksien lukumäärä.
- `no_of_previous_bookings_not_canceled`: Aiempien toteutuneiden käyntien lukumäärä.
- `no_of_special_requests`: Lisäpyyntöjen lukumäärä.
- `lead_time`: Varauksentekopäivän ja yöpymisen välillä olevien päivien lukumäärä.
- `arrival_year`: Saapumisvuosi (2017 tai 2018).
- `arrival_month`: Saapumiskuukausi (1-12).
- `arrival_date`: Saapumispäivä (1-31).
- `avg_price_per_room`: Huoneen keskimääräinen hinta kyseisenä päivänä.

[4] https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset

Sisääntulokerroksen kategoriset muuttujat one-hot -enkoodataan, jolloin sisääntulokerroksen dimensioksi saadaan 32. Numeeriset muuttujat skaalataan välille $[0, 1]$ jakamalla ne kyseisen muuttujan maksimiarvolla. Aineisto jaetaan opetus- ja testijoukkoihin siten, että testijoukossa on 20 % koko aineistosta. Joukot rakennetaan niin, että peruttuja varauksia on molemmissa joukoissa sama osuus.

Vertaillaan jälleen erilaisia neuroverkon rakenteita, minkä jälkeen vertaillaan tähän rakenteeseen eri optimointialgoritmeja. Neuroverkon ulostulokerrokseen valitaan yksi Sigmoid-aktivoitu neuroni. Virhefunktiona käytetään binääristä ristientropiaa. Valitaan verkkoon yksi piilokerros, jossa käytetään ReLu-aktivointia. Vaihdellaan neuronien lukumäärää ja opetetaan verkkoa 20 epookin ajan stokastisella gradienttimenetelmällä, jonka oppimisnopeutena on 0.1. Valitaan minisatsien kooksi 100. Vertailusta saadaan seuraavat tulokset.

Kerros	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus
ReLu-32	0.4259	0.7971	0.3852	0.8278
ReLu-64	0.4246	0.8000	0.3921	0.8248
ReLu-128	0.4223	0.8002	0.3817	0.8291
ReLu-256	0.4202	0.7981	0.3810	0.8266
ReLu-512	0.4228	0.7964	0.3888	0.8225

Neuronien lukumäärällä ei vaikuta olevan suurta merkitystä oppimisen kannalta. Kokeillaan vielä muokata sisääntulokerrosta niin, että käsitellään osa sisääntulokerroksen numeerisista muuttujista kategorisiksi. Monet numeerisista muuttujista saavat vain muutamia erilaisia arvoja, joten niitä voidaan pitää kategorisina. One-hot -enkoodataan kaikki muut muuttujat paitsi `lead_time`, `avg_price_per_room` ja `no_of_previous_bookings_not_canceled`. Tällöin sisääntulokerroksen dimensio on 120. Tehdään tälle verkolle sama vertailu, jolloin saadaan oheiset tulokset.

Kerros	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus
ReLu-32	0.3484	0.8436	0.3852	0.8444
ReLu-64	0.3630	0.8297	0.3425	0.8398
ReLu-128	0.3389	0.8467	0.3306	0.8524
ReLu-256	0.3506	0.8443	0.3554	0.8371
ReLu-512	0.3307	0.8532	0.3230	0.8516

Huomataan, että tällä verkolla saadaan merkittävästi aiempaa paremmat tulokset. Vaikuttaisi myös siltä, että 512 neuronia olisi paras vaihtoehto, vaikka erot ovatkin pieniä. Tämän neuroverkon rakennetta on havainnollistettu alla olevassa taulukossa.

Kerros	Neuronien lkm.	Aktivointi	Opittavien parametrien lkm.
Sisääntulo	120	-	-
1	512	ReLu	$120 \cdot 526 + 526 = 63646$
Ulostulo	1	Sigmoid	$526 \cdot 10 + 1 = 527$

Vertaillaan tähän neuroverkkoon eri optimointialgoritmeja samoin kuin Mnist-aineiston kanssa. Käytetään vertailun algoritmeissa samoja parametreja. Nyt saadaan kolmen ajon keskiarvoiksi seuraavat arvot.

