Neuroverkot ja koneoppiminen

Joonas von Lerber
Otaniemen lukio

Johdanto

Tekoäly mullistaa maailmaa monilla eri tavoilla. Tekoälyä voi pyytää kirjoittamaan romaaneja, koodia, runoja tai luomaan maalauksia. Lähes lukematon määrä erilaisia malleja on luotu eri tehtäviä varten. Neuroverkot ovat yksi varhaisimmista tekoälyn malleista, joilla voidaan ratkaista hankalia regressio- ja luokitteluongelmia. Neuroverkot koostuvat pienemmistä tekoneuroneista, jotka ovat yksinkertaisia malleja biologisista hermosoluista.

Tekoneuronit ja perseptronit (Perceptron)

Tekoneuroni on neuroverkon pienin osanen, sen yksinkertaisin solu. Tekoneuroni on algoritmi, joka pyrkii arvioimaan biologisen neuronin käyttäytymistä. Perseptroni on tietynlainen tekoneuroni, joka pystyy luokittelemaan datan kahteen eri kategoriaan, eli se on binäärinen luokittelija. Nämä kategoriat voivat olla vaikka ristit ja nollat tai 1 ja 0, tai vaikka hymynaama ja surunaama [9].

Perseptronimallia voidaan käyttää luokitteluun, jos hermoverkon data eli sen käsittelemä tieto on lineaarisesti separoituvaa eli kategorioiden raja voidaan kuvailla suorana. Jos ehto ei täyty, perseptroni ei voi oppia dataa kunnollisesti [9].

Tekoneuronin idean alkuperä on biologisessa hermosolussa ja sen käyttäytymisessä, joskin myöhempi tutkimus [5] on osoittanut hermosolujen olevan huomattavasti monimutkaisempia kuin tekoneuronit.

Tekoneuroni-algoritmi ottaa syötteenä yhden dataalkion, laskee sen painotetun summan ja syöttää sen aktivointifunktioon (perseptronin tilanteessa askelfunktioon). Perseptronilla askelfunktion ulostulo on sitten klassifikaatio kahden eri luokan välillä. Kaava 1 on kaksiulotteiselle perseptronille, joka ottaa syötteenä pisteen (x_1, x_2) ja antaa ulos luokan

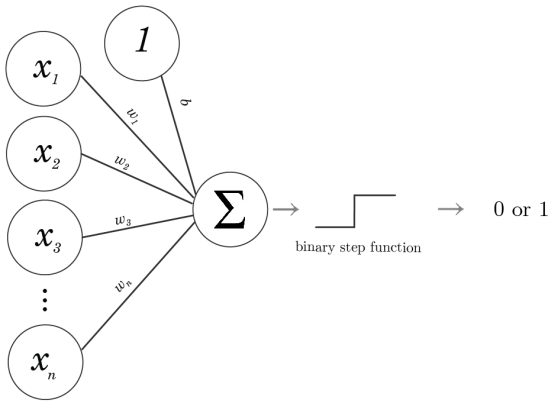
$$H(w_1x_1 + w_2x_2 + b) = 0 \text{ tai } 1,$$
$$H(x) = \begin{cases} 1, \text{ jos } x > 0, \\ 0, \text{ jos } x \leq 0. \end{cases} \tag{1}$$

Tämä lauseke voidaan yleistää n-ulotteiseen perseptroniin

$$H(b + \sum_{i=1}^{n} w_i x_i). \tag{2}$$

Yleisen tekoneuronin kaava saadaan muuntamalla kaavan 2 askelfunktio H yleiseksi aktivointifunktioksi $\varphi\colon \mathbb{R} \to \mathbb{R}$,

$$\varphi\left(b + \sum_{i=1}^{n} w_i x_i\right). \tag{3}$$



Kuva 1: Perseptronin rakenne, lähde: [1].

Huomaamme, että voimme esittää pisteen ja perseptronin parametrit vektoreina, ja voimme tehdä siitä laskennallisesti helpomman hyödyntämällä vektorien pistetuloa

$$b + \sum_{i=1}^{n} w_i x_i = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

Täten saamme n-ulotteisen perseptronin yleisen kaavan

$$H(\mathbf{w} \cdot \mathbf{x} + b), \quad \mathbf{x}, \mathbf{w} \in \mathbb{R}^n. \tag{4}$$

Voimme siis järjestää perseptronin parametrit vektoriin, jonka avulla saamme vastauksen käyttämällä pistetuloa.

Geometrisesti perseptronin voidaan ajatella piirtävän rajaviivan. Rajaviivan alapuoliset pisteet ovat ensimmäistä luokkaa ja yläpuoliset pisteet toista luokkaa. Huomaamme, että perseptroni on lineaarinen avaamalla sen kaksiulotteista kaavaa

$$w_1 x_1 + w_2 x_2 + b = 0.$$

Kuten huomaamme, yhtälö on sama kuin kaksiulotteisella viivalla suhteessa muuttujiin x_1, x_2. Sama geometrinen intuitio pätee myös korkeampiin ulottuvuuksiin. Yksi n-ulotteisen perseptronin raja on n-ulotteinen hypertaso.

Perseptronin toimintaa voi ymmärtää tarkastelemalla muutamaa esimerkkiä.

