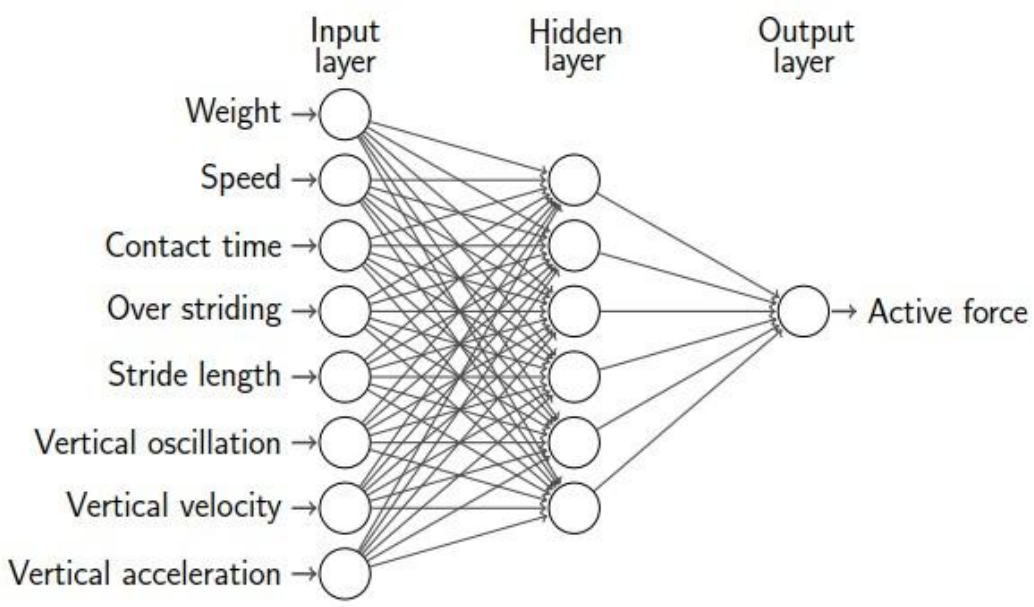


Neuroverkko

Lisätietoa koneoppimisesta

1. T.Kohonen: MATLAB Implementations and Applications of the Self-Organizing Map
2. R.S. Sutton ja A.G. Barto: Reinforcement Learning, An Introduction
3. Y. Zhang: New Advances in Machine Learning
4. H. Daumé: A Course in Machine Learning (University on Maryland)
5. Machine Learning Mastery: Your First Machine Learning Project in Python Step-By-Step
6. Machine Learning Mastery: Machine Learning Algorithms
7. Machine Learning Mastery: How To Implement The Decision Tree Algorithm From Scratch In Python
8. A Complete Tutorial on Tree Based Modeling from Scratch in R and Python
9. Decision Trees (Scikit learn)
10. Python and Machine Learning

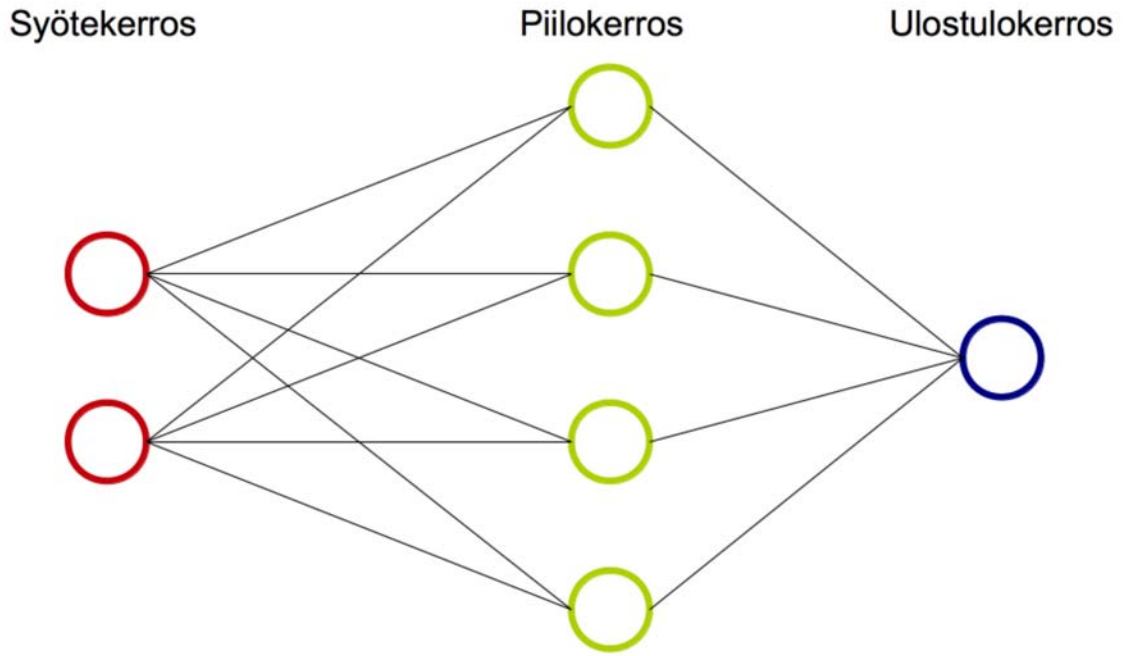
4 Neuroverkkojen matemaattiset perusteet *(Heli Tuominen)*

(Heli Tuominen)

Tässä luvussa tutustutaan neuroverkkojen rakenteeseen, toimintaan ja matemaattisiin perusteisiin. Modernien tekoälysovellusten taustalla on monenlaisia neuroverkkoja, esimerkiksi kuvantunnistukseen hyvin soveltuvia konvoluutioneuroverkkoja. Yksinkertaisuuden vuoksi tässä luvussa käsitellään vain eteenpäinsyöttäviä “tavallisia” neuroverkkoja.

4.1 Keinotekoiset neuroverkot

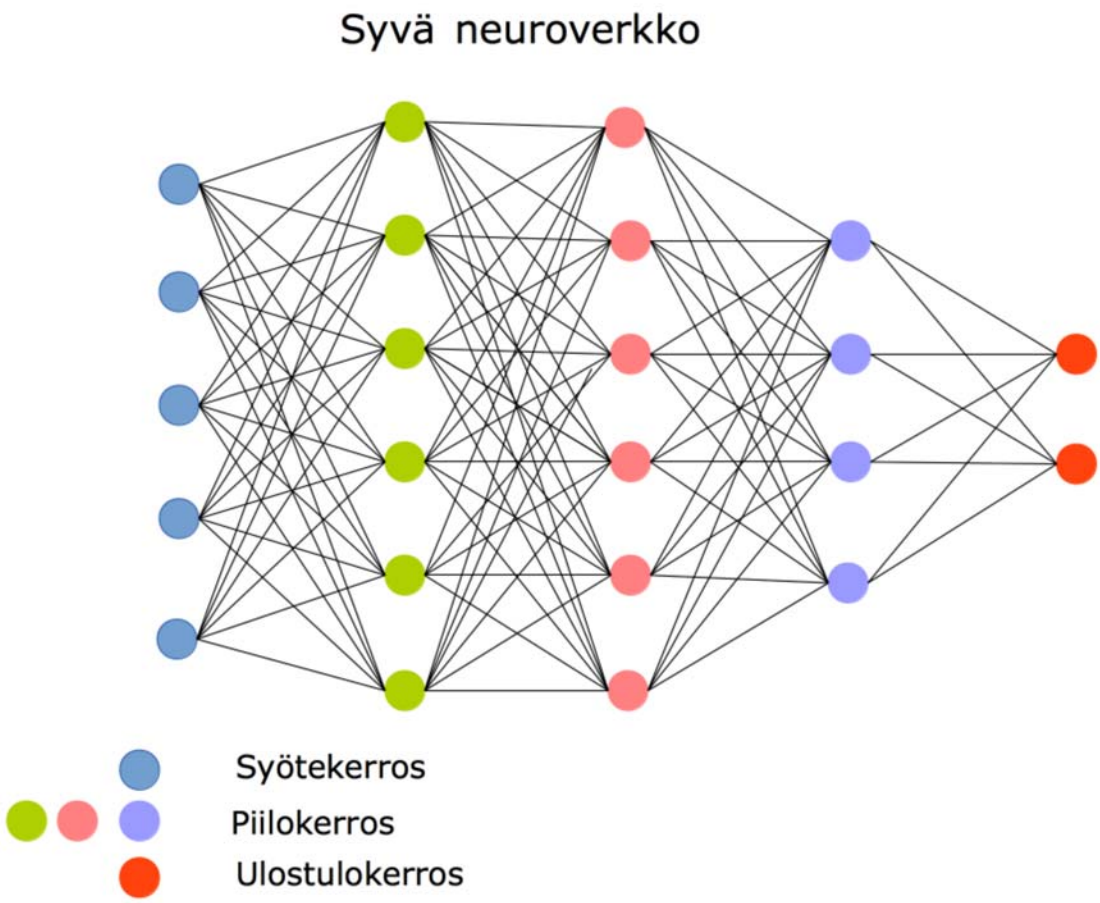
Keinotekoinen neuroverkko *(Artificial neural network)* jäljittelee ihmisen aivoja. Neuroverkko koostuu syöte- ja ulostulokerroksesta *(input layer, output layer)* ja niiden välissä olevista piilokerroksista *(hidden layer)*. Kerrokset puolestaan rakentuvat neuroneista *(neuron)*, joihin liittyy verkon opetuksessa muutettavia parametreja.



Neuroverkko koostuu syötekerroksesta, ulostulokerroksesta ja niiden välissä olevista piilokerroksista.

Data annetaan neuroverkon käsiteltäväksi syötekerroksessa. Syötekerroksen neuroneiden määrä riippuu esimerkiksi siitä, montaako ominaisuutta syötteestä tutkitaan. Piilokerroksien ja ulostulokerroksen kaikissa neuronissa lasketaan syötekerroksesta tai piilokerroksesta tulleiden syötteiden painotettu summa ja siihen lisätään neuronin vakiotermi. Ennen neuronin tuloksen lähettämistä seuraavalle neuronille summa viedään aktivointifunktioon, joka muuttaa lineaarisen (affiinin eli ensimmäisen asteen polynomin) syötteen epälineaariseksi.

Usean piilokerroksen neuroverkkoja sanotaan syviksi neuroverkoiksi *(deep neural network)*.

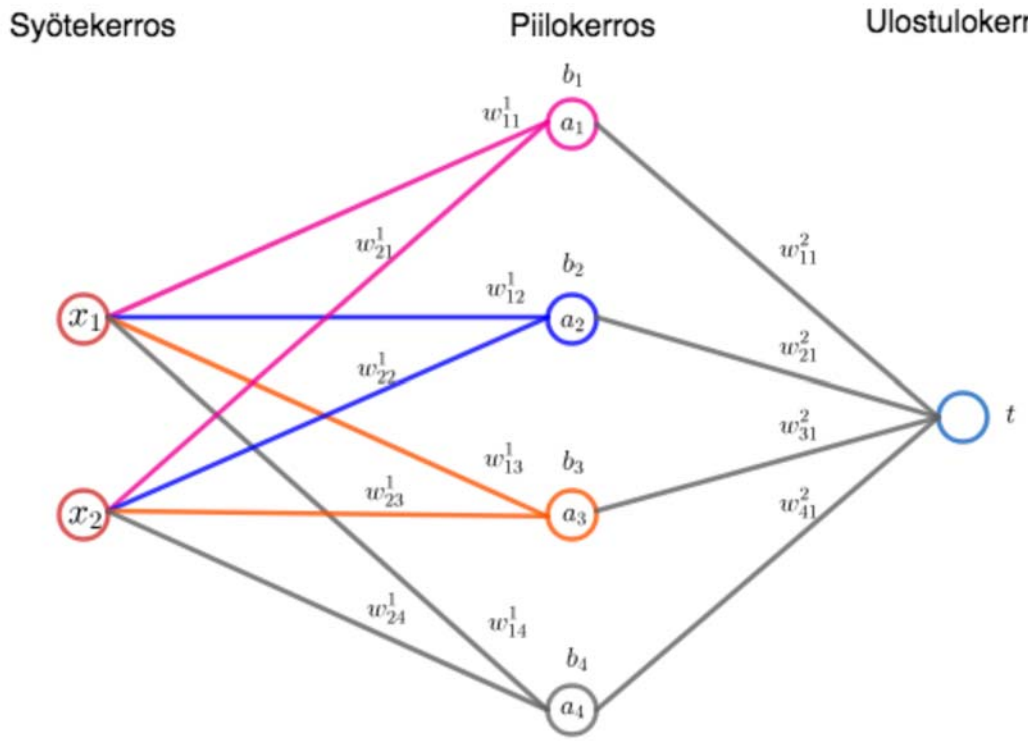


Syvässä neuroverkossa on useita (jopa tuhansia) piilokerroksia.

Neuroverkon toimintaan liittyviä kaavoja tarkastellaan ensin esimerkin avulla ja sitten yleisemmin luvussa.

Esimerkki

Verkossa on yksi piilokerros, jossa on neljä neuronia. Syöte on vektori $x = (x_1, x_2) \in \mathbb{R}^2$. Ulostulokerroksessa on yksi neuroni, jolta saadaan syötettä x vastaava tulos t.



Yhden piilokerroksen neuroverkko.

Syötevektorin komponentit kerrotaan piilokerroksen neuroneiden painoilla w_{ij}^1, tulot lasketaan yhteen ja summaan lisätään piilokerroksen neuronin i vakiotermi b_i:

$$z_1^1 = w_{11}^1 x_1 + w_{21}^1 x_2 + b_1 = \sum_{i=1}^{2} w_{i1}^1 x_i + b_1,$$

$$z_2^1 = w_{12}^1 x_1 + w_{22}^1 x_2 + b_2 = \sum_{i=1}^{2} w_{i2}^1 x_i + b_2,$$

$$z_3^1 = w_{13}^1 x_1 + w_{23}^1 x_2 + b_3 = \sum_{i=1}^{2} w_{i3}^1 x_i + b_3 \text{ ja}$$

$$z_4^1 = w_{14}^1 x_1 + w_{24}^1 x_2 + b_4 = \sum_{i=1}^{2} w_{i4}^1 x_i + b_4.$$

Nämä summat viedään piilokerroksen aktivointifunktiolle, jolloin piilokerroksen neuronien antamat syötteet ulostulokerrokselle ovat

$$a_1 = \varphi(z_1), a_2 = \varphi(z_2), a_3 = \varphi(z_3) \text{ ja } a_4 = \varphi(z_4).$$

Verkon antama tulos saadaan käyttämällä piilokerroksen ja ulostulokerroksen välisiä painoja ja aktivointifunktiota:

$$t = \varphi(z_1^2) = \varphi\Big(\sum_{i=1}^{3} w_{i1}^2 a_i\Big).$$

4.1.1 Neuroverkkoihin liittyviä käsitteitä ja merkintöjä

Neuroni

Neuroverkon kerrokset koostuvat neuroneista *(neuron)*. Jokaiseen piilokerrosten ja ulostulokerroksen neuroniin liittyy kahdenlaisia parametreja, neuroneiden välillä olevat painot *(weight)* ja neuronikohtainen kynnysarvon/vakiotermi *(bias)*.