Algoritmi	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus
SGM	0.3397	0.8480	0.3502	0.8406
Momentti	0.2986	0.8700	0.3322	0.8481
Nesterov	0.3048	0.8668	0.3333	0.8490
AdaGrad	0.2940	0.8730	0.3215	0.8535
AdaDelta	0.2736	0.8817	0.3240	0.8534
RMSProp	0.2475	0.8944	0.3273	0.8549
Adam	0.2336	0.9010	0.3339	0.8552

Yhden epookin opettaminen kesti kaikilla algoritmeilla keskimäärin 7-8 sekuntia. Tuloksista havaitaan, että stokastinen gradienttimenetelmä ei löydä yhtä hyvää minimiä kuin muut algoritmit, mikä saattaa johtua huonosti valitusta oppimisnopeudesta. Muut algoritmit vaikuttavat pääsevän suunnilleen samaan testijoukon virheeseen. Erityisesti RMSProp- ja Adam-algoritmit pääsevät huomattavasti muita pienempään opetusvirheeseen. Ylisovitusta on siis voinut tapahtua. Seuraavassa taulukossa on esitetty algoritmien parhaimpien testivirheiden keskiarvot ja epookit, joilla paras testivirhe on keskimäärin saavutettu.

Algoritmi	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus	Epookki
SGM	0.3369	0.8485	0.3408	0.8450	19.33
Momentti	0.2997	0.8693	0.3290	0.8492	19.33
Nesterov	0.3022	0.8687	0.3287	0.8511	19.33
AdaGrad	0.2950	0.8719	0.3212	0.8540	19.33
AdaDelta	0.2819	0.8778	0.3223	0.8541	17.67
RMSProp	0.2629	0.8879	0.3207	0.8570	16
Adam	0.2595	0.8894	0.3216	0.8552	13

Adam-algoritmi vaikuttaa pääsevän nopeimmin parhaaseen testivirheeseen. Kaikki kehittyneemmät algoritmit päätyvät kuitenkin lopulta lähes samaan testivirheeseen. Seuraavassa kuvaajassa on esitetty Adam-algoritmilla opetetun neuroverkon antamat ennusteet.

		Ennuste		
		1	**0**	**Yhteensä**
Oikea arvo	**1**	1815	562	2377
	0	473	4405	4878
	Yhteensä	2288	4967	

Neuroverkko luokittelee $1815/2377 \approx 76.4$ % peruutuksista oikein, kun taas peruuttamattomista varauksista verkko saa luokiteltua oikein $4405/4878 \approx 90.3$ %. Tulos

