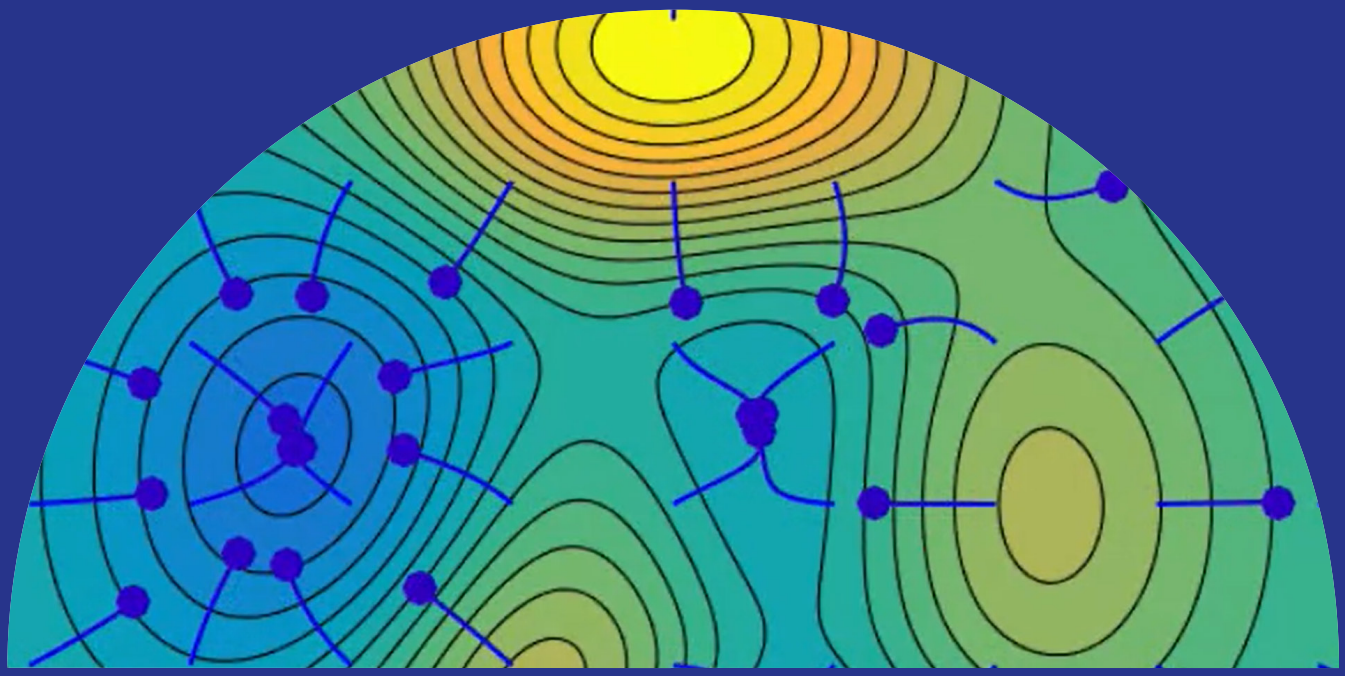
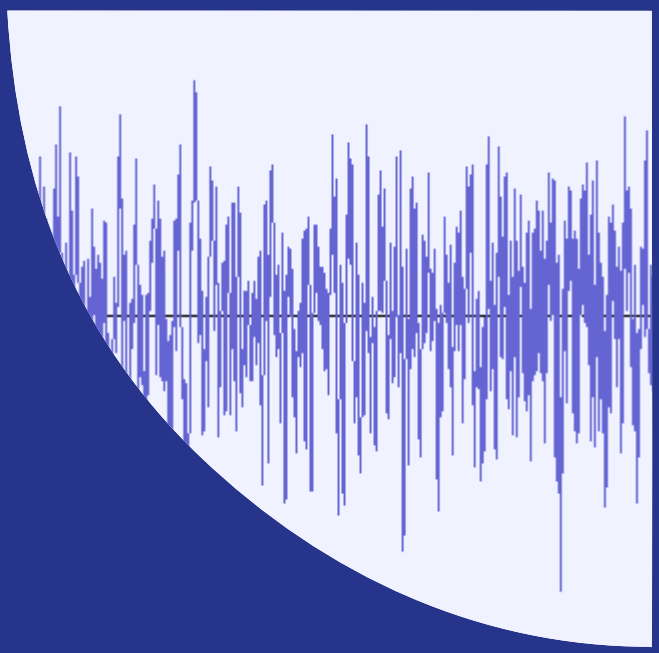
TRÆK VIRKSOMHEDERNE IND I UNDERVISNINGEN

Matematikken bag Machine Learning





BJØRN GRØN (RED), MADS PETER STEENSTRUP,
PER ROSENQVIST

matematik
i arbejde

Matematikken bag Machine Learning

Undervisnings og Projektmaterialer til filmen

Bjørn Grøn (red.), Per Rosenqvist og Mads Peter Steenstrup

Grafisk koncept: Ulla Korgaard, Designeriet.

Filmene og de tilhørende projektmaterialer hostes af Lindhardt og Ringhof / Praxis på websitet:
Træk virksomhederne ind i undervisningen (praxis.dk)

Undervisningsmaterialer til filmen *Matematikken bag Machine Learning.*
Forfattere: Bjørn Grøn (red.), Per Rosenqvist og Mads Peter Steenstrup

Omslagets illustration af Gradient Descent er lånt fra en gif på Wikipedias opslag om dette, se her: File:Gradient Descent in 2D.webm - Wikipedia

Projektet *Træk virksomhederne ind i undervisningen* er forankret på Rysensteen Gymnasium.
Film og tilhørende materialer er produceret med støtte fra Novo Nordisk Fonden

0. Indledning

I projektet *Træk virksomhederne ind i undervisningen* foreligger der 12 film med tilhørende undervisningsmaterialer til hver film. Dette hæfte er skrevet i tilknytning til filmen ***Matematikken bag Machine Learning.*** Enkelte steder opfordres til at se korte sekvenser i filmen, men hovedparten af materialet kan sagtens anvendes uafhængigt af filmen. Så man behøver ikke have filmen kørende, mens man arbejder med de matematiske problemer. Men det er en stor fordel for begrebsindlæringen, at man har set filmen, før man begynder at arbejde med emnet.

Materialet er opdelt i to hovedafsnit, en I.del, hvor vi (stort set) ikke inddrager differentialregning og i stor udstrækning anvender regneark. Og en II. del hvor differentialregning og programmering kommer mere i spil. I del kan man arbejde med på C-niveau og indledende B-niveau, eller sagt på en anden måde: i 1. og 2. g.

Vi har foretaget denne opdeling, fordi vores fornemmelse er, at der er et stort behov i ungdomsuddannelserne for at diskutere, præsentere og arbejde med problemstillinger indenfor kunstig intelligens, også selv om man ikke (endnu) har matematik på A.

Fokus er machine learning (ML), mens de store sprogmodeller (LLM's) er udeladt. Ikke fordi disse ikke er vigtige at forholde sig til og drøfte med elever, men det hører mere hjemme i almendannelse, end i matematik. LLM's har i langt højere grad en ML karakter af 'black boxes', og det udfordrer ikke i stor grad det matematiske ræsonnement.

Da vi samtidig har bevaret strukturen fra alle andre hæfter, hvor man skal kunne gå ind i et kapitel uden at have arbejdet med alt det foregående, så betyder det, at der er nogle overlap af emner.

Endelig har vi skilt nogle af de udfordrende emner indenfor funktioner af flere variable ud og lagt i III.del som oplæg til SRP. Men det er udfoldet i en sådan grad, at man i en 3. g godt kan tage emnet ind fx som valgfrit emne indenfor differentialregning.

Niveauerne er markeret med symbolerne:

		
9. klasse og 1.g	Slut 1.g og 2.g	3.g

Film og materialer er til fri download til anvendelse i undervisning og selvstudier. Bliver undervisningsmaterialet downloadet og dele af det kopieret, skal der angives kilde.

Forkortelsen HEM står for lærebogssystemet: *Hvad er matematik*? Fra HEM's website: Hvad er matematik - LRU.dk (praxis.dk) kan der bla. frit hentes mange projekter.