Parametreistä käytetään seuraavia merkintöjä. Kerrosindeksiä merkitään kirjaimella l. Indeksi $l = 0$ viittaa syötekerrokseen ja indeksi $l = L$ ulostulokerrokseen.

- N_l = kerroksen l neuronien lukumäärä,
- $x_1, \dots, x_{N_0}$ syötteen x komponentit (N_0 kappaletta),
- w_{ij}^l = kerroksen $l-1$ neuronin i ja kerroksen l neuronin j välillä oleva paino,
- b_j^l = kerroksen l neuronin j vakiotermi,
- z_j^l = kerroksen l neuronia j vastaava painotettu summa

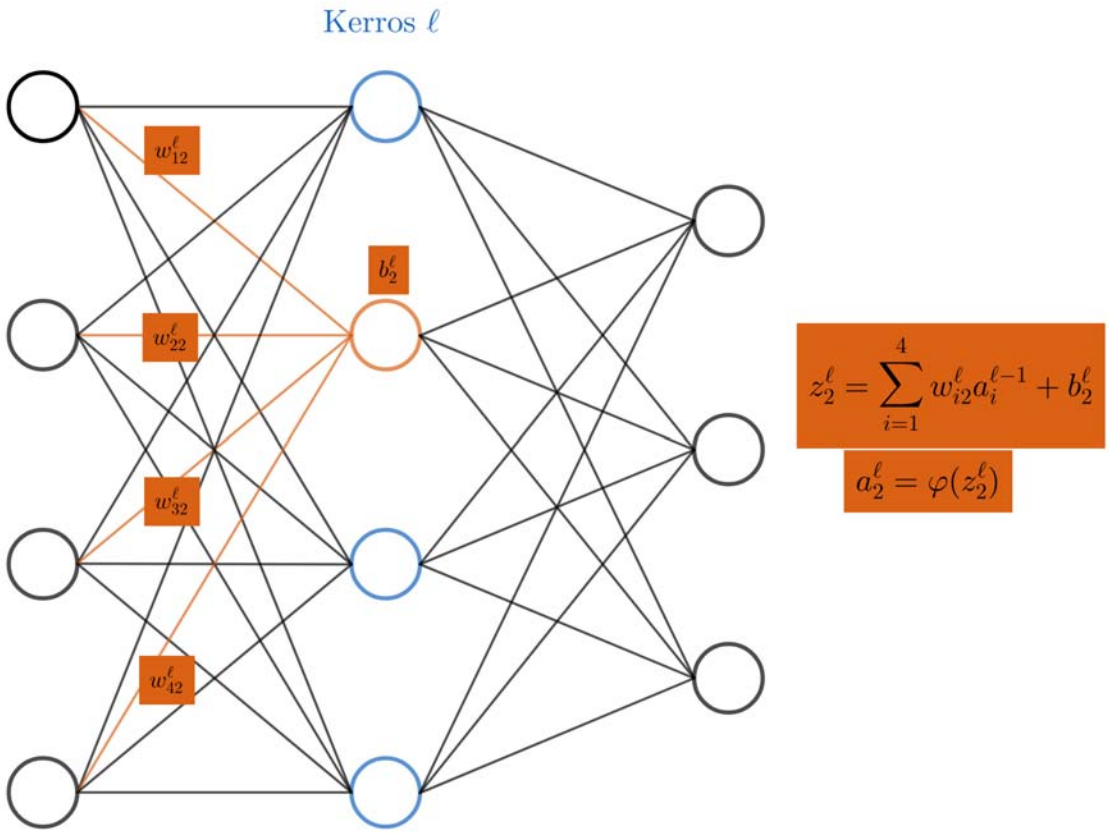
$$z_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^l a_i^{l-1} + b_j^l,$$

- a_j^l = kerroksen l neuronin j tulos eli syöte seuraavaan kerrokseen

$$a_j^l = \varphi(z_j^l) = \varphi\Big(\sum_{i=1}^{N_{l-1}} w_{ij}^l a_i^{l-1} + b_j^l \Big),$$

  missä φ on aktivointifunktio (joka voi vaihdella kerroksesta toiseen).

Huomaa, että $x_j = a_j^0$ kaikilla $j = 1, \dots, N_0$.



Kerroksen ℓ toiseen neuroniin liittyviä kaavoja.

Kaavat vektorimuodossa

Merkintöjen yksinkertaistamiseksi neuroverkon kaavat kirjoitetaan monesti vektori- ja matriisimuodossa.

Kerroksen l kynnysarvoja/vakiotermejä vastaa vektori

$$b^l = (b_1^l, \dots, b_{N_l}^l),$$

kerroksen l neuronien painotettuja summia vektori

$$z^l = (z_1^l, \dots, z_{N_l}^l),$$

ja kerroksen l neuronien tuloksia vektori

$$a^l = (a_1^l, \dots, a_{N_l}^l).$$

Kerroksen l painoja vastaa $N_{l-1} \times N_l$-matriisi

$$W^l = \begin{pmatrix} w_{11}^l & w_{12}^l & \dots & w_{1N_l}^l \\ w_{21}^l & w_{22}^l & \dots & w_{2N_l}^l \\ & & \dots & \\ w_{N_{l-1}1}^l & w_{N_{l-1}2}^l & \dots & w_{N_{l-1}N_l}^l. \end{pmatrix}$$

Painotettujen summien ja neuronien tuloksien vektorit saadaan esitettyä lyhyesti muodossa

$$z^l = a^{l-1}W^l + b^l \text{ ja } a^l = \varphi(z^l) = (\varphi(z_1^l), \dots, \varphi(z_{N_l}^l)).$$

Huomaa, että jos edellä vektorit a, b ja z määriteltäisiin pystyvektoreina, niin olisi

$$z^l = (W^l)^T a^{l-1} + b^l.$$

Joissain lähteissä painojen w_{ij}^l neuroni-indeksit i ja j ovat päinvastaisessa järjestyksessä. Tällöin vastaava matriisi W^l on $N_l \times N_{l-1}$-matriisi ja

$$(z^l)^T = W^l (a^{l-1})^T + (b^l)^T,$$

missä v^T on vektorin v transpoosi.

Matriisien ja vektoreiden ominaisuuksia kerrataan liitteessä Appendix A.

Neuroni ja neuroverkko funktioina

Neuroverkkoa voi ajatella funktiona $f\colon \mathbb{R}^n \to \mathbb{R}^m$; syöte on n-ulotteinen vektori $x = (x_1, x_2, \dots, x_n)$, piilokerrokset hoitavat laskutehtävän ja funktion arvo $f(x) = t = (t_1, t_2, \dots, t_m) \in \mathbb{R}^m$ saadaan ulostulokerroksesta. Verkon käyttötarkoitus määrää, miten funktion arvo t tulkitaan.

Piilokerroksen neuronit voidaan tulkita funktioiksi $f_j^l\colon \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_{l+1}}$,

$$f_j^l(v) = (\varphi_l(g_j^l(v)), \dots, \varphi_l(g_j^l(v))),$$

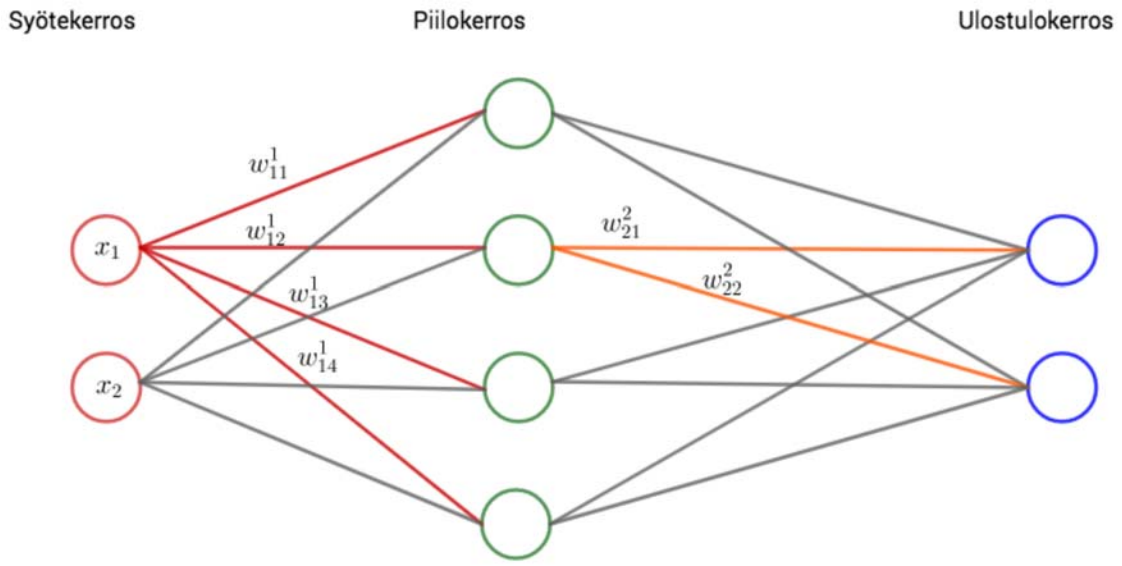
$j \in \{1, 2, \dots, N_l\}$, missä g_j^l on yleensä edellisen kerroksen painotettu summa lisättynä vakiotermillä eli

$$g_j^l(v) = \sum_{i=1}^{N_{l-1}} w_{ij}^l v_i + b_j^l$$

ja φ_l on kerroksen l aktivointifunktio. Ulostulokerroksen funktioille f_j^L arvojoukko on $\mathbb{R}$.

4.1.2 Harjoitus

Tarkastellaan neuroverkkoa, jonka syöte on $x = (x_1, x_2) \in \mathbb{R}^2$, jossa on yksi neljän neuronin piilokerros, jonka ulostulokerroksessa on kaksi neuronia ja jonka aktivointifunktio sekä piilo- että ulostulokerroksessa on $\varphi\colon \mathbb{R} \to \mathbb{R}$. (Kuvassa vain osa painoista merkitty.)

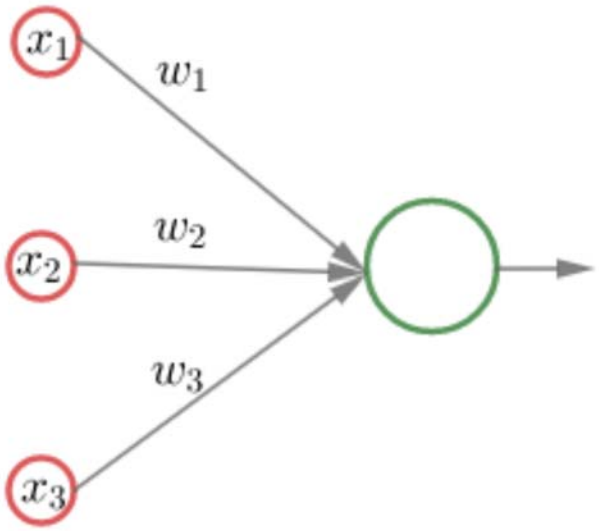


Yhden piilokerroksen neuroverkko.

Kirjoita verkkoon liittyvät neuronien painotetut summat z_i^l ja neuronien tulokset a_i^l vektoreiden ja painomatriisien avulla.

4.1.3 Perseptroni

Perseptroni *(perceptron)* on syötekerroksen ja yhden neuronin muodostama minimaalinen neuroverkko, jonka syöte on $x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ ja tulos on $t \in \{0, 1\}$.



Perseptroni on yksinkertaisin neuroverkko.

Perseptonia, jonka painojen muodostama vektori on $w = (w_1, w_2, \dots w_n) \in \mathbb{R}^n$, vakiotermi on b ja aktivointifunktio on yksikköporrasfunktio (Heavisiden funktio) $h \colon \mathbb{R} \to \{0, 1\}$,

$$h(s) = \begin{cases} 1, & \text{jos } s > 0 \\ 0, & \text{jos } s \leq 0, \end{cases}$$

vastaa funktio $P \colon \mathbb{R}^n \to \{0, 1\}$,

$$P(x) = \begin{cases} 1, & \text{jos } w \cdot x + b > 0 \\ 0, & \text{jos } w \cdot x + b \leq 0, \end{cases}$$

missä $w \cdot x$ on vektoreiden w ja x sisätulo.

Kysymykseen, millaiset funktiot voidaan esittää perseptronilla, on yksinkertainen vastaus nollan ja ykkösen alkukuvien lineaarisen erotettavuuden avulla.

Joukot $A \subset \mathbb{R}^n$ ja $B \subset \mathbb{R}^n$ ovat lineaarisesti erotettavat *(linearly separable)*, jos on vakiot $c_1, c_2, \dots, c_n \in \mathbb{R}$ ja $b \in \mathbb{R}$, joille

$$\sum_{i=1}^{n} c_i x_i > b \quad \text{kaikilla } x \in A$$

ja

$$\sum_{i=1}^{n} c_i x_i \leq b \quad \text{kaikilla } x \in B.$$

Tasossa $\mathbb{R}^2$ tämä tarkoittaa sitä, että joukkoja A ja B vastaavat pisteet voidaan erottaa suoralla ja $\mathbb{R}^3$:ssa sitä, että pistejoukot voidaan erottaa tasolla.