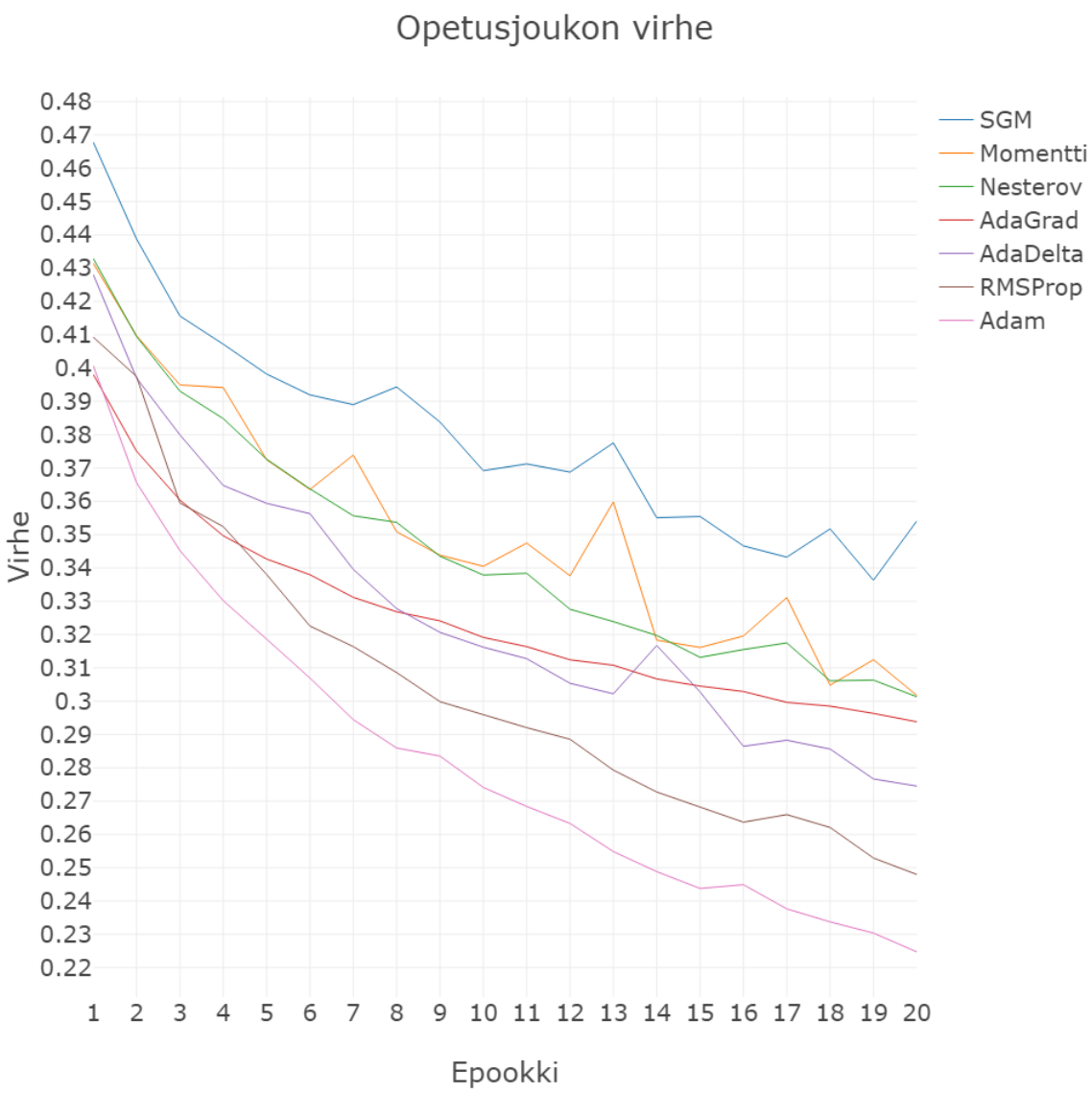
on käyttökelpoinen eli hotelli pystyisi neuroverkon avulla arvioimaan tulevien peruutuksien määrää. Osion lopussa olevissa kuvaajissa esitetään opetuksen tulokset eri epookeilla.

