Eelmine kord

- Neurovõrk on lihtsastest elementidest (*neuronitest*) ehitatud funktsioon.
- Neurovõrk on *parametriseritud* funktsioon, Funktsiooni parameetrid (*kaalud*) otsitakse võrgu *treenimisega* etteantud andmete peal.
- Treenitud võrk realiseerib andmetes väljendatud seaduspärasust.
- Neurovõrke on palju erinevaid tüüpe.
- Neurovõrgud on rohkem kui poolsajandit vanad, kuid siialaani on see aktiivselt arendatud valdkond.

Seekord

- Vektorid, maatriksid ja tuletised
- Kiirema languse meetod
- Tehisneuron lineaarse regressiooni jaoks
- Adaptiivne lineaarne neuron (*Adaline*), LMS algoritm
- Tehisneuron kui lineaarne klassifikseerija
- Rosenblatt’i pertseptron

Vektorid ja maatriksid

\[
\mathbf{x} = \begin{pmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_n
\end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix}
    a_{11} & \cdots & a_{1n} \\
    \vdots & \ddots & \vdots \\
    a_{m1} & \cdots & a_{mn}
\end{pmatrix}
\]

- Transponeerimine: \( \mathbf{A}^T, \mathbf{x}^T \)
- Korrutamine: \( \mathbf{Ax}, \mathbf{w}^T \mathbf{x} \)
- Arvuga korrutamine: \( \alpha \mathbf{x} \)
- Lüttmine: \( \mathbf{A} + \mathbf{B}, \mathbf{x} + \mathbf{y} \)
- Vektori norm (pikkus): \( \| \mathbf{x} \| \)
- Rauatmaatriksi pööramine: \( \mathbf{A}^{-1} \)
Vektorid ja maatriksid

- $m$-elemendiline vektor on ruumi $\mathbb{R}^m$ element
- $n \times m$ maatriks on lineaariteised $\mathbb{R}^m \to \mathbb{R}^n$
- Teisendus $f$ on lineaarne kui $f(\alpha x + y) = \alpha f(x) + f(y)$
- $Ax$ on vektor $x$ teisendatud $A$ poolt.
- $w$ on vektor, $w^T$ on teisendus
- Kui $\|w\| = 1$ siis $w^T x$ on vektori $x$ projektsioon vektorile $w$.

Diferentseeruvus

- Tuletise omadused:
  - $f'(x)_{ij} = \frac{\partial f_i(x)}{\partial x_j}$
  - $(\alpha f(x) + g(x))' = \alpha f'(x) + g'(x)$
  - $(f(g(x)))' = f'(g(x))g'(x)$
- Mõned tuletised:
  - $(Ax)' = A$
  - Sümmeetrilise $A$ korral $(x^T Ax)' = 2x^T A$

Diferentseeruvus

Olgu $f : \mathbb{R}^m \to \mathbb{R}^n$. Funktsiooni $f$ nimetatakse
diferentseeruvaks punktis $x_0$, kui leidub lineaariteisendus
$A(x_0)$ selline, et
$$\Delta f(x_0) = f(x_0 + \Delta x) - f(x_0) = A(x_0)\Delta x + o(\Delta x)$$
Maatriksit $A(x_0)$ nimetatakse funktsiooni $f$ tuletiseks
(punktis $x_0$) ning tähistatakse $\frac{\partial f(x_0)}{\partial x}$ vool $f'(x_0)$.
Kui $f$ on diferentseeruv igas punktis, ütleme lubalt, et $f$ on
diferentseeruv. Diferentseeruv funktsioon on pidev.
Võrdle juhuga $f : \mathbb{R} \to \mathbb{R}$:
$$\Delta f(x_0) = f'(x_0)\Delta x + o(\Delta x)$$

Gradient

Olgu niiüd $f : \mathbb{R}^m \to \mathbb{R}$ diferentseeruv funktsioon.
Funktsiooni $f$ tuletiseks punktis $x_0$ on siis $1 \times m$ maatriks
$A$, kusjuures
$$A = \begin{pmatrix} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{pmatrix}$$
Vektorit $A^T$ nimetatakse funktsiooni $f$ graendiiks punktis
$x_0$, ning tähistatakse $\nabla_x f(x_0)$ või lihtsalt $\nabla f(x_0)$. 

**Gradient**

Gradient on vektor, mis näitab funktsiooni tõusu poole

![Gradient Diagram](image)

**Kiirema languse meetod**

1. Funktsiooni (lokaalset) miinimumis on gradient null.
2. Kui mingis punktis $x_0$ funktsiooni $f$ gradient pole null, siis tehes väikse sammus $\eta$ gradiendi vastassuunas ("allapoole") saabume punkti

$$x_1 = x_0 - \eta \nabla f(x_0)$$

kus funktsiooni väärtsus on väiksem kui algse punktis $x_0$.