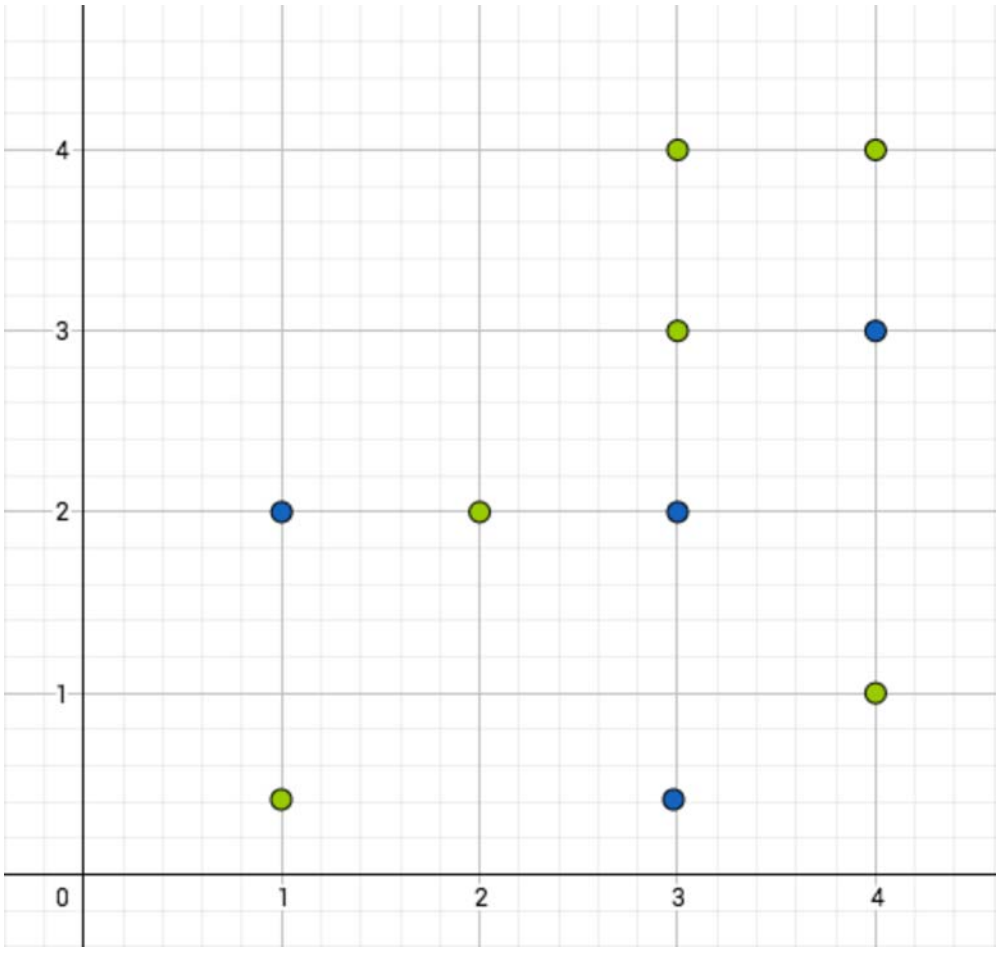
Lause

Funktio $f\colon \mathbb{R}^n \to \{0,1\}$ voidaan esittää perseptronilla jos ja vain jos alkukuvat $f^{-1}(\{0\})$ ja $f^{-1}(\{1\})$ ovat lineaarisesti erotettavat.

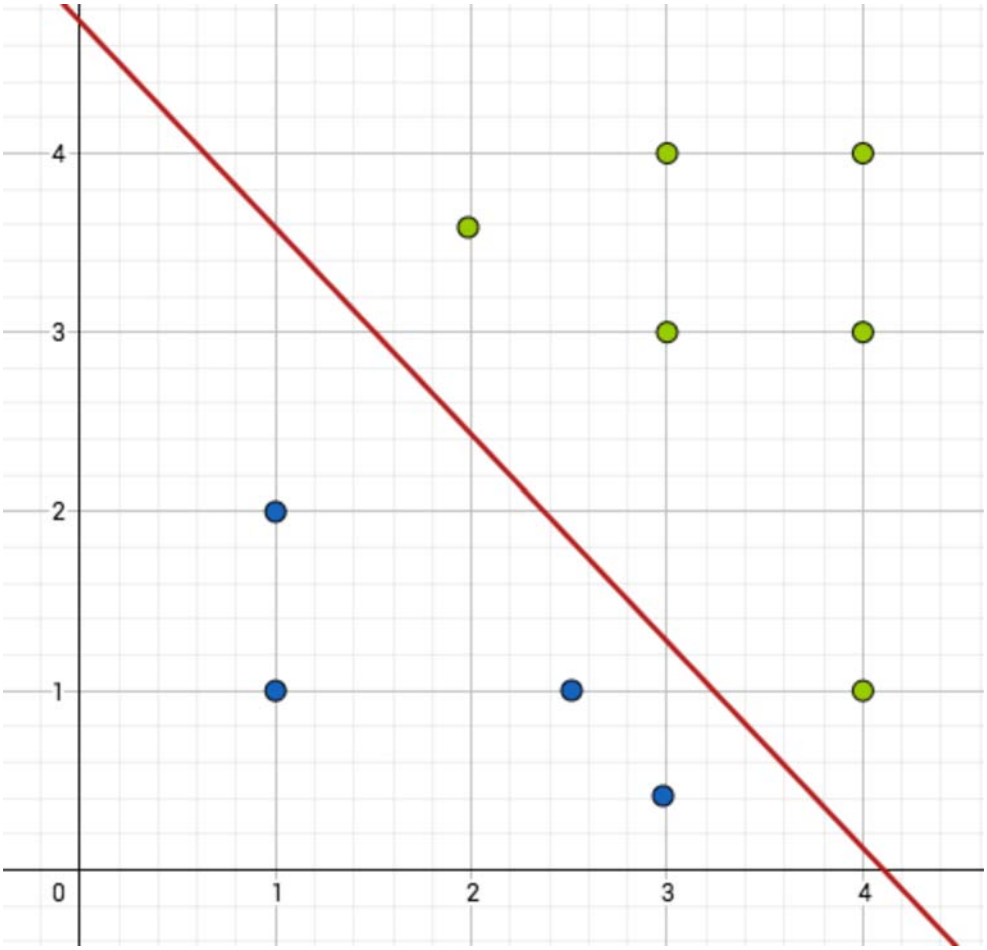
Esimerkki

Ensimmäisen kuvan pistejoukko ei ole lineaarisesti erotettava, toisen kuvan on.

Funktio, jonka arvot vihreitä palloja vastaavissa tason pisteissä on 1 ja sinisiä palloja vastaavissa pisteissä on 0, voidaan siis toisessa tapauksessa esittää perseptronilla, ensimmäisessä ei.



Ei voida esittää perseptronilla.



Voidaan esittää perseptronilla.

Esimerkki

Looginen konnektiivi AND (JA) voidaan esittää yhdellä perseptronilla mutta konnektiivia XOR (poissulkeva TAI) ei voi. Konnektiiveja vastaavat funktiot ovat $AND\colon \{0,1\} \times \{0,1\} \to \{0,1\}$ ja $XOR\colon \{0,1\} \times \{0,1\} \to \{0,1\}$

$$AND(0,0) = AND(0,1) = AND(1,0) = 0$$
$$AND(1,1) = 1$$

ja

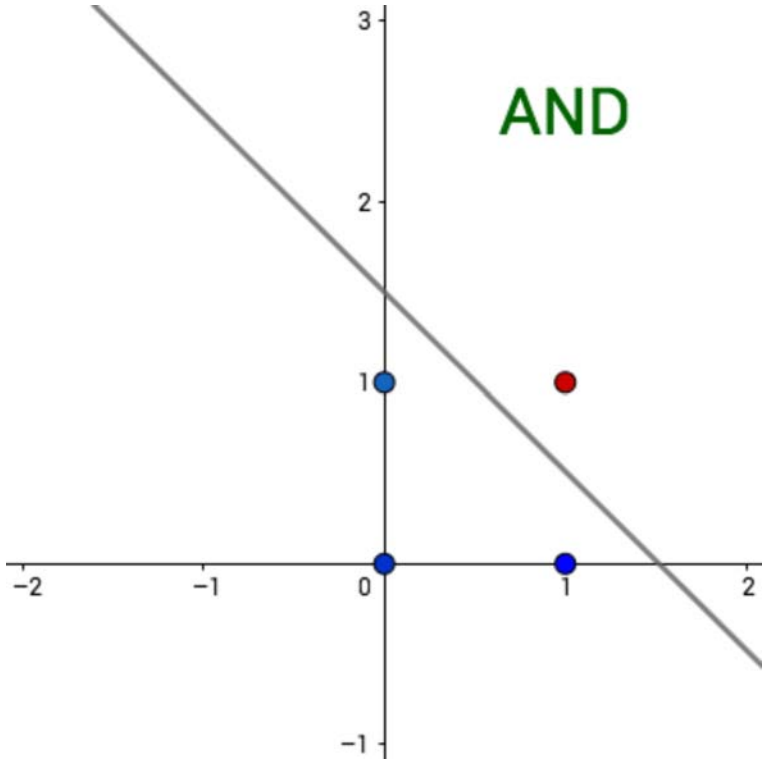
$$XOR(0,0) = XOR(1,1) = 0$$
$$XOR(0,1) = XOR(1,0) = 1.$$

Nollan ja ykkösen alkukuvat ovat siis

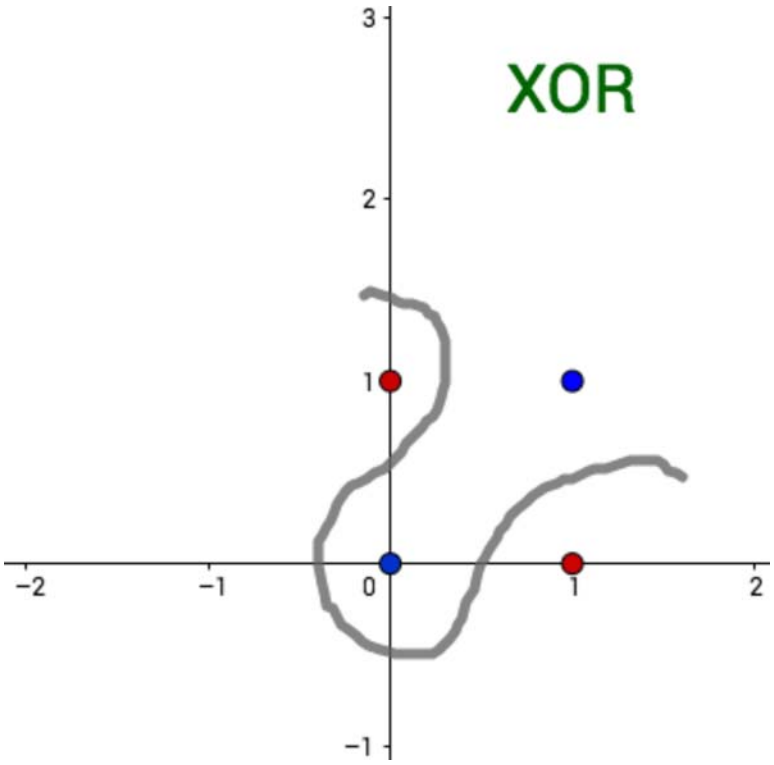
$$AND^{-1}(\{0\}) = \{(0,0),(0,1),(1,0)\}, \quad AND^{-1}(\{1\}) = \{(1,1)\}$$

ja

$$XOR^{-1}(\{0\}) = \{(0,0),(1,1)\}, \quad XOR^{-1}(\{1\}) = \{(0,1),(1,0)\}.$$

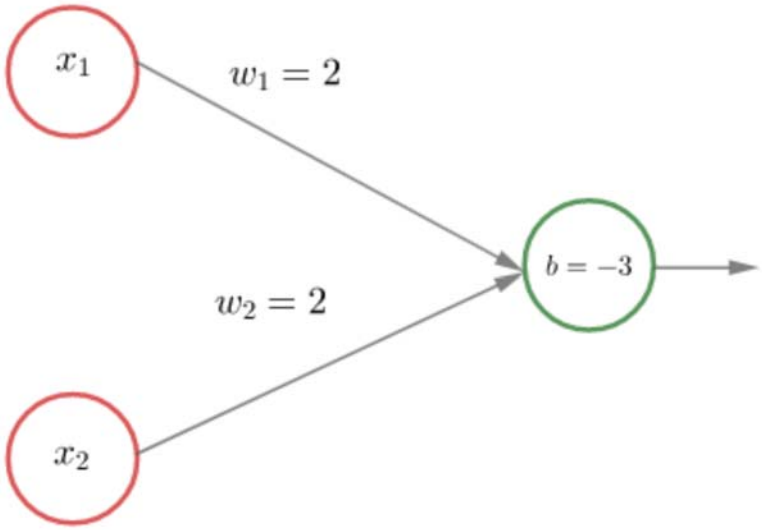


Looginen konnektiivi AND.



Looginen konnektiivi XOR.

Edellisen esimerkin looginen konnektiivi AND seuraavan kuvan perseptronilla.



Loogista konnektiivia AND vastaava perseptroni.

Muuttamalla painoiksi $w_1 = w_2 = -2$ ja vakiotermiksi $b = 3$, saadaan AND-konnektiivin negaatio, NAND, jonka arvo parille $(1, 1)$ on 0 ja muille lukupareille 1. NAND konnektiivin esityksen olemassaolosta seuraa, että perseptronien avulla voidaan rakentaa verkko, joka tekee minkä tahansa halutun loogisen päättelyn.

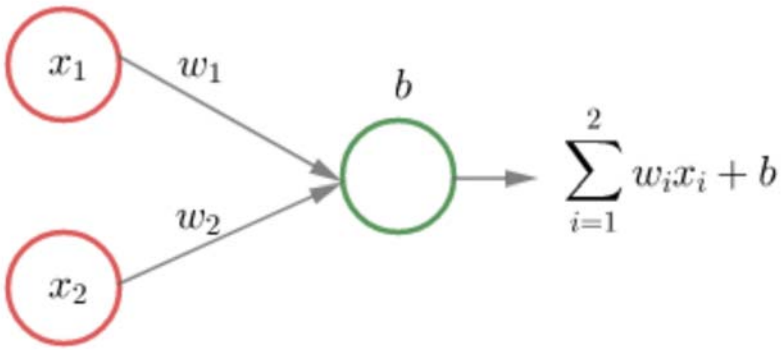
Perseptronin ongelma on se, että pienet muutokset painoissa tai syötteissä aiheuttavat ison muutoksen tuloksessa (0/1). Tämä on huono asia verkon opettamisen kannalta. Perseptronin yksikköporrasfunktion sijaan käytetäänkin yleensä verkon opettamiseen paremmin soveltuvia aktivointifunktioita.

4.1.4 Harjoitus

Loogista konnektiivia OR (TAI) vastaava funktio on
$OR\colon \{0,1\} \times \{0,1\} \to \{0,1\}$,

$$OR(1,1) = OR(0,1) = OR(1,0) = 1 \quad \text{ja} \quad OR(0,0) = 0.$$

1. Määritä alkukuvat $OR^{-1}(\{0\})$ ja $OR^{-1}(\{1\})$.
2. Voidaanko OR esittää perseptronilla? Jos voidaan, niin etsi kertoimet w_1 ja w_2 ja perseptronin vakiotermi b.



Voidaanko OR esittää perseptronilla?

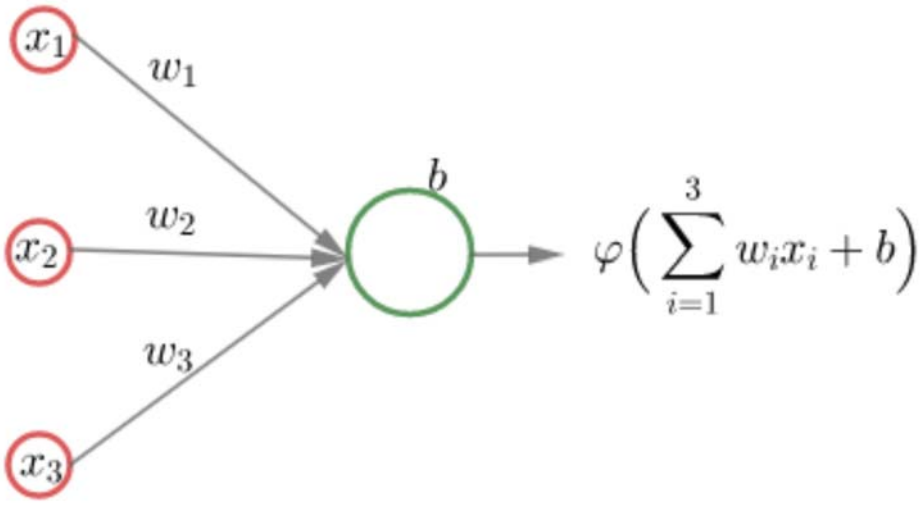
4.1.5 Aktivointifunktiot

Neuroverkon piilo- ja ulostulokerroksissa käytetään aktivointifunktioita *(activation function)*.