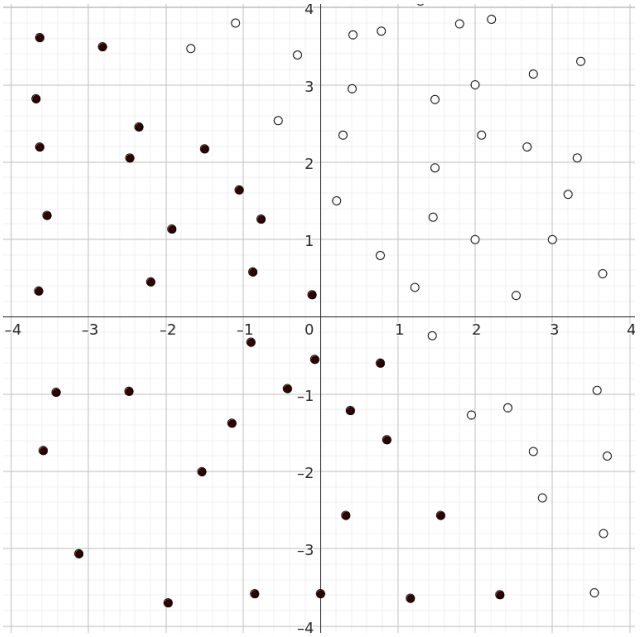
Kuvassa 2 näemme datajoukon, jolla on kaksi eri kategoriaa, valkoiset ja mustat. Huomaamme, että datajoukko on lineaarisesti separoituva, koska voimme piirtää viivaimella suoran erottamaan luokat toisistaan.

Mutta hipsuliveisulat sentään! Tämähän on juuri perseptronin käyttötarkoitus. Jos säädämme perseptronin, jolla kyseiset parametrit ovat $w_1 = 4$, $w_2 = 3$, $b = -2$, voimme erottaa pisteet toisistaan. Asettamalla mustien pisteiden ulostuloksi 0 ja valkoisten pisteiden ulostuloksi 1 voimme sijoittaa kaavaan 1 pisteitä. Esimerkiksi sijoittamalla pisteen $(2, 1)$ saamme ulostuloksi:

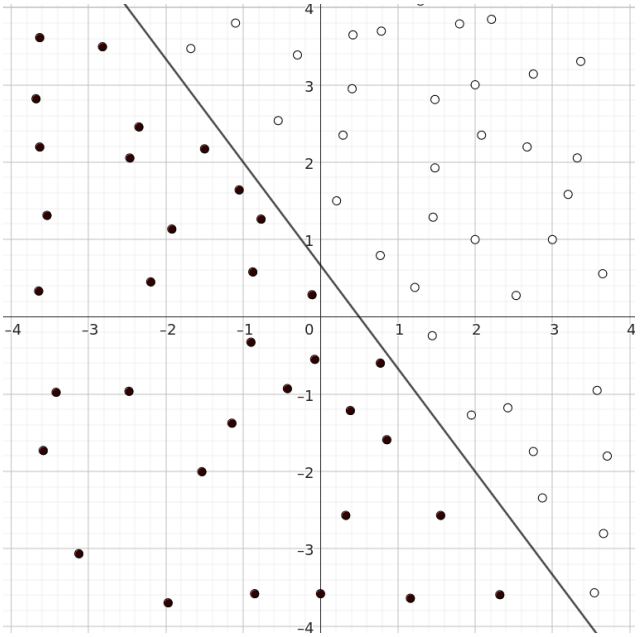
$$H(4 \cdot 2 + 3 \cdot 1 - 2) = H(8 + 3 - 2) = H(9) = 1.$$

Tämän voimme tarkistaa katsomalla kuvaa, ja huomaamme, että perseptroni osasi kertoa oikean luokan. Kokeillaanpa seuraavaksi joukkomme ulkopuolista pistettä $(-2, 3)$:

$$H(4 \cdot (-2) + 3 \cdot 3 - 2) = H(-8 + 9 - 2) = H(-1) = 0.$$



Kuva 2: Eräs lineaarisesti separoituva datajoukko A koordinaatistossa.



Kuva 3: Datajoukon A rajaviiva piirrettynä.

Tästä näemme perseptronin kyvyn luokitella eri pisteitä eri luokkiin. Yhdistämällä useita luokittelijoita voimme piirtää useita eri viivoja ja muodostaa mielivaltaisen määrän luokkia.

Perseptronin opettaminen

Seuraavaksi voimme kysyä, miten saamme sopivat luokittelun parametrit perseptronille?

Käytämme tähän perseptronin opetuskaavaa [9]

$$\begin{aligned} y_j &= H(\mathbf{w}(t) \cdot \mathbf{x}_j), \\ \mathbf{w}(t+1) &= \mathbf{w}(t) + r \cdot (d_j - y_j) \cdot \mathbf{x}_j, \end{aligned} \tag{5}$$

missä

$\mathbf{w}(t)$ on painovektori ajankohdassa t,

$\mathbf{x}_j$ on datajoukon alkio j,

d_j on odotettu ulostulo datajoukon alkiolle j,

y_j on laskettu ulostulo datajoukon alkiolle j,

r on askelkoko.

Kaavassa laskemme ensimmäiseksi, mikä on kyseisen pisteen luokka, ja jos se on väärin, parametreja muutetaan askelkoon suuruisesti. Askelkoko on meidän itse määrittelemämme hyperparametri, joka on yleensä välillä $]0, 1]$. Askelkoko voi olla myös suurempi kuin 1.