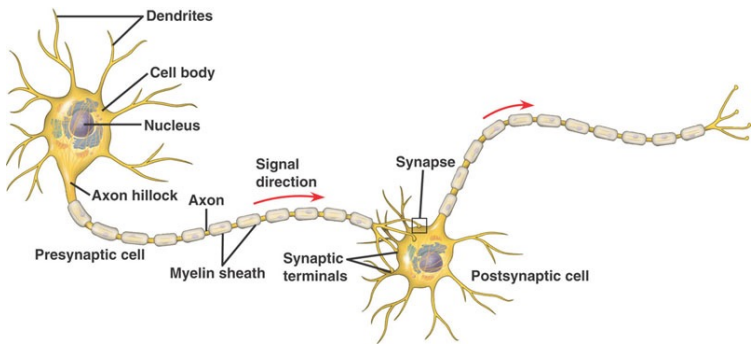
Hvor der er markeret *Link til 7* (og andre tal) angives, at her kan der hentes ekstra materialer ind. Disse tilgås via website for filmen. Find nummeret og klik.

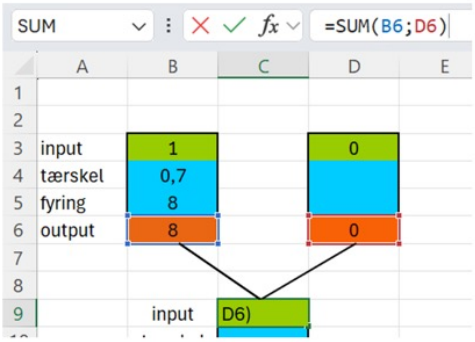
Vi vil meget gerne have feedback med kommentarer, fx hvis du finder fejlmog mngler noglem steder i materialet.

Film og undervisningsmaterialer produceres med støtte fra Novo Nordisk Fonden.

Indholdsfortegnelse

I.
Machine Learning på C og B-niveau





Machine Learning er som en del af AI/Kunstig Intelligens et ambitiøst forsøg på at efterligne, hvordan nerveceller gør det muligt for levende organismer at orientere sig i og om den verden den lever i, først og fremmest ved at lære af erfaringen. Så i machine learning bliver computere ikke programmeret til at løse specifikke problemer, men programmeret til at lære. I denne første del simuleres hjernecellers aktivitet og deres indbyrdes kommunikation ved hjælp af regneark. Celler i et regneark er naturligvis meget simple i forhold til biologiske neuroner, men vi kan alligevel udforske nogle grundlæggende ting om denne kommunikation, som fx hvad det vil sige at lære noget; eller hvordan et netværk undgår at bryde sammen under den massive informationsstrøm, som vi fx får ind gennem sanserne, og hvordan input sorteres, så kun visse dele sendes videre til andre celler.

1. Kunstige og naturlige neurale netværk

1.1 Programmering

Før vi giver os i kast med at forstå kunstig intelligens så lad os starte med at se på et utroligt simpelt problem. Kig på de 10 cifre bestående af 0'er og 1'er i linjen herunder:

1 0 1 0 1 1 0 0 1 0

Øvelse 1.1
Er der et sted i linjen ovenover, hvor der står to 1'er lige ved siden af hinanden? Kan du beskrive, hvordan du løser opgaven?

Opgaven er jo nærmest ingen opgave. Du kan omgående spotte, om det er tilfældet bare med et enkelt blik på linjen. Så det er svært at beskrive, hvad hjernen gør.

For at nærme os en forståelse af hvad kunstig intelligens er, og hvordan det virker, vil vi i de følgende afsnit diskutere

- Hvordan vil man løse opgaven ovenfor med traditionel programmering
- Hvordan virker en enkelt hjernecelle (også kaldet en neuron) i en naturlig dyrehjerne
- Hvordan er en biologisk hjerne opbygget
- Hvordan kan man i en computer efterligne de naturlige neuroner og den biologiske hjernes måde at virke på?

Det sidste fører os til de såkaldte kunstige neurale netværk, der løser problemet på en helt anden måde end en traditionelt kodet computer.

1.1.1 Konventionel programmering

Computeren har siden sin fremkomst ført sig frem som en imponerende talknuser, der ved hjælp af kun to tal - nemlig nul og et - har løst umådeligt mange problemer for ingeniører, fysikere, biologer og forskere kloden rundt. Selv om en computer kan klare imponerende beregninger på næsten ingen tid, er der dog ingen tvivl om vores vurdering af den traditionelle computer: Den gør nøjagtig som den får besked på. Den ræsonnerer ikke selvstændigt.

Et traditionelt program er bygget op af en lang række af linjer med instruktioner til, hvad computeren skal gøre. Den udfører derfor linjerne én for én i den rækkefølge de står, men indimellem kan en af instrukserne være en ordre om, at den skal gentage ordrerne i en eller flere linjer, eller at den skal hoppe til en linje længere nede i programmet og udføre den.

Eksempel Står der 2 et-taller ved siden af hinanden?
Følgende programstump viser hvordan en traditionelt programmeret computer kan løse vores første opgave, dvs. undersøge om der i en stribe på 10 cifre bestående af lutter 0'er og 1'er forekommer to 1'er ved siden af hinanden. Programmet er skrevet i et pseudo programmeringssprog, der ikke findes, men det giver et indtryk af, hvordan almindelige programmeringssprog som C-sharp, Python eller Pascal virker.

```
    Program er_der_11
For i = 1 to 10 do
    Read x(i)              *Indlæsning af den binære talstreng
Enddo

Y = 0                      *variabel der fortæller hvor mange 11- kombinationer der findes
For i = 1 to 9 do
    If ( x(i) = x(i+1) and x(i) = 1 ) then
        Write " der står to ettaller efter hinanden på plads nummer ", i , "og", i+1
        Y = Y+1
    Enddif
Enddo

    If Y = 0 then
        Write " Der stod ikke to ettaller efter hinanden noget sted i denne serie"
    Else
        Write " Der var",Y," pladser med to ettaller lige efter hinanden"
    Endif

  Stop
```

```
x = [1,0,1,0,1,1,0,0,1,0]
y = 0
i = 0
while i < 8:
    if x[i] == x[i+1] and x[i] == 1:
        y = y + 1
    i = i + 1
print('der er',y,'steder med to ettaller lige efter hinanden.')
```

Øvelse 1.2 Hvordan virker programmet?
Prøv at forklare hvordan programmet virker.

Det er ovenfor skrevet i Python og kan køres online på https://www.online-python.com/ eller I Jupyter filen, du kan hente *her* (link til filen 1a).

NB. Hvis I kopierer fra PDF, vil indryk ikke komme med, sørg selv for at lave alle indryk i linjerne 5 til 7.

Kør programmet og se, om det faktisk giver det I forventer.

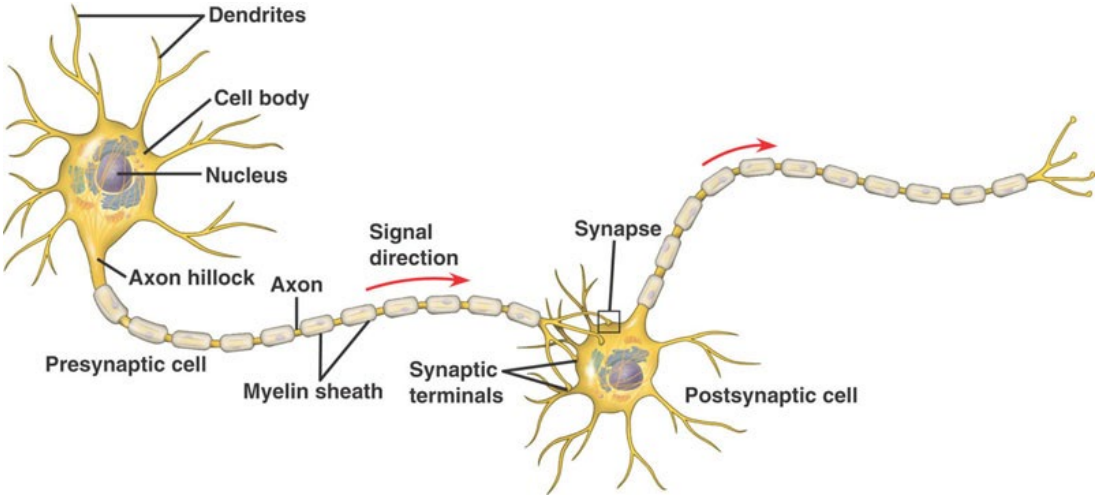
Lav om i den binære kode og se om programmet virker, lav også om i linje 4, hvis I ændrer på antallet af cifre.