Ennen neuronin tuloksen lähettämistä seuraavalle neuronille tai ulostulokerroksesta ulos, edellisen kerroksen syötteistä laskettu painotettu summa viedään aktivointifunktioon $\varphi\colon \mathbb{R} \to \mathbb{R}$. Aktivointifunktiot muuttavat lineaarisen (affiinin eli ensimmäisen asteen polynomin) syötteen epälineaariseksi ja niillä olisi toivottavaa olla seuraavia ominaisuuksia:

- **epälineaarisuus**: Koska summa ja yhdistetty funktio lineaarisista funktioista on lineaarinen ja affiineista affiini ja neuroneiden summalausekkeet $a \mapsto wa + b$ ovat affiineja, niin lineaarisilla (tai affiineilla) aktivointifunktioilla saadaan affiini kuvaus.
- **(jatkuvasti) derivoituvuus**: Vastavirta-algoritmissa ja muissa virhefunktion minimointitavoissa tarvitaan aktivointivointifunktion derivaattaa. Jos aktivointifunktio ei ole derivoituva, niin virhefunktion minimoinnissa pitää käyttää muita kuin gradienttiin perustuvia keinoja.
- **identtisen funktion approksimointi**: Jos aktivointifunktio on nollan lähellä lähellä identtistä funktiota $i\colon \mathbb{R} \to \mathbb{R}$, $i(x) = x$ kaikilla x, niin neuroverkko oppii tehokkaasti kun painot alustetaan satunnaisluvuilla. Muussa tapauksessa painot pitää alustaa huolellisesti.

Se, onko aktivointifunktio rajoitettu vai ei, vaikuttaa verkon oppimisnopeuteen ja oppimisen vakauteen. Rajoitetuilla aktivointifunktioilla oppiminen on yleensä vakaata ja rajoittamattomilla monesti tehokasta. Rajoittamattomia aktivointifunktioita käytettäessä kannattaa käyttää pieniä oppimisnopeuksia.



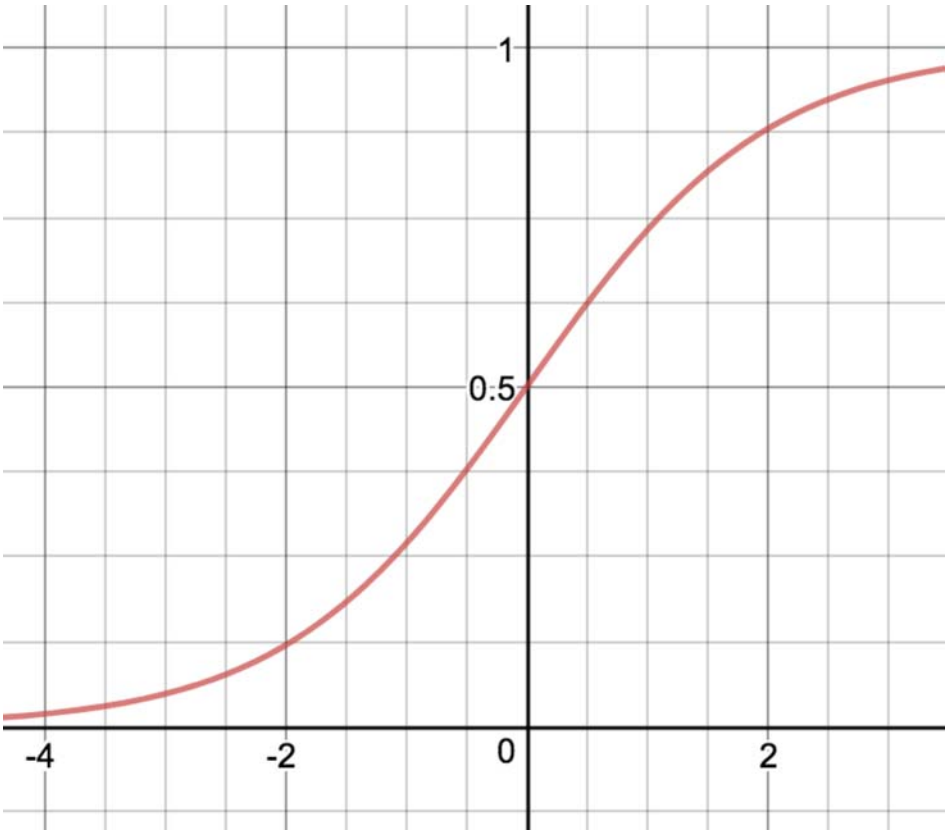
Neuronin painotettu summa viedään aktivointifunktioon φ.

Esimerkkejä aktivointifunktioista ovat sigmoid-funktio, hyperbolinen tangentti ja ReLu-funktio. Tutustutaan näihin lyhyesti.

Sigmoid-funktio (logistinen funktio)

Sigmoid-funktiolla $\sigma\colon \mathbb{R} \to]0,1[$,

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



Sigmoid-funktio

on seuraavat ominaisuudet:

- rajoitettu, aidosti kasvava ja jatkuva
- $\lim_{x\to-\infty} \sigma(x) = 0$, $\lim_{x\to\infty} \sigma(x) = 1$
- $\sigma \in C^\infty(\mathbb{R})$ eli funktiolla σ on kaikkien kertalukujen jatkuvat derivaatat ja

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)).$$

Sigmoid-funktio σ on yksikköporrasfunktion $h\colon \mathbb{R} \to [0,1]$,

$$h(x) = \begin{cases} 0, & \text{kun } x \leq 0, \\ 1, & \text{kun } x > 0, \end{cases}$$

silotettu versio. Sigmoid-funktion huonoin ominaisuus johtuu siitä, että se kasvaa hyvin hitaasti kun x kasvaa ja vähenee hyvin hitaasti kun x vähenee. Sen derivaatta on hyvin lähellä nollaa kun x on suuri tai pieni. Tästä seuraa ongelmia silloin kun verkkoa opetetaan derivaattoihin perustuvilla menetelmillä.

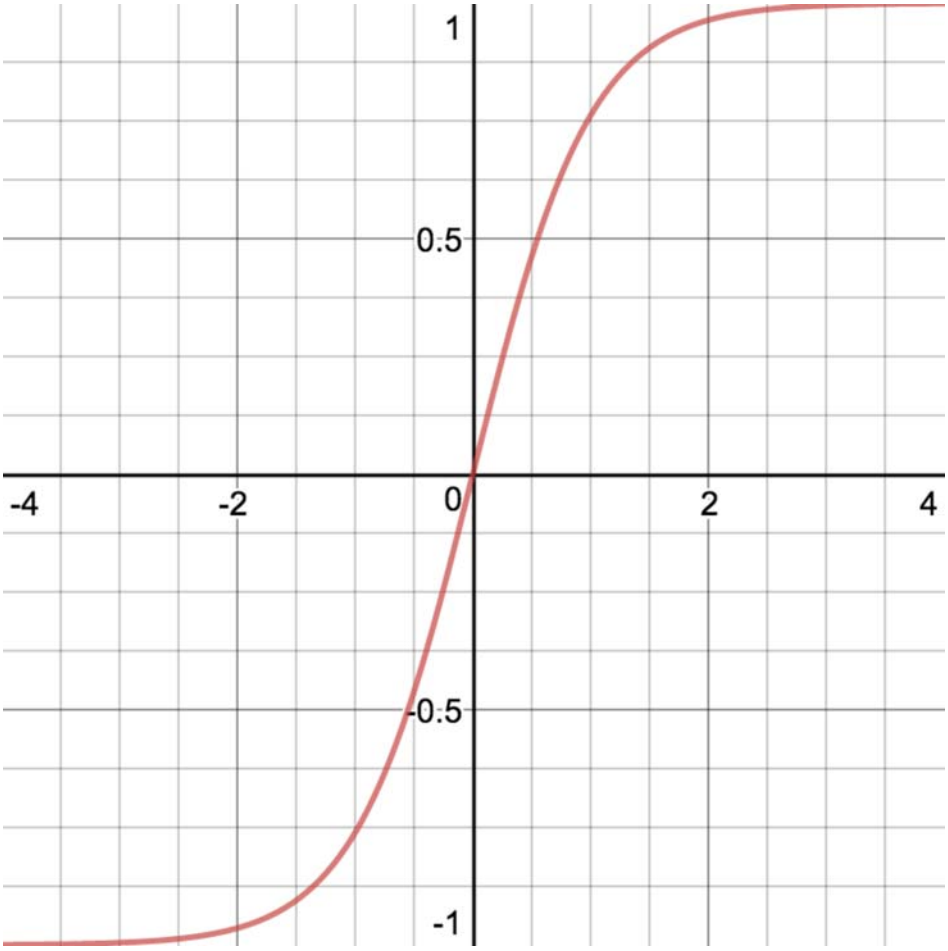
Vastavirta-algoritmin kaavoista nähdään, että virhefunktion osittaisderivaatat neuronin painojen ja vakiotermien suhteen riippuvat aktivointifunktion derivaatasta ja että painojen ja vakioiden muutoksen koulutettaessa ovat pieniä jos osittaisderivaatat ovat pieniä. Tällöin verkko oppii hitaasti.

Toinen sigmoid-funktion huono puoli on se, että se ei ole symmetrinen nollan suhteen. Nykyisin sitä käytetään lähinnä ulostulokerroksessa varsinkin jos verkon tulokset ovat välillä $[0, 1]$.

Hyperbolinen tangentti (tanh)

Hyperbolisella tangenttilla $\tanh\colon \mathbb{R} \to]-1, 1[$,

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



tanh

on monia samoja ominaisuuksia kuin sigmoid-funktiolla mutta se on symmetrinen nollan suhteen ja se kasvaa nopeammin nollan lähellä, jolloin sen derivaatta on suurempi.

Hyperbolinen tangentti on

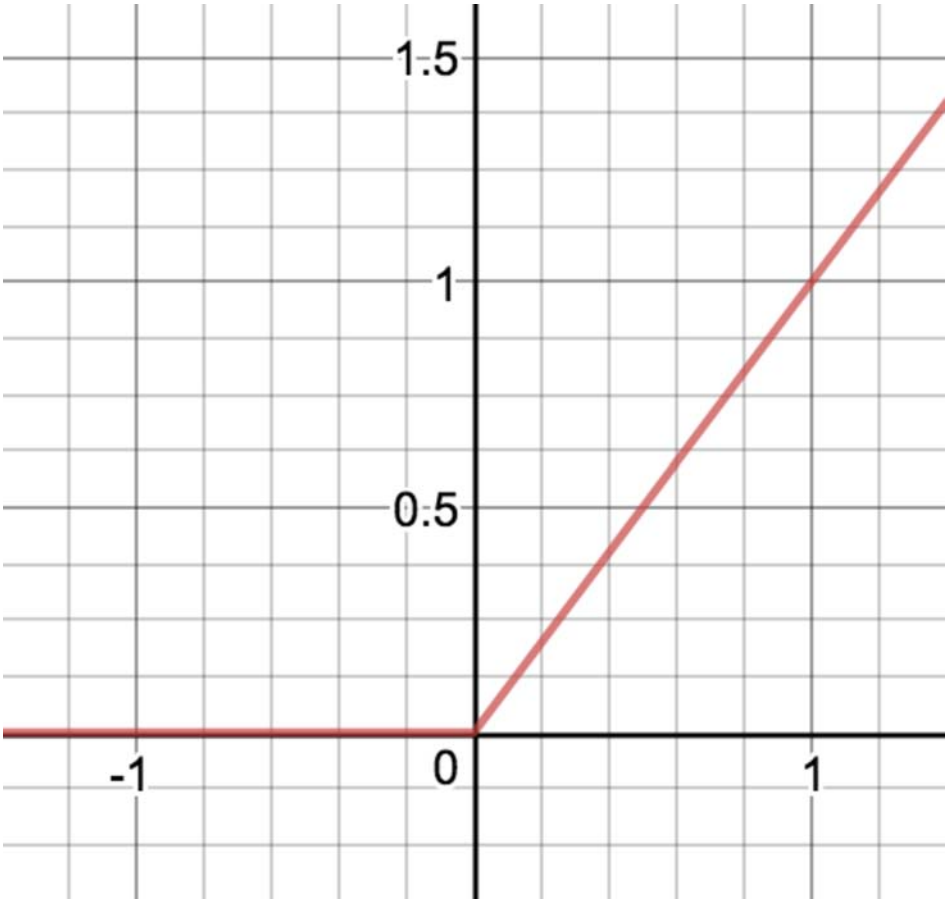
- rajoitettu, aidosti kasvava ja jatkuva
- $\lim_{x\to-\infty} \tanh(x) = -1$, $\lim_{x\to\infty} \tanh(x) = 1$
- $\tanh \in C^\infty(\mathbb{R})$ ja $\tanh'(x) = 1 - \tanh^2(x)$.

Gradientin pienuus isoilla ja pienillä arvoilla on myös hyperbolisen tangentin ominaisuus, joten sen käyttö aktivointifunktiona saattaa aiheuttaa verkon oppimisen hitautta.