Huomaamme vakiotermin b kadonneen, mutta todellisuudessa se on kiinnitettynä parametrivektorin indeksiin 0. Olen samalla kiinnittänyt datapisteen indeksiin 0 numeron 1, jotta se ei vaikuta vakiotermiin, sillä

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + b.$$

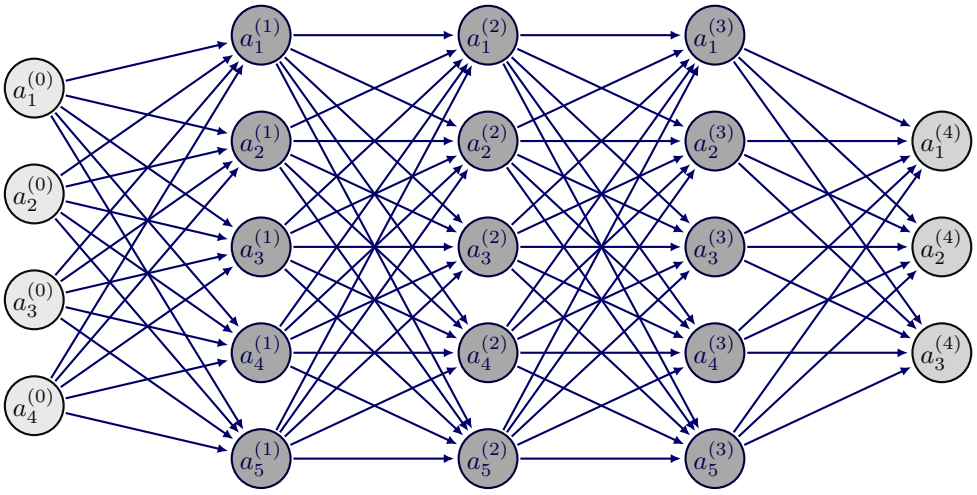
Kaavan 5 voi rakentaa vastavirta-algoritmin avulla, jota käsittelen hieman luvussa Neuroverkkojen opettaminen.

Neuroverkot

Kuten mainitsin artikkelin alkupuolella, tekoneuronien voidaan ajatella olevan neuroverkkojen rakennuspalikoita. Neuroverkoissa käytetään aktivointifunktioina ”pehmeitä” funktioita, kuten Sigmoid-perheen- tai lineaarisen suoristimen mukaisia funktioita. Yksinkertaisin variaatio tällaisesta funktiosta on nimeltään ReLU (Rectified Linear Unit), joka määritellään $\mathrm{ReLU}(x) = \max\{0, x\}$ [16].

Neuroverkot rakentuvat siis monista tekoneuroneista, jotka voivat olla järjestettyinä riveihin tai olla järjestelemättömässä asetelmassa. Eteenpäin syötettävissä neuroverkoissa neuronit ovat järjestyneet riveihin, joissa yhteydet kulkevat rivien välissä, mutta eivät rivien sisällä. Jokainen neuroni edelliseltä riviltä on yhdistetty jokaiseen neuroniin seuraavalla rivillä. Kuvassa 4 havainnollistetaan yksinkertaista eteenpäin kytkettyä neuroverkkoa. On myös olemassa neuroverkkomalleja, joissa neuronit eivät ole riveissä, vaan ne voivat olla erilaisissa järjestyksissä. Esimerkki tällaisesta on Hopfieldin verkko [10].

Neuroverkon ensimmäistä kerrosta kutsutaan syötteeksi, seuraavaksi on yksi tai enemmän piilokerroksia, ja viimeinen kerros on nimeltään ulostulo. Kuvassa 4 syöttö on väriltään vaaleampi, piilokerrokset tummempia ja ulostulo taas vaaleampi.



Kuva 4: Yksinkertainen eteenpäin syöttävä neuroverkko.

Neuroverkot ovat erityisen hyviä regressio- ja luokitteluongelmissa, mutta niiden teoreettinen hyöty tulee niiden kyvystä toimia yleisenä funktion approksimoijana. Kaikki laskettavissa olevat funktiot voidaan approksimoida neuroverkolla [11]. Tämä kattaa esimerkiksi oikean biologisen neuronin toiminnan arvioinnin, kuvien luokittelun, talouden arvioinnin ja muut vastaavat sovellukset. Tämä mahdollistaa neuroverkkojen käytön moneen eri sovellukseen [6]. Neuroverkon ominaisuudesta approksimoida jokaista funktiota $f\colon \mathbb{R}^n \to \mathbb{R}^m$ kerrotaan tarkemmin esitelmässä [12].

Eteenpäin syöttäviä neuroverkkoja käytetään yleisesti. Monet modernit tekoälyt, kuten ChatGPT, perustuvat GPT-4-malliin, jossa eteenpäin syöttävä neuroverkko on yksi osa monien muiden joukossa [17]. Eteenpäin syöttäviä neuroverkkoja käytetään monissa muissa tekoälymalleissa, ja ne ovat yksi yksinkertaisimmista ja parhaiten tutkituista malleista.

Puhun paljon arvoista, ulostuloista, painoista ja vakiotermeistä. Arvo tarkoittaa tekoneuronin arvoa ennen sen syöttämistä aktivointifunktioon, kuten sigmoidiin, ReLUun tai askelfunktioon. Sitä merkitään tässä symbolilla x. Ulostulo on aktivointifunktion läpi syötetty arvo ja sen symboli on o. Paino on kahden tekoneuronin välisen yhteyden eli synapsin kerroin, joka kertoo yhteyden voimakkuudesta. Vakiotermi, englanniksi bias, on tekoneuronin ominainen parametri, joka kertoo, kuinka aktiivinen tekoneuroni on. Painon symboli on w ja vakiotermin b. Painoa edellisen kerroksen ulostulon i ja seuraavan kerroksen arvon j välillä merkitään $w_{i,j}$.

