Föreläsning 5 - Neurala Nätverk

Josef Wilzen

2021-09-08

Outline

1. Neurala nätverk
2. Feature learning
3. Optimering av neurala nätverk
4. Hyperparameterar

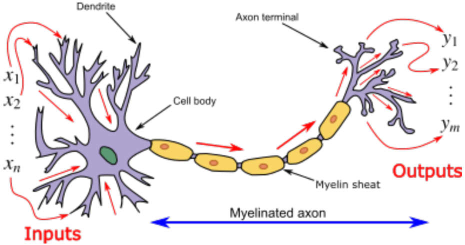
Neurala nätverk

Denna föreläsning utgår ifrån att ni har:

- Sett dessa videor: länk
- Läsning i **ISL**:
  - 10 intro, 10.1-10.2 10.7 intro, 10.7.1, 10.7.2, 10.7.4
- Läsning i **IDM**
  - 6.7 till 6.8.2

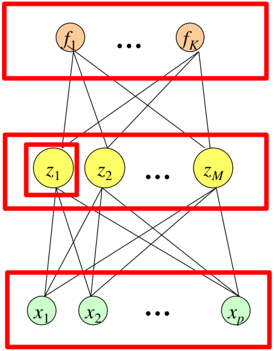
Neurala nätverk

- Neuroner
- Axoner
- Dendriter
- Synapser



Terminologi

Feed-forward nätverk: Inlager - Gömda lager - Utlager



Terminologi

- Feed-forward nätverk
  - Noder i ett lager är bara kopplade till noder i nästa lager
- Återkopplande nätverk:
  - Noder i ett lager kan vara kopplade till noder i samma, föregående eller nästa lager

Neurala nätverk

- Finns **många** olika sorters nätverk! Se här för en sammaställning.
- De används för många olika saker
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning
  - Generativa modeller

Neurala nätverk

- Supervised learning
  - Feed-forward/mult-layer peceptron (MLP)
  - Radial basis networks
  - Faltade (Convolutional) nätverk: bilder, videor, tidserier.

Neurala nätverk

- Unsupervised learning
  - Dolda representationer: Autoencoders
  - Clustering: Self Organizing Map (SOM)
- Generativa modeller:
  - Används för att lära sig komplexa sannolikhetsfördelningar: sampla bilder, text, mm
  - Generative adversarial network (GAN)

Feature learning

Linjär regression

$$y = X\beta + \varepsilon \qquad E[\varepsilon] = 0 \qquad V[\varepsilon] = \sigma^2$$

Feature learning

Linjär regression:

- Givet $X = (x_1, x_2, \ldots, x_p)$, y: $y = X\beta$
- Vi kan transformera variablerna i X
- Polynomregression: $X = (x, x^2, x^3, \ldots, x^p)$
- Andra exempel: $log(x)$, $\sqrt{x}$, $cos(x)$, $exp(x)$, interaktioner, stegfunktioner, diskretisering, dummy-kodning
- Kallas i maskininlärning för "feature engineering"
  - Svårt att veta vilken transformation vi ska göra för ett givet problem!
  - Svårt med komplexa datastrukturer: text, bilder mm

Feature learning

- Vi har $X = (x_1, x_2, \ldots, x_p)$
- Transformationer är funktiner av $(x_1, x_2, \ldots, x_p)$
  - Ex: $h(x) = log(x)$, $h(x_1, x_2) = log(x_1) + sin(x_2)$
- Anta en x variabel, vi kan låta $h(x)$ 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:

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

- Hur ska vi välja $h_i(x)$?

Feature learning

- Linjär transformation: bestäm värden på W och V

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

- Neurala nätverk: Vill kunna modellera icke-linjära funktioner
  - Sätt samman många "enkla" icke-linjära funktioner för att göra en komplex funktion!

Feature learning

Neurala nätverk:
Låt $\sigma()$ vara en enkel icke-linjär funktion, och låt $h_i(x_1, x_2, \ldots, x_p)$ vara en linjär funktion: $h_i(x_1, x_2, \ldots, x_p) = \beta_{0i} + \beta_i^T \boldsymbol{x}$

$$z = \sigma(h_i(x_1, x_2, \ldots, x_p)) = \sigma\left(\beta_{0i} + \beta_i^T \boldsymbol{x}\right)$$

Nästla sedan många sådana funktioner för bygga upp en godtyckligt komplicerad icke-linjär funktion.

Feature learning

- För MLP brukar det skrivas som

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

- $Wa^{(0)} + b$ ger en vektor som är $n \times 1$
- $\sigma()$ opererar elementvis på inputvektorn
- Historiskt, $\sigma(x)$ har valts till sigmoid eller hyperbolic tangent
  - Dessa nätverken visade sig vara svåra att skatta
- Nu används ofta Rectified Linear (ReLu) eller varianter
  - $\sigma(x) = max(0, x)$
  - Funkar bättre med SGD
  - Kan skatta djupa modeller!

Feature learning

Vi kan se neurala nätverk som att vi

1. Automatiskt lär oss en lämplig transformation av de förklarande variablerna
2. Gör linjär (logistik) regression på transformationen = sista lagret

Notera!

- Komplexa funktioner kräver mycket data att lära sig!
- Neurala nätverk kan lätt överanpassa träningsdata!
- Funkar när vi har stort antal förklarande variablerna
- Om vi låter de gömda lagren ha mindre dim än förklarande variablerna: icke-linjär variabelreduktion innan vi når sista lagret (output)

Universal approximation theorem

Universal approximation theorem $\approx$
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

Svårt problem!

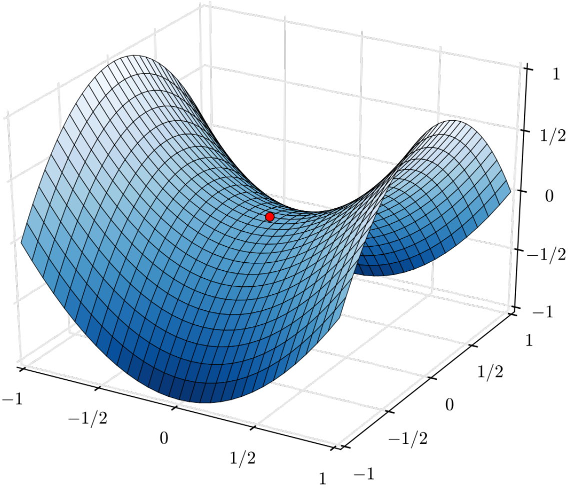
- Lokala minima
  - Kan ha hög kostnad eller låg
  - Model identifiability problem
    - Weight space symmetry
    - Scaling between layers
  - Kan leda till oräkneligt antal lokala minima

Optimering av neurala nätverk

Platåer och sadelpunkter

- Ställen där gradienten är noll (eller nästan noll), fast vi inte är på ett lokalt min/max
- Sadelpunkter:
  - Ta tvärsnitt längs några dimensioner och då har vi lokalt minima i sadelpunkten
  - Ta tvärsnitt längs några andra dimensioner då har vi lokalt maxima i sadelpunkten
- Antalet sadelpunkter tenderar att öka med antalet dimensioner!
- Stora områden som är platta
- Platåer och sadelpunkter: gör optimeringen med gradient decent svårare

Sadelpunkt



Optimering av neurala nätverk

Gradient descent: hitta minimum på en funktion

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

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

Stochastic gradient descent (SGD)

- SGD:
  - Dyrt att beräkna $\nabla f(a_n)$ för alla datapunkter
  - Gör en väntevärdesriktig skattning av $\nabla \hat{f}(a_n)$ genom att ta ett slumpmässigt sample från data (mini-batch)
  - Det finns varians i $\nabla \hat{f}(a_n)$, större batch ger mindre varians men blir dyrare att beräkna.
  - Kräver många iterationer och liten learning rate (γ)
  - Kräver att vi har oberoende observationer i likelihoodfunktionen.
  - Funkar bra för neurala nätverk !

Stochastic gradient descent (SGD)

Require: Learning rate, Initial parameter a

- $k \leftarrow 1$
- **while** stopping criterion not met **do**
  - Sample a minibatch of m examples from the training set $\left(x^{(1)}, \ldots, x^{(m)}\right)$, with corresponding y values.
  - Compute gradient estimate:

$$\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L\left(f\left(x^{(i)}, a\right), y^{(i)}\right)$$

  - Apply update:

$$a \leftarrow a - \gamma \cdot \hat{g}$$

  - $k \leftarrow k + 1$
- **end while**

Hyperparameterar

Det finns många hyperparameterar för neurala nätverk!

- Arkitektur:
  - Antal gömda lager
  - Antal neuroner i varje lager
  - Aktiveringsfunktioner
  - (Specialla typer av neuroner/lager)
- Optimeringen:
  - Mini-batchstorlek
  - Learning rate
  - Antal epoker (antalet gånger som hela träningsmängden används i SGD)

Hyperparameterar

- Hur ska vi bestämma deras värden?
- Svår fråga!
- Mycket trail and error!
- Valideringsdata
- För stora problem/data kan det kan lång tid att hitta bra hyperparameterar

Avslut

- Frågor? Kommentarer?
- Kurshemsidan
- Labben