

Forelesning 3 - Maskinlæring for økonomer

Til nå:

- Tensorer $\stackrel{\text{rank} = 0}{\leftarrow} \stackrel{\text{rank} = 1}{\leftarrow} \dots$
- Prediksjon versus inferens
- Irreducible versus reducible error
- Regresjon versus klassifikasjon
- Trenings/valid. / test-sett
- Diskutert overfitting \rightarrow ulike metrikker
- Confusion matrix

$$E[(Y - \hat{Y})^2] = [f(X) - \hat{f}(X)]^2 + \text{Var}(\epsilon)$$

- Strukturer for et nevrat nettverk

- Neuron, aktivitetsfunksjon, vektorer, bølgjer

- Ulke lag

- Aktivitetsfunksjon, loss-funksjon

- Notesjon, neurone back- og forward propagasjon

- At hele nettverket basically er en funksjon

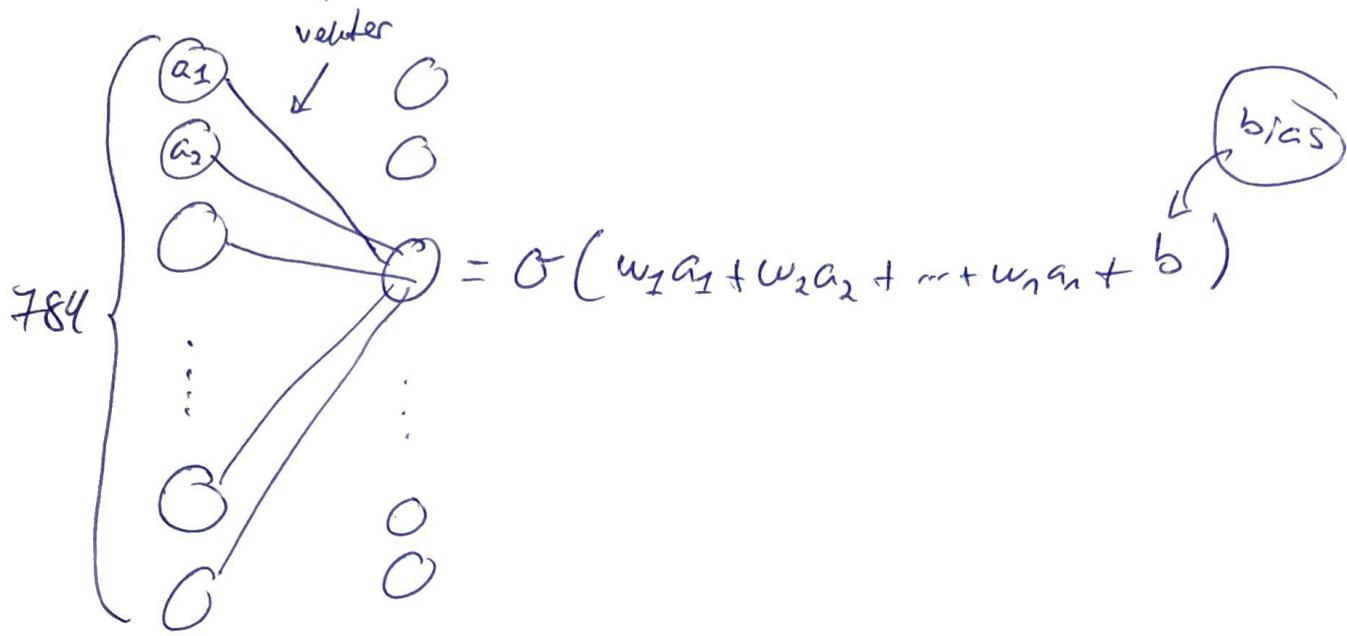
$$f(a_0, \dots, a_{783}) = \begin{bmatrix} y_0 \\ \vdots \\ y_q \end{bmatrix}$$

- At "læring" handler om å fine "rike" verdi for vektorer og bølgjene - til å få ønsket prediksjon.

↓
Grade nettkjernen!

Idag: Skal vi fokusere på hvannetverk "lærer"?

Husk at vi hadde:



Slik nettverket gjør det:

- Tilfeldig gi vektorer (w_1, \dots, w_n) og bias (b) random verdier.

=> Da vil (selvsagt) output et bli klarlig!

↳ Da harer vi en måte å "straffe" funksjonen på.

Da definerer vi en cost-funksjon!