Neuroverkko koostuu kerroksista. Jokainen kerros sisältää rivin tekoneuroneita, joista jokainen liittyy jokaiseen seuraavan kerroksen tekoneuroniin kuvan 4 mukaisesti. Jokaisella neuronilla on omat painot ja vakiotermit. Seuraavan kerroksen arvot lasketaan käyttäen tekoneuronin kaavaa 3. Yläindeksit viittaavat kerroksen indeksiin eli monesko kerros neuroverkossa on kyseessä. Alaindeksit taas viittaavat tekoneuronin indeksiin, eli monesko neuroni kyseisessä kerroksessa on kyseessä:

$$\begin{aligned} b_j^L + \sum_{i=1}^{n} w_{i,j}^L o_i^L &= x_j^{L+1}, \\ \varphi(x_j^{L+1}) &= o_j^{L+1}, \end{aligned} \tag{6}$$

missä

j, i on neuronin indeksi,
L on kerroksen indeksi,
x on neuronin arvo,
o on neuronin ulostulo,
w on kyseisen synapsin paino,
b on kyseisen neuronin vakiotermi,
$\varphi(x)$ on aktivointifunktio.

Siirrymme siis kerroksesta toiseen tietäen edellisen kerroksen ulostulon, painot ja vakiotermit. Algoritmi jatkuu, kunnes olemme saavuttaneet viimeisen kerroksen eli neuroverkon ulostulon.

Kaava 6 on vektorimuodossaan

$$\begin{bmatrix} \sum_{i=1}^{n} w_{i,1}^L o_i^L \\ \vdots \\ \sum_{i=1}^{n} w_{i,m}^L o_i^L \end{bmatrix} + \begin{bmatrix} b_1^L \\ \vdots \\ b_m^L \end{bmatrix} = \begin{bmatrix} x_1^{L+1} \\ \vdots \\ x_m^{L+1} \end{bmatrix},$$

$$\varphi \begin{pmatrix} x_1^{L+1} \\ \vdots \\ x_m^{L+1} \end{pmatrix} = \begin{bmatrix} o_1^{L+1} \\ \vdots \\ o_m^{L+1} \end{bmatrix}.$$

Yhden neuronin tapauksessa voimme järjestää sen painot vektoriin ja laskea neuronin arvon pistetulona datapisteen kanssa. Monen neuronin systeemissä toimimme samoin, joskin painot järjestetään matriisiin:

$$\begin{bmatrix} w_{1,1}^L & \dots & w_{1,n}^L \\ \vdots & \ddots & \vdots \\ w_{m,1}^L & \dots & w_{m,n}^L \end{bmatrix} \begin{bmatrix} o_1^L \\ \vdots \\ o_n^L \end{bmatrix} + \begin{bmatrix} b_1^L \\ \vdots \\ b_m^L \end{bmatrix} = \begin{bmatrix} x_1^{L+1} \\ \vdots \\ x_m^{L+1} \end{bmatrix}.$$

Huomaamme tässä esitetyn kaavan olevan saman kuin aikaisemmin esitetty summaa käyttävä kaava ja voimme kirjoittaa sen ytimekkäämmin

$$\mathbf{W}^L \mathbf{o}^L + \mathbf{b}^L = \mathbf{x}^{L+1}, \tag{7}$$

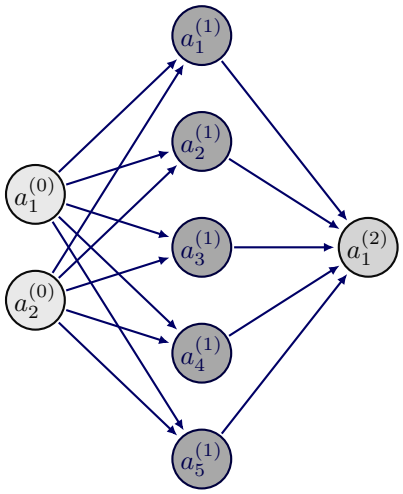
$$\varphi(\mathbf{x}^{L+1}) = \mathbf{o}^{L+1}, \tag{8}$$

$$\mathbf{W} \in \mathbb{R}^{m \times n}, \mathbf{o} \in \mathbb{R}^n, \ \mathbf{b}, \mathbf{x} \in \mathbb{R}^m.$$

Kuten huomaamme, mystiset ja ihmeelliset tekoälyt ovat oikeasti vain kasa matriiseja ja vektoreita ryydittynä epälineaarisilla aktivointifunktioilla.

Tiivistyksenä: Eteenpäin syöttävä neuroverkko muodostuu riveistä tekoneuroneita, joista jokainen edellisen kerroksen neuroneista on kytköksissä kaikkiin seuraavan kerroksen neuroneihin. Jokaisella yhteydellä on paino, jotka voidaan esittää painomatriisina. Tekoneuroneilla on myös vakiotermi. Voimme syöttää neuroverkolle dataa, ja neuroverkko syöttää dataa eteenpäin kerrosten välillä käyttäen annettuja kaavoja antaen lopulta ulos haluamamme vastauksen.

Esimerkin avulla voimme saada paremman ymmärryksen eteenpäin syöttävien neuroverkkojen toiminnasta. Neuroverkkomme voisi olla vaikka tämän muotoinen:



Neuroverkkomme koostuu siis sisääntulosta, yhdestä piilotetusta kerroksesta ja ulostulosta. Kerrosten välillä on painomatriisi ja vakiotermivektori. Piilotetun kerroksen ja ulostulon aktivointifunktiona toimii yksi sigmoidfunktioista nimeltään logistinen funktio, joka määritellään

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

Sen muoto koordinaatistossa muistuttaa S-kirjainta. Se rajoittaa neuroverkkomme ulostulot välille $]0,1[$.