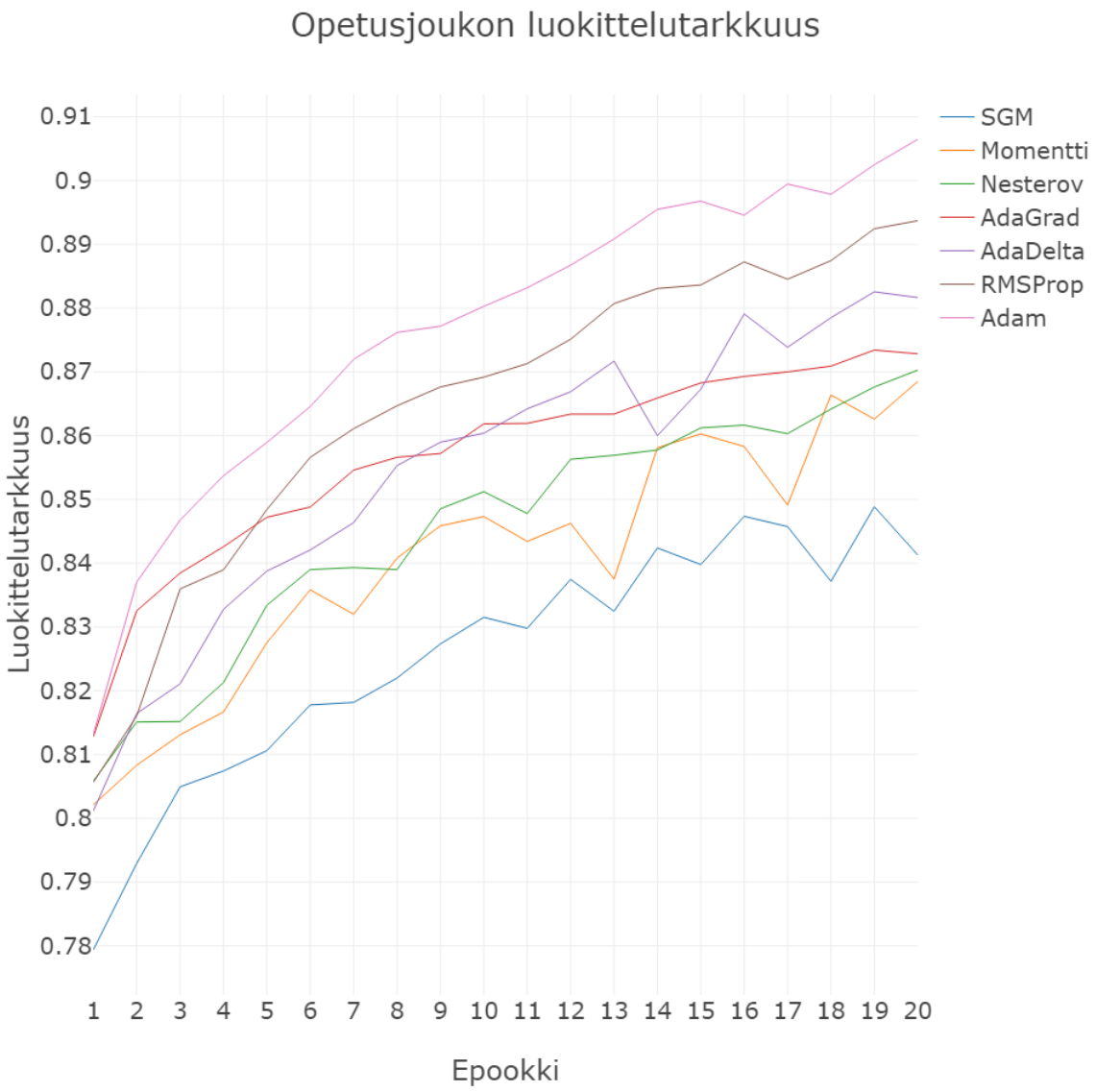
Vertailu Keras-kirjaston kanssa

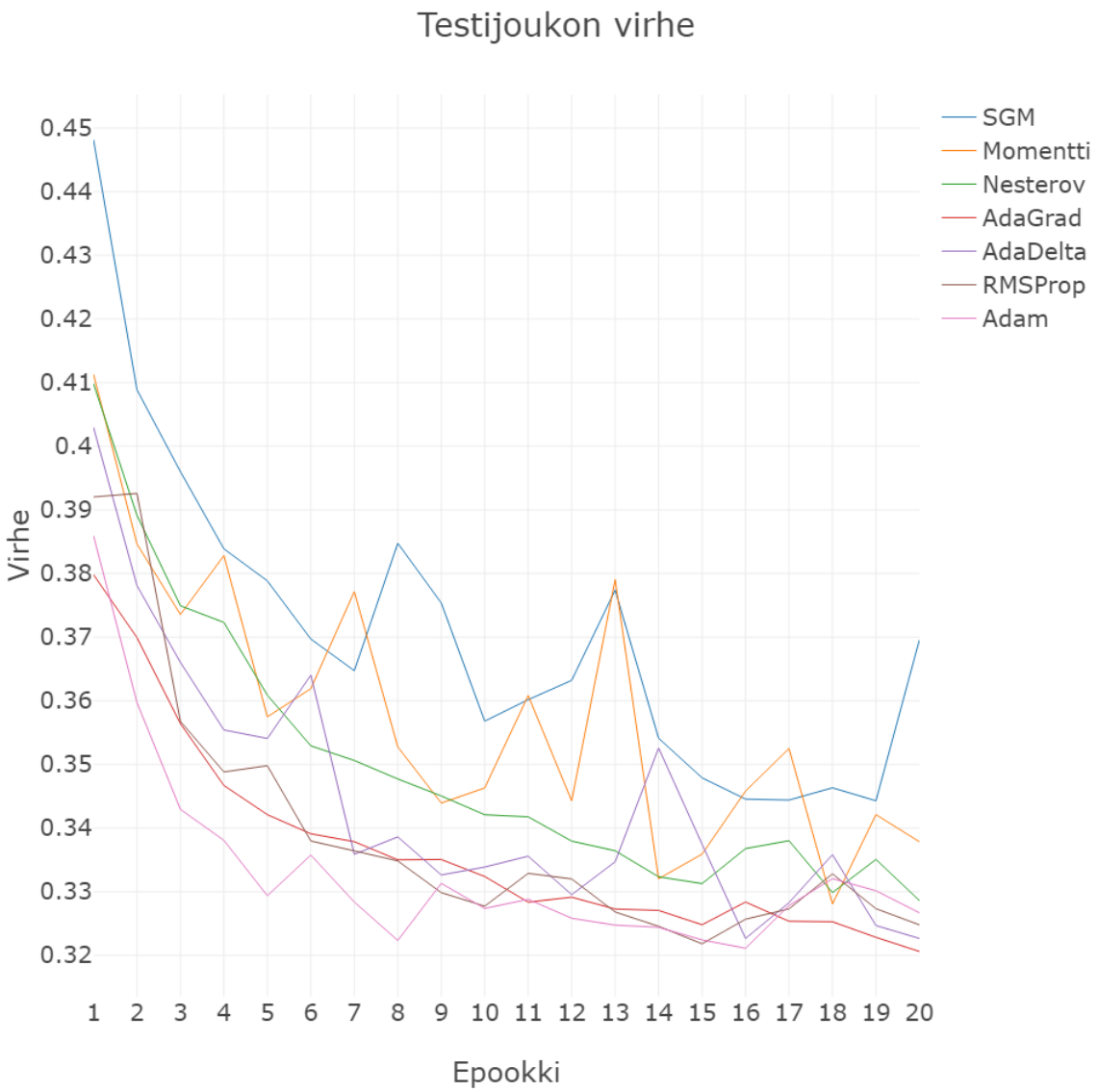
Kun opetetaan vastaava neuroverkko Keras-kirjastolla, saadaan oheiset tulokset.