ReLu

Neuroverkkojen piilokerroksissa paljon käytetty aktivointifunktifunktio on ReLu-funktio *(Rectified Linear Unit)* $f\colon \mathbb{R} \to [0, \infty[$,

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

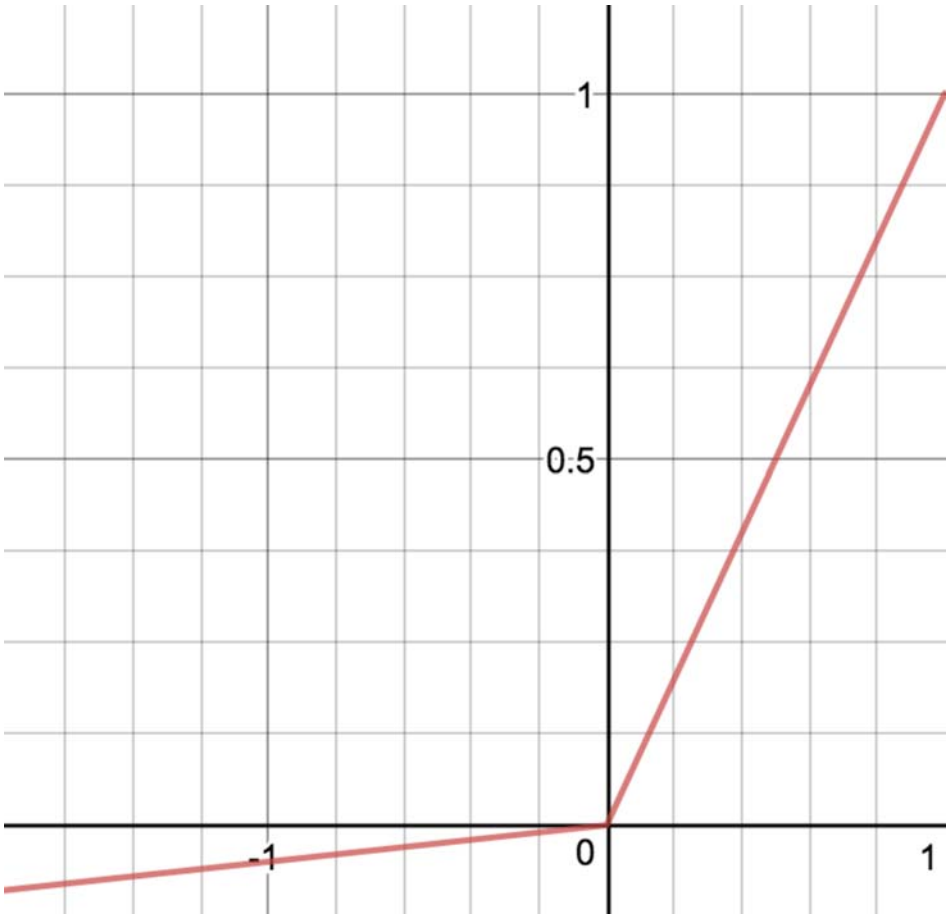


ReLu

Jotta verkon toimintaan saadaan epälineaarisuutta, niin ulostulokerroksessa käytetään epälineaarista aktivointifunktiota. ReLu-funktio ei ole derivoituva nollassa. Sen toinen huono ominaisuus on se, että se on nolla ja sen derivaatta on nolla negatiivisilla arvoilla. Tästä syystä joidenkin neuronien painot saattavat päivittyä oppimisen aikana nollaksi jolloin neuronit "kuolevat". Neuronien kuoleentumisongelmaa pyritään välttämään muuttamalla aktivointifunktiota hieman.

Yksi ReLun variantti on "Leaky ReLu", $f\colon \mathbb{R} \to \mathbb{R}$,

$$f(x) = \max\{ax, x\},\ 0 < a < 1.$$



Leaky ReLu

Toimivimpien aktivointifunktioiden valinta riippuu siitä, mitä verkolla ollaan tekemässä eli mitä funktiota sillä approksimoidaan. Jos verkkoa vastaavalla funktiolla on samoja ominaisuuksia kuin aktivointifunktiolla, niin oppiminen on nopeampaa. Esimerkiksi sigmoid-funktiota kannattaa käyttää ulostulokerroksessa jos verkkoa käytetään luokitteluun 0/1.

4.1.6 Universaali approksimointilause

Funktionaalianalyysin keinoin voidaan todistaa neuroverkkojen universaali approksimointilause, joka sanoo, että jos aktivointifunktio on rajoitettu, kasvava ja jatkuva, niin mille tahansa $\mathbb{R}^n$:n kompaktin joukon jatkuvalle funktiolle on tätä aktivointifunktiota käyttävä neuroverkko, joka approksimoi haluttua funktiota hyvin. (Joukko on kompakti, jos se on suljettu ja rajoitettu.)

Universaali approksimointilause

Olkoon φ rajoitettu, kasvava ja jatkuva funktio. Olkoon $K \subset \mathbb{R}^n$ kompakti joukko. Olkoon $\varepsilon > 0$ ja olkoon $f\colon K \to \mathbb{R}$ jatkuva funktio. Tällöin on $N \in \mathbb{N}$, $v_i, b_i \in \mathbb{R}$, ja $w_i \in \mathbb{R}^n$, $i = 1, \dots, N$, siten, että

$$|F(x) - f(x)| < \varepsilon$$

kaikilla $x \in K$ funktiolle

$$F(x) = \sum_{i=1}^{N} v_i \varphi(w_i^T x + b_i).$$

4.1.7 Harjoitus

1. Näytä, että jos aktivointifunktiona käytetään affiinia funktiota $\varphi\colon \mathbb{R} \to \mathbb{R}$,

   $$\varphi(z) = az + b, \quad a, b \in \mathbb{R},$$

   niin neuroverkkoa vastaava kuvaus on affiini. Huomaa, että riittää todeta, että affiinien kuvausten summa ja yhdistetty kuvaus ovat affiineja.
2. Laske sigmoid-funktion ja hyperbolisen tangentin derivaatat osamäärän derivointisäännön ja ketjusäännön avulla. Muista, että eksponenttifunktiolle $f\colon \mathbb{R} \to]0, \infty[$, $f(x) = e^x$ on $f'(x) = f(x) = e^x$ kaikilla $x \in \mathbb{R}$.

Lisätietoa aktivointifunktioista

- Taulukko aktivointifunktioiden ominaisuuksista (Wikipedia)
- Activation functions and it's types-Which is better?
- A visual proof that neural nets can compute any function
- Understanding Activation Functions in Neural Networks

4.1.8 Neuroverkon opettaminen

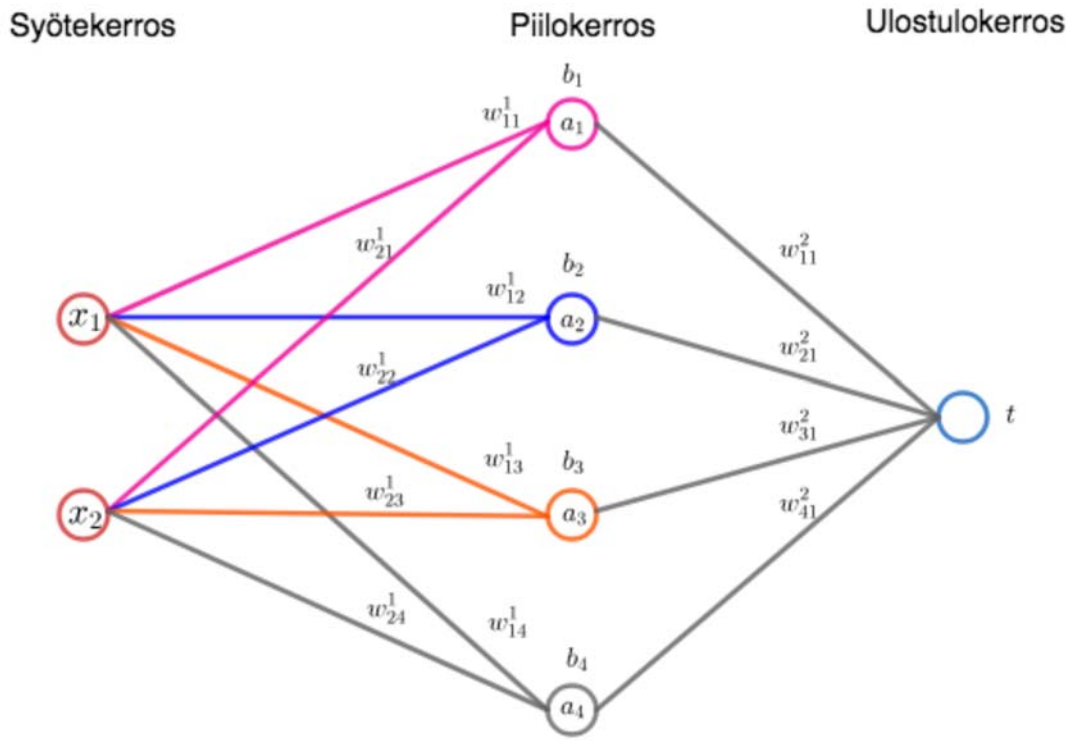
Ohjattua oppimista käytettäessä neuroverkkoa opetetaan syöte-tavoite-pareilla (x, y) eli opetusesimerkeillä *(training examples)*. Verkon syötteelle x antamaa tulosta t verrataan valitulla virhefunktiolla tavoitteeseen y. Opettamisen aikana yritetään minimoida virhefunktioita ja piilokerroksen parametreja muutetaan esimerkiksi vastavirta-algoritmin avulla.

Verkon toiminta varmistetaan ja oppimisnopeus- ja muita hyperparametreja säädetään vahvistus- eli validointiesimerkkijoukon *(validation examples)* avulla.

Kun verkko toimii halutulla tavalla opetusesimerkeille, sen toimintaa tarkastetaan testiesimerkeillä *(test examples)*.

4.1.9 Vastavirta-algoritmi

Eteenpäin kytketyssä neuroverkossa syötekerroksen syötteen komponentit viedään ensimmäisen piilokerroksen neuroneille. Jokaista syötekerroksen neuronia vastaa täsmälleen yksi syötteen komponetti. Jokaisessa ensimmäisen piilokerroksen neuronissa komponentit kerrotaan piilokerroksen neuroneita vastaavilla painoilla, tulot lasketaan yhteen ja summaan lisätään neuronin vakiotermi. Tämä summa syötetään aktivointifunktioon, joka antaa kyseisen neuronin syötteen seuraavalle kerrokselle. Seuraava kerros käyttää omia painojaan, vakiotermejään ja aktivointifunktioitaan. Näin jatketaan kaikkien kerrosten läpi.



Neuroneissa lasketaan painotetut summat edellisen kerroksen syötteistä.

Esimerkiksi kuvan ainoan piilokerroksen toiseen (siniseen) neuroniin tulee syötteen kaksi komponenttia, x_1 ja x_2. Komponentit kerrotaan vastaanottavan neuronin painoilla w_{12}^1 ja w_{22}^1, missä alaindeksit kertovat syötekerroksen ja piilokerroksen neuronin järjestysluvun ja yläindeksi 1 kertoo piilokerroksen järjestysluvun (1. piilokerros). Painotettuun summaan lisätään piilokerroksen toisen neuronin vakiotermi b_2 ja saatu summa

$$z_2^1 = \sum_{i=1}^{2} w_{i2}^1 x_i + b_2 = w_{12}^1 x_1 + w_{22}^1 + b_2$$

viedään ensin aktivointifunktiolle φ ja sitten luku $a_2^1 = \varphi(z_2^1)$ lähetetään ulostulokerrokseen.

Kun neuroverkon laskutoimitukset on tehty, niin syötteen (tai syötejoukon) antamaa tulosta verrataan tavoitteeseen ja lasketaan virhefunktion arvo. Tavoitteena on minimoida opetusesimerkkijoukkoa vastaava virhefunktio ja löytää minimointia vastaavat painot neuroneille.

Useissa virhefunktion minimointikeinoissa kuten gradienttimenetelmässä *(gradient descent)* tarvitaan virhefunktion E osittaisderivaatat $\frac{\partial E}{\partial w}$ ja $\frac{\partial E}{\partial b}$ verkon kaikkien painojen w ja vakiotermien b suhteen. Osittaisderivaatat ja jokaisen neuronin vaikutus virheeseen lasketaan usein vastavirta-algoritmilla *(backpropagation)*. Gradienttimenetelmässä neuroneille saadaan uudet painot ja vakiotermit muuttamalla edellisiä arvoja neuronien parametrien osittaisderivaatoista koostuvan gradientin vastavektorin suuntaan (eli virhefunktion nopeimman pienenemisen suuntaan).

Monesti yksittäistä syötettä x vastaavan tavoitteen $y \in \mathbb{R}^m$ ja verkon antaman tuloksen

$t \in \mathbb{R}^m$ virhefunktiona käytetään erotuksen euklidisen normin neliötä

$$E = \frac{1}{2}\|t - y\|^2 = \frac{1}{2}\sum_{k=1}^{m}(t_k - y_k)^2$$

ja opetusesimerkkijoukon A virhefunktiona keskineliösummaa *(mean squared error)*

$$E_A = \frac{1}{2N}\sum_{x \in A}\|(t(x) - y(x)\|^2,$$

missä N on joukon A opetusesimerkkien lukumäärä.

Seuraavaksi lasketaan virhefunktion E osittaisderivaatat $\frac{\partial E}{\partial w}$ ja $\frac{\partial E}{\partial b}$ verkon kaikkien painojen w ja vakiotermien b suhteen vastavirta-algoritmilla. Derivaatan ja osittaisderivaattojen määritelmät esimerkkeineen löytyvät luvusta.

Ulostulokerroksen osittaisderivaatat

Ulostulokerroksen parametreihin liittyvät osittaisderivaatat on helppo laskea. Aloitetaan esimerkillä.

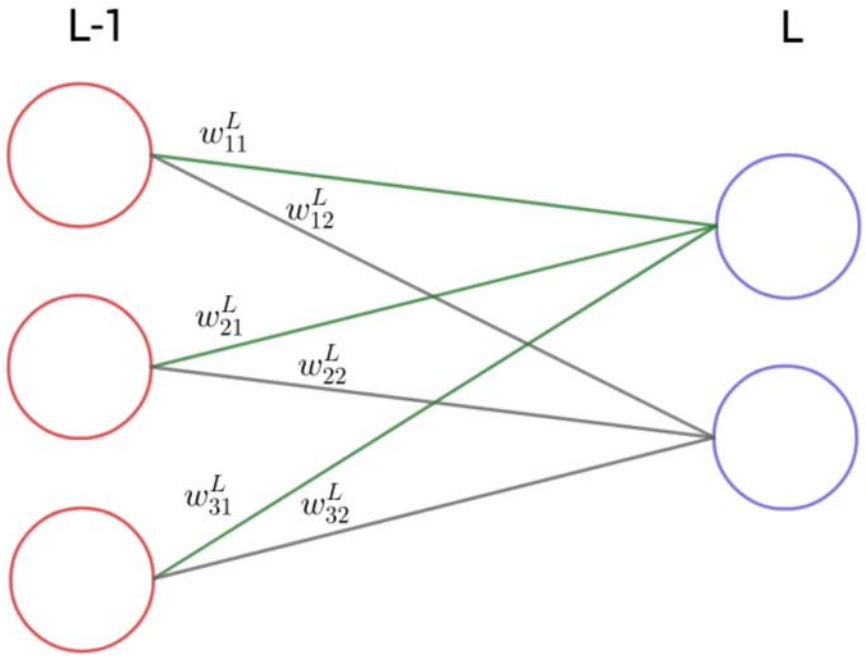
Esimerkki

Ulostulokerroksessa (L. kerros) on kaksi ja viimeisessä piilokerroksessa ($(L-1)$. kerros) kolme neuronia. Virhefunktio on

$$E = \frac{1}{2}\Big((t_1 - y_1)^2 + (t_2 - y_2)^2\Big)$$

ja ulostulokerroksen neuronien tulokset ovat

$$z_j = \sum_{k=1}^{3} w_{kj}^L a_k^{L-1} + b_j^L \text{ ja } t_j = \varphi(z_j), \quad j = 1, 2.$$



Ulostulokerroksen ja viimeisen piilokerroksen väliset painot.