---

**Kiirema languse meetod**

1. Kui ka selles punktis pole gradient null, teeme järgmise sammu allapoole:

$$x_2 = x_1 - \eta \nabla f(x_1)$$

ja kordame seda seni kuni leiame punkti $x_k$ kus gradient $\nabla f(x_k)$ on nulllähedane.

2. Sobivate $\eta$ ja $x_0$ korral leiab selline algoritm suvalise diferentseeruvu funktsiooni $f$ mõnda lokaalse miinimumi (kui selline üldse leidub). Seda algoritmi nimetatakse „kiirema languse meetodiks“ (gradient descent, steepest descent).
**Kiirema languse meetod**

Edastides elu lihtsustamiseks kaotame *bias* parameetrit, modelleerides teda veel ühe kaaluga, mille vastav sisend on alati 1:

\[ x_0 = 1 \]

\[ y = \phi(w^T x + b) \]

**Tehisneuron**

Aktivatsioonifunksioon \( \phi \)

Tehisneuronit saab kasutada:
- Lineaarne regressiooni jaoks, siis \( \phi(x) = x \)
- Klassifikatsioonist jaoks, siis \( \phi(x) = \text{sign}(x) \), s.t.

\[
\phi(x) = \begin{cases} 
+1 & \text{kui } x \geq 0 \\
-1 & \text{kui } x < 0
\end{cases}
\]
Aktivatsioonifunktsoon $\phi$

- Mõnele meeldib klassifikseerimise puhul "lävefunktsoon":
  \[
  \phi(x) = \begin{cases} 
  1 & \text{kui } x \geq 0 \\
  0 & \text{kui } x < 0
  \end{cases}
  \]

- Lävefunktsoon ja sign ei ole pidevad. Sobiv pidev asendus on logistiline sigmoid $\frac{1}{1+e^{-ax}}$ või hüperboolne tangens $\tanh(ax) = \frac{1-e^{-2ax}}{1+e^{-2ax}}$.

Lineaarne neuron

- Lineaarsete neuroni väljund on siis $y = f(x) = \mathbf{w}^T \mathbf{x}$
  (bias-i enam ei kasuta)
- Meil on olemas andmed: $\{(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)\}$
  (kus x-i eksemple komponent on alati 1).
- Andmed esitavad mingit meie teadmatu juhusliku funksiooni: $d_i$ on selle funksiooni väärtus punktis $x_i$.
- Me tahame leida "kõige sobivama" lineaarse funksiooni, mis igas punktis $x_i$ annaks midagi lähest $d_i$-le. (Muide $d$ siin tähendab "desired output").

Sigmoid

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

Lineaarne regressioon

- Oletame siis et meie andmete puhul
  \[
  d = \mathbf{w}^T \mathbf{x} + \epsilon
  \]
  kus $\epsilon$ on juhuslik $\mathbf{x}$-ist sõltumatu mür.
- Meie ülesandeks on leida $\mathbf{w}$ nii et mür oleks minimaalne.
- Saab näidata (suurima tõepärast printsibiga) et selleks peame me minimiseerima vigade ruutude summat:
  \[
  \mathcal{E}(\mathbf{w}) = \sum_{i=1}^{n} (d_i - \mathbf{w}^T \mathbf{x}_i)^2
  \]
**Lineaarne regressioon**

**Vigade ruutide summa minimiseerimine**

Tähistame

\[
X = \begin{pmatrix} x_1^T \\ \vdots \\ x_n^T \end{pmatrix}, \quad d = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}
\]

siis

\[
\mathcal{E}(w) = (d - Xw)^T (d - Xw)
\]

ning

\[
\nabla_w \mathcal{E}(w) = 2X^T Xw - 2X^T d
\]

**Lineaarse neuroni treenimise algoritm**

- Saadud tulemus on siis lühidalt kirjapandav järgmiselt:
  - Vektor \( w \) mille puhul
    \[
    Xw \approx d
    \]
  - avaldub kujul
    \[
    w = X^+ d
    \]
- See valem ongi lineaarse neuroni treenimise algoritm.
**Iteratiivne versioon**

- Eelmine valem nõuab suhteliselt mahukaid arvutusi, eriti kui andmeid on palju ja andmete dimensionaalsus on suur. Lihtsama iteratiivse meetodi saame, kui minimiseerime $E(w)$ kiirema languse meetodi abil,

- Alustame siis suvaliselt valitud $w_0$ väärtusest, ja igal sammul $k > 0$ lidadme talle juurde suurust

$$\Delta w_k = -\eta \nabla_w E(w_{k-1})$$

- Valemit nagu see eelmne nimetatakse *kaalude kohendamisreegliks* (weight update rule) või ka *õppimisreegliks*. Selliseid tuleb ka edaspidi ette.

---

**Iteratiivne versioon**

Arvutame $\nabla_w E$ nüüd mitte-matrikskujul:

$$\nabla \left( \sum_{i=1}^{n} (d_i - w^T x_i)^2 \right) = \sum_{i=1}^{n} \nabla (d_i - w^T x_i)^2$$

$$= \sum_{i=1}^{n} 2(d_i - w^T x_i) \nabla (d_i - w^T x_i) = - \sum_{i=1}^{n} 2e_i x_i$$

kus $e_i$ on neuroninäi $i$ treeningnäite $i$ jaoks, Konstandi 2 saame välja visata, ning saame sellist õppimisreeglil:

$$\Delta w_k = \eta \sum_{i=1}^{n} e_i x_i$$

---

**Batch vs online**

Veelkord siis:

- **Batch update**, vastab normaalsele kiirema languse meetodile:

$$\Delta w_k = \eta \sum_{i=1}^{n} e_i x_i$$

- **Online update**, vastab stohastiliselle kiirema languse meetodile, mis on pärimetoodi lähend:

$$\Delta w_k = \eta e_i x_i \quad \text{kus } i \text{ valitud juhuslikult}$$
Adaptiivne lineaarne neuron

- Linearse neuroni koos eelnevas tuletatud online-öppimisreeglaga nimetatakse Adaline (ADAptive LINEar Element).
- Vastava reegli nimetatakse LMS (Least-Mean-Squares) öppimisreegliks (ka Widrow & Hoff-i reegliks).
- Adaline põhiamused:
  - Lihtne
  - Ei vaja öppimiseks tervet andmete komplekti korraga ning sobib hästi pidevaks öppimiseks" kui andmed tulevad keskkonnast järgmis. Kui keskkond muutub ajaga, siis „adapteerub” selleks ka adaline.

Lineaarne klassifitseerija

- Teine asi mida tõeb tehisneuron hästi on klassifitseerimine. Selleks aga tuleb valida ta aktivatsioonifunktsiooniks $\phi$ midagi mis oleks $-1$ ja $1$ (või $0$ ja $1$) vahel, näiteks funktsioon $\text{sign}()$.
- Paneme tähele et vaatem
  \[ w^T x + b = 0 \]
  määrab hüperkantid ruumis $\mathbb{R}^m$. Punktid $x$, mille puhul $w^T x + b > 0$ asuvad ühel pool sellest tasandist, ning need, mille puhul $w^T x + b < 0$ teisel pool.

Lineaarne klassifitseerija
Lineaarne klassifitseerija

- Adaline jaoks tuletasime me õppimisreegli:
  \[ \Delta w_k = \eta e_i x_i \]
  (ära unusta et \( w \) on tegelikult kogu aeg \( w \), s.t. \( w_0 = b \)).
- Proovime kasutada sama reegli ka klassifitseerija jaoks.
  Sel juhul on aga
  \[ e_i = d_i - \text{sign}(w^T x_i) \]
  ning \( e_i \) võimalikud väärtused on 2, 0 või –2, kusjuures kui \( e_i \neq 0 \), siis on ta sama märgiga kui \( d_i \).

Rosenblatt-i pertseptron

- Kui \( e_i = 0 \), siis klassifitseerib neuron näite \( x_i \) oieti ja kaalude vektorit ei muudeta, vastasel juhul saab õppimisreegli panna kirja kui:
  \[ \Delta w_k = \eta d_i x_i \]
- Kokkuvõttes saame järgmise algoritmi:
  1. Leia näide \( i \), mis on neuroni poolt valesti klassifitseeritud
  2. Kohenda kaalud vastavalt
- See on Rosenblatt-i pertseptroni õppimisreegel

Pertseptron: märkused

- Pertseptoni algoritmi ei ole kairema languse meetod
- Pertseptoni algoritmi ei minimiseeri mingid vigade ruutude summasid vaid leiab suvast eralduvat hüperpindset.
- Pertseptoni algoritmi ei oma põhjendatud statistilist alust.
- Kuna pertseptron nõuab lineaarselt eralduvaid klasse ei suuda ta õppida mõisteid nagu ring, kumer kuju, jms.
- Muidu on pertseptoni algoritmi üks lihtsamatest ning töötab õllatavalt hästi.
Kokkuvõte

a. Lineaärne neuron:
\[ w = X^+d \]

b. Widrow & Hoffi Adaline:
\[ \Delta w_k = \eta e_i x_i \]

c. Rosenblatt'i Perceptron:
\[ \Delta w_k = \eta d_i x_i \quad \text{kui } x_i \text{ on valesti klassifitseeritud} \]