Syötämme sisään vektorin $\begin{bmatrix}1.2\\0.4\end{bmatrix}$. Ensimmäisen kerroksen arvot:

$$\mathbf{o}^1 = \begin{bmatrix}1.2\\0.4\end{bmatrix},\ \mathbf{W}^1 = \begin{bmatrix}-1.2 & -1.1\\-0.6 & 1.0\\0.8 & -0.5\\0.3 & 1.1\\0.4 & 0.5\end{bmatrix},\ \mathbf{b}^1 = \begin{bmatrix}1.4\\0.1\\-0.5\\-0.3\\-0.5\end{bmatrix}.$$

Piilotetun kerroksen arvot $\mathbf{x}^2$ lasketaan edellisen kerroksen painomatriisin matriisitulosta ulostulojen kanssa käyttäen kaavaa 7:

$$\mathbf{x}^2 = \begin{bmatrix}-1.2 & -1.1\\-0.6 & 1.0\\0.8 & -0.5\\0.3 & 1.1\\0.4 & 0.5\end{bmatrix}\begin{bmatrix}1.2\\0.4\end{bmatrix} + \begin{bmatrix}1.4\\0.1\\-0.5\\-0.3\\-0.5\end{bmatrix}$$

$$= \begin{bmatrix}-1.88\\-0.32\\0.76\\0.80\\0.68\end{bmatrix} + \begin{bmatrix}1.4\\0.1\\-0.5\\-0.3\\-0.5\end{bmatrix} = \begin{bmatrix}-0.48\\-0.22\\0.26\\0.5\\1.18\end{bmatrix}.$$

Nämä ovat siis seuraavan kerroksen arvot. Käyttämällä kaavaa 8 ja sijoittamalla funktion φ paikalle σ:

$$\mathbf{o}^2 = \sigma\begin{pmatrix}-0.48\\-0.22\\0.26\\0.5\\1.18\end{pmatrix} \approx \begin{bmatrix}0.38\\0.45\\0.56\\0.62\\0.76\end{bmatrix}.$$

Voimme tämän nyt syöttää seuraavaan kerrokseen käyttämällä kaavaa 7:

$$\mathbf{W}^2 = \begin{bmatrix}-0.3 & 0.8 & -1.4 & -1.4 & 1.5\end{bmatrix},\ \mathbf{b}^2 = [-0.3],$$

jolloin

$$\mathbf{x}^3 = \begin{bmatrix}-0.3 & 0.8 & -1.4 & -1.4 & 1.5\end{bmatrix}\begin{bmatrix}0.38\\0.45\\0.56\\0.62\\0.76\end{bmatrix} + [-0.3]$$

$$= -0.04 - 0.3 = -0.34.$$

Pyöritämme tämän vielä sigmoidin kautta ja saamme neuroverkon ulostuloksi

$$\mathbf{o}^3 = \sigma(-0.34) \approx 0.41.$$

Tämä ulostulo ei kerro mitään järkevää ja on lähes satunnainen, sillä neuroverkkoa ei oltu opetettu suorittamaan mitään tehtävää, vaan painomatriisit ja vakioarvot luotiin satunnaislukugeneraattorilla.

Neuroverkot luokittelijoina

Eteenpäin syöttäviä neuroverkkoja käytetään luokitteluun perustuvissa ongelmissa kuten objektien tunnistaminen kuvista tai videosta. Yksi tunnetuimmista koneoppimisen ongelmista oli saada kone tunnistamaan kokoa 28×28 olevista kuvista käsin piirrettyjä numeroita käyttäen MNIST-datajoukkoa. Tätä ongelmaa ja datajoukkoa on jopa kutsuttu koneoppimisen ”Hello World!” -ongelmaksi. Nykytekniikoita käyttäen eteenpäin syöttävä neuroverkko pystyy luokittelemaan 10 000 uudesta kuvasta noin 99,65 %, ja konvolutionaalisella neuroverkolla on saatu jopa 99,77 % oikein [18]. Nykyään monia uusia malleja ja tekniikoita kokeillaan ja mitataan tällä datajoukolla.

Neuroverkon ulostulo luokitteluongelmissa on yleensä One-Hot -muodossa, jossa luokat on järjestelty vektoriin. Neuroverkolle annetaan jokin alkio datajoukosta ja neuroverkon tulee antaa ulostulona vektoriin nollia, mutta toivotun luokan kohdalla tulisi olla arvo 1. Kuvassa 5 on esimerkki One-Hot -muodosta väreillä.

Kuvien tunnistamisessa käytetään usein paremmin soveltuvia konvolutionaalisia neuroverkkoja, joissa yhdistelmä konvoluutio-operaatioita ja eteenpäin syöttäviä neuroverkkoja kykenevät tunnistamaan ympyrämäisiä muotoja, suoria viivoja ja monia muita kuvien muodollisia ominaisuuksia [7].

id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Kuva 5: One-Hot -koodaus väreillä, lähde: [2].

Neuroverkot regressio-ongelmissa

Eteenpäin syöttävät neuroverkot soveltuvat regressio-ongelmiin mainiosti ja niitä sovelletaan monilla eri aloilla arvioimaan tuotteiden hintoja, taloutta tai vaikka biologisia neuroneita. Neuroverkkojen etu perinteiseen lineaariseen regressioon on neuroverkon kyky approksimoida epälineaarista funktiota saaden tarkemman arvion suureesta [6].

Neuroverkkojen opettaminen

Koneoppiminen on laaja käsite, joka käsittelee eri malleja ja tapoja, joilla tietokoneet voivat ”oppia”. Neuroverkot oppivat tietyn datajoukon käyttämällä gradienttimenetelmää, joka on optimointimenetelmä käyttäen gradienttioperaattorin ominaisuutta minimoimaan häviöfunktiota.