Oletetaan, että aktivointifunktio on identtinen funktio $\varphi(x) = x$ ja että ulostulokerroksen vakiotermit ovat nollia. Tällöin $t_1 = z_1$ ja $t_2 = z_2$.

Lasketaan virhefunktion E osittaisderivaatat painojen $w_{ij} = w_{ij}^L$ suhteen. Koska kaavan perusteella painot w_{11}, w_{21} ja w_{31} eivät vaikuta ulostuloon t_2, niin virhefunktion termi $(t_2 - y_2)^2$ on vakio osittaisderivoinnessa painojen w_{11}, w_{21} ja w_{31} suhteen. Siten derivoinnin ketjusäännön avulla nähdään, että kaikilla $i = 1, 2, 3$ on

$$\frac{\partial E}{\partial w_{i1}^L} = \frac{\partial}{\partial w_{i1}^L}\frac{1}{2}(t_1 - y_1)^2 = (t_1 - y_1)\frac{\partial}{\partial w_{i1}^L}(t_1 - y_1).$$

Koska summan termit, joissa on kertoimena w_{k1}^L, $k \neq i$, ovat muuttujan w_{i1} suhteen vakiota, niin kaikilla $i = 1, 2, 3$ on

$$\frac{\partial}{\partial w_{i1}^L}(t_1 - y_1) = \frac{\partial}{\partial w_{i1}^L}\sum_{k=1}^{3} w_{k1}^L a_k^{L-1} = a_i^{L-1}.$$

Vastaavasti saadaan, että

$$\frac{\partial E}{\partial w_{i2}^L} = \frac{\partial}{\partial w_{i2}^L}\frac{1}{2}(t_2 - y_2)^2 = (t_2 - y_2)\frac{\partial}{\partial w_{i2}^L}(t_2 - y_2) = \frac{\partial}{\partial w_{i2}^L}\sum_{k=1}^{3} w_{k2}^L a_k^{L-1}$$

ja

$$\frac{\partial}{\partial w_{i2}^L}(t_2 - y_2) = \frac{\partial}{\partial w_{i2}^L}\sum_{k=1}^{3} w_{k2}^L a_k^{L-1} = a_i^{L-1}.$$

Palataan nyt yleiseen tilanteeseen. Olkoon ulostulokerros verkon L. kerros ja olkoon siinä m neuronia.

Osittaisderivaatat painojen w_{ij}^L suhteen

Koska virhefunktiossa termit

$$(t_k - y_k)^2 = \Big(\varphi\Big(\sum_{i=1}^{N_{L-1}} w_{ik}^L a_i^{L-1} + b_j^L\Big) - y_k\Big)^2$$

ovat vakioita painon w_{ij}^L suhteen kun $j \neq k$, niin derivoinnin ketjusääntöä käyttämällä saadaan

$$\frac{\partial E}{\partial w_{ij}^L} = \frac{\partial}{\partial w_{ij}^L}\frac{1}{2}\sum_{k=1}^{m}(t_k - y_k)^2 = (t_j - y_j)\frac{\partial}{\partial w_{ij}^L}(t_j - y_j).$$

Huomaa, että syötteiden tulokset y_k ovat vakioita kaikkien painojen w_{ij}^L suhteen ja siten niiden osittaisderivaatat ovat nollia. Siten kaikilla $j = 1, \ldots m$ saadaan ketjusäännön avulla

$$\frac{\partial}{\partial w_{ij}^L}(t_j - y_j) = \frac{\partial}{\partial w_{ij}^L}t_j = \frac{\partial}{\partial w_{ij}^L}a_j^L = \frac{\partial}{\partial w_{ij}^L}\varphi(z_j^L) = \varphi'(z_j)\frac{\partial}{\partial w_{ij}^L}z_j^L.$$

Koska $z_j^L = \sum_{k=1}^{N_{l-1}} w_{kj}^l a_k^{L-1} + b_j^l$ ja muut termit summassa paitsi $w_{ij}^L a_i^{L-1}$ ovat vakioita painon w_{ij}^L suhteen, niin

$$\frac{\partial}{\partial w_{ij}^L}z_j^L = \frac{\partial}{\partial w_{ij}^L}\Big(\sum_{k=1}^{N_{L-1}} w_{kj}^L a_k^{L-1} + b_j^L\Big) = \frac{\partial}{\partial w_{ij}^L}(w_{ij}^L a_i^{L-1}) = a_i^{L-1}$$

Yhdistämällä nämä laskut saadaan

$$\boxed{\frac{\partial E}{\partial w_{ij}^L} = (t_j - y_j)\varphi'(z_j)a_i^{L-1}}.$$

Tämän kaavan indeksistä j riippuvaa osaa merkitään usein

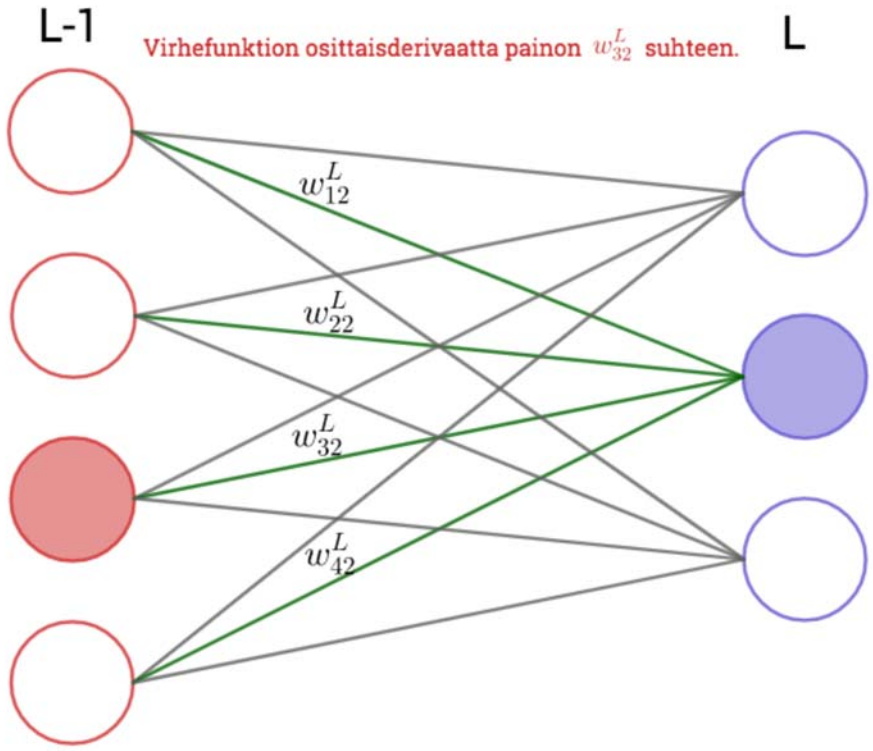
$$\delta_j^L = (t_j - y_j)\varphi'(z_j).$$

Laskemalla huomataan, että

$$\delta_j^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\varphi'(z_j^L).$$

Siten on

$$\frac{\partial E}{\partial w_{ij}^L} = \delta_j^L a_i^{L-1}.$$



Virhefunktion osittaisderivaatta viimeisen piilokerroksen painon suhteen.

Osittaisderivaatat vakiotermien b_j^L suhteen

Ulostulokerroksen osittaisderivaatat vakiotermien suhteen saadaan laskettua samaan tapaan kuin painojen suhteen. Virhefunktiossa termit

$$(t_k - y_k)^2 = \Big(\varphi\Big(\sum_{i=1}^{N_{L-1}} w_{ik}^L a_i^{L-1} + b_j^L\Big) - y_k\Big)^2$$

ovat vakioita termin b_j^L suhteen kun $j \neq k$, joten ketjusääntöä käyttämällä saadaan

$$\begin{aligned}\frac{\partial E}{\partial b_j^L} &= \frac{\partial}{\partial b_j^L}\frac{1}{2}\sum_{k=1}^{m}(t_k - y_k)^2 = (t_j - y_j)\frac{\partial}{\partial b_j^L}(t_j - y_j) \\ &= (t_j - y_j)\frac{\partial}{\partial b_j^L}\varphi(z_j^L) = (t_j - y_j)\varphi'(z_j)\frac{\partial}{\partial b_j^L}z_j^L \\ &= (t_j - y_j)\varphi'(z_j),\end{aligned}$$

sillä viimeisessä osittaisderivoinnissa ainoastaan summan termi b_j^L vaikuttaa derivointiin ja sen osittaisderivaatta b_j^L:n suhteen on 1. Siten saadaan

$$\boxed{\frac{\partial E}{\partial b_j^L} = (t_j - y_j)\varphi'(z_j) = \delta_j^L}.$$

Huomautus

Ulostulokerroksen j. neuroniin liittyvää virhettä $\delta_j^L = \frac{\partial E}{\partial z_j^L}$, ketjusääntöä ja z_j^L:n laskukaavaa käyttäen saadaan vastaavat kaavat myös muille virhefunktioille, joita merkitään tässä myös E:llä,

$$\frac{\partial E}{\partial w_{ij}^L} = \frac{\partial E}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{ij}^L} = \delta_j^L a_i^{L-1} \quad \text{ja} \quad \frac{\partial E}{\partial b_j^L} = \frac{\partial E}{\partial z_j^L}\frac{\partial z_j^L}{\partial b_j^L} = \delta_j^L.$$

Seuraavaksi lasketaan virhefunktion osittaisderivaatat piilokerroksien painojen ja vakiotermien suhteen. Laskun avulla nähdään, että osittaisderivaatat kerroksen l suhteen saadaan laskettua rekursiivisesti kun tiedetään yhtä ylemmän kerroksen osittaisderivaatat. Osittaisderivaattoja laskettaessa lähdetään siis liikkeelle ulostulokerroksen osittaisderivaatoista ja niitä käytetään ensimmäisen piilokerroksen derivaattojen laskemiseen. Osittaisderivaatat viimeisen piilokerroksen painojen ja vakioiden suhteen antavat vastaavat osittaisderivaatat viimeistä edelliselle piilokerrokselle. Näin jatketaan kunnes virhefunktion kaikki osittaisderivaatat saadaan laskettua. Nimi vastavirta-algoritmi tulee siitä, että osittaisderivaattoja lasketaan takaperoisesti ulostuloskerroksesta syötekerrosta kohti vastavirtaan.

Osittaisderivaatat piilokerroksen painojen w_{ij}^l suhteen

Lasketaan virhefuntion osittaisderivaatat piilokerroksen painojen suhteen Huomatuksen tyylillä. Lasku on teknisesti hieman haastavampi sillä paino, jonka suhteen osittaisderivoidaan, vaikuttaa virheeseen yhden tai useamman piilokerroksen kautta. Siksi laskussa tarvitaan tavallisen ketjusäännön lisäksi osittaisderivaattojen ketjusääntöä.

Käytetään tässäkin merkintää

$$\delta_j^l = \frac{\partial E}{\partial z_j^l}$$

kerroksen l neuroniin j liittyvälle virheelle.

Ketjusääntöä ja kaavaa $z_j^l = \sum_{n=1}^{N_{l-1}} w_{nj}^l a_n^{l-1} + b_j^l$ käyttäen saadaan

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{ij}^l} = \delta_j^l a_i^{l-1}$$

Osittaisderivaattojen ketjusäännön, ketjusäännön ja kaavojen

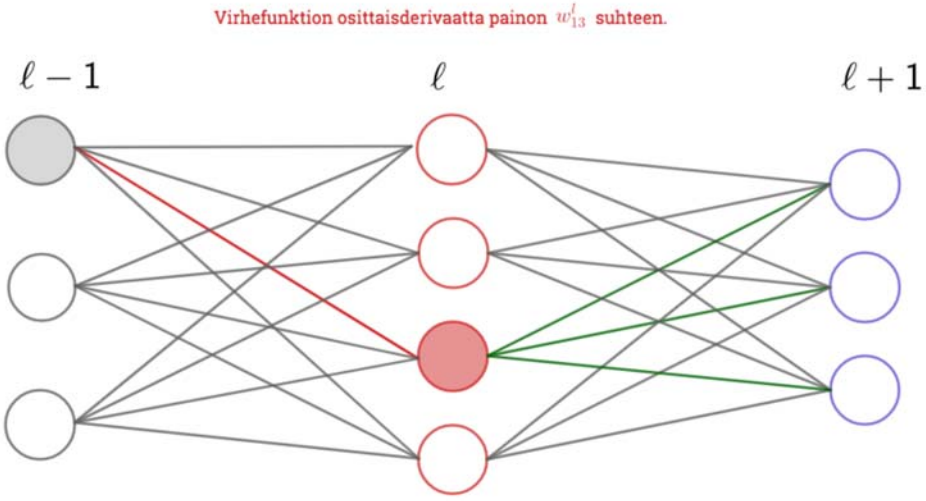
$$z_k^{l+1} = \sum_{n=1}^{N_l} w_{nj}^{l+1} a_n^l + b_j^{l+1} \text{ ja } a_j^l = \varphi(z_j^l)$$

perusteella on

$$\begin{aligned}\delta_j^l = \frac{\partial E}{\partial z_j^l} &= \sum_{k=1}^{N_{l+1}} \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_k^l} \\ &= \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{jk}^{l+1} \varphi'(z_j^l)\end{aligned}$$

ja siten

$$\boxed{\frac{\partial E}{\partial w_{ij}^l} = a_i^{l-1} \varphi'(z_j^l) \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{jk}^{l+1}}.$$



Virhefunktion osittaisderivaatta kerroksen ℓ painon suhteen.

