Kjerneregelen og Maskinlæring
MAT1110 Obligatorisk oppgave 1 av 2

Øyvind Ryan

13. februar 2024

Innleveringsfrist

Torsdag 29. februar 2024, klokken 14:30 i Canvas (canvas.uio.no).

Instruksjoner

Du velger selv om du skriver besvarelsen for hånd og skanner den, eller om du skriver løsningen direkte inn på datamaskin (for eksempel ved bruk av LaTeX). Skannede ark må være godt lesbare. Det forventes at man har en klar og ryddig besvarelse med tydelige begrunnelser. Besvarelsen skal leveres som én PDF-fil. Husk å inkludere eventuell kode og kjøreeksempel, samt relevante plott og figurer i PDF-filen.

Merk at man har kun ett forsøk på oppgaven. Samarbeid og alle slags hjelpemidler er tillatt, men den innleverte besvarelsen skal være skrevet av deg og reflektere din forståelse av stoffet. Er vi i tvil om du virkelig har forstått det du har levert inn, kan du bli bedt om en muntlig redegjørelse.

Søknad om utsettelse av innleveringsfrist

Hvis du blir syk eller av andre grunner trenger å søke om utsettelse av innleveringsfristen, må du ta kontakt med studieadministrasjonen ved Matematisk institutt (e-post: studieinfo@math.uio.no) før innleveringsfristen. Vitenskapelig ansatte kan ikke innvilge utsettelser.

For å få adgang til avsluttende eksamen i dette emnet, må man bestå alle obligatoriske oppgaver i ett og samme semester.

For å få godkjent denne første obligatoriske oppgaven må du ha gjort seriøse forsøk på å løse alle deloppgavene, og minst halvparten av oppgavene må være tilfredsstillende besvart.

For fullstendige retningslinjer for innlevering av obligatoriske oppgaver, se her:

www.uio.no/studier/admin/obligatoriske-aktiviteter/mn-math-oblig.html

LYKKE TIL!

Denne obligatoriske oppgaven går ut på på å løse 6 korte oppgaver rundt kjerneregelen, Jacobimatriser, og lineær algebra. Det viser seg at disse oppgavene henger tett sammen med et generelt oppsett innen maskinlæring der man lager det som kalles *nevrale nett*. Dette generelle oppsettet er litt mer komplisert. Stoffet knyttet til dette kan du finne etter de 6 oppgavene, men det anbefales mest for spesielt interesserte. Uansett så er dette av stor viktighet, da det er en viktig byggeblokk innen maskinlæring, og kommer med en helt konkret algoritme.

For deg som vil vite mer kan du også ta en titt på videoserien til 3Blue1Brown om nevrale nett. Her forklares nevrale nettverk på en relativt enkel visuell måte (samtidig som det nødvendige av lineær algebra er med). Fordelen med denne obligen er at vi bruker notasjon som er kompatibel med læreboka (mye litteratur i maskinlæring er ikke kompatibelt på samme måte). Fordelen med videoserien er at den ikke er like matematisk tung.

Oppgave 1

Anta at $f_1, ..., f_n$ er deriverbare funksjoner fra $\mathbb{R}^n$ til $\mathbb{R}$. Forklar at

$$\nabla(f_1 + \cdots + f_n) = \nabla f_1 + \cdots + \nabla f_n,$$

det vil si at gradienten til en sum er lik summen av gradientene.

Oppgave 2

Vi vil kalle funksjonen fra $\mathbb{R}$ til $\mathbb{R}$ definert ved $\sigma(x) = 1/(1 + e^{-x})$ for *aktiveringsfunksjonen*. Vis at $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

Oppgave 3

Vi utvider definisjonen av σ slik den også er definert på $\mathbb{R}^n$ (og tar verdier i $\mathbb{R}^n$). Dette gjør vi ved å definere den komponentvis ved

$$\sigma(\mathbf{x}) = \sigma \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_n) \end{pmatrix}. \tag{1}$$

Finn $n \times n$-Jacobimatrisen til σ, d.v.s. $\sigma'(\mathbf{x})$.

Oppgave 4

Anta at $f(\mathbf{x}) = |\mathbf{x} - \mathbf{y}|^2$, der $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Finn gradienten til f.