Gradienttimenelmä (Stochastic Gradient Decent)

Gradienttimenetelmä [4] on optimointialgoritmi, joka voi löytää funktion minimin käyttäen funktion gradienttia. Gradientti on pinnalle piirretyn tangenttitason jyrkin muutossuunta. Kuvittele olevasi laskeutumassa vuorta alas ja tehtäväsi on päästä laakson matalimpaan kohtaan. Tehtävään on monia ratkaisuja, mutta yksi yksinkertaisimmista tavoista on ensin löytää jyrkin suunta alaspäin ja ottaa askel tähän suuntaan. Tätä jatketaan, kunnes jyrkin suunta alaspäin on vaakatasossa. Tämä algoritmi varmistaa, että löydät aina **jonkinlaisen** laakson. Algoritmi voidaan esittää matemaattisesti näin:

$$\begin{aligned} &\text{toista } \mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla f(\mathbf{x}_n), \\ &\text{kunnes } \|\nabla f(\mathbf{x}_n) - \vec{0}\| < \delta, \end{aligned} \tag{9}$$

missä $\delta > 0$. Algoritmi pyörii N kertaa. Algoritmin lopussa $\mathbf{x}_N$ on funktion lokaali minimi tarkkuudella δ eli

$$\exists \epsilon > 0 \text{ siten, että } \|\mathbf{x} - \mathbf{x}_N\| < \epsilon \implies f(\mathbf{x}_N) \leq f(\mathbf{x}).$$

Edellä α on hyperparametri, joka kuvaa askeleen suuruutta. Se kuvailee, kuinka suuren harppauksen otamme. Jos α on liian suuri, saatat hypätä koko minimin yli.

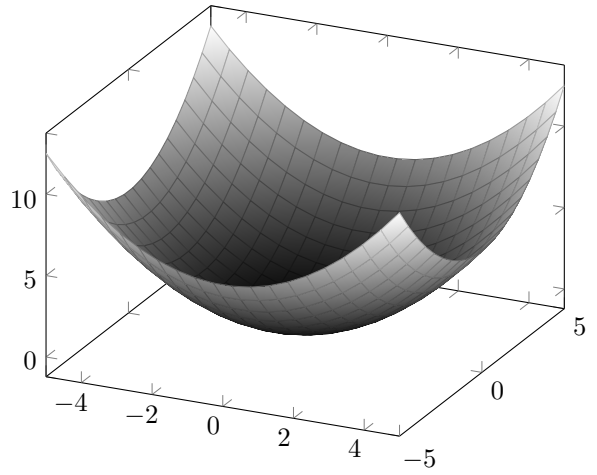
Konvekseilla ja pseudokonvekseilla funktioilla algoritmi löytää lähes aina globaalin minimin [13], mutta ei-konvekseilla funktioilla algoritmi yleensä tökkähtää lokaaliin minimiin. Lokaali minimi ei ole aina hyvä minimi, sillä vuoren harjanteessakin on pieniä kuoppia. Huonojen lokaalien minimien lisäksi moniulotteisissa ei-konvekseissa optimointiongelmissa ovat myös satulapisteet merkittävä ongelma [8].

Neuroverkolle voidaan määritellä häviöfunktio, joka määrittelee suureen kuvaamaan neuroverkon vääryyttä. Neuroverkon treenaamisen alussa se ei vielä tunne datajoukkoa, joten sen antamat ulostulot ovat lähes sattumia. Neuroverkon häviö on siis korkea, koska sen ulostulot ovat roimasti väärässä haluamastamme ulostulosta. Häviöfunktioita on monia ja yksi käytetyimmistä on MSE (Mean Squared Error):

$$\begin{aligned} MSE(\mathbf{o}, \mathbf{t}) =& \frac{1}{N} \sum_{i=1}^{N} (o_i - t_i)^2 \\ =& \frac{1}{N} \mathbf{e}^\top \mathbf{e}, \text{ missä } \mathbf{e} = \mathbf{o} - \mathbf{t}. \end{aligned} \tag{10}$$

MSE on siis saadun ulostulon ja haluamamme ulostulon erotuksien neliöiden keskiarvo. Jos neuroverkkomme ulostulo olisi $\begin{bmatrix} 2.3 \\ 0.5 \end{bmatrix}$, mutta haluaisimme ulostulon $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, olisi häviö

$$\frac{(2.3-1)^2 + (0.5-0)^2}{2} = \frac{1.69 + 0.25}{2} = 0.97.$$



Kuva 6: Häviö kasvaa mitä kauempana odotetusta vektorista ulostulomme on, $z = MSE\left(\begin{bmatrix}x\\y\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}\right)$.

Neuroverkon häviö datajoukolle A määritellään laskemalla keskiarvo jokaisen datajoukon alkion häviöstä. Valitaan symboli $E(\theta)_A$ kuvaamaan tätä häviötä, θ kuvaa neuroverkon parametreja.

Häviöpinta määritellään laskemalla häviö kaikille saman muotoisille neuroverkoille ja merkitsemällä jokainen häviö koordinaatistoon, jossa muuttujina toimivat neuroverkon parametrit. Saman muotoiset neuroverkot ovat neuroverkkoja, joilla on sama määrä kerroksia ja jokaisessa kerroksessa on sama määrä neuroneita. Neuronien painot ja vakioarvot voivat vaihdella.

Neuroverkon häviöpinnan esittäminen ihmiselle ymmärrettävässä muodossa on vaikeaa, sillä neuroverkon parametrien määrää voi olla miljoonia. Kuva 7b on viipale neuroverkon häviöpinnasta [14].

Kuvan 7a musta piste on neuroverkkomme tässä koordinaatistossa. Musta nuoli voisi kuvata treenattavan neuroverkon vaellusta vuoriston pohjalle. Kun häviöfunktio on mahdollisimman alhainen, on neuroverkko tarkimmissa parametreissa. Tällöin musta piste on alhaisimmassa kohdassa ja neuroverkko on oppinut tehtävän. Käytännössä emme laske häviöpintaa, sillä gradienttimenetelmän ei tarvitse nähdä häviöpintaa toimiakseen.