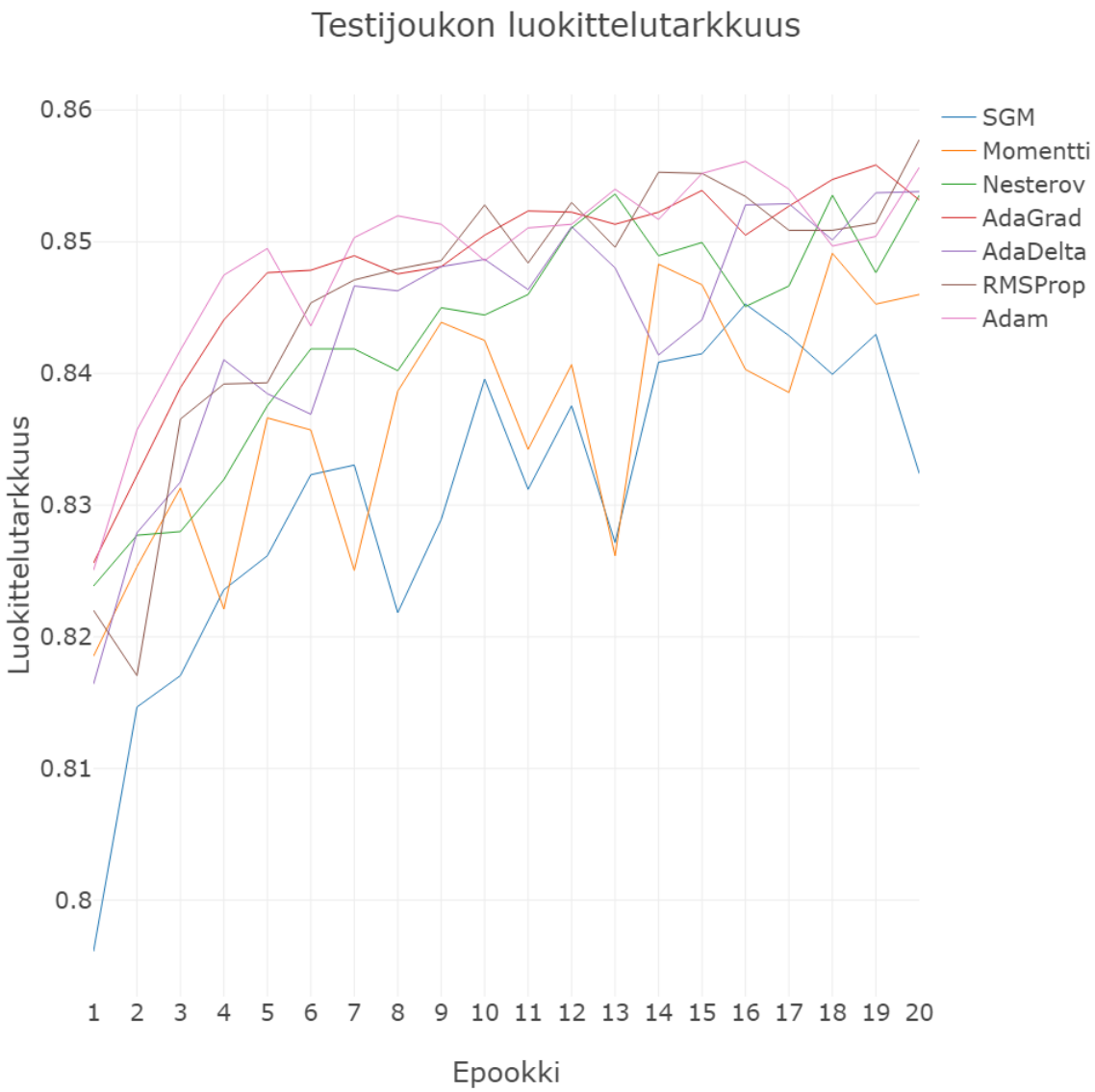
Algoritmi	Opetusvirhe	Opetustarkkuus	Testivirhe	Testitarkkuus
SGM	0.3268	0.8553	0.3539	0.8354
Momentti	0.3045	0.8660	0.3309	0.8514
Nesterov	0.2927	0.8738	0.3326	0.8517
AdaGrad	0.2741	0.8763	0.3370	0.8487
AdaDelta	0.2783	0.8796	0.3252	0.8553
RMSProp	0.2599	0.8879	0.3211	0.8594
Adam	0.2341	0.8983	0.3204	0.8580

Havaitaan jälleen, että Keras-kirjastolla saadut tulokset ovat lähellä Javascript-toteutuksen antamia tuloksia. Opetusvirhe on kaikilla algoritmeilla hieman pienempi. Yhden epookin opettaminen kesti kaikilla algoritmeilla noin 0.7 sekuntia eli Keras-toteutus oli tämänkin aineiston kanssa noin 10 kertaa nopeampi.


Opetusjoukon virhe
Virhe
Epookki
SGM
Momentti
Nesterov
AdaGrad
AdaDelta
RMSProp
Adam
0.48
0.47
0.46
0.45
0.44
0.43
0.42
0.41
0.4
0.39
0.38
0.37
0.36
0.35
0.34
0.33
0.32
0.31
0.3
0.29
0.28
0.27
0.26
0.25
0.24
0.23
0.22
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Opetusjoukon luokittelutarkkuus
Luokittelutarkkuus
0.91
0.9
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0.82
0.81
0.8
0.79
0.78
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Epookki
SGM
Momentti
Nesterov
AdaGrad
AdaDelta
RMSProp
Adam


Testijoukon virhe
Virhe
0.45
0.44
0.43
0.42
0.41
0.4
0.39
0.38
0.37
0.36
0.35
0.34
0.33
0.32
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Epookki
SGM
Momentti
Nesterov
AdaGrad
AdaDelta
RMSProp
Adam


Testijoukon luokittelutarkkuus
Luokittelutarkkuus
0.86
0.85
0.84
0.83
0.82
0.81
0.8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Epookki
SGM
Momentti
Nesterov
AdaGrad
AdaDelta
RMSProp
Adam

7 Johtopäätökset

Tässä tutkielmassa esiteltiin neuroverkkojen taustalla olevat matemaattiset mallit, joiden avulla ratkaistiin yksinkertainen XOR-ongelma. Lisäksi esiteltiin neuroverkkojen opettamisessa yleisimmin käytettyjä optimointialgoritmeja ja sovellettiin niitä luokitteluongelmiin.