F.eks. MSE

$$\text{Cost} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

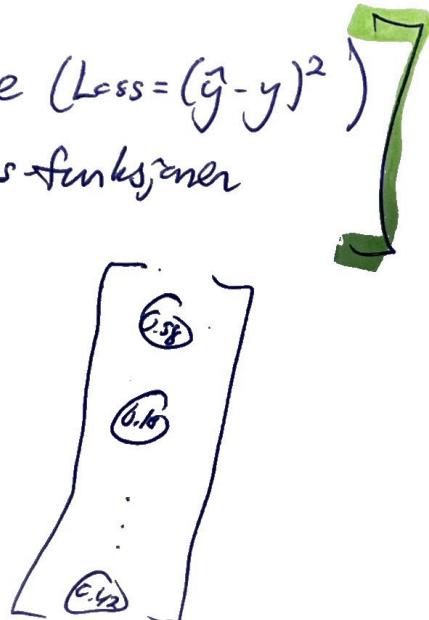
Dette evaluerer performance til modellen og optimiseringssalg. Minimerer dette.

NB: Loss-funksjon = feil for en enkelt hendelse ($\text{Loss} = (\hat{y} - y)^2$)

Cost-funksjon: Gjennomsnitt (eller sum) av loss-funksjoner over hele dataseksett

$$\text{Cost} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

hva modellen gir
hva sannheten er?



Nettverke-funksjoner

Input: 784

Output: 10

Parametere: 13.002 vektorer/bas

Kost-funksjoner

Input = 13.002 vektorer/bas

Output: I tall ("koster")

Parametere = Treningsdata'er (mange eksempler)

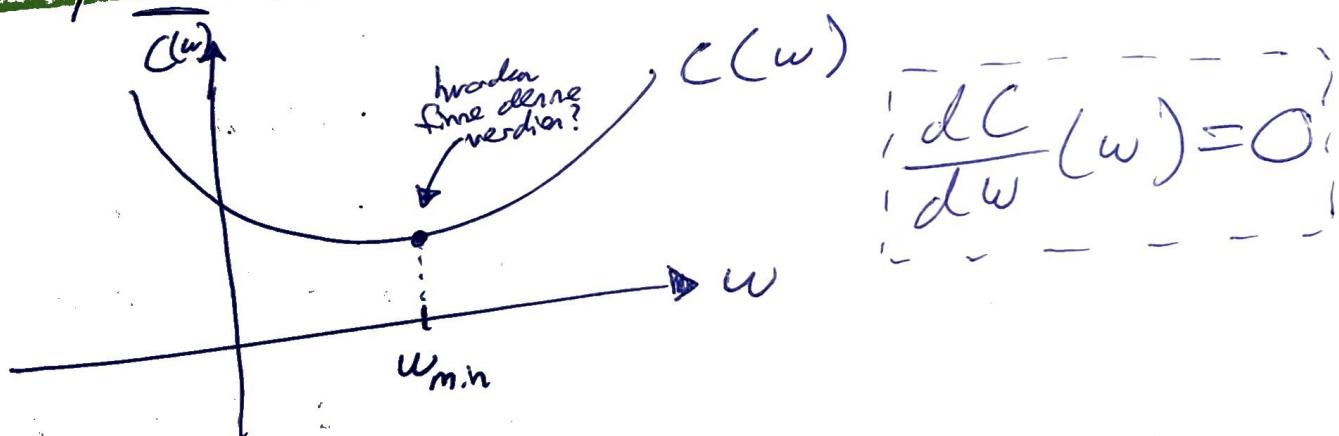
$(x_1, y_1), (x_2, y_2), \dots$

V_i kan se på cost-funksjoner, og skrive der

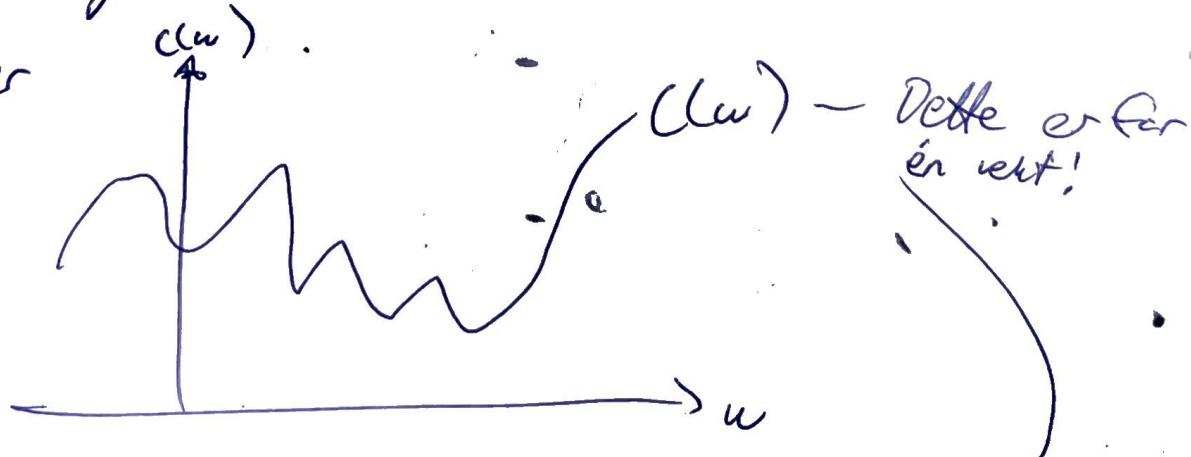
$$C(w_1, w_2, \dots, b)$$

Vanligvis er det enkelte
verdier som vi ønsker
at funksjonen skal få?