Oppgave 5

Anta at W er en $n \times n$-matrise med elementer w_{ij}, og at $\mathbf{b} \in \mathbb{R}^n$ er en søylevektor med komponenter b_i. Vi har lært at Jacobimatrisen til den affine avbildningen $\mathbf{F}(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$ er $\mathbf{F}'(\mathbf{x}) = W$. La oss nå i stedet se på funksjonen $\mathbf{G} : \mathbb{R}^{n^2+n} \to \mathbb{R}^n$ definert ved

$$\mathbf{G}\left(w_{11}, ..., w_{1n}, ..., w_{n1}, ..., w_{nn}, b_1, ..., b_n\right) = W\mathbf{x} + \mathbf{b}.$$

Med andre ord, vi ser nå på elementene i W og $\mathbf{b}$ som variable (med disse listet opp rad for rad, fra venstre mot høyre), og $\mathbf{x}$ som en konstant. Forklar at Jacobimatrisen til $\mathbf{G}$ er

$$\left(\begin{array}{cccc|cccc} \mathbf{x}^T & O & \cdots & O & 1 & 0 & \cdots & 0 \\ O & \mathbf{x}^T & \cdots & O & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ O & O & \cdots & \mathbf{x}^T & 0 & 0 & \cdots & 1 \end{array}\right), \tag{2}$$

der O står for en vektor eller matrise med bare nuller. Over blir altså $\mathbf{x}$ repetert på hver rad.

Blokkmatriser

Anta at vi har 4 matriser A, B, C, og D. Vi skriver

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

for matrisen der elementene i disse 4 matrisene er plassert over, under, til venstre, eller til høyre for hverandre (I Matlab ville dette bli `[A B;C D]`). Vi kaller dette for en *blokkmatrise*. Vi bruker altså streker til å adskille blokkene.

Hvis antall rader i A og C til sammen er forskjellig fra antall rader i B og D til sammen, så gir ikke dette mening. Men hvis

- A og B har like mange rader,
- C og D har like mange rader,

og tilsvarende for søylene, så gir det mening. Videre gir multiplikasjon av blokkmatriser mening: Vi har at

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)\left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right) = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

hvis i tillegg

- antall søyler i A er lik antall rader i E,
- antall søyler i B er lik antall rader i G.

I det følgende vil vi anta at dette er oppfylt, slik at vi kan multiplisere blokkmatriser på samme måte som for matriser. En blokkmatrise kan også ha flere enn 4 blokker - det som står over lar seg greit generalisere.

Oppgave 6

Anta U, V, og X er matriser. Vis at

$$\left(\begin{array}{c|c|c} I_1 & O & O \\ \hline O & U & V \end{array}\right)\left(\begin{array}{c|c|c} I_1 & O & O \\ \hline O & I_2 & O \\ \hline O & O & X \end{array}\right) = \left(\begin{array}{c|c|c} I_1 & O & O \\ \hline O & U & VX \end{array}\right)$$

Her er I_1 og I_2 identitetsmatriser, og det antas at dimensjonene til matrisene er slik at blokkmatrisemultiplikasjonen over gir mening.

Oppsettet for maskinlæring (for spesielt interesserte!)

I maskinlæring forsøker man å finne en funksjon som tilnærmer et sett med målingsdata best mulig. For at dette skal være mulig å regne på, krever vi at kandidatfunksjonene har en gitt form. Hvis målingsdataene er $\mathbf{x}_i \in \mathbb{R}^n$, $\mathbf{y}_i \in \mathbb{R}^n$, så vil vi finne en $\mathbf{F}$ på den gitte formen slik at "den totale feilen"

$$\sum_{i=1}^{n} |\mathbf{F}(\mathbf{x}_i) - \mathbf{y}_i|^2$$

i målingsdataene er minst mulig. Vi antar at funksjoner på den gitte formen er unikt beskrevet ved et sett *parametre* $w_1, w_2, ...$, og vi skriver da

$$\mathbf{F}_{w_1,w_2,...} : \mathbb{R}^n \to \mathbb{R}^n$$

for $\mathbf{F}$. Vårt spørsmål blir: Hvilket valg av parametre gir oss en funksjon $\mathbf{F}_{w_1,w_2,...}$ med total feil

$$f(w_1, w_2, ...) = \sum_{i=1}^{n} |\mathbf{F}_{w_1,w_2,...}(\mathbf{x}_i) - \mathbf{y}_i|^2 \tag{3}$$

minst mulig? Dette er et minimeringsproblem der $w_1, w_2, ...$ er variablene våre (ikke $\mathbf{x}_i$ og $\mathbf{y}_i$, disse er gitte målingsdata!).