Osittaisderivaatat piilokerroksen vakiokertoimien b_j^l suhteen

Samaan tapaan kuin painojen tapauksessa saadaan

$$\frac{\partial E}{\partial b_j^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \cdot 1$$

ja

$$\boxed{\frac{\partial E}{\partial b_j^l} = \varphi'(z_j^l) \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{jk}^{l+1}}.$$

Huomioita osittaisderivaattojen kaavoista

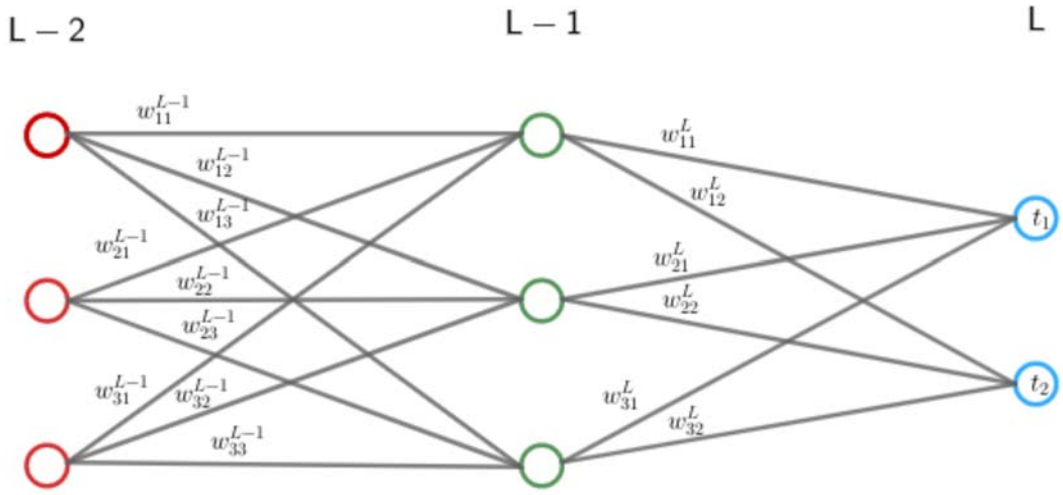
Kaavasta nähdään, että jos edellisen kerroksen $l-1$ syöte a_i^{l-1} on pieni, niin kerroksen l painoa vastaava virheen osittaisderivaatta $\frac{\partial E}{\partial w_{ij}^l}$ on pieni. Tällaiset painot muuttuvat vastavirta-algoritmin aikana vähän, monesti sanotaan, että ne oppivat hitaasti.

Kaavoista nähdään myös, että aktivointifunktion derivaatat vaikuttavat virheen osittaisderivaattoihin ja siten neuroneiden parametrien muutokseen. Jos derivaatta on hyvin pieni, niin parametrit muuttuvat vähän ja neuronit oppivat hitaasti. Tästä syystä verkon käyttötarkoituk-seen sopivan virhefunktion valinta on tärkeää.

Verkon eri kerroksissa voidaan käyttää eri aktivointifunktioita. Jos näin on, niin äskeisissä laskuissa ja kaavoissa aktivointifunktioon φ lisätään verkon kerrosta vastaavat alaindeksit l.

4.1.10 Harjoitus

1. Laske virhefunktion osittaisderivaatat piilokerroksen $L-1$ painojen w_{ij}^{L-1} suhteen virhefunktiolle E samaan tapaan kuin ulostulokerroksen osittaisderivaatat. Mieti, mitkä painoista w_{ij}^{L-1} vaikuttavat ulostuloon t_1.



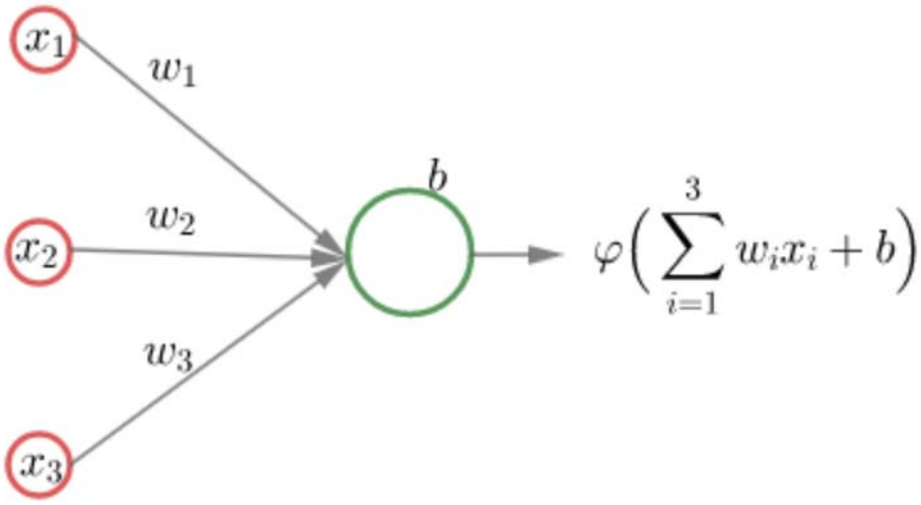
Virhefunktion osittaisderivaatta viimeisen piilokerroksen painojen suhteen.

2. Koodissa on kätevää ja nopeaa käyttää vastavirta-algoritmin kaavojen vektori- ja matriisiversioita. Lue näistä esimerkiksi linkkilistan lähteestä. Lähteessä pohditaan myös sitä, miksi vastavirta-algoritmi on paljon nopeampi tapa laskea tarvittavat osittaisderivaatat kuin osittaisderivaattojen erotusosamäärien raja-arvomääritelmään pohjautuva tapa.

3. Tarkastellaan neuroverkkoa, jonka syöte on $x = (x_1, x_2, x_3) \in \mathbb{R}^3$, jossa ei ole piilokerroksia, jonka ulostulokerroksessa on yksi neuroni ja jonka ulostulokerroksen aktivointifunktio on derivoituva funktio $\varphi\colon \mathbb{R} \to \mathbb{R}$. Käytetään syöte-tavoiteparin x, y ja verkon antaman tuloksen $t = \varphi(\sum_{i=1}^3 x_i w_i + b)$ vertailuun virhefunktiota

$$E = \frac{1}{2}\|t-y\|^2 = \frac{1}{2}(t-y)^2 = \frac{1}{2}\Big(\varphi(\sum_{i=1}^{3} x_i w_i + b) - y\Big)^2.$$

Laske virhefunktion osittaisderivaatat painojen w_1, w_2 ja w_3 suhteen.



Virhefunktion osittaisderivaatta perseptronille.

Lisätietoa vastavirta-algoritmista

Vastavirta-algoritmista löytyy paljon monentasoista luettavaa, esimerkkejä ja koodia. Osassa selitetään matemaattinen tausta ja painojen muutoksen vaikutus verkon toimintaan hyvin, osan selitys on turhan monimutkaista. Kaavoissa indeksien käyttö on monesti epämatemaattista (osittaisderivaatan indeksit ja summausindeksit samoja).

- M. Nielsen: Neural Networks and Deep Learning, Chapter 2
- B. Rohrer: How deep neural networks work
- M. Zabarauskas: Backpropagation Tutorial
- Andrej Karpathy: Hacker's guide to Neural Networks
- Convolutional Neural Networks for Visual Recognition (Stanfordin yliopisto)
- Calculus on Computational Graphs
- Online demo

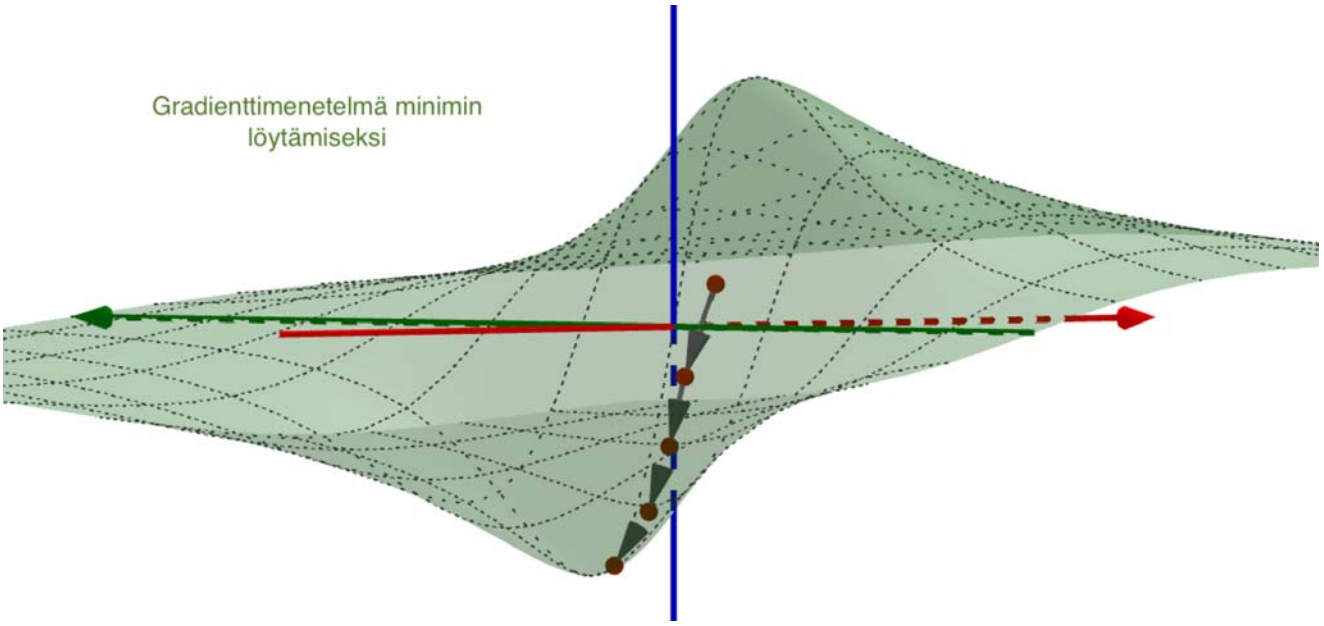
4.1.11 Gradienttimenetelmä

Tavoitteena on minimoida verkon parametreista riippuvaa virhefunktiota eli syötteiden ja verkon antamien tulosten välistä virhettä. Virhefunktio on monen muuttujan (kaikkien neuronien painojen ja vakiotermien) funktio, jolle etsitään pienintä arvoa.

Matemaattisen analyysin keinoin monen muuttujan funktion ääriarvoja etsitään riittävän siistille funktiolle gradientin nollakohdista ja niistä pisteistä, joissa funktiolla ei ole osittaisderivaattaa. Gradientin nollakohtien etsimisen sijaan virhefunktion minimoinnissa käytetään erilaisia algoritmeja kuten gradienttimenetelmää *(gradient descent)*. Siinä minimin etsiminen aloitetaan laskemalla tarkasteltavan funktion arvo aloitusparametreilla.

Funktion gradientti kertoo nopeimman kasvun ja siten gradientin vastavektori nopeimman vähenemisen suunnan. Sopivilla askelilla nopeimman vähenemisen suuntaan siirtymällä löydetään (menetelmään sopiville funktioille) lokaali minimi.

Minimin etsimistä gradienttimenetelmällä havainnollistetaan usein yhden tai kahden muuttujan funktiolla. Kahden muuttujan tilanteessa funktion kuvaajan voi ajatella kumpuilevaksi maastoksi, missä rinteellä seisova ihminen haluaa mennä laakson pohjalle jyrkkyydestä välittämättä. Gradienttimenetelmän keinolla alas mennään vähän matkaa jyrkintä rinnettä (gradientin vastavektorin suuntaan), pysähdytään ja valitaan taas jyrkin suunta. Näin jatketaan, kunnes päästään laakson pohjalle. Huomaa, että jos maastossa on useita laaksoja, niin liian pitkä siirtymä yhteen suuntaan voi johtaa väärän laakson pohjalle.



Gradienttimenetelmällä laakson pohjaa eli minimiä etsitään etenemällä askelittain jyrkimmän alamäen suuntaan.

Neuroverkon opettaminen vastavirta-algoritmilla ja gradienttimenetelmällä

Vastavirta-algoritmia ja gradienttimenetelmää käytettäessä suoritetaan seuraavat tehtävät:

1. Syötetään opetusesimerkkijoukon A kaikki opetusesimerkit x neuroverkolle.
2. Kaikille opetusesimerkeille $x \in A$:
   (i) Lasketaan vastavirta-algoria varten neuronikohtaiset summat z_j^l ja ulostulot a_j^l.
   (ii) Lasketaan syötettä vastaavan virhefunktion osittaisderivaatat vastavirta-algoritmin avullla (ensin ulostulokerroksen painojen ja vakiotermien suhteen, sitten kerros kerrallaan alaspäin).
3. Korjataan neuronien parametrit gradienttimenetelmän avulla. Matriisi- ja vektorimuodossa ilmoitettuna parametrien muutokset ovat

$$w^l \rightsquigarrow w^l - \frac{\alpha}{N} \sum_{x \in A} \delta_x^l (a_x^{l-1})^T \quad \text{ja} \quad b^l \rightsquigarrow b^l - \frac{\alpha}{N} \sum_{x \in A} \delta_x^l,$$

missä α on verkon oppimisnopeus ja N opetusesimerkkijoukon A alkioiden lukumäärä.

Huomaa, että jos opetusesimerkkijoukko koostuu yhdestä syötteestä, niin yksittäiten neuronien uudet painot vastavirta-algoritmin jälkeen ovat

$$w_{ij}^l \rightsquigarrow w_{ij}^l - \alpha \frac{\partial E}{\partial w_{ij}^l} \quad \text{ja} \quad b_j^l \rightsquigarrow b_j^l - \alpha \frac{\partial E}{\partial b_j^l},$$

Gradienttimenetelmän eri versioita

(Satsi)gradienttimenetelmä *((Batch/Vanilla) Gradient descent)*

Perinteisessä gradienttimenetelmässä yksittäistä opetusesimerkkiä x^i vastaava virhe

$$\mathcal{E}_i(x^i)$$

lasketaan jokaisen opetusesimerkin jälkeen ja minimoitavana virhefunktiona käytetään opetusesimerkkien virheiden summaa

$$\mathcal{E} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{E}_i,$$

missä N on opetusesimerkkijoukon alkioiden lukumäärä. Verkon parametrit päivitetään vasta, kun koko opetusesimerkkijoukko on käyty läpi.

Jos parametrit laitetaan jonoon ja niistä muodostetaan vektori w, niin parametrien päivityskaava on

$$w \rightsquigarrow w - \alpha \nabla \mathcal{E}(w),$$

missä α on verkon oppimisnopeus ja virhefunktion osittaisderivaatat parametrien suhteen ovat gradientissa samassa järjestyksessä kuin parametrit vektorissa w.

Gradienttimenetelmässä koko opetusesimerkkijoukon tiedot ovat kerralla muistissa ja verkko saattaa oppia hitaasti isoilla opetusesimerkkijoukoilla. Päivityksiä on vähän, joten menetelmä on virheen pienenemisen suhteen vakaa mutta se saattaa supeta liian aikaisin ja huonommilla parametreilla kuin stokastinen versio. Gradienttimenetelmällä löydetään globaali minimi konvekseille virhefunktioille (harvinainen tilanne) ja lokaali minimi ei-konvekseille virhefunktioille.

Stokastinen gradienttimenetelmä *(Stochastic gradient descent)*

Stokastisessa gradienttimenetelmässä virhe lasketaan ja neuronien parametrit päivitetään opetusesimerkkijoukon jokaisen syötteen jälkeen. Tällä menetelmällä saadaan nopea tieto verkon oppimisesta, sillä verkko oppii koko ajan. Menetelmä on helppo ymmärtää ja toteuttaa. Tiheä päivittäminen on kuitenkin hidasta, parametrien arvot saattavat heilua paljon päivittämisen aikana ja häiriöherkkyys voi hidastaa virhefunktion lokaalin minimin löytymistä. Joissain tilanteissa heiluminen on etu tavalliseen gradienttimenetelmään verrattuna - stokastinen versio saattaa päätyä pienempään lokaaliin minimiin.