Ser på en veit! Da det er lettare å se følgende!

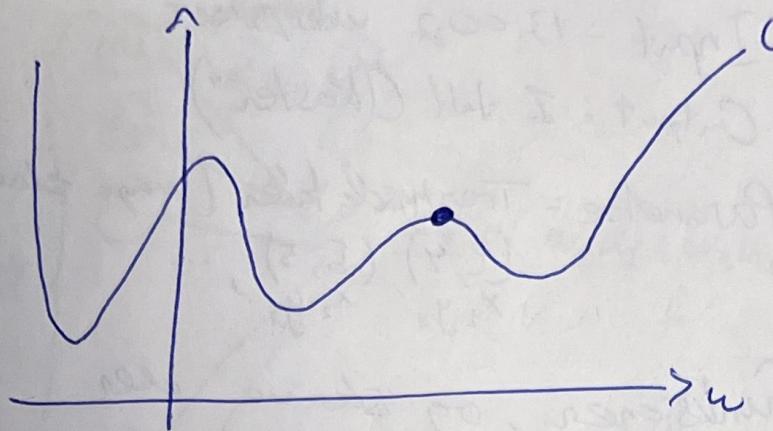


Dette lar seg ikke alltid gjøre for svært komplekse
funksjoner



og enda verre blir alt med 13.002 vektorer/bas.

En strategi



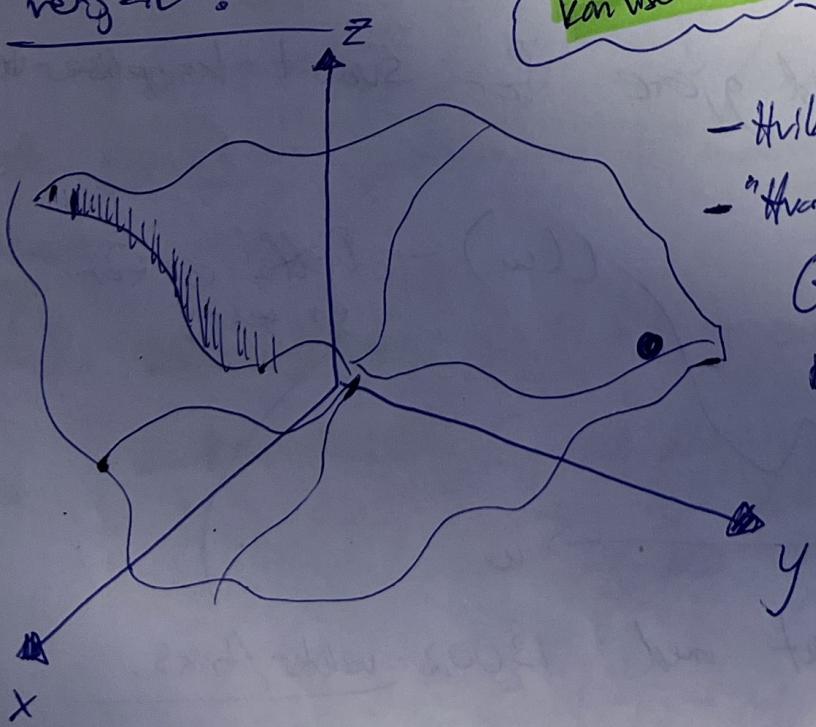
En bedre teknikk:

- ① Bygge et ikke-lin. w
- ② Finn stigningen til funksjonen i w
- ③ "Gå" i den retningen hvor w minimeres. Da følger du en ny w
- ④ Gjentk 2 og 3, og du kommer i et lokalt minimum.

OBS: Lokalt minimum \neq globalt minimum.

Man kan "get trapped" i lokale minimum.

Så nå venter vi å se på en litt mer kompleks + verden:



Kan vi se i detalj?

- $C(x,y)$ går i z -retning
- hvilken retning minimerer $C(x,y)$ først?
 - "Hva driller bollen?"