Vastavirta-algoritmi (Backpropagation)

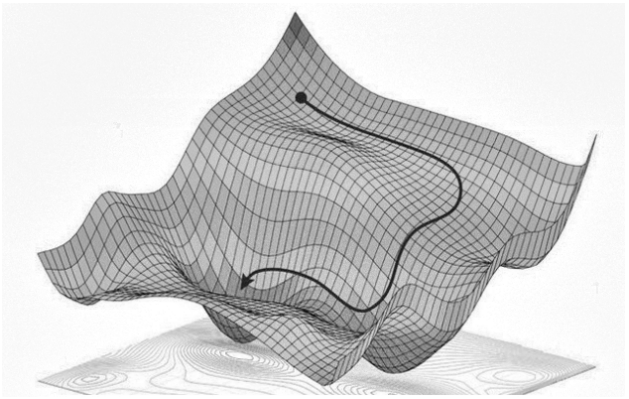
Neuroverkon gradientin tietylle pisteelle voi laskea eri tavoilla. Numeerisesti laskettu gradientti voi olla joskus hyödyllinen, mutta usein käytetään vastavirta-algoritmia, jossa lasketaan häviöfunktion osittaisderivaatat jokaisen parametrin kanssa käyttäen apuna derivaatan ketjusääntöä ja matriisilaskennan työkaluja [4].

Jos neuroverkkojen opettaminen on herättänyt kiinnostuksesi, voit tutustua tarkemmin vastavirta-algoritmiin. YouTube-kanava 3Blue1Brown teki intuitiivisen ja laskennallisen selityksen vastavirta-algoritmista sarjassaan Deep learning. Ehdotan näiden videoiden katsomista ymmärtääksesi algoritmin toiminnan.

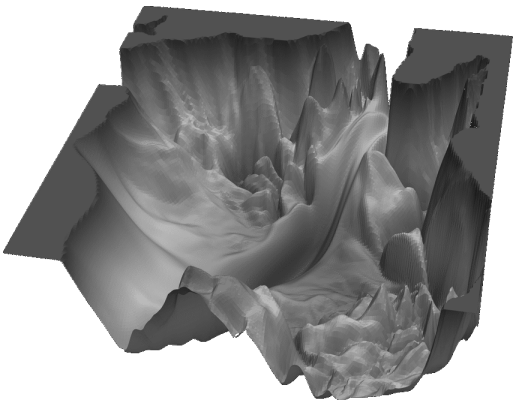
Gradienttimenetelmän optimisaatiota

Tutustumalla tarkemmin gradienttimenetelmään ja koneoppimiseen saattaa törmätä gradienttimenetelmän optimisaatioalgoritmeihin. Optimisaatioalgoritmissa muunnellaan gradienttimenetelmää paremman minimin saavuttamiseksi nopeammin. Monet näistä algoritmeista hyödyntävät vanhojen gradienttien muistamista. Yksi intuitiivisimmista ja yksinkertaisimmista on gradienttimenetelmä liikemäärällä (Momentum), jossa käytetään edellisten gradienttien eksponentaalisesti vähenevää summaa [15]:

$$v_t = \beta v_{t-1} + (1-\beta)\nabla\theta_t(x),$$
$$\theta_{t+1}(x) = \theta_t(x) - v_t,$$



(a) Möykkyinen häviöpinta, lähde: [3].



(b) Oikean neuroverkon häviöpinta, lähde: [14], kuva 3.

Kuva 7: Esimerkkejä neuroverkkojen häviöpinnoista.

missä β on liikemäärän termi, joka on puoliaukinaisella välillä $[0, 1[$. Mitä suurempi beta sitä paremmin algoritmi muistaa vanhat gradientit ja kasvattaa gradientin kokoa.

Algoritmi toimii paremmin kuin perinteinen gradienttimenetelmä, ja se suppenee nopeammin minimiin kuin perinteinen gradienttimenetelmä. Se saavuttaa minimin nopeammin, koska siihen on lisätty nopeuden tyyppinen termi. Tämä termi kasvattaa gradientin suuruutta, jos gradientti on pitkään samaan suuntaan.

Intuitiivisesti Momentum-algoritmin voi ajatella olevan pallo, jolla on kitkaa ja massaa ja joka pyörii häviöpintaa alas. Kuten pallo, voi se pyöriä minimin ympärillä, kunnes hiljalleen osuu minimin kohdalle.

On myös monia muita tapoja kuten Adam (Adaptive Moment Estimation), jotka lisäävät lisää erilaisia nopeus- ja arviointitermejä, minkä avulla voidaan poistaa askelkoon säätäminen.

Tiivistelmä

Tekoneuroni on algoritmi, jonka idea syntyi tutkimalla biologisia neuroneita. Perseptroni on eräänlainen tekoneuroni, jonka aktivointifunktio on askelfunktio. Perseptroni on binäärinen luokittelija, joka voi jakaa datajoukon eri kategorioihin. Perseptronit saavuttavat tämän laskemalla painotetun summan datapisteen komponenteista. Tämän jälkeen tulos syötetään binääriseen askelfunktioon, joka antaa ulos luokan. Voimme organisoida perseptronin painot vektoriin ja nopeuttaa painotetun summan laskemista pistetulon kautta.

