732G57 Maskininlärning för statistiker

Föreläsning 7

Josef Wilzén
IDA, Linköping University, Sweden

Dagens föreläsning

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

Neuralt nätverk

När vi pratar om neurala nätverk kan vi prata om lite vad som helst.
Finns **väldigt många** olika sorters nätverk. Se här för en sammaställning.

Kan använda neurala nätverk för att lösa många olika problem:

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

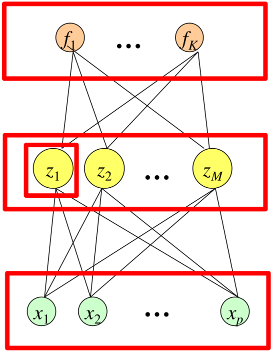
Neurala nätverk

För övervakad inlärning kan vi t.ex. använda

- Feed-forward network / Multiple layer perceptron (MLP)
- Radial basis network
- Faltade (Convolutional) nätverk (CNN): bilder, videor, tidserier.
- Recurrent neural networks, LSTM
- Transformer networks

Neurala nätverk

Feed-forward nätverk: Inlager $\rightarrow$ Gömda lager $\rightarrow$ Utlager



Neurala Nätverk

För oövervakad inlärning:

- Dolda representationer: Autoencoders, Variational 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)
- Exempel: chatGPT, Gemini, Llama, stable diffusion

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?

Feature learning

Vanligt i linjär regression,

- Givet data $\mathbf{X} = (x_1, x_2, \ldots, x_p)$ och $y = \mathbf{X}\beta$.
- Vi kan transformera variablerna i $\mathbf{X}$:
  - Polynomregression, $\mathbf{X} = (x, x^2, x^3, \ldots, x^p)$
  - Funktioner, $\log(x), \sqrt{x}, \exp(x), \ldots$
  - Interaktioner, $x_1 x_2$
  - Stegfunktioner
  - Disktretisering
  - 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 $\mathbb{X} = (x_1, x_2, \ldots, x_p)$.
- 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 $\mathbb{X} = (x_1, x_2, \ldots, x_p)$.
- 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,

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

- Hur ska vi välja $h_i(x)$?

Feature learning

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

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

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 som opererar elementvis och låt $h_i(x_1, \ldots, x_p)$ vara en linjär funktion,

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

Låt nu

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

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

Definiera lager

För MLP brukar vi skriva

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

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,

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

Sista lagret - output layer

Hur sista lagret ser ut och vilken aktivetsfunktion som används beror på repsonsvariabeln.

Regression:

- $y \in \mathbb{R} \rightarrow$ använd identitetsfunktionen
- $y \in \mathbb{R}^+ \rightarrow$ använd $exp()$ eller softplus $log(1 + exp(x))$
- Om y är multivariat $\rightarrow$ låt utlagret ha flera noder

Klassificering

- y binär $\rightarrow$ sigmoid $1/(1 + exp(-x))$ - flera sigmoid-noder kan användas om y är multivariat
- y nominell $\rightarrow$ en nod per klass $\rightarrow$ Softmaxfunktionen:

$$\sigma(\mathbf{z})_i = \frac{exp(z_i)}{\sum_{j=1}^{K} exp(z_j)}$$

Notera: ett nätverk kan utföra regression och klassificering samtidigt om vi vill!

Neurala nätverk

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".
- "Black box"

Universal approximation theorem

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

Universal approximation theorem (informellt): En MLP med ett lager och en icke-linjär aktiveringsfunktion kan approximera godtycklig kontinuerlig eller diskret funktion med ett godtyckligt litet fel givet tillräckligt många gömda neuroner.

Optimering av neurala nätverk

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

Vill hitta

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

där a är nätverkets parametrar och $L(a)$ är kostandsfunktionen. Löser detta genom sekvensen

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

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

Optimering av neurala nätverk

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 $\rightarrow$ annan ordning på gömda noder ger samma modell/output
        - Skalning mellan lager
    - Kan ha oräkneligt antal lokala minima.

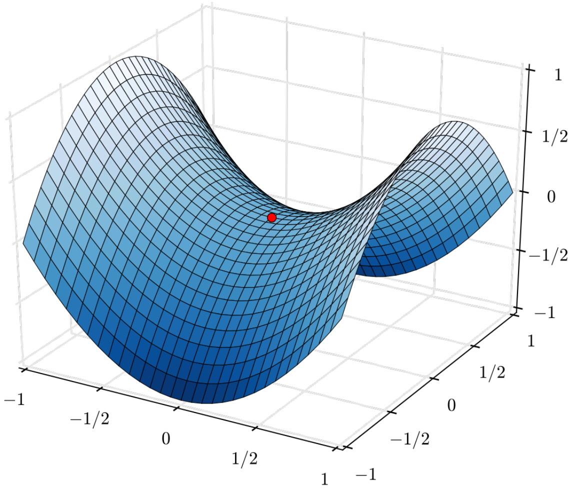
Optimering av neurala nätverk

Platåer och sadelpunkter

- Ställen där gradienten är noll (eller nära), fast vi inte är på ett loaklt 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.

Optimering av neurala nätverk

Exempel på sadelpunkt i två dimensioner:



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 urval 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 epok (epoch) är en genomgång av all träningsdata i SGD.

Hyperparametrar

I neruala 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 optimeringsalgoritm som används)

Hyperparametrar

Hur ska vi bestämma deras värden?

- Svår fråga utan exakt svar.
- Mycket trial and error.
- Valideringsdata
- Använda föreslagna nätverksarkitekturer från litteraturen
  - Finns en mängd förtränade modeller som kan anpassas för nya problem/data
- För stora problem kan det ta lång tid att hitta bra hyperparametrar

Kommentarer - neurala nätverk

- Mycket flexibelt ramverk för att modellera data
  - Många olika kostandsfunktioner för olika situationer
  - Många specialiserade arkitekturer
  - Klarar av många olika sorts data
- Har ofta bra prediktiv förmåga
- Finns risk för överanpassning
- Funkar ofta bra med många observationer och variabler
  - Tabelldata: på mindre dataset så funkar ofta andra enklare metoder lika bra eller bättre.
- Låg tolkningsbarhet på de flesta nätverk: "black box modeling"
- Kan ta lång tid att träna och bestämma hyperparametrar