Luokitteluongelmien ratkaisemisessa käytettiin itse tehtyä toteutusta neuroverkosta. Tuloksista havaittiin, että ne vastaavat hyvin Keras-kirjastolla saatuja tuloksia. Javascript-toteutuksen suorituskyky oli hyvä, mutta kuitenkin noin 10 kertaa hitaampi kuin Keras-kirjastolla. Toteutusta on mahdollista optimoida lisää, mutta valmiiden kirjastojen käyttäminen on huomattavasti järkevämpää.

Käsinkirjoitettujen numeroiden luokittelussa päästiin hyvään, lähes 98 %, testijoukon tarkkuuteen. Parhaimmat tulokset saatiin RMSProp- ja Adam-algoritmeilla ja huonoimmat stokastisella gradienttimenetelmällä. Parhaiten suoriutuvilla algoritmeilla oli havaittavissa lievää ylisovitusta. Algoritmien välillä ei havaittu eroja suorituskyvyissä.

Hotellivarauksien peruuntumisten ennustamisessa saatiin kelvollisia tuloksia. Varaukset luokiteltiin oikein melkein 86 % tarkkuudella. Optimointialgoritmien väliset erot olivat tämän aineiston kanssa samaa luokkaa kuin numeroiden tunnistuksessa. Kaikkien algoritmien kohdalla havaittiin ylisovittamista, joten suuremmasta aineistosta olisi hyötyä paremman tuloksen saamiseksi.

Usein sanotaan, että koneoppimisessa ei ole ilmaisia lounaita (*No Free Lunch theorem*), mikä tarkoittaa sitä ettei yksikään menetelmä tai algoritmi ole kaikissa tilanteissa muita parempi. Koneoppimismenetelmät ja niissä käytetyt parametrit täytyy aina valita tapauskohtaisesti, jotta saadaan paras mahdollinen lopputulos. Tässä tutkielmassa sovellettiin valittuihin aineistoihin neuroverkkoa, mutta muilla koneoppimismenetelmillä voitaisiin saada vielä parempia tuloksia. Vaikuttaisi kuitenkin sille, että neuroverkkojen optimoinnissa kannattaa useimmiten käyttää jotakin kehittyneempää algoritmia, kuten RMSProp-, Adam- tai AdaGrad-algoritmia. Muut algoritmit voivat toimia joissain tapauksissa hyvin, mutta ne todennäköisesti vaativat enemmän parametrien manuaalista säätämistä. Algoritmin valinnalla ei ole yhtä suurta merkitystä, jos neuroverkko ylisovittaa.

Viitteet

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Michael David. Spivak. *Calculus on manifolds : a modern approach to classical theorems of advanced calculus*. Mathematics monograph series. Benjamin, New York, NY, 1965.

[3] Jyrki Lahtonen. Vektorilaskenta, 2020. `https://users.utu.fi/lahtonen/VL2020/VL2020.pdf`.

[4] Michael A. Nielsen. Neural networks and deep learning, 2018.

[5] M. S. Bazaraa. *Nonlinear programming : theory and algorithms*. J. Wiley & Sons, Hoboken, NJ, 3rd edition, 2006.

[6] Ethem Alpaydin. *Introduction to machine learning*. The MIT Press, Cambridge, Massachusetts, fourth edition, 2020.

[7] Tom M. Mitchell. *Machine learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, 1997.

[8] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

[9] Tomasz Szandala. Review and comparison of commonly used activation functions for deep neural networks. *CoRR*, abs/2010.09458, 2020.

[10] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks, 2015.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[12] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[13] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[14] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

[15] Geoffrey Hinton. Neural networks for machine learning. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Haettu 3.4.2023.

[16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[18] Josef Steppan. Mnist examples. https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png, 2017. Haettu 10.4.2023.