1.2 Biologiske neuroner og hjerner (naturlige neurale netværk)

Et naturligt neuralt netværk er det netværk af celler – neuroner – som en dyrehjerne (herunder også menneskers) består af. En hjernes virkemåde er på én gang både nogenlunde simpel og samtidig umådeligt kompliceret. Det simple består i, at de enkelte neuroners virkemåde er forholdsvis let at forstå, ud fra ganske elementære begreber fra el-lære og (bio)kemi. Det komplicerede ligger i hjernens enorme kompleksitet og de fænomener, der opstår, fordi den er sammensat af et enormt antal neuroner.

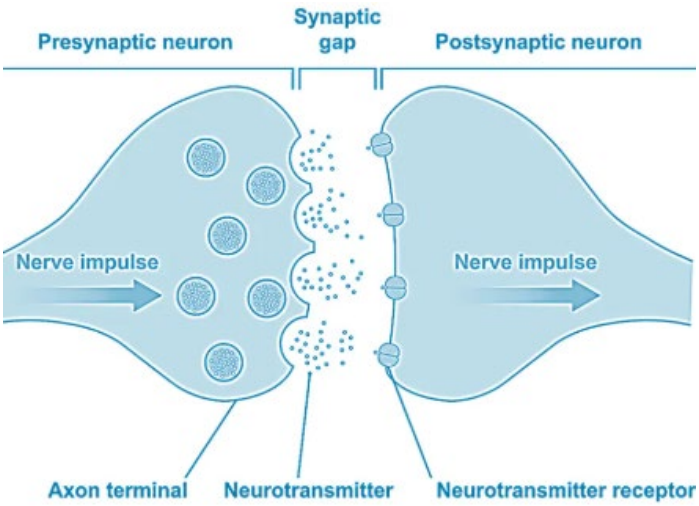
1.2.1 Den biologiske neuron

Lad os starte med at kigge på et par neuroner der er i kontakt med hinanden:



De tilsvarende vigtigste danske betegnelser er: celle, cellekerne, dendritter, axoner og synapser

Som figuren viser, består en biologisk neuron af et cellelegeme med to slags udløbere: dendritter og axoner. Neuronen kan udsende elektriske (elektrokemiske) signaler til de andre neuroner, den er i forbindelse med, gennem **axoner**. Axonerne kan være meget lange, over 1 meter, fx fra hjernen eller rygmarven ud til en muskel eller kirtel et sted i kroppen. Hvis et axon er i forbindelse med en anden neuron findes der et kontaktpunkt hvor signalet fra axonet kan overføres til den næste neurons ”følere”. Kontaktpunktet kaldes *synapsegabet* se figuren her



”Følerne” på den næste neuron, der modtager signalet, kaldes dendritter. Dendritter er korte udvækster tæt på "deres" neurons kerne.

Axoner *udsender* signaler, dendritter *modtager* signaler.

For enden af både dendritter og axoner sidder altså synapser og axonet kan forgrene sig i spidsen, så hver neuron kan have kontakt til mange andre neuroner i det neurale netværk. Størrelsen af synapsegabet dvs. afstanden mellem synapsen og den næste neurons synapse er typisk 0,1 mikrometer dvs. 10^{-7} m, og det tager det elektrokemiske signal ca. 0,1 millisekund at passere det synapsegabet.

1.2.2 Kommunikation mellem neuroner

Neuronerne kommunikerer som sagt med hinanden ved at udsende elektrokemiske impulser. Ved en neurons ***aktivitetsniveau*** forstås den hyppighed, hvormed den udsender disse impulser. Naturlige neuroner har tendens til at have enten meget høj eller meget lav aktivitet, men sjældent en aktivitet, der ligger midt imellem. Man siger, at en neuron "fyrer"[1]. hvis den er meget aktiv, eller at neuronen "ikke fyrer", hvis den er forholdsvis inaktiv.

Hvis man skulle beskrive hele netværkets aktivitetstilstand, kunne man i princippet gøre dette ved at lave en (meget lang) liste over, hvilke neuroner der fyrer, og hvilke der ikke fyrer. Efter kort tid vil dette billede selvfølgelig være ændret radikalt, fordi hjernen hurtigt tænker på noget nyt, og aktivitetsmønstret kan altså opfattes som en slags snapshot af netværkets eller hjernens tilstand til et bestemt tidspunkt. Hvis man tildeler de neuroner, der fyrer, et 1-tal og de neuroner, der ikke fyrer, et 0, kunne man altså i princippet beskrive netværkets fyringsmønster ved en lang række nuller og et-taller!

Øvelse 1.3 Opsummér hvad du har lært

Prøv at forklare, hvordan hjernen er opbygget, og hvordan neuronerne virker

Øvelse 1.4 Antallet af mulige tilstande for en hjerne

Vi kigger på et neuralt netværk der har 10 neuroner (et lillebitte udsnit af en hjerne)

a) Hvis hver neuron kan have tilstanden 0 = ikke aktiv eller 1 = aktiv, hvor mange mulige tilstande kan netværket så være i?

En primitiv orm har ca. 300 neuroner

b) Hvor mange mulige fyringsmønstre er der for ormens hjerne?

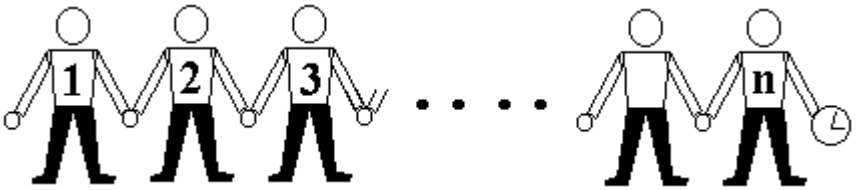
Når hjernen, også hos det vi kalder primitive væsener, er så utrolig effektiv, skyldes det fundamentalt set to ting: Antal og hastighed. *Antal* neuroner og deres indbyrdes forbindelse, og *hastigheden* af processeringen af signaler.

Kan vi på nogen måde visualisere, hvor hurtigt neuroner kommunikerer? Vi kan ikke direkte måle på vores hjerner, men vi kan alligevel måle nerveimpulsernes hastighed i et snedigt eksperiment:

[1] Den proces, vi under ét betegner med begrebet "neutronen fyrer", er i sig selv ganske kompleks og modelleres matematisk med koblede differentialligninger. Teorien kan findes behandlet i Bjørn Grøn og Torsten Tranum Møller: *Matematisk modellering af signaler i nerve- og muskelceller,* et projektmateriale skrevet i tilknytning til en film om og med Susanne Ditlevsen: *Statistiske metoder i hverdagsliv og i neurovidenskab*. Kan hentes her: https://lru.praxis.dk/Lru/microsites/10danskematematikere/index.html

Eksperiment: Måling af nerveimpulsernes hastighed i vores arme

Opstilling i første forsøgsgang: Alle elever stiller sig op i en lang række med hinanden i hænderne som vist på figuren:



Forsøget kører over 2 forsøgsgange á ca. 5 runder, idet der i første runde er fem forsøgspersoner i rækken, i anden runde er der 10 osv. op til, vi har alle i klassen med. Hvis dette giver under fem runder, kan man nøjes med at øge antallet af deltagere med 3 eller 4 for hver runde.

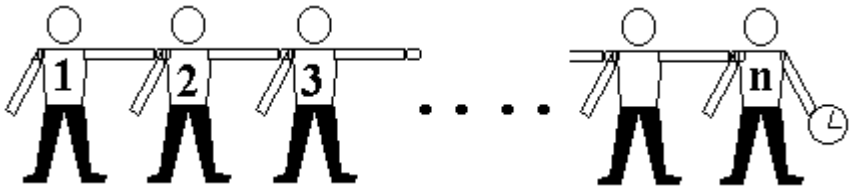
Personen længst til højre har et stopur i hånden. **Meget vigtigt:** Alle forsøgspersonerne lukker nu øjnene. Personen med stopuret tæller højt til 3 og på "tre" giver personen længst til venstre nr. 1 nu sin sidemand nr. 2 et mærkbart klem i hånden samtidig med, at personen med stopuret starter dette. Lige så snart forsøgspersonerne mærker et klem fra sidemanden i den ene hånd, sender man så hurtigt som muligt signalet videre til den næste sidemand. Når signalet når den sidste mand i rækken, stopper han/hun stopuret.