Neuroverkko on kasa tekoneuroneita järjestettynä kerroksiin. Neuroverkkojen läpi syötetään dataa laskemalla edellisestä kerroksesta seuraava kerros ja suorittamalla tätä, kunnes päästään viimeiseen kerrokseen. Neuroverkkoja voi opettaa tunnistamaan ja klassifikoimaan kuvia, suorittamaan regressio-ongelmia ja tekemään monenlaisia operaatioita, sillä ne ovat pohjimmiltaan universaaleja funktion approksimoijia.

Neuroverkkoja opetetaan gradienttimenetelmällä, jossa hypimme häviötasoa alaspäin gradientin suuntaan eli siis minimoimme häviöfunktion. Häviöfunktio määrittelee, kuinka väärässä neuroverkko oli näillä parametreilla. Esimerkki tällaisestä funktiosta on vaikka MSE (Mean Squared Error). Gradientti lasketaan vastavirta-algoritmilla.

Gradienttimenetelmää voi optimoida lisäämällä nopeus- ja arviointitermejä, joiden avulla neuroverkko suppenee nopeammin minimiin.

Loppuajatukset

Toivon, että tämän artikkelin avulla sain kiinnostuksen herätettyä koneoppimisen ja erityisesti neuroverkkojen monimutkaiseen maailmaan. Tein itse Rust-ohjelmointikielellä neuroverkkokirjaston, jossa pystyy luomaan neuroverkkoja ja opettamaan niitä käyttämällä yksinkertaisia funktioita.

Lisämateriaaleja (linkkejä) aiheista:

- Perseptronit: Samuli Siltasen sarja Tekoälyn Taikaa
- Neuroverkot: 3Blue1Brownin sarja neuroverkoista
- Neuroverkot: StatQuestin with Josh Starmerin sarja neuroverkoista
- Neuroverkot: Brilliantin kurssi neuroverkoista

Omia neuroverkkoja voi joko testailla omalla kirjastollani tai käyttää Googlen omaa Tensorflow- tai Metan Pytorch-kirjastoa. Nämä kirjastot on optimoitu käytännöllisyyteen ja nopeuteen, ja uskon kirjastojen ominaisuuksien olevan sinulle uusia oppimiskokemuksia.

Kuka olen?

Hei, olen Joonas. Olen toisen vuosikurssin oppilas Otaniemen lukiosta ja minulla heräsi kiinnostus neuroverkkoihin vuoden 2022 lokakuussa, kun tein kaverini Pyry Jäppisen kanssa Rust-ohjelmointikurssin projektiksi pienen neuroverkkoesittelyn Rust-ohjelmointikielellä. Jatkoin tämän tekemistä ja projekti hiljalleen kehittyi omaksi harrastukseksi. Olen ollut aina kiinnostunut matematiikasta ja tietokoneista, ja aihe tuntui täydelliseltä yhdistelmältä. Pyrin jatkamaan tiedonjanoni tyydyttämistä ja oppimaan lisää neuroverkoista ja erilaisista koneoppimisen malleista.

Jos herää kysymyksiä artikkelista ja sen aiheista, niitä voi lähettää sähköpostiin

joonas.vonlerber@gmail.com

Minut löytää githubista, jossa voit seurata tulevaisuuden projektejani tai nykyistä neuroverkkoprojektiani. Teen tätä omalla vapaa-ajallani ja kirjoittamani kirjasto voi muuttua ajan myötä huomattavasti.

Viitteet

[1] URL: `https://miro.medium.com/v2/resize:fit:720/format:webp/1*Z1_IgFO1c6tq4Tz1iwJraw.png`.

[2] URL: `https://miro.medium.com/v2/resize:fit:720/format:webp/1*ggtP4a5YaRx6l09KQaYOnw.png`.

[3] URL: https://www.chegg.com/homework-help/questions-and-answers/10-consider-path-gradient-descent-takes-image-3d-loss-surface-label-parts-path-momentum-wo-q85140081.

[4] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[5] David Beniaguev, Idan Segev ja michael London. Single Cortical Neurons as Deep Artificial Neural Networks. *SSRN Electronic Journal*, tammikuu 2020. DOI: 10.2139/ssrn.3717773.

[6] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[7] Dan Cireşan, Ueli Meier ja Juergen Schmidhuber. Multi-column Deep Neural Networks for Image Classification, 2012. arXiv: 1202.2745 [cs.CV].

[8] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli ja Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014. arXiv: 1406.2572 [cs.LG].

[9] Yoav Freund ja Robert Schapire. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37, helmikuu 1999. DOI: 10.1023/A:1007662407062.

[10] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. en. *Proc. Natl. Acad. Sci. U. S. A.*, 79(8):2554–2558, huhtikuu 1982.

[11] Kurt Hornik, Maxwell Stinchcombe ja Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[12] Kurt Hornik, Maxwell Stinchcombe ja Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. URL: https://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2016/notes/Sonia_Hornik.pdf.

[13] Yaohua Hu, Jiawen Li ja Carisa Kwok Wai Yu. Convergence Rates of Subgradient Methods for Quasi-convex Optimization Problems, 2019. arXiv: 1910.10879 [math.OC].

[14] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer ja Tom Goldstein. Visualizing the Loss Landscape of Neural Nets, 2018. arXiv: 1712.09913 [cs.LG].

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. arXiv: 1609.04747 [cs.LG].

[16] Tomasz Szandala. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. *CoRR*, abs/2010.09458, 2020. arXiv: 2010.09458. URL: https://arxiv.org/abs/2010.09458.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser ja Illia Polosukhin. Attention Is All You Need, 2017. arXiv: 1706.03762 [cs.CL].

[18] Christopher J.C. Burges Yann LeCun Corinna Cortes. THE MNIST DATABASE of handwritten digits. URL: http://yann.lecun.com/exdb/mnist/. Accessed: 2023.