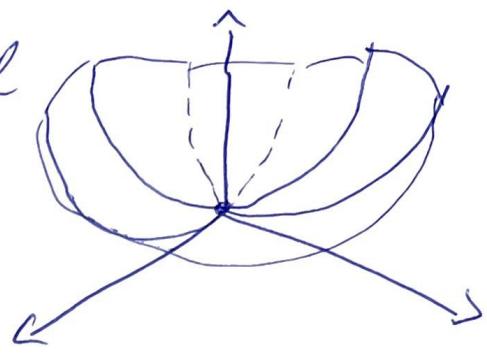
Gradiensen av en funksjon gir retning for brattest stigning.

Skrives $\nabla C(x,y)$

Gradient Descent

Et enkelt Gradient Descent eksempel

Vi ønsker å minne $f(x, y) = x^2 + y^2$



$$\textcircled{1} \quad \frac{\partial f}{\partial x} = 2x, \quad \frac{\partial f}{\partial y} = 2y$$

$$\textcircled{2} \quad \nabla f(x, y) = (2x, 2y)$$

$$\textcircled{3} \quad (x_0, y_0) = (3, 4), \quad \text{leirhysrate } \eta = 0.1$$

$$(x_1, y_1) = (x_0, y_0) - \eta \cdot \nabla f(x_0, y_0)$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} - 0.1 \cdot \begin{bmatrix} 2 \cdot 3 \\ 2 \cdot 4 \end{bmatrix}$$

$$= \begin{bmatrix} 3 \\ 4 \end{bmatrix} - 0.1 \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$

$$= \begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix}$$

Og gjent... for $\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}, \begin{pmatrix} x_3 \\ y_3 \end{pmatrix}, \dots$

Gradienten kan regnes ut

$$\nabla C(x, y) = \left(\frac{\partial C}{\partial x}, \frac{\partial C}{\partial y} \right)$$

for to parametere.

Denne vektoren gir retninger cost-funksjonen stiger mest.
Vi er da interessert; $-\nabla C(x, y)$.

Dette kan vi gjøre for
villekurs antall parametere / input.

$$\vec{w} = \begin{bmatrix} 2.25 \\ 1.62 \\ \vdots \\ -1.41 \\ 2.42 \end{bmatrix}$$

alle wekter/
bias

$$-\nabla C(\vec{w}) = \begin{bmatrix} 0.42 \\ -0.45 \\ \vdots \\ 0.52 \\ -0.41 \end{bmatrix}$$

bestemmer steget
som tas i retning av gradienten.

"Learning rate"

Vil fine
balanse mellom
"liten skritt" og
"stort skritt".

$$\vec{w} := \vec{w} - \eta \cdot \nabla C(\vec{w}) \quad \text{— og repeat!}$$

"Lærings" = minimere en cost-funksjon

Før at dette skal slye, må cost-funksjoner være glatt og smooth.
Dette er grunnen til at "artificial" neuroner har en røye med løft.
verdier/aktiviteter istedenfor 0/1.

Men det er ikke kun cost-funksjon i implementering.
Sam skal fil først at NN løver?

For å beregne den sikt komponerte graden
bruker vi backpropagasjon.

Denne algoritmen for å regne ut

$$-\nabla C(\vec{w})$$

$$-\nabla C(\vec{w}) = \begin{bmatrix} \vdots \\ 2.40 \\ \vdots \\ 0.20 \\ \vdots \end{bmatrix}$$

$$C(\dots w_n \dots w_w \dots) = \dots$$

Endrer vi w_n , er cost-funksjonen $C(\vec{w})$ 12 ganger mer
sensitiv, enn hvis vi endrer w_u .

- Denne sikt lange vektorer, $-\nabla C(\vec{w})$, kan sees på som et verdier til vektorene/briksene sør om hvor sensitiv cost-funksjonen er til en endring i disse vektorene/briksene.

Kodeklat - til forelesning 3

- Vi har diskutert MNIST, NN-arkitektur og gradient descent.
 - ↳ Her så bruker vi ADAM som vår optimisator.

- Regularisering (tilby ord til prosjektet)

- ↳ Regularisering er en teknikk for å hindre overfittning, altså at modellen lærer treningsdataene for godt. Overfittning siger ofte når et modell er for kompleks, som f.eks. for mange parametere.

Kan være:

- Dropout: Dropper tilfeldig noen neuroner under trening for å hindre at nettverket blir "utøvd" av verste neuroner.
 - ↳ Hinder memorizing av mønstre i X-train.
 - ↳ Låtter lage mer "robuste" ferdigheter.

- Early stopping: Stopper treningen tidlig, hvis modellen slutter å forbedre seg på validitetssettet.

- ↳ Settes ved "patience" er lik en eller annen verd.: F.eks. etter 3 epocher uten forbedring → stopp trening.

- Andre teknikker: L1, L2-reg., Batch-normalisering, Data-Augmentation, store batcher, redusjon i nettverkets størrelser...

Mer treningsdata, bedre treningsdata, kryssovalidering.