Øvelse 1.5

Der er n personer, "svarende til" n neuroner. Overvej, at vi nu har målt, hvor lang tid det tager for et nervesignal at gå gennem $2n$ - 1 arme og blive behandlet i n hjerner.

Tiden og *antallet af deltagere* i runden noteres ned i et skema, og man fortsætter til næste runde.

Opstilling i anden forsøgsgang :



Forsøget er magen til det forrige bortset fra, at den hånd, man sender beskeden videre med til sin sidemand, nu er placeret på dennes skulder. Det er meget vigtigt, at der i hver runde er præcis det samme antal deltagere som i den tilsvarende runde i forrige forsøgsgang. Tiderne noteres igen ned i et skema. Når de to forsøgsserier er slut, laver man en ny række hvor de to tidsserier trækkes fra hinanden (brug regneark). Tidsforskellen Δt må skyldes, at signalet ikke skal igennem en hel masse arme i andet forsøg, så med andre ord fortæller Δt, hvor lang tid det tager for nervesignalet at gå igennem de arme, der var med i første forsøg, men som ikke blev brugt i andet forsøg! Under Δt -rækken laves en række med dette antal.

Øvelse 1.6

Argumentér for, at hvis der er *n* personer med i forsøget, så er det *n* –1 arme, der tages tid på.

Indtegn antallet af arme som funktion af Δt i et regneark. Find en gennemsnitsarmlængde og brug den til at beregne den samlede armlængde, der svarer til tiderne Δt. Indtegn også dette i regneark som funktion af Δt. Beregn hastigheden af nervesignalerne ved at lave lineær regression.

Øvelse 1.7 Hjernens processeringstid
Hvordan kan man bruge forsøgets data til at estimere, hvor lang tid det tager for hjernen at modtage et signal, processere det og give signal til armen om, at den skal sende et klem videre?

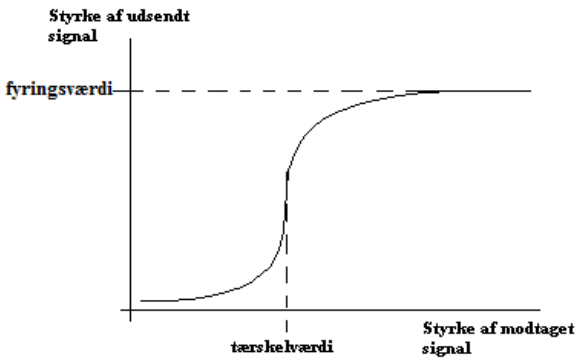
1.2.3 Tærskelværdi og fyringsværdi

Svagheden ved forsøget, vi gennemførte, er naturligvis, at her er bevidstheden ind over – vi reagerer med en beslutning om at levere et klem videre. En meget stor del af vores nervesystem handler uden vores bevidsthed er ind over. Og det virker umiddelbart sandsynligt, at hastigheden, hvormed neuroner kommunikerer, er lidt mindre, når bevidstheden involveres.

Den enkelte neuron bestemmer normalt ikke selv, om den vil fyre eller ej. Man kan - måske lidt dramatisk - sige, at den enkelte neuron ikke har nogen fri vilje. Det, der afgør om en neuron sender de impulser videre, som den modtager gennem dendritterne, er simpelthen hvor kraftigt, det signal er, den modtager. Enhver neuron har nemlig en vis tærskelværdi eller tolerancetærskel, der afgør, om den viderebringer signalet.

Hvis alle de milliarder og trilliarder af signaler, vi modtager, blev sendt videre umiddelbart, så ville systemet bryde sammen. Der må ske en sortering, og i neuronerne sker der en første grovsortering ved hjælp af *tærskelværdierne*: Hvis styrken af det signal, der modtages, overstiger tærskelværdien, bliver neuronen provokeret til selv at blive aktiv, og "fyre" signalet videre til de neuroner, som dens axoner er i forbindelse med, hvis ikke lægger den signalet dødt. Tærskelværdien og **fyringsværdien,** dvs. styrken af det signal den enkelte neuron udsender under fyring, afhænger af, i hvor god "træning" neuronen er.

Øvelse 1.8 Grafisk model over tærskelværdi og fyringsværdi
Betragt følgende graf, og sæt dig ind i, hvilke variable der er i spil. Forklar med dine egne ord, hvad det er grafen illustrerer



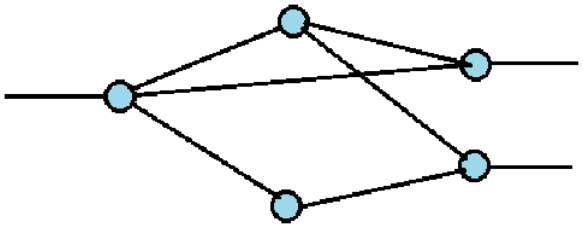
1.3 Matematisk model af neuroner og neurale netværk

En matematisk model for et neuralt netværk kaldes også for et kunstigt neuralt netværk og kan udformes på flere forskellige måder. Man skal tage stilling til to ting:

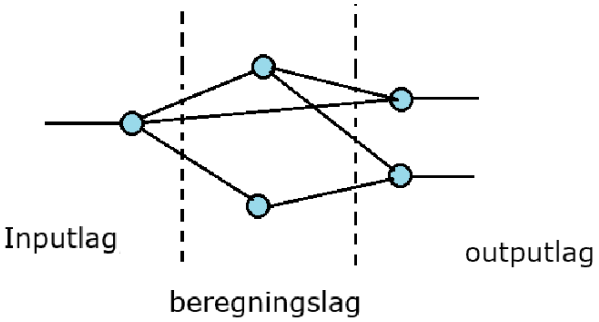
a) Hvor mange neuroner skal der være, og hvilke neuroner skal være i forbindelse med hinanden?

b) Hvordan skal vi modellere den enkelte neurons virkemåde?

Lad os først starte med at overveje hvordan vi kan beskrive neuronernes placering, og hvordan de er forbundet med hinanden i netværket. Et sådant netværk kaldes for en *graf* og består altså af en række punkter, der hver svarer til kernen i en neuron og så linjer, der forbinder punkterne, og som fortæller, hvilke neuroner der skal kunne fyre til hvilke andre:



En sådan graf-oversigt over neuronerne giver os netværkets *arkitektur*. På figuren har vi ikke vist hvilken retning neuronerne fyrer, men vi underforstår, at netværket kun kan sende signaler fra venstre mod højre. Med den vedtægt kan vi se, at netværket ovenfor har 3 *lag*: Et inputlag (også kaldet et sanselag), et skjult beregningslag og et outputlag:



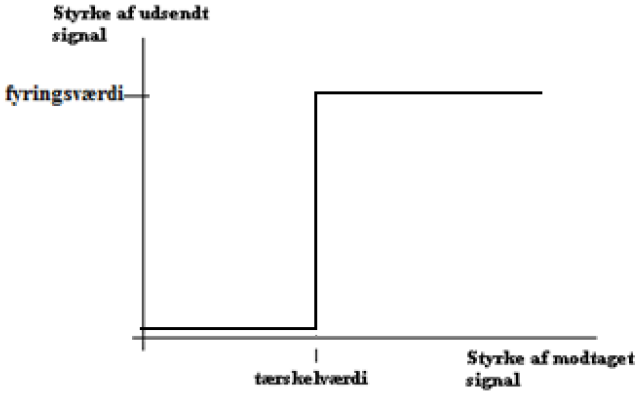
Det er klart, at der bare med nogle få neuroner bliver ufatteligt mange muligheder for, hvordan arkitekturen dvs. kombinationen af lag, neuroner og forbindelser kan skrues sammen.

Øvelse 1.9 Mulige netværk

a) Tegn så mange mulige arkitekturer du kan finde på for netværk med: 1 neuron, 2 neuroner, 3 neuroner og 4 neuroner.

b) Konkurrence: hvor mange forskellige arkitekturer kan i finde på for et neuralt netværk med 5 neuroner?