Å finne $w_1, w_2, ...$ som gir minimum for (3) viser seg å være vanskelig. Det å finne gode tilnærminger til minimum kan være enklere: Vi har jo lært at gradienten til en funksjon peker i retningen funksjoner vokser raskest. Derfor vil funksjonen avta raskest hvis vi går i motsatt retning av gradienten, og det er nettopp det vi skal gjøre: Vi regner ut gradienten til f (Oppgave 1-6 hjelper oss med dette) gitt ved (3), og følger denne et lite stykke for å komme til et annet punkt, der f har mindre verdi.

I vår setting er det lettere å tenke på variablene våre som komponenter i flere matriser og vektorer, selv om kjerneregelen krever en vektor (Oppgave 5 forklarer hvordan man konverterer dette til en lang vektor). Hvis $\mathbf{F}$ er en funksjon der variablene er samlet i en matrise W og en vektor $\mathbf{b}$, så vil vi skrive $\mathbf{F}(W, \mathbf{b})$ i stedet for $\mathbf{F}(w_1, w_2, ...b_1, b_2, ...)$.

Formen vi krever for kandidatfunksjonen $\mathbf{F}_{w_1,w_2,...}(\mathbf{x})$ er

$$\mathbf{F}_{W^{(1)},\mathbf{b}^{(1)},W^{(2)},\mathbf{b}^{(2)}...}(\mathbf{x}) = \mathbf{F}_1(\mathbf{F}_2(\cdots \mathbf{F}_r(\mathbf{x}))) \tag{4}$$

der

$$\mathbf{F}_i(\mathbf{x}) = \sigma(W^{(i)}\mathbf{x} + \mathbf{b}^{(i)}) \tag{5}$$

($\sigma : \mathbb{R}^n \to \mathbb{R}^n$ er (den komponentvise) aktiveringsfunksjonen). Alle $W^{(i)}$ er $n \times n$-matriser, og alle $\mathbf{b}^{(i)}$ er vektorer med lengde n. Vi kaller en slik $\mathbf{F}$ for et *nevralt nett*, og F_i'ene for *lagene* til $\mathbf{F}$. Siden $\mathbf{F}$ er en sammensetning av mange funksjoner, så må vi bruke kjerneregelen.

Gitt målingsdata $\mathbf{x}_i \in \mathbb{R}^n$, $\mathbf{y}_i \in \mathbb{R}^n$, så skal vi finne matriser $W^{(i)}$ og vektorer $\mathbf{b}^{(i)}$ som gir et nevralt nett (4) med minimum total feil

$$f\left(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, ...\right) = \sum_{i=1}^{m} |\mathbf{F}_{W^{(1)},\mathbf{b}^{(1)},W^{(2)},\mathbf{b}^{(2)}...}(\mathbf{x}_i) - \mathbf{y}_i|^2, \tag{6}$$

i målingene. Når vi regner ut gradienten til denne vil vi skrive $\mathbf{F}_1 \circ \mathbf{F}_2$ for den sammensatte funksjonen $\mathbf{x} \to \mathbf{F}_1(\mathbf{F}_2(\mathbf{x}))$, og mer generelt $\mathbf{F}_1 \circ \cdots \circ \mathbf{F}_r$ for den sammensatte funksjonen $\mathbf{x} \to \mathbf{F}_1(\mathbf{F}_2(\cdots \mathbf{F}_r(\mathbf{x})))$. Denne notasjonen sparer oss for å skrive mange parenteser.

Den ytterste funksjonen i f er $\mathbf{x} \to |\mathbf{x} - \mathbf{y}|^2$. Denne regnet du ut gradienten til i Oppgave 4 (du må bruke Oppgave 1 i tilfelle det er flere målepunkter). La oss nå se på de andre lagene. For å anvende kjerneregelen riktig må vi sette opp punktene vi beregner hver Jacobimatrise i. For dette definerer vi

$$\begin{aligned} \mathbf{z}^{(r)} &= \mathbf{x} \\ \mathbf{y}^{(k)} &= W^{(k)}\mathbf{z}^{(k)} + \mathbf{b}^{(k)} && 1 \leq k \leq r \\ \mathbf{z}^{(k-1)} &= \sigma(\mathbf{y}^{(k)}) && 1 \leq k \leq r. \end{aligned} \tag{7}$$

Det første laget kan skrives

$$\mathbf{F}_r : \begin{pmatrix} W^{(1)} \\ \mathbf{b}^{(1)} \\ \vdots \\ W^{(r-1)} \\ \mathbf{b}^{(r-1)} \\ W^{(r)} \\ \mathbf{b}^{(r)} \end{pmatrix} \to \begin{pmatrix} W^{(1)} \\ \mathbf{b}^{(1)} \\ \vdots \\ W^{(r-1)} \\ \mathbf{b}^{(r-1)} \\ \sigma\left(W^{(r)}\mathbf{x} + \mathbf{b}^{(r)}\right) \end{pmatrix}, \tag{8}$$

Vi har her redefinert $\mathbf{F}_r$ i den forstand at variablene, med unntak av $W^{(r)}, \mathbf{b}^{(r)}$, blir sendt videre til neste lag. Vi fortsetter slik, slik at vi lag for lag eliminerer W og $\mathbf{b}$-variable. La oss finne Jacobimatrisen $\mathbf{F}'_r$, og la oss starte med de siste n komponentfunksjonene, som har formen

$$\mathbf{H}\left(W^{(1)}, \mathbf{b}^{(1)}, ..., W^{(r)}, \mathbf{b}^{(r)}\right) = \sigma(\mathbf{G}(W^{(r)}, \mathbf{b}^{(r)}))$$

med $\mathbf{G}(W^{(r)}, \mathbf{b}^{(r)}) = W^{(r)}\mathbf{x} + \mathbf{b}^{(r)}$. Bruker vi oppgave 5 i kombinasjon med kjerneregelen får vi en del nuller først (siden $\mathbf{H}$ ikke er avhengig av $W^{(1)}$, $\mathbf{b}^{(1)}$,..., $W^{(r-1)}$, $\mathbf{b}^{(r-1)}$), etterfulgt av

$$\sigma'(\mathbf{y}^{(r)}) \left(\begin{array}{cccc|cccc} \mathbf{x}^T & O & \cdots & O & 1 & 0 & \cdots & 0 \\ O & \mathbf{x}^T & \cdots & O & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ O & O & \cdots & \mathbf{x}^T & 0 & 0 & \cdots & 1 \end{array}\right)$$

Vi kaller denne matrisen for $U^{(r)}$. Denne er altså en $n \times n$-matrise ganget med en $n \times (n^2 + n)$-matrise. Aktiveringsfunksjonen regnet du ut Jacobimatrisen til i Oppgave 3. Siden de første komponentfunksjonene bare er identitetsavbildningen, så får vi at

$$\mathbf{F}'_r = \left(\begin{array}{c|c} I & O \\ \hline O & U^{(r)} \end{array}\right), \tag{9}$$

På samme måte skriver vi de andre lagene $\mathbf{F}_1$,...,$\mathbf{F}_{r-1}$ som

$$\mathbf{F}_k : \begin{pmatrix} W^{(1)} \\ \mathbf{b}^{(1)} \\ \vdots \\ W^{(k-1)} \\ \mathbf{b}^{(k-1)} \\ W^{(k)} \\ \mathbf{b}^{(k)} \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \to \begin{pmatrix} W^{(1)} \\ \mathbf{b}^{(1)} \\ \vdots \\ W^{(k-1)} \\ \mathbf{b}^{(k-1)} \\ \sigma\left(W^{(k)}\mathbf{x} + \mathbf{b}^{(k)}\right) \end{pmatrix} \tag{10}$$

Den eneste forskjellen fra tilfellet over er at det er n ekstra variable x_i her. De første søylene i $\mathbf{F}'_k$ får dermed samme form som over (med r byttet ut med k). Siden $W^{(i)}, \mathbf{b}^{(i)}$ ikke avhenger av x_i, og siden Jacobimatrisen til $\mathbf{x} \to \sigma\left(W^{(k)}\mathbf{x} + \mathbf{b}\right)$ er lik $V^{(k)} := \sigma'(\mathbf{y}^{(k)})W^{(k)}$, så får vi $\mathbf{F}'_k$ ved å legge til søylene $\begin{pmatrix} O \\ V^{(k)} \end{pmatrix}$ i (9), slik at vi får

$$\mathbf{F}'_k = \left(\begin{array}{c|c|c} I & O & O \\ \hline O & U^{(k)} & V^{(k)} \end{array}\right), \tag{11}$$

for $1 \leq k \leq r-1$.

Nå som vi har Jacobimatrisen for hvert enkelt lag så kan vi bruke kjerneregelen til å finne Jacobimatrisen for hele det nevrale nettet. Bruker vi først kjerneregelen for å finne Jacobimatrisen til $\mathbf{F}_{r-1} \circ \mathbf{F}_r$ får vi

$$\left(\begin{array}{c|c|c} I_1 & O & O \\ \hline O & U^{(r-1)} & V^{(r-1)} \end{array}\right) \left(\begin{array}{c|c|c} I_1 & O & O \\ \hline O & I_2 & O \\ \hline O & O & U^{(r)} \end{array}\right).$$

Her splittet vi opp identitetsmatrisen I i $\left(\begin{array}{c|c} I_1 & O \\ \hline O & I_2 \end{array}\right)$, for at radene skal matche med søylene i matrisen foran. Det er nå Oppgave 6 kommer inn. Den sier at det kommer inn en ekstra ikke-null blokk i siste rad. En U-blokk kommer først, og den andre blokken får en V foran seg:

$$\left(\begin{array}{c|c|c} I_1 & O & O \\ \hline O & U^{(r-1)} & V^{(r-1)}U^{(r)} \end{array}\right)$$

Jacobimatrisen til $\mathbf{F}_{r-2} \circ \mathbf{F}_{r-1} \circ \mathbf{F}_r$ blir så

$$\left(\begin{array}{c|c|c} I_1 & O & O \\ \hline O & U^{(r-2)} & V^{(r-2)} \end{array}\right) \left(\begin{array}{c|c|c|c} I_1 & O & O & O \\ \hline O & I_2 & O & O \\ \hline O & O & U^{(r-1)} & V^{(r-1)}U^{(r)} \end{array}\right),$$

som Oppgave 6 sier at blir

$$\left(\begin{array}{c|c|c|c} I_1 & O & O & O \\ \hline O & U^{(r-2)} & V^{(r-2)}U^{(r-1)} & V^{(r-2)}V^{(r-1)}U^{(r)} \end{array}\right)$$

(igjen en ekstra blokk: En U-blokk først, mens den siste blokken(e) får en V foran seg). Fortsetter vi dette, når alle lag tas med får vi at blokk k i siste rad til slutt blir

$$V^{(1)} \cdots V^{(k-1)} U^{(k)}.$$

Vi mangler nå bare å bake inn Jacobimatrisen(=gradienten) til den ytre funksjonen. På grunn av Oppgave 4 må vi regne ut

$$2(\mathbf{z}_0 - \mathbf{y})^T V^{(1)} \cdots V^{(k-1)} U^{(k)},$$

for alle k. Man kan regne ut dette ved hjelp av mange matrisemultiplikasjoner fra høyre mot venstre, men det er lurere å regne dette ut fra venstre mot høyre: Da får man en serie vektor-matrise multiplikasjoner i stedet, som er betydelig raskere å regne ut. Til og med den siste multiplikasjonen med (den større) matrisen $U^{(k)}$ kan regnes ut effektivt: Hvis vi fikk $\mathbf{a}^T = 2(\mathbf{z}_0 - \mathbf{y})^T V^{(1)} \cdots V^{(k-1)}$, så skal vi altså regne ut $\mathbf{a}^T U^{(k)}$. Sett først $\mathbf{c} = \mathbf{a}^T \sigma'(\mathbf{y}^{(k)})$. Hvis c_i er komponentene i $\mathbf{c}$, så er det greit å se at $\mathbf{c}^T U^{(k)}$ blir radvektoren med komponenter

$$c_1 z_1, \ldots, c_1 z_n, c_2 z_1, \ldots c_2 z_n, \cdots, c_n z_1, \ldots, c_n z_n, c_1, \ldots, c_n.$$

Husk nå at variablene var listet opp linje for linje i $W^{(k)}$ først (fra venstre mot høyre), etterfulgt av variablene i $\mathbf{b}^{(k)}$. La oss liste opp tilhørende variable:

$$\underbrace{c_1 z_1, \ldots, c_1 z_n}_{\frac{\partial f}{\partial w_{1j}^{(k)}}}, \underbrace{c_2 z_1, \ldots c_2 z_n}_{\frac{\partial f}{\partial w_{2j}^{(k)}}}, \cdots, \underbrace{c_n z_1, \ldots, c_n z_n}_{\frac{\partial f}{\partial w_{nj}^{(k)}}}, \underbrace{c_1, \ldots, c_n}_{\frac{\partial f}{\partial b_i^{(k)}}}.$$

Regner vi oss tilbake til matriseform får vi da

- matrisen ΔW med elementer $\Delta W_{ij} = \frac{\partial f}{\partial w_{ij}^{(k)}}$ er lik $\mathbf{c}\mathbf{z}^T$,
- vektoren $\Delta \mathbf{b}$ med elementer $\Delta \mathbf{b}_i = \frac{\partial f}{\partial b_i^{(k)}}$ er lik $\mathbf{c}$.

En Matlabimplementasjon

Det vi trenger å gjøre først er å regne ut vektorene $\mathbf{y}^{(k)}$ og $\mathbf{z}^{(k)}$ i (7). Dette kan gjøres med følgende kode, der matrisene $W^{(k)}$ lagres i en tredimensjonal $n \times n \times r$-matrise, og vektorene $\mathbf{b}^{(k)}$ i en $n \times r$-matrise.

```
function [z,y,z_0,val] = f_forward(W, b, xv, yv)
    nl = size(W,3);
    n = size(b,1);
    z = zeros(n,nl);
    y = zeros(n,nl);
    z(:,nl) = xv;
    y(:,nl) = W(:,:,nl)*z(:,nl) + b(:,nl);
    for k=(nl-1):(-1):1
        z(:,k)= sigma(y(:,k+1));
        y(:,k) = W(:,:,k)*z(:,k) + b(:,k);
    end
    z_0 = sigma(y(:,1));
    val = sum(abs(z_0-yv).^2);
```

```
end

function y=sigma(x)
    y = 1./(1+exp(-x));
end
```

Disse vektorene kan så brukes til å regne ut gradienten til det nevrale nettet. Følgende kode regner først ut verdien for det nevrale nettet. Deretter regner koden ut gradienten, og tar et et steg i retning av denne. Koden regner til slutt ut verdien i det nye punktet, og verifiserer at verdien faktisk er mindre.

```
nl = 10;
n = 10;
W = rand(n,n,nl);
b = rand(n, nl);
xv = rand(n,1);
yv = rand(n,1);

dW = zeros(size(W));
db = zeros(size(b));

[z,y,z_0,val0] = f_forward(W, b, xv, yv);
temp = 2*(z_0-yv).*sigmader(y(:,1));
for k=1:(nl-1)
    dW(:,:,k) = temp*z(:,k)';
    db(:,k) = temp;
    temp = (temp'*W(:,:,k).*sigmader(y(:,k+1))')';
end
dW(:,:,nl) = temp*z(:,r)'; db(:,nl) = temp;

[z,y,z_0,val] = f_forward(W-dW, b-db, xv, yv);
val0
val

function y = sigmader(x)
    y = sigma(x).*(1-sigma(x));
end
```

Legg merke til at startverdiene i $W^{(i)}$ og $\mathbf{b}^{(i)}$, samt målingsdataene, er randomgenerert.

En ting som er utelatt her er hvor langt vi skal følge gradienten i motsatt retning. Det å følge gradienten et lite stykke svarer til å følge en førsteordens tilnærming til funksjonen, men denne tilnærmingen er god bare når punktet vi står i. Innen maskinlæring bruker man en egen algoritme for å finne en god steglengde.

Koden over er også forenklet i den forstand at vi bruker bare et målingspar $(\mathbf{x}_1, \mathbf{y}_1)$. Oppgave 1 hjelper oss når vi har flere målinger, men vi må også oppdatere algoritmen slik at den effektivt kan takle flere målinger. Ganges matriser og vektorer sammen i samme rekkefølge så er dette mulig å få til.