Minisatsi gradienttimenetelmä *(Mini batch gradient descent)*

Minisatsigradienttimenetelmä on perinteisen ja stokastisen gradienttimenetelmän välimuoto. Siinä opetusesimerkkijoukko jaetaan osajoukkoihin, jotka syötetään verkolle, lasketaan virhefunktio ja päivitetään parametrit. Tämä vähentää parametrien heiluntaa päivityksissä ja mahdollistaa paremman ja vakaamman suppenemisen lokaaliin minimiin kuin toisilla versioilla. Menetelmässä voidaan käyttää ohjelmakirjastojen tehokkaita lineaarialgebran laskurutiineja.

Lisätietoa gradienttimenetelmästä

- Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent
- An Introduction to Gradient Descent and Linear Regression
- Ruder: An overview of gradient descent optimization algorithms
- A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size
- Raschka: Single-Layer Neural Networks and Gradient Descent
- 3Blue1Brown: Gradient descent, how neural networks learn | Deep learning, chapter 2

4.1.12 Virhefunktiot

Verkon oppimisen kannalta on tärkeää, että pieni muutos neuronin painossa aiheuttaa vain pienen muutoksen ulostulossa. Vastavirta-algoritmin vaiheita tutkiessa huomataan, että jos opetusesimerkkijoukon virhe saadaan keskiarvona yksittäisten opetusesimerkkien virheistä, niin opetusesimerkkijoukon virheen osittaisderivaatat saadaan laskettua opetusesimerkkien virheiden avulla.

Vastavirta-algoritmin yhteydessä käytettiin yksittäisen syötteen x tavoitteen $y \in \mathbb{R}^m$ ja verkon antaman tuloksen $t \in \mathbb{R}^m$ välisenä virheenä erotuksen euklidisen normin neliötä

$$E = \frac{1}{2}\|t - y\|^2 = \frac{1}{2}\sum_{k=1}^{m}(t_k - y_k)^2$$

ja N:n alkion opetusesimerkkijoukon A virhefunktiona keskineliösummaa

$$E_A = \frac{1}{2N}\sum_{x \in A}\|t - y\|^2.$$

Logistisen regression virhefunktio

Jos ulostulokerroksen arvot kuuluvat välille $[0, 1]$, niin voidaan käyttää myös logistisen regression virhefunktiota (ristientropian virhefunktio), *(cross-entropy cost function)*,

$$E = -\frac{1}{N}\sum_{x}\sum_{k=1}^{m}\Big(y_k \log t_k + (1 - y_k)\log(1 - t_k)\Big),$$

missä vektorit $y = (y_1, \dots, y_m)$ ovat syötteiden x tavoitteita, vektorit $t = (t_1, \dots, t_m)$ neuroverkon syötteille x antamia tuloksia ja N on opetusesimerkkijoukon koko.

Laskemalla nähdään, että sigmoid-aktivointifunktiota käytettäessä tämän virhefunktion osittaisderivaatat neuroneiden painojen ja vakiotermien suhteen eivät riipu aktivointifunktion derivaatoista vaan pelkästään tavoitteiden ja tulosten erotuksista,

$$\frac{\partial E}{\partial w_{ij}^L} = \frac{1}{N} \sum_x a_i^{L-1}(a_j^L - y_j)$$

ja

$$\frac{\partial E}{\partial b_i^L} = \frac{1}{N} \sum_x (a_j^L - y_j).$$

(Muista, että ulostulokerroksen tulos $a^L = (a_1^L, \dots, a_m^l)$ on syötettä x vastaava tulos $t = (t_1, \dots, t_m)$.) Siksi sigmoid-funktion derivaatan pienuus suurilla ja pienillä arvoilla ei hidasta verkon oppimista niissä tapauksissa, joissa tavoitteet erovat paljon syötteistä.

Joissain lähteissä syöte-tavoite-parin välisistä virhefunktioista käytetään nimeä tappiofunktio *(loss function)* ja opetusesimerkkijoukon virhefunktiosta virhe-/maksufunktio *(cost function)*.

Lisätietoa virhefunktioista

- M. Nielsen: Neural Networks and Deep learning, Chapter 3
- A. Ng: Machine Learning, Cost function (Coursera)

4.1.13 Yli- ja alisovittaminen

Neuroverkon ja yleisemmin koneoppimisen opettaminen suoritetaan opetusesimerkkien avulla, oppiminen varmistetaan ja oppimisnopeus- ja muita hyperparametreja säädetään vahvistusesimerkkijoukon avulla ja lopuksi toiminta testataan testiesimerkkijoukolla. Tarkoitus on, että verkko osaa yleistää oppimansa ja toimii lopulta riittävän tarkasti tuntemattomalle datalle. Joskus käy niin, että verkko tuntuu oppivan hyvin mutta sitten tulee ongelmia:

- Opettamisen edetessä virhefunktion pieneneminen hidastuu tai tarkkuus huononee.
- Verkko toimii hyvin opetusesimerkeille mutta ei (opetusesimerkkien kaltaisille) vahvistus- tai testiesimerkeille.

Tätä ilmiötä sanotaan ylisovittamiseksi *(overfitting)*. Siinä verkko on oppinut opetusesimerkkijoukon liian hyvin ja säätänyt parametrinsa sen erityisominaisuuksien ja häiriöiden mukaan. Ylisovittaminen on yleinen ongelma suurissa tuhansien parametrien neuroverkoissa joissa opetusesimerkkijoukko ei ole ole riittävän suuri suhteessa verkon kokoon.

Ylisovittamista voidaan estää seuraavilla tavoilla:

- opetusesimerkkijoukon kasvattaminen
- (verkon koon pienentäminen)
- opettamisen lopettaminen riittävän aikaisin *(early stopping)*
- neuroneiden osittainen poistaminen verkosta *(dropout layer)*
- painojen pienentämien L_2- ja L_1- säännöstelyllä *(regularization)*

Opetusesimerkkijoukon kasvattaminen

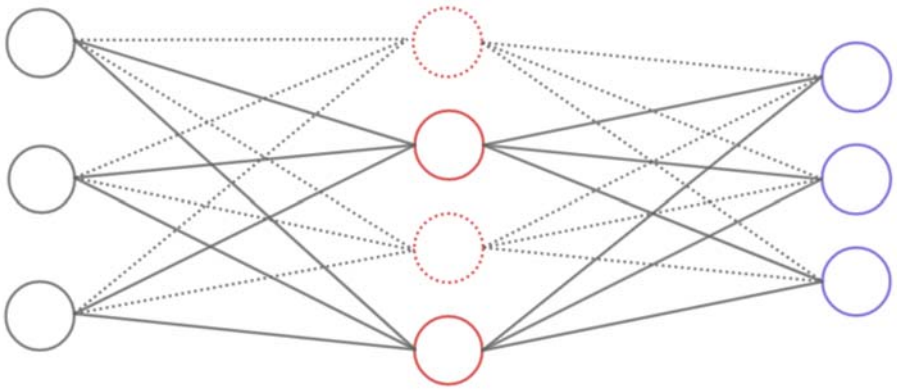
Opetusesimerkkijoukon koon kasvattaminen saattaa olla vaikeaa mutta joissain tilanteissa sitä voi kasvattaa olemassaolevan datan avulla. Esimerkiksi uusia tunnistettavia kuvia saadaan helposti kiertojen, siirtojen ja skaalauksen avulla.

Aikainen lopettaminen

Verkon toimintaa testattaessa vahvistusesimerkkijoukolla opetusesimerkkijoukon jälkeen tarkastetaan tulosten tarkkuus jokaisen osajoukon jälkeen. Kun tarkkuus pienenee, lopetetaan.

Osittainen poistaminen

Yksi tapa pienentää ylisovittamista on neuroneiden hetkellinen poistaminen verkosta. Tässä tekniikassa osa piilokerrosten neuroneista poistetaan väliaikaisesti. Vajaaseen verkkoon syötetään opetusesimerkkejä, käytetään vastavirta-algoritmia ja päivitetään verkon parametrit. Tämän jälkeen poistetut neuronit palautetaan, poistetaan uusi neuronijoukko ja jatketaan opettamista. Menetelmässä jälkeen verkko on tavallaan keskiarvo monesta samaa tehtävää tekevästä verkosta. Koska neuroneiden lähellä olevat neuronit eivät välttämättä ole mukana jokaisella opetuskierroksella niin neuroneista tulee itsenäisempiä ja verkosta robustimpi.



Neuronit poistetaan hetkellisesti verkosta, päivitetään parametrit ja palautetaan neuronit.

Säännöstely

Virhefunktion muuttaminen niin, että minimi löytyy pienillä painoilla perustuu siihen, että monesti verkot toimivat itseisarvoiltaan pienillä painoilla paremmin kuin suurilla. Säännöstelyssä virhefunktiota muutetaan niin, että minimi löytyy pienemmillä painoilla.

Jos verkossa on käytössä virhefunktio $\mathcal{E}$, niin L_2-säännöstelyn virhefunktio on

$$\mathcal{F} = \mathcal{E} + \frac{\lambda}{2N} \sum_{i,j} w_{ij}^2,$$

missä N on opetusesimerkkijoukon koko, w_{ij} ovat neuroneiden painot ja $\lambda > 0$ on säännöstelyparametri. Neuroneiden vakiotermejä ei oteta mukaan säännöstelyosaan.

Virhefunktion $\mathcal{F}$ jälkimmäinen osa on pieni kun painot ovat itseisarvoltaan pieniä. Minimoinnissa suuret painot ovat hyviä vain jos niillä saadaan alkuperäinen virhefunktio $\mathcal{E}$ hyvin pieneksi.

Virhefunktion $\mathcal{F}$ osittaisderivaatat painojen w_{ij} suhteen ovat

$$\frac{\partial \mathcal{F}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial w_{ij}} + \frac{\lambda}{N} w_{ij}$$

ja vakiotermien suhteen samat kuin alkuperäisellä virhefunktiolla $\mathcal{E}$. Siten gradienttimenetelmän antamat uudet painot saadaan kaavalla

$$w \rightsquigarrow \Big(1 - \frac{\alpha\lambda}{N}\Big) w - \alpha \frac{\partial \mathcal{E}}{\partial w},$$

missä α on verkon oppimisnopeus.

Säännöstelyssä yritetään siis samanaikaisesti käyttää mahdollisimman pieniä painoja ja saada virhefunktio pieneksi.

L_1-säännöstelyssä käytetään painojen neliöiden sijaan itseisarvoja. Virhefunktio on

$$\mathcal{E}_1 = \mathcal{E} + \frac{\lambda}{N} \sum_{i,j} |w_{ij}|,$$

missä $\mathcal{E}$ on alkuperäinen virhefunktion, N on opetusesimerkkijoukon koko, w_{ij} ovat neuroneiden painot ja $\lambda > 0$ on säännöstelyparametri.

Virhefunktion $\mathcal{E}_1$ osittaisderivaatat painojen w_{ij} suhteen ovat

$$\frac{\partial \mathcal{E}_1}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial w_{ij}} + \operatorname{sign}(w_{ij}),$$

missä $\operatorname{sign}(w_{ij}) = 1$, kun $w_{ij} > 0$, $\operatorname{sign}(w_{ij}) = -1$, kun $w_{ij} < 0$ ja nolla kun paino on nolla. Vakiotermien suhteen osittaisderivaatat ovat samat kuin alkuperäisellä virhefunktiolla $\mathcal{E}$.

Gradienttimenetelmän antamat uudet painot saadaan kaavalla

$$w \rightsquigarrow w - \frac{\alpha\lambda}{N} \operatorname{sign}(w) - \alpha \frac{\partial \mathcal{E}}{\partial w},$$

missä α on verkon oppimisnopeus.

L_1-säännöstelyssä painot pienenevät askelilla, joiden pituus ei riipu painon koosta. L_2-säännöstelyssä askeleen koko on paino kerrottuna vakiolla.

Alisovittamisessa *(underfitting)* verkon parametrit päivittyvät hyvin hitaasti ja verkko oppii huonosti.

Lisätietoa yli- ja alisovittamisesta

- A. Ng: Machine Learning, The Problem of Overfitting (Coursera)
- A. Ng: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization, Regularization (Coursera)
- A. Ng: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization, Why regularization reduces overfitting? (Coursera)

- A. Ng: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization, Dropout Regularization (Coursera)
- A. Ng: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization, Understanding Dropout (Coursera)
- A. Ng: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization, Other regularization methods(Coursera)
- Chatbot's Life: Regularization in deep learning
- J. D. McCaffrey: Implementing Neural Network L2 Regularization
- J. D. McCaffrey: Neural Network L2 Regularization using Python, Visual Studio Magazine 09/2017
- M. Nielsen: Neural Networks and Deep learning, Chapter 3

4.1.14 Muita virhefunktion minimointikeinoja

Vastavirta-algoritmi ja muut osittaisderivaattoihin (gradientteihin) perustuvat menetelmät ovat monesti hitaita. Vastavirta-algoritmin tapauksessa hitaus johtuu paljosta laskemisesta: neuronien parametrien muutoksessa tarvittavia osittaisderivaattoja lasketaan koko ajan iteratiivisesti. Vastavirta-algoritmin erilaisilla muunnoksilla ja muilla verkon opetusmenetelmillä haetaan lisää nopeutta. Osassa menetelmiä käytettään ensimmäisen kertaluvun osittaisderivaattojen lisäksi toisen kertaluvun derivaattoja, joiden avulla saadaan tietoa ensimmäisen kertaluvun osittaisderivaattojen kasvusta.

Lisätietoa verkon opettamisesta

- 5 algorithms to train a neural network
- H. Daumé: A Course in Machine Learning, Chapter 2 (University on Maryland)
- M. Nielsen: Neural Networks and Deep learning, Chapter 3

5 Prosessienlouhinta *(Toni Ruohonen)*

(Toni Ruohonen)

Tässä luvussa tutustutaan prosessinlouhintaan ja simulointimalleihin sosiaali- ja terveydenhuollon kehittämisen menetelminä.

Prosessienlouhinta on menetelmä, joka mahdollistaa prosessien automaattisen kuvaamisen visuaalisessa ja tilastollisessa muodossa. Päätavoitteena prosessienlouhinnalla on identifioida, monitoroida sekä kehittää kirjattuun tietoon perustuvia faktapohjaisia prosesseja (ei oletettuja prosesseja) louhimalla tapahtumalokeja tai tietorekisterejä. Prosessienlouhintaa on kolmea eri tyyppiä. Nämä eri louhintatyypit ovat:

1. Automaattisen prosessien tunnistaminen *(Automated Process Discovery)*
2. Mallien vertailu *(Conformance Checking)*
3. Mallien laajentaminen *(Enhancement)*.

Automaattisella prosessien tunnistamisella tarkoitetaan mallien louhimista lokeista tai rekistereistä ilman tarkempia ennakkomääritteitä. Mallien vertailussa olemassa olevaa prosessia verrataan lokeista louhittuun prosessiin ja katsotaan noudattaako lokeista louhittu