Nu mangler vi så bare at finde ud af, hvad der skal til for at modellere den enkelte neuron. Hvis vi kigger på figuren i øvelse 1.8, kan vi se, at hvis vi kunne finde en funktion, hvis graf har den rigtige facon, så ville vi have en god model. Der findes flere forskellige klasser af funktioner - såkaldte *sigmoidfunktioner* - der har grafer, der ligner ovenstående graf, men lad os først prøve at se på den simpleste model:



Funktionen vist på figuren er en Heaviside stepfunktion der har gaffelforskriften:

$$f(x)=\begin{cases} y_1 & \text{for } x \geq x_1 \\ 0 & \text{for } x < x_1 \end{cases}$$

Her er y_1 fyringsværdien som opnås når inputtet x er større end *tærskelværdien* x_1 og for alle andre inputværdier er funktionsværdien bare 0.

I stedet for, at vi for hver neuron skal opstille en hel funktion, kan vi bare nøjes med at angive to tal (x_1, y_1), dels hvad tærskelværdien skal være, og dels hvad fyringsstyrken eller aktivitetsniveauet er.

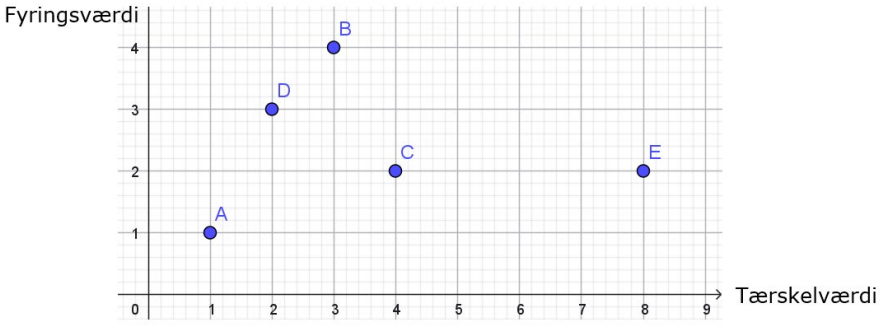
På den måde kan vi tænke på den enkelte neuron som et punkt i et koordinatsystem med koordinater (x_1, y_1), og hvis vi har et helt neuralt netværk, kan vi beskrive det som en række punkter i koordinatsystemet.

Øvelse 1.10 Grafisk fremstilling af forskellige neuroner

I koordinatsystemet her er vist 5 neuroners tærskel og fyringsværdier:

a) Hvilken neuron har den største fyringsværdi?

b) Hvilken neuron har den laveste tærskelværdi?



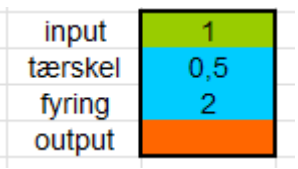
1.4 Neuroners aktivitetsniveau

Neuroners aktivitetsniveau

Når vi så skal afgøre om den enkelte neuron fyrer eller er inaktiv, ser vi på *det samlede input* ind i neuronen dvs. summen af alle de input neuronen modtager: Hvis det samlede input i neuronen (x_1, y_1) er større end neuronens tærskelværdi x_1 fyrer den med værdien y_1, og ellers er den inaktiv (med fyringsværdi på 0).

1.4.1 Modellering med regneark

Gennem den enkelte neuron løber der en strøm af information, og vi opstiller i et regneark en model af én neuron med en figur som denne. Her angiver:



- input, det som neuronen modtager via en af sine dendritter
- tærskel, som er første trin i behandlingen af input, en slags "Stop / Go"
- fyring, som er andet trin, hvor der fyres, hvis første trin sagde "Go", ellers ikke.
- Output er så *værdien* af fyringen, hvis vel at mærke der bliver fyret.

Tærskel og Fyring repræsenterer tilsammen neuronens "beregning" eller processering af det modtagne signal.

Neuronen, der er angivet her som eksempel, vil sende signalet videre med en styrke på 2, fordi det modtagne signal har en styrke, der overstiger tærskelværdien. Så i dens outputcelle skal skrives 2.

På figuren nedenfor er vist en række neuroner der er tegnet i et regneark. På grund af regnearkets opbygning, så går signaler her oppefra og ned (hvor vi tidligere fastlagde, de bevægede sig fra venstre mod højre)

- Første lag kaldes inputlaget.
- Andet lag kaldes beregningslaget – og også ofte for "de skjulte lag". Her vil normalt være en række lag efter hinanden.
- Tredje lag kaldes for outputlaget

Øvelse 1.11 Signaler, der sendes gennem netværk

For hver neuron er vist input på neuronerne i inputlaget. Der er også vist tærskelværdien samt den fyringsværdi de sender signalet videre med, hvis de bliver aktive. Neuronen herover vil f.eks. sende signal videre med en styrke på 2 (dens fyringsværdi) fordi dens input (1) er større end dens tærskelværdi (0,5).

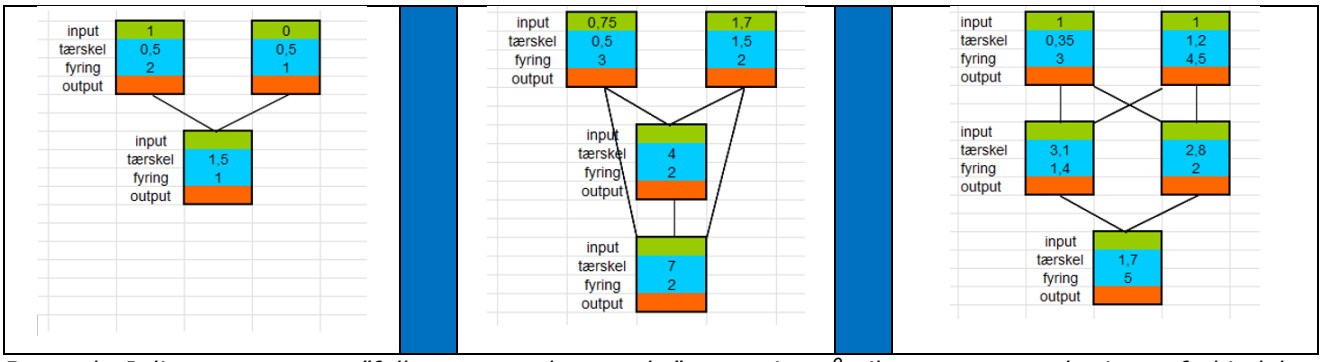
For hvert af de tre neurale netværk skal du:

a) Først udfylde de orange outputfelter i inputlaget (første lag).

b) Dernæst beregne de grønne inputfelter i 2. lag, og:

c) Afgøre, om neuronerne i 2. lag fyrer og udfyld det/de orange outputfelter i 2- lag.

d) Beregne de grønne inputfelter i 3. lag (for de to sidste netværk) og afgøre om neuronen i 3. lag fyrer.

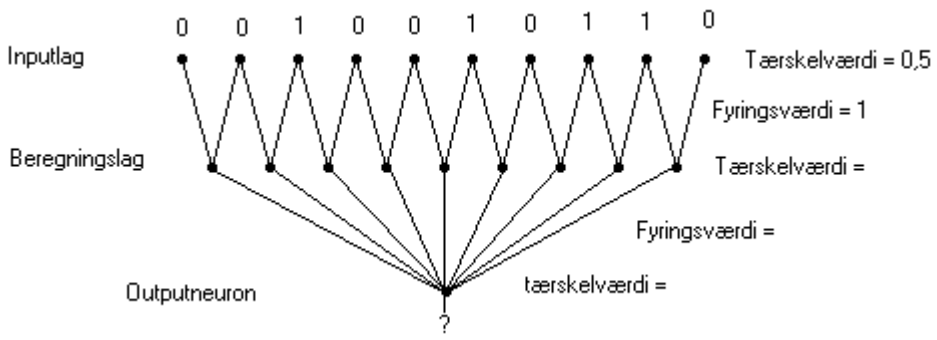


Bemærk: I diagrammer over "fully connected networks", som vi også vil se senere, er der ingen forbindelse der springer et lag over, som det sker i nr. 2 ovenfor. Men det vil ske i praksis, hvis et signal rammer en neuron med tærskelværdi på 0 og fyringsværdi = input værdi

