

# 732G12 Data Mining

## Föreläsning 7

---

Josef Wilzén

IDA, Linköping University, Sweden

- Neurala nätverk
- Feature learning
- Optimering av neurala nätverk
- Hyperparametrar

Finns några gamla tentor på Lisam. Den tenta ni får kommer vara liknande men inget som har med associationsanalys/sekventiell data är med i kursen nu.

Inte bara kod som ska lämnas in utan även lösningar. Använd Rmarkdown för att skriva fina lösningar med plottar och kod. All kod ska bifogas i inlämningen.

På Lisam finns mapp med alla hjälpfiler för tentan. Dessa kommer finnas på datorerna när tentan börjar. Kolla igenom dessa i god tid innan tentan för att få en uppfattning om vad som finns och var!

När vi pratar om neurala nätverk kan vi prata om lite vad som helst.  
Finns **väldigt många** olika sorters nätverk.

Kan använda neurala nätverk för att lösa många olika problem:

- Övervakad inlärning
- Oövervakad inlärning
- Reinforcement learning
- Generativa modeller
- Representation learning

För övervakad inlärning kan vi t.ex. använda

- Feed-forward network / Multiple layer perceptron (MLP)
- Radial basis network
- Convolutional neural networks (CNN)
- Recurrent neural networks

För oövervakad inlärning:

- Dolda representationer: Autoencoders
- Clustering: Self Organizing Map

Generativa modeller:

- Används för att lära sig komplexa fördelningar för att sen dra nya samples.
  - Sampla nya bilder
  - Skriva text
- Generative adversarial network (GAN)

Vi går tillbaka till linjär regression,

$$y = \mathbf{X}\beta + \varepsilon, \quad \mathbb{E}[\varepsilon] = 0, \quad \mathbb{V}[\varepsilon] = \sigma^2.$$

Vad kan vi göra om data inte följer denna linjära modell?

Vanligt i linjär regression,

- Givet data  $\mathbf{X} = (x_1, x_2, \dots, x_p)$  och  $y = \mathbf{X}\beta$ .
- Vi kan transformera variablerna i  $\mathbf{X}$ :
  - Polynomregression,  $\mathbf{X} = (x, x^2, x^3, \dots, x^p)$
  - Funktioner,  $\log(x), \sqrt{x}, \exp(x), \dots$
  - Interaktioner,  $x_1 x_2$
  - Stegfunktioner
  - Diskretisering
  - Dummy-kodning
- Kallas i maskininlärning för "feature engineering"
  - Svårt att veta vilken transformation som vi ska göra för varje problem.
  - Svårt med komplexa datastrukturer som text eller bild.



# Feature learning

- Vi har data  $\mathbb{X} = (x_1, x_2, \dots, x_p)$ .
- Transformationer är funktioner av data.
  - Ex.  $h(x) = \log(x)$ ,  $h(x_1, x_2) = \exp(x_1 \cdot x_2)$ .
- Anta en  $x$ -variabel, låt  $h(x)$  vara en viktad summa av andra funktioner,

$$z = h(x) = \sum_{i=1}^M w_i h_i(x),$$

där  $h_i(x)$  är godtyckliga funktioner.

# Feature learning

- Vi har data  $\mathbb{X} = (x_1, x_2, \dots, x_p)$ .
- Transformationer är funktioner av data.
  - Ex.  $h(x) = \log(x)$ ,  $h(x_1, x_2) = \exp(x_1 \cdot x_2)$ .
- Anta en  $x$ -variabel, låt  $h(x)$  vara en viktad summa av andra funktioner,

$$z = h(x) = \sum_{i=1}^M w_i h_i(x),$$

där  $h_i(x)$  är godtyckliga funktioner.

- Om vi har många  $x$ -variabler får vi,

$$z = h(x_1, x_2, \dots, x_p) = \sum_{i=1}^M w_i h_i(x_1, x_2, \dots, x_p).$$

- Hur ska vi välja  $h_i(x)$ ?

För en linjär transformation hitta matriserna  $\mathbf{W}$  och  $\mathbf{V}$ ,

$$\underset{n \times m}{\mathbf{Z}} = \underset{n \times p}{\mathbf{X}} \cdots \underset{p \times m}{\mathbf{W}}, \quad \underset{n \times g}{\mathbf{Z}} = \underset{n \times p}{\mathbf{X}} \cdots \underset{p \times m}{\mathbf{W}} \cdot \underset{m \times g}{\mathbf{V}}.$$

För neurala nätverk vill vi kunna modellera icke-linjära funktioner.

Idé: Använd många "enkla" icke-linjära funktioner för att skapa en komplex icke-linjär funktion!

## Neurala nätverk

Låt  $\sigma(\cdot)$  vara en enkel icke-linjär funktion och låt  $h_i(x_1, \dots, x_p)$  vara en linjär funktion,

$$h_i(x_1, x_2, \dots, x_p) = \beta_{0i} + \beta_i^\top \mathbf{x}.$$

Låt nu

$$z = \sigma(h_i(x_1, x_2, \dots, x_p)) = \sigma(\beta_{0i} + \beta_i^\top \mathbf{x}),$$

nästla sedan många sådana funktioner för att bygga upp en godtyckligt komplex icke-linjär funktion.

# Feature learning

För MLP brukar vi skriva

$$\mathbf{a}_{k \times 1}^{(p+1)} = \sigma \left( \mathbf{W}_{k \times n}^{(p)} \cdot \mathbf{a}_{n \times 1}^{(p)} + \mathbf{b}_{k \times 1}^{(p)} \right).$$

Här är:

$k$  Dimension av nya lagret

$n$  Dimension av föregående lager

$\mathbf{W}$  Viktmatris

$b$  Bias

$(p)$  Vilket lager

$\sigma()$  Vår funktion som opererar elementvis

Historiskt har sigmoid eller hyperbolic tangent varit vanliga aktiveringsfunktioner. Numera är ReLu (eller varianter) den vanligaste,

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

Vi kan se ett neuralt nätverk som att vi

1. Automatiskt lär oss transformationer av de förklarande variablerna.
2. Gör linjär (logistisk, multinomiell) regression på transformationerna (sista lagret).

OBS!

- Komplexa funktioner kräver mycket data att lära sig!
- Neurala nätverk kan lätt överanpassa träningsdata!
- Funkar när vi har stort antal förklarande variabler.
- Om vi låter gömda lager ha mindre dimension än förklarande variabler får vi "icke-linjär variabelreduktion".

# Universal approximation theorem

Vilka funktioner kan vi då lära oss med ett neuralt nätverk av detta slag?

**Universal approximation theorem:** Let  $C(X, \mathbb{R}^m)$  denote the set of continuous functions from  $X \subset \mathbb{R}^n$  to  $\mathbb{R}^m$ . Let  $\sigma \in C(\mathbb{R}, \mathbb{R})$ . Note that  $(\sigma \otimes x)_i = \sigma(x_i)$ , so  $\sigma \otimes x$  denotes  $\sigma$  applied to each component of  $x$ . Then  $\sigma$  is not polynomial if and only if for every  $n \in \mathbb{N}$ ,  $m \in \mathbb{N}$ , compact  $K \subseteq \mathbb{R}^n$ ,  $f \in C(K, \mathbb{R}^m)$ ,  $\varepsilon > 0$  there exists  $k \in \mathbb{N}$ ,  $A \in \mathbb{R}^{k \times n}$ ,  $b \in \mathbb{R}^k$ ,  $C \in \mathbb{R}^{m \times k}$  such that

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon,$$

where  $g(x) = C \cdot (\sigma \otimes (A \cdot x + b))$ .

# Optimering av neurala nätverk

Gradient decent: Hitta minimum på en funktion genom att gå dit den lutar mest!

Vill hitta

$$a^* = \arg \min_a L(a) = \sum_i L_i(f(x^{(i)}, a), y^{(i)}).$$

Löser detta genom sekvensen

$$a_{n+1} = a_n - \gamma \cdot \nabla L(a_n).$$

- Vi behöver gradienter (partiella derivator)
- Backpropagation: kedjeregeln för derivator på neurala nätverk.
- Gradient decent: dyrt när vi har många observationer!



Svårt problem med många fallgropar.

- Lokala minima
  - Ställen som ser ut som ett minima (eller grannar har högre kostand) men inte är det bästa som finns.
  - Kan ha hög kostand eller låg.
  - Identifikationsproblem:
    - Viktsymmetri
    - Skalning mellan lager
  - Kan ha oräkneligt antal lokala minima.

## Platåer och sadelpunkter

- Ställen där gradienten är noll (eller nära), fast vi inte är på ett lokalt min/max.
- Sadelpunkter:
  - Lokalt minima i några riktningar.
  - Lokalt maxima i andra riktningar.
- Antalet sadelpunkter tenderar att öka med antalet dimensioner.
- Platåer är stora områden som är platta (gradient nära noll).
- Platåer och sadelpunkter gör optimeringen med gradient decent svårare.

# Stochastic gradient decent (SGD)

Det är dyrt att beräkna  $\nabla L(a_n)$  för alla datapunkter.

Gör istället en väntesvärdesriktig skattning  $\nabla \hat{L}(a_n)$  av gradienten genom att ta ett slumpmässigt sample från data (mini-batch).

- Större batch ger mindre varians i skattningen men blir dyrare att beräkna.
- Kräver fler iterationer och mindre learning rate.
- Kräver att vi har oberoende observationer.
- Funkar bra för neurala nätverk!
- En epoch är en genomgång av all träningsdata.

I neurala nätverk finns massvis med Hyperparametrar!

- Arkitektur:
  - Antal gömda lager
  - Antal neuroner i varje lager
  - Aktiveringsfunktioner
  - (Speciella typer av neuroner/lager)
- Optimeringen:
  - Mini-batch storlek
  - Learning rate (fix eller föränderlig)
  - Antal epoker
  - (Vilken optimeringsalgorithm som används)

Hur ska vi bestämma deras värden?

- Svår fråga utan exakt svar.
- Mycket trial and error.
- Valideringsdata
- För stora problem kan det ta lång tid att hitta bra hyperparametrar