732G57 Maskininlärning för statistiker

Föreläsning 1B

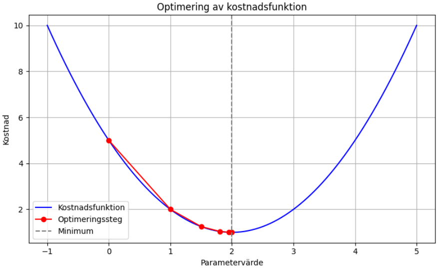
Josef Wilzén
IDA, Linköping University, Sweden

Dagens föreläsning

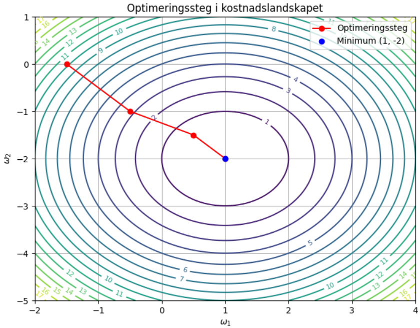
- Optimering
- Modellering, modellval

Introduktion till optimering

- I maskininlärning vill vi ofta minimera en **kostnadsfunktion**: $f(\omega)$
- Parametrar, ω, styr värdet på funktionen – målet är att hitta de värden som ger lägst kostnad.
- Optimering innebär att vi vill hitta värden på ω som ger så låga/höga värden som möjligt på kostnadsfunktionen.
- En variant: stegvis förbättra parametrarna för att närma oss ett minimum eller maximum.



Introduktion till optimering



Optimeringssteg i kostnadslandskapet för $f(\omega_1, \omega_2) = (\omega_1 - 1)^2 + (\omega_2 + 2)^2$

- Konturlinjer visar nivåer av $f(\omega_1, \omega_2)$
- Röda punkter och linjer visar tänkta optimeringssteg
- Minimum vid $(1, -2)$ är markerat med blå punkt

Gradient och Euklidisk norm

Gradient:

- Gradienten av en funktion $f(x_1, x_2, \ldots, x_n)$ är en vektor med partiella derivator:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

- Gradientvektorn pekar i riktningen där funktionen ökar snabbast.
- Kan användas i optimering för att hitta minimum eller maximum.

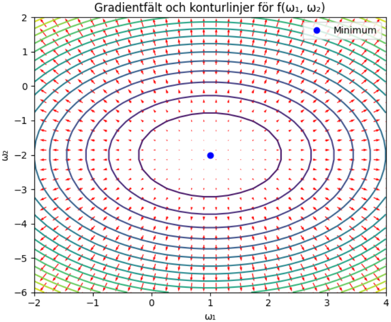
Euklidisk norm:

- Normen av en vektor $v = (v_1, v_2, \ldots, v_n)$ är dess längd:

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

- Kallas även L^2-norm eller den euklidiska normen.
- Används för att mäta storleken på gradienten.

Gradientens norm och riktining i ett kostnadslandskap



Gradientfält och konturlinjer för $f(\omega_1, \omega_2) = (\omega_1 - 1)^2 + (\omega_2 + 2)^2$

- Konturlinjer visar nivåer av $f(\omega_1, \omega_2)$
- Röda pilar visar gradientens riktning och storlek
- Minimum vid $(1, -2)$ där gradienten är noll

Hur hittar vi de bästa värdena på ω?

- Det finns många olika sätt att lösa olika optimeringsproblem på
- Brute force: vi testar många värden på ω och ser vilket som ger bäst värde på $f(\omega) \rightarrow$ oftast inte praktiskt genomförbart
- Vissa problem har enkla/analytiska lösningar
- Optimeringsalgoritmer som stegvis förbättrar värdet på $f(\omega)$

Minimum för en andragradsfunktion

Vi undersöker funktionen:

$$f(x) = x^2 - 4x + 5$$

- Derivera funktionen:

$$f'(x) = 2x - 4$$

- Sätt derivatan lika med noll:

$$2x - 4 = 0 \Rightarrow x = 2$$

- Undersök andraderivatan:

$$f''(x) = 2 > 0$$

- Eftersom $f''(x) > 0$ har funktionen ett **minimum** vid $x = 2$
- Värdet vid minimum:

$$f(2) = 2^2 - 4 \cdot 2 + 5 = 4 - 8 + 5 = 1$$

Slutsats: Funktionen har ett minimum vid $(x, f(x)) = (2, 1)$

Minimum för ett godtyckligt polynom

Antag att vi har ett polynom av grad n:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- **Steg 1:** Beräkna den första derivatan:

$$f'(x) = n a_n x^{n-1} + (n-1) a_{n-1} x^{n-2} + \cdots + a_1$$

- **Steg 2:** Sätt $f'(x) = 0$ och lös ekvationen för att hitta kritiska punkter.
- **Steg 3:** Beräkna den andra derivatan:

$$f''(x) = n(n-1) a_n x^{n-2} + \ldots$$

- **Steg 4:** Undersök tecknet på $f''(x)$ vid varje kritisk punkt:
  - Om $f''(x) > 0 \Rightarrow$ lokalt minimum
  - Om $f''(x) < 0 \Rightarrow$ lokalt maximum
  - Om $f''(x) = 0 \Rightarrow$ vidare undersökning krävs

Slutsats: Minimum hittas där $f'(x) = 0$ och $f''(x) > 0$

OLS-lösning med linjär algebra

Vi utgår från modellen för linjär regression:

$$y = X\beta + \varepsilon$$

Den klassiska OLS-lösningen för β fås genom att minimera residualsumman:

$$\min_{\beta} \|y - X\beta\|^2$$

Lösningen ges av:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

- $X^\top X$ är en (p $\times$ p) matris
- $(X^\top X)^{-1}$ är inversen (om den existerar)
- $X^\top y$ är en (p $\times$ 1) vektor

Slutsats: Vi kan beräkna $\hat{\beta}$ direkt med linjär algebra om $X^\top X$ är inverterbar.

Antaganden om värde- och definitionsmängd

Antagande:

- Kostnadsfunktionen $f(\omega)$ har de reella talen $\mathbb{R}$ som **värdemängd**.
- Parametern ω har de reella talen $\mathbb{R}^p$ som **definitionsmängd**.

Implikationer för optimering:

- Vi kan använda **gradientbaserade metoder** som gradient descent.
- Derivator och normer är definierade och kontinuerliga.
- Möjligt att använda **analytiska verktyg** som första och andra derivatan.
- Lösningen kan ligga var som helst i det reella rummet – inga diskreta begränsningar.

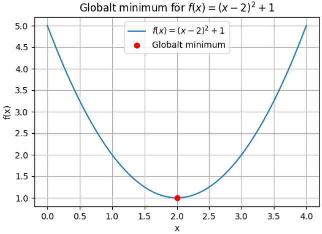
Reell värde- och definitionsmängd möjliggör effektiv optimering med kontinuerliga metoder.

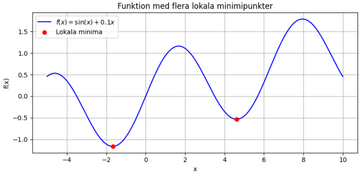
Globala och lokala minimipunkter

Definitioner:

- **Lokalt minimum:** En punkt x_0 där $f(x_0) \leq f(x)$ för alla x i en omgivning kring x_0.
- **Globalt minimum:** En punkt x^* där $f(x^*) \leq f(x)$ för alla x i hela definitionsmängden.

Exempel:





Vänster: globalt minimum. Höger: flera lokala minima.

Vad är en konvex funktion?

Intuitiv förklaring:

- En konvex funktion är "skålformad" – den böjer uppåt.
- Om du ritar en linje mellan två punkter på kurvan, så ligger hela linjen ovanför kurvan.
- Det finns bara ett minimum, och det är det lägsta värdet i hela funktionen.

Exempel:

- $f(x) = x^2$ är en konvex funktion.
- $f(x) = |x|$ är också konvex – även om den har ett hörn.

Varför är det bra?

- Enklare att hitta minimum – vi vet att det inte finns några "fällor".
- Optimeringsalgoritmer fungerar bättre och snabbare.

Konvexa och icke-konvexa funktioner

Icke-konvex funktion:

- Bryter mot konvexitetsvillkoret.
- Kan ha **flera lokala minima** och maxima.
- Exempel: $f(x) = \sin(x)$, $f(x) = x^4 - x^2$

Konsekvenser för optimering:

- Konvexa funktioner: enklare att optimera, globalt minimum kan hittas med gradientmetoder.
- Icke-konvexa funktioner: kräver mer avancerade metoder (t.ex. randomisering, flera startpunkter). Vi har inga garantier att hitta ett globalt minimium.

En generisk optimeringsalgoritm

- Starta med en initial parametervektor ω_0
- Sätt ett max antal iterationer k

För varje iteration $i = 1, 2, \ldots, k$:

1. Uppdatera ω_i baserat på ω_{i-1} enligt en specifik regel
2. Beräkna kostnaden $f(\omega_i)$
3. Undersök konvergens:
   - Om konvergerat: avbryt loopen

Returnera: ω_i och $f(\omega_i)$

Vad menas med konvergens?

- En optimeringsalgoritm sägs ha **konvergerat** när den når ett tillstånd där ytterligare iterationer inte leder till någon meningsfull förbättring.
- Det finns flera sätt att definiera konvergens:
    - **Liten förändring i kostnadsfunktionen:** $|f(\omega_i) - f(\omega_{i-1})| < \varepsilon_1$
    - **Liten förändring i parametrarna:** $\|\omega_i - \omega_{i-1}\| < \varepsilon_2$
    - **Gradientens norm är nära noll:** $\|\nabla f(\omega_i)\| < \varepsilon_3$
    - **Maximalt antal iterationer har uppnåtts**
- Valet av konvergenskriterium påverkar både hur bra den föreslagna lösningen blir och beräkningstiden.

Gradient Descent – En optimeringsmetod

Vad är Gradient Descent?

- En metod för att hitta minimum av en funktion.
- Används ofta för att minimera kostnadsfunktioner i maskininlärning.
- Bygger på att följa den negativa riktningen av gradienten.

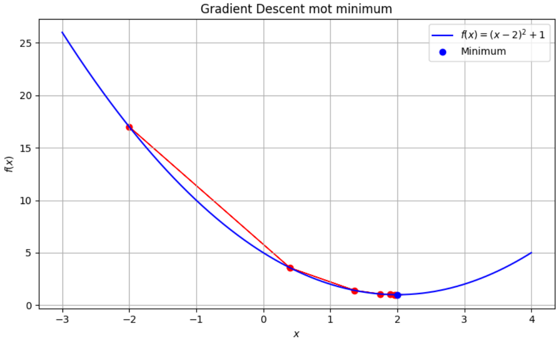
Hur fungerar det?

- Starta från en initial punkt ω_0.
- Uppdatera enligt:
  $$\omega_{t+1} = \omega_t - \gamma \nabla f(\omega_t)$$
  där γ är inlärningshastigheten (stegets storlek).
- Upprepa tills gradienten är nära noll (dvs. vi når ett minimum).

Nyckelidéer:

- Gradient = riktning där funktionen ökar mest.
- Negativ gradient = riktning mot lägre värden.
- Inlärningshastigheten påverkar hur snabbt vi rör oss mot bättre värden.

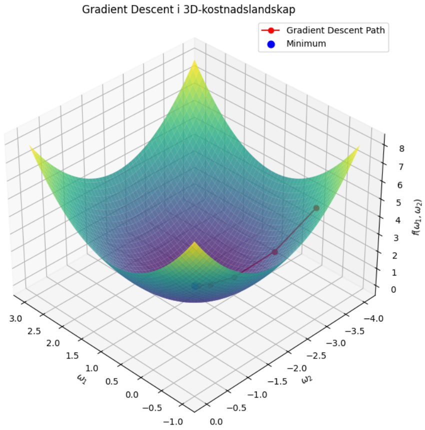
Gradient Descent – Visuell illustration



Gradient descent mot minimum för $f(x) = (x-2)^2 + 1$

- Startpunkt vid $x = -2$
- Pilar visar hur algoritmen rör sig mot lägre värden
- Stegen följer den negativa gradienten
- Minimum nås vid $x = 2$

Gradient Descent i flera dimensioner – 3D-visualisering



Gradient descent i ett 3D-kostnadslandskap: $f(\omega_1, \omega_2) = (\omega_1 - 1)^2 + (\omega_2 + 2)^2$

- Ytan visar kostnadsfunktionen som beror på två variabler.
- Den röda linjen visar optimeringsstegen från startpunkt till minimum.

När konvergerar Gradient Descent?

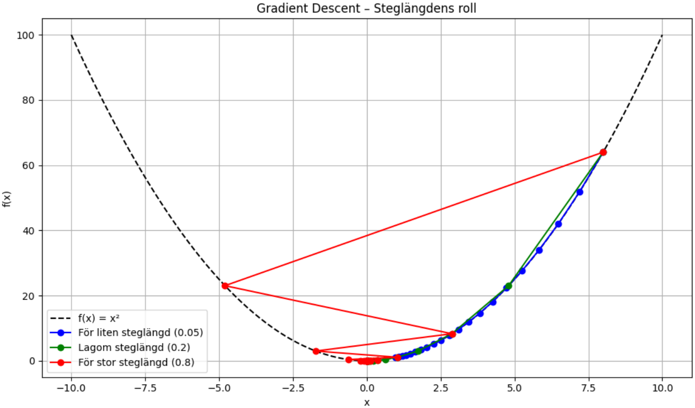
För att Gradient Descent ska konvergera till ett lokalt minimum krävs:

- **Funktionen $f(\omega)$ är differentiabel:** Vi måste kunna beräkna gradienten.
- **Gradienten är Lipschitz-kontinuerlig:** Det finns en konstant L så att
$$\|\nabla f(\omega_1) - \nabla f(\omega_2)\| \leq L\|\omega_1 - \omega_2\|$$
vilket ger stabilitet i uppdateringarna.
- **Inlärningshastigheten γ är tillräckligt liten:** Om $\gamma < \frac{2}{L}$ så garanteras konvergens.
- **Startpunkt nära minimum:** För icke-konvexa funktioner kan algoritmen fastna i lokala minima.
- **Funktionen är konvex (för global konvergens):** Då är varje lokalt minimum också ett globalt minimum.

Med rätt förutsättningar leder gradient descent till ett (lokalt) minimum.

Gradient Descent – Steglängdens roll

- Vi optimerar funktionen $f(x) = x^2$ med gradient descent.
- Startpunkt: $x = 8$, antal iterationer: 20.
- **För liten steglängd (0.05):** långsam konvergens.
- **Lagom steglängd (0.2):** snabb och stabil konvergens.
- **För stor steglängd (0.8):** hoppar över minimum, risk för divergens.



Gradient Descent: Fördelar och nackdelar

Fördelar

- Enkel att implementera
- Fungerar för stora datamängder
- Flexibel för olika typer av modeller
- Kan användas med olika förbättringar/utökningar

Nackdelar

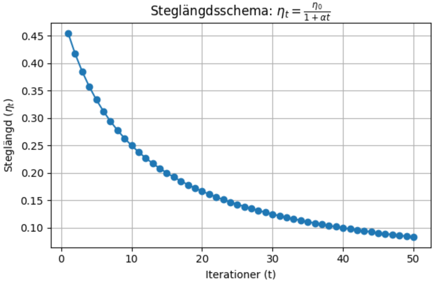
- Känslig för val av steglängd
- Kan fastna i lokala minima
- Långsam konvergens
- Kräver derivator av målfunktionen

Vanliga förbättringar av Gradient Descent

- Stochastic Gradient Descent (SGD)
- Momentum
- Nesterov Accelerated Gradient
- Adagrad
- RMSprop
- Adam
- AdaDelta
- Learning rate scheduling/Steglängdsscheman

Gradient Descent – Steglängdsscheman

- Ett **steglängdsschema** är en metod för att ändra steglängden γ över iterationer i gradient descent.
- Syftet är att börja med en större steglängd för snabb konvergens, och sedan minska den för stabilitet.
- Vanligt schema: $\gamma_t = \frac{\gamma_0}{1+\alpha t}$ där γ_0 är initial steglängd och α styr minskningstakten.
- Hjälper algoritmen att undvika att studsa runt minimum i senare iterationer.



Gradient Descent för Multipel Linjär Regression

- Vi modellerar sambandet mellan flera variabler med:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$$

  där:
  - $\mathbf{y}$ är en $n \times 1$-vektor med utfall,
  - $\mathbf{X}$ är en $n \times p$-matris med prediktorer (inkl. intercept),
  - $\boldsymbol{\beta}$ är en $p \times 1$-vektor med koefficienter.
- Kostnadsfunktion (MSE):

$$f(\boldsymbol{\beta}) = \frac{1}{n}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

- Gradienten ges av:

$$\nabla f(\boldsymbol{\beta}) = -\frac{2}{n}\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

- Uppdateringsregel:

$$\boldsymbol{\beta} := \boldsymbol{\beta} - \gamma \cdot \nabla f(\boldsymbol{\beta})$$

  där γ är steglängden.

Coordinate Descent – En variabel i taget

Vad är Coordinate Descent?

- En optimeringsmetod där man optimerar en variabel i taget.
- Alla andra variabler hålls fasta under varje steg.
- Itererar över variablerna tills konvergens uppnås.

Nyckelidéer:

- Enkel att implementera, särskilt för stora problem.
- Effektiv när varje delproblem (en variabel) är lätt att lösa.
- Används ofta i t.ex. Lasso-regression (mer om det senare) och konvex optimering.
- Kräver inte beräkning av hela gradienten.

Första och andra ordningens metoder

- **Första ordningens metoder:**
  - Använder endast gradienten (första derivatan) av kostandsfunktionen.
  - Exempel: Gradient Descent, Stochastic Gradient Descent.
- **Andra ordningens metoder:**
  - Använder både gradienten och Hessianen (andra derivatan).
  - Exempel: Newtons metod, Quasi-Newton (t.ex. BFGS).
- Val mellan metoder beror på problemets struktur, resurser och krav på noggrannhet.

Vad är en Hessian?

- Hessianen är en matris med alla andra ordningens partialderivator av en funktion.
- Används i optimering för att analysera kurvatur och hitta extrema punkter.
- **Exempel:** Funktionen $f(x, y) = x^2 + xy + y^2$

$$\nabla^2 f(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Exempel på Hessian: Funktion med tre variabler

- Funktion: $f(x, y, z) = x^2 + xy + y^2 z + \sin(z)$
- Hessianen innehåller alla andra ordningens partialderivator:

$$\nabla^2 f(x, y, z) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2z & y^2 \\ 0 & y^2 & -\sin(z) \end{bmatrix}$$

Viktiga andra ordningens optimeringsmetoder

- Newtons metod
- Quasi-Newton-metoder
  - BFGS (Broyden–Fletcher–Goldfarb–Shanno)
  - L-BFGS (Limited-memory BFGS)
  - DFP (Davidon–Fletcher–Powell)
- Gauss-Newton-metoden
- Conjugate Gradient (för icke-linjär optimering)

Begränsningar via reparameterisering

- Ibland måste parametrar uppfylla vissa villkor, t.ex. vara positiva.
- Istället för att optimera direkt på parametern, kan vi omformulera problemet.
- **Exempel:** Om $\theta > 0$, optimera istället över ϕ där:

$$\theta = \exp(\phi)$$

- Nu är θ alltid positiv, oavsett värde på ϕ.
- Reparameterisering gör det möjligt att använda gradientbaserade metoder utan att hantera begränsningar direkt.

Värdemängd och definitionsmängd i optimering

Värdemängd för $f(\omega)$:

- Värdemängden är alla möjliga utfall av kostnadsfunktionen $f(\omega)$.
- Exempel: Om $f(\omega) = \|\omega\|^2$ är värdemängden $[0, \infty)$.

Definitionsmängd för ω:

- **Reella tal:** $\omega \in \mathbb{R}^p \Rightarrow$ gradientbaserade metoder som gradient descent kan användas.
- **Heltal:** $\omega \in \mathbb{Z}^p \Rightarrow$ diskreta metoder som branch-and-bound eller dynamisk programmering krävs.
- **Intervall:** $\omega \in [a, b]^p \Rightarrow$ optimering med begränsningar, t.ex. projicerad gradient descent eller L-BFGS-B.
- **Blandade variabler:** Kombination av reella och heltal $\Rightarrow$ mixed-integer programming (MIP).

Valet av optimeringsalgoritm beror på både värdemängden för $f(\omega)$ och definitionsmängden för ω.

Optimering i modellering

- Optimering är centralt för att träna/skatta modeller inom statistik och maskininlärning.
- Målet är ofta att minimera en förlustfunktion eller maximera en sannolikhet.
- **Exempel:**
    - Linjär regression: minimera residualsumma
    - Logistisk regression: maximera log-likelihood
    - Regulariserade modeller: balansera förlust och komplexitet
    - Neurala nätverk: minimera förlust via gradientbaserade metoder
- Val av optimeringsmetod påverkar både noggrannhet och effektivitet.

Parametrar vs. Hyperparametrar

- **Parametrar:**
  - Lärs direkt från data under träning.
  - Exempel: parametrar i en linjär (logistisk) regressionsmodell
- **Hyperparametrar:**
  - Ställs in före träning av parametarna
  - "Styr modellens/skattningens övergripande egenskaper"
  - Kan handla dels om modellen, men också om hur vi skattar modellens parametrar
  - Skattning/optimiering, exempel: val av algoritm, lärhastighet, antal iterationer, regulariseringsstyrka
  - Modellens struktur kan vara en hyperparameter:
    - Antal variabler eller features
    - Val av modelltyp (t.ex. linjär vs. icke-linjär)
    - Antal lager i ett neuralt nätverk
- Vi kan inte skatta hyperparametrar på "vanligt" sätt $\rightarrow$ leder ofta till överanpassning

Val av hyperparametrar med validering

- Hyperparametrar styr modellens träning och struktur.
- För att välja bra värden testar vi olika alternativ och utvärderar modellen.
- **Validering:**
  - Dela upp data i tränings- och valideringsdel.
  - Träna modellen på träningsdata, utvärdera på valideringsdata.
- **Korsvalidering:**
  - Dela upp data i flera delar (folds).
  - Träna och utvärdera modellen flera gånger med olika uppdelningar.
  - Vanligt: 5- eller 10-faldig korsvalidering.
- Välj de hyperparametrar som ger bäst genomsnittlig prestanda.
- Vilka värden ska vi välja? Grid search, random search, ...

Frågor?