1.4.2 Løsning af problemer ved hjælp af et kunstigt neuralt netværk

Lad os prøve at se på et neuralt netværk, der kan løse problemet, som vi tidligere løste med et konventionelt computerprogram, nemlig eksemplet: **Står der 2 ettaller ved siden af hinanden?**

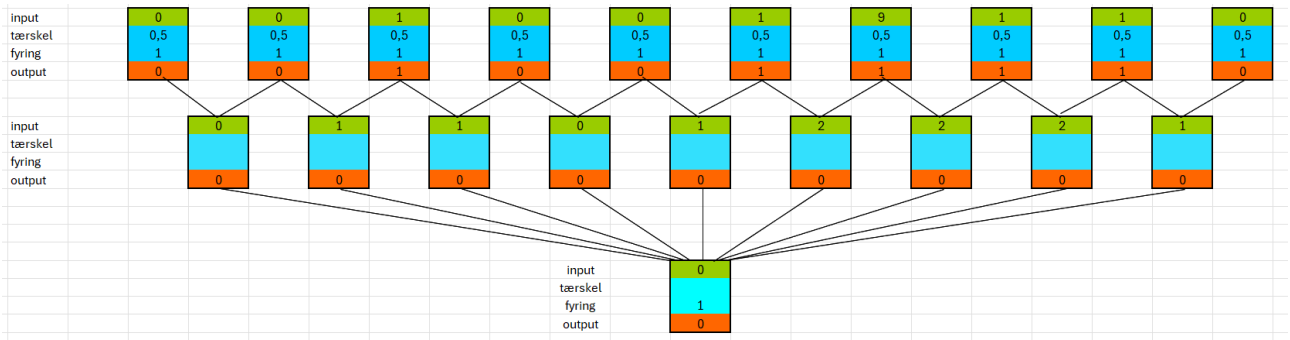
Vi starter med at sende vores række af 0'er og 1'er ind i et inputlag med 10 neuroner:



I dette netværk er det underforstået, at alle signalerne går nedad. Tærskelværdien for alle inputneuronerne sætter vi til $x_1 = 0{,}5$ så de fyrer altså, med værdien $y_1 = 1$, hvis de modtager et ettal og ellers forbliver de inaktive. På figuren er der ikke angivet tærskelværdi for det næste neuronlag, ligesom der heller ikke er angivet hverken fyringsværdi for laget eller tærskelværdi for den sidste neuron.

Øvelse 1.12 Er der to et-taller ved siden af hinanden.

Hent filen 1b (*Øvelse 1.10 - Er der to ettaller ved siden af hinanden.xls*) her. I filen er dette netværk med neuroner og deres virkemåde indtastet i et regneark:



Du skal selv eksperimentere med tærskel- og fyringsværdier i de to næste lag for at løse følgende opgaver:

a) Vælg en fælles tærskelværdi for beregningslaget (det midterste lag i regnearket) således, at beregningslagets neuroner fyrer hvis de modtager signal fra **to** 1-taller, men ikke hvis de kun modtager signal fra ét (eller ingen) et-tal

b) Vælg en fyringsværdi for beregningslagets neuroner

c) Vælg en tærskelværdi for output-neuronen i nederste lag, således at den fyrer, hvis der findes to ettaller ved siden af hinanden oppe i inputlaget.

d) Tjek at dit netværk løser problemet, hvis du også indtaster andre sekvenser af 0'er og 1'er i inputlaget.

e) Kunne opgaven være løst med andre tærskel- og fyringsværdier?

f) Kan du finde et netværk med en anden arkitektur der løser det samme problem?

Træning af netværk
Når vi har valgt tærskelværdier og fyringsværdier så netværket fungerer korrekt, siger vi, at vi har *trænet* netværket.

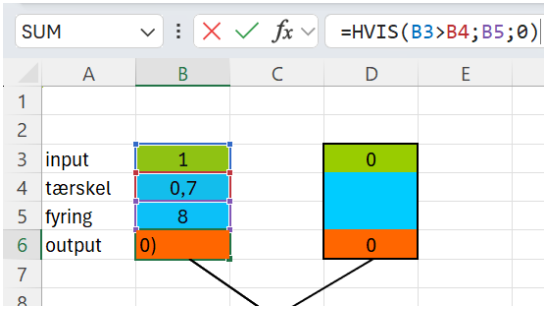
Øvelse 1.13 Hvordan neuroners aktivitet kan beskrives i et regneark - 1

På figuren her er vist et udsnit af et regneark. Vi fokuserer på neuronen til venstre.

I formellinjen er der vist den beregning, regnearket vil foretage i outputcellen B6, når der tastes enter.

a) Forklar hvordan outputberegningen afspejler neuronens stepfunktion

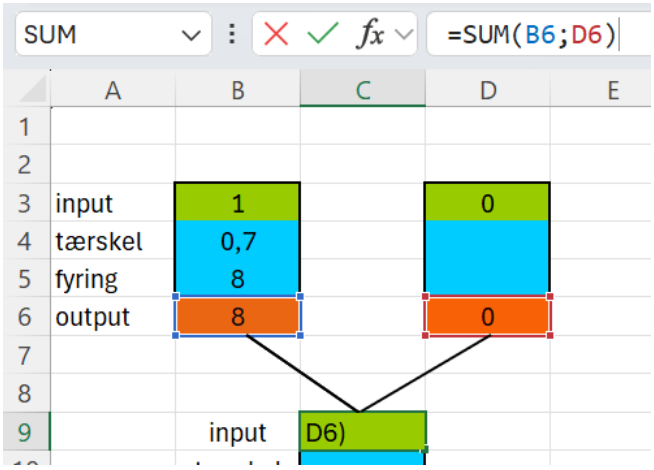
b) Hvad bliver neuronens output når der tastes enter?



Øvelse 1.14. Hvordan neuroners aktivitet kan beskrives i et regneark - 2

Her ser vi et større udsnit af netværket:

c) Forklar ud fra formellinjen hvordan inputfeltet i C9 beregnes ud fra de to øverste neuroners aktivitet



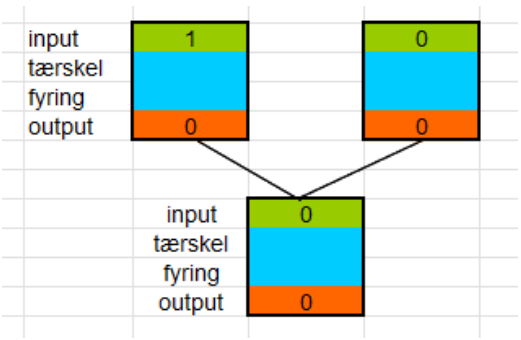
1.5 Løsning af problemer vha. neurale netværk i Excel

Som tidligere nævnt kalder vi processen, hvor vi finder tærskel- og fyringsværdier for et neuralt netværk, så det løser et problem korrekt, at "træne netværket".

I følgende opgaver skal du igen arbejde i et regneark, og for hver neuron anvende de koder, som vi lærte i øvelserne 1.13 og 1.14. Regnearket er stadig konstrueret så der "regnes nedad", og vi antager, at hver neuron i inputlaget (sanselaget) modtager enten et 0 eller 1 tal i signalværdi. De netværk, du skal træne ligger i en excelfil, du kan hente *her* (link til filen 1b- *Opgaver med Excel neuroner.xls*). Du skal træne netværkene til at løse følgende opgaver:

Opgave 1 Træning i at anvende logisk "eller" – 1. version

Du har et netværk med følgende arkitektur:



Netværket skal have den egenskab, at den nederste neuron fyrer, hvis der er mindst et 1-tal blandt de to neuroner i øverste række.

Altså neuronen i nederste række skal fyre, hvis det ene eller det andet eller begge input er 1-taller, men ikke hvis begge input er 0.

Opgave 2 Træning i at anvende logisk "eller" – 2. version

Træn et netværk med samme arkitektur som i opgave 1. Netværket skal have den egenskab, at den nederste neuron fyrer hvis der er et 0 i input hos mindst en af de to neuroner i inputlaget. (*Vink: fyringsværdierne må gerne være negative*)

Opgave 3 Træning i at anvende logisk "og"

Et netværk har samme arkitektur som i opgave 1. "Træn" netværket så den nederste neuron kun fyrer hvis begge de øverste neuroner et 1-tal som input.

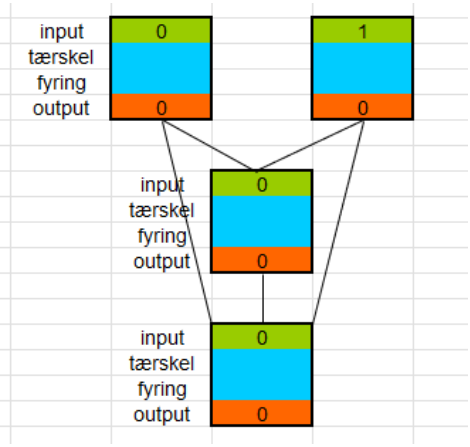
Opgave 4 Træning i at anvende logisk "medfører"

Træn et netværk med samme arkitektur som i opgave 1. Netværket skal have den egenskab, at den nederste neuron fyrer hvis der IKKE forekommer kombinationen 10 i de to neuroner i inputlaget. (*Vink: fyringsværdierne må gerne være negative*)

Opgave 5 Træning i at anvende logisk XOR

Træn et netværk med følgende arkitektur:

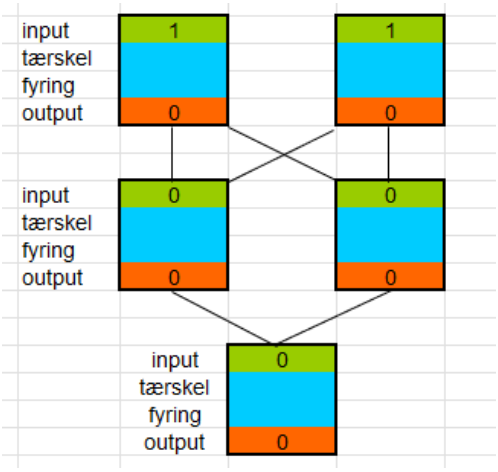
Netværket skal have den egenskab, at nederste neuron skal fyre hvis der er PRÆCIS et ettal blandt inputs i de to neuroner i inputlaget



Opgave 6 Træning i at anvende enten 0'er eller 1'er

Træn et netværk med følgende arkitektur:

Netværket skal have den egenskab, at den nederste neuron fyrer, hvis de to neuroner i øverste lag modtager samme signal.



2. Sigmoidfunktioner

Netværk af biologiske neuroner har gennem evolutionen trænet sig op til at kunne håndtere situationer som:

- genkende mønstre
- afkode sammenhænge

Det (og meget mere) har været forudsætningen for at overleve.

Vi vender tilbage til mønstergenkendelse, men vil i det følgende opholde os lidt ved dette begreb, *at afkode sammenhænge*. Bestemte kemiske spor og koncentrationen heraf fortæller historier om føde og om fare. Den mest simple sammenhæng er en proportionalitet, og en anelse mere kompleks så er det lineært. Det er jo også det første matematiske værktøj vi griber til, når vi søger at afkode sammenhænge.

Men som Jes Frellsen fortæller i filmen, så er verden ikke lineær. På korte distancer og over korte tidsrum kan det være rigtig gode tilnærmelser, men for at få fat i de rigtige sammenhænge, skal der et twist til. Vi ved, at sammensætningen af to funktioner ofte kan give noget ret kompliceret, men sammensætningen af to lineære giver en ny tredje lineær funktion.

Øvelse 2.1 Sammensætning af to lineære funktioner giver noget lineært
Vis dette! Dvs vis, at hvis $f(x) = a \cdot x + b$ og $g(x) = c \cdot x + d$, så er $f(g(x))$ også en lineær funktion.

Skal vi kunne fange noget mere komplekst, skal der et ikke lineært element ind over. Og i den matematiske værktøjskasse byder en særlig klase af funktioner sig til, *sigmoid-funktionerne*.

En sigmoid-funktion er en matematisk funktion, hvis graf er en S-formet kurve, som stiger gradvist og flader ud i toppen og i bunden af kurven. I hovedafsnit II vender vi tilbage til "Sigmoid classic", og undersøger nogle af disse med brug af differentialregning.

Her går vi videre ind i sigmoidfunktionernes verden, via en funktion, der godt nok ikke lever op til den præcise definition, men som både er simpel og har nogle af de egenskaber, vi efterspørger.

2.1 Relu-funktionen

Definition: Relu-funktionen

Relu-funktionen kan defineres som: $r(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{for } x<0 \end{cases}$

Eller den kan defineres som: $r(x) = \max(0, x)$

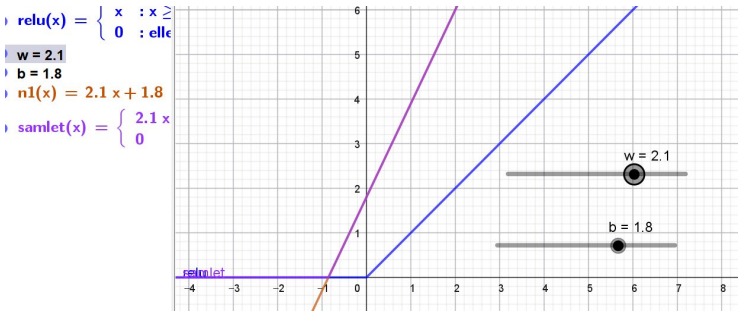
Øvelse 2.8 De to definitioner er ækvivalente
Argumenter for, at de to funktionsudtryk i definitionen giver samme funktion

Øvelse 2.9 Grafen for relu og for relu sammensat med en lineær.

a) Tegn selv grafen for relufunktionen.

b) Sammensæt relu-funktionen med den lineære $f(x) = w \cdot x + b$ og tegn grafen for den samlede funktion $samlet(x) = relu(w \cdot x + b)$ med skydere for w og b

c) Hvilke af parametrene w og b styrer henholdsvis *tærskelværdien* og *fyringsværdien*?



I filmen nævner Jes Frellsen, at man i et dybt neuralt netværk godt kan klare sig med simple relufunktioner som ikke-lineariteter, hvis bare netværket er stort nok. Vi antyder sammenhængen i næste øvelse.

Øvelse 2.10 Sammensætning af to relu'er giver en sigmoid-lignende funktion!

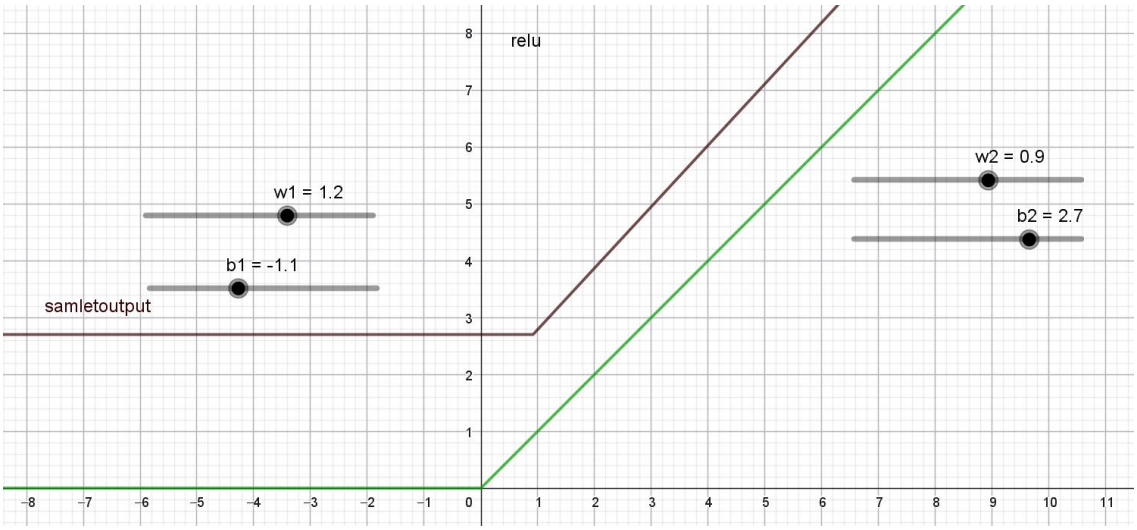
Hent filen ned sammensætning af to relu-funktioner *her* (link til *2a samensætning af to relu giver sigmoid simpel.ggb*.)

Vi har givet to lineære funktioner: $f_1(x) = w_1 \cdot x + b_1$ og $f_2(x) = w_2 \cdot x + b_2$.

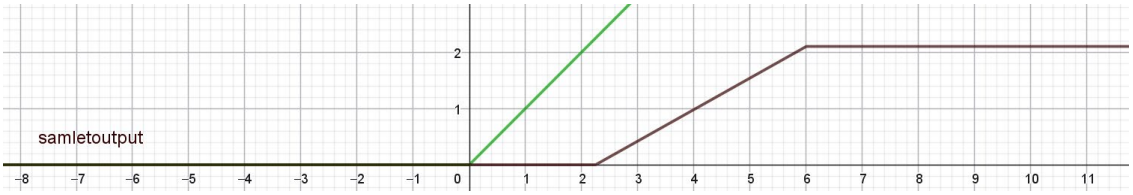
Sammensæt nu de to lineære med to standardreluer efter hinanden, således:

$$samletoutput(x) = r(f_2(r(f_1(x)))) = r\left(w_2 \cdot r\left(w_1 x + b_1\right) + b_2\right)$$

Grafen for $samletoutput(x)$, hvor parametrene er angivet med skydere



a) Leg med tærskel og fyringsværdierne w_1, b_1, w_2, b_2, til du får sammensætningen til at ligne en sigmoidfunktion dvs. se således ud:



b) Hvilke(n) parameter styrer tærskelværdien for den fremkomne sigmoidfunktion?

c) Hvilke(n) parameter styrer fyringsværdien for den fremkomne sigmoidfunktion?

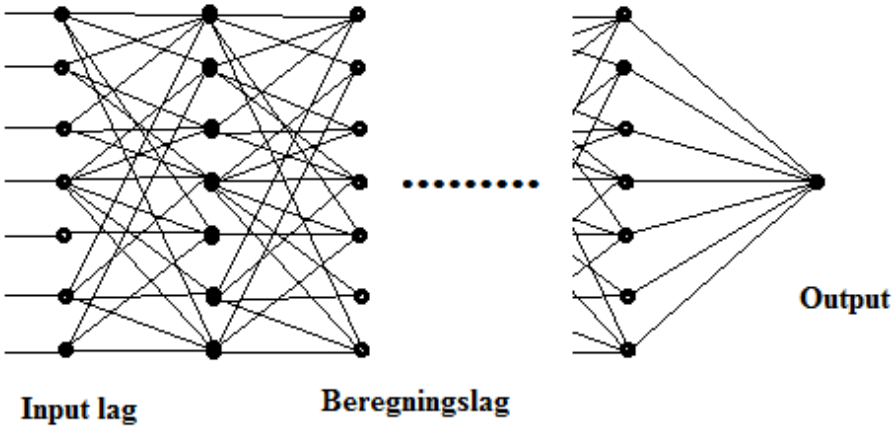
3. Træning af neurale netværk – Back propagation

Som I sikkert opdagede ved at løse opgaverne ovenfor, så er det ret svært at træne et netværk, så det svarer rigtigt i alle situationer, - også selvom der kun er nogle få neuroner i netværket. Ved store netværk med flere hundrede neuroner bliver opgaven betydeligt meget sværere og kan ikke overskues af noget menneske. Man er derfor nødt til at benytte sig af *algoritmer*.

En algoritme er en metode med nogle få overskuelige regler man bliver ved med at

følge, til man har nået det ønskede resultat. En måde at træne netværk på er metoden *back propagation*. For at udføre denne er man nødt til at have et træningssæt at træne netværket på. Sådan et sæt består af en række input, hvor man kender det rigtige svar.

Algoritmen fungerer ved, at man for hvert input måler forskellen på det output, neuronen sender ud, og det rigtige svar. Dette definerer en slags "fejlfunktion" *loss-function* der for hvert input viser hvor dårligt (eller godt) netværket er lige nu. Hvis netværket fungerer godt, er fejlfunktionen lille og hvis det fungerer skidt, er fejlen stor. For et givet input holder man nu alle tærskel og fyringsværdier fast i alle neuronerne på nær én i det sidste beregningslag.



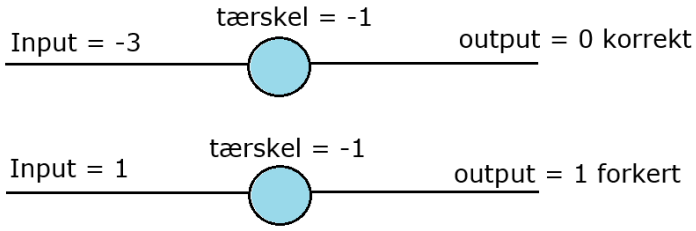
På skift ændrer man nu en lille smule på tærskel og fyringsværdien og ser, om det gør fejlen mindre eller større. Hvis fejlen er blevet større, skal tærskel/fyringsværdien ændres i modsat retning af hvad man gjorde og omvendt – hvis fejlen er blevet mindre er man på rette spor. *Man stiller nu neuronen tilbage i den oprindelige tilstand* og undersøger så den næste neuron i laget. På den måde får man for hele beregningslaget et vink om i hvilken retning hver enkelt neurons tærskel og fyringsværdi skal ændres, for at fejlen bliver mindre. Og det gør man så…. Altså ændrer dem et lille skridt i den retning man fandt ud af. Nu er alle neuronværdier i hele det sidste beregningslag ændret en lille smule – fejlen er blevet formindsket ved *back propagation*.

Man fortsætter nu til næstsidste beregningslag og gentager metoden og når dette lag også er forbedret, er fejlen altså blevet endnu mindre. Algoritmen fortsætter nu, til man er kommet igennem alle beregningslagene og man starter så forfra med næste input i træningssættet. Når man er færdig med at træne netværket på alle input i træningssættet, starter man helt forfra med træningssættet – for det kan jo være, at nogle af de sidste ændringer i neuronværdier har ødelagt de forbedringer man lavede ved hjælp af de første input. Som regel er det umuligt at få netværket til at svare korrekt for alle inputs netop af denne grund. Normalt tillader man derfor en fejlmargin på f.eks. 10%, så processen stopper, når netværket svarer korrekt på 90% af alle inputs.

Øvelse 3.1 Træning af en enkelt neuron

Lad os kigge på et simpelt eksempel hvor vi med én enkelt neuron skal løse problemet: ”er inputtallet større end 2?”

Vi ønsker at neuronen skal fyre, hvis svaret er ja, og være inaktiv, hvis tallet er mindre end 2. Det er tærskelværdien, vi skal ændre på, så fejlen bliver så lille som muligt:



Øvelse 3.2

Hvad skal tærskelværdien være for at neuronen svarer korrekt for alle inputværdier?

I denne simpel situation er det næsten indlysende hvad tærskelværdien skal være, men

Lad os prøve at træne den på et helt sæt af inputs. Hent filen *her* (link til *3a-træning af en enkelt neuron med batch.xls*).

Prøv at træne neuronen med et *batch* dvs: du beregner *gennemsnitsfejlen* for hele træningssættet (et lille et), før du laver justeringen af tærsklen:

E	F	G	H	I	J	K	L	M	N	O
			tærskel =	5		fejl = forskel ^2		9		
træningssæt		korrekt	neuronens							
	input	svar	svar		hvor mange der reelt er større end 2:					7
1	-10	0	0		hvor mange neuronen siger der er større end 2 :					4
2	-9	0	0							
3	-8	0	0							

I det viste eksempel er tærsklen sat til t = 5 og for hele træningssættet på 20 inputs (heltallene fra -10 til 9) er der 7 der faktisk er større end 2 men neuronen siger, at der er 4 fordi tærsklen er sat til 5. Loss-funktionen (fejlfunktionen) sætter vi til at være forskellen på det korrekte antal og neuronens antal i 2. potens (da forskellen kan være negativ) så her er fejlfunktionen 9.

Øvelse 3.3

a) Juster lidt på tærskelværdien (med heltalsskridt) til du kan se at fejlen bliver mindre. Skal tærsklen gøres større eller mindre?

b) Forstæt med at justere tærskelværdien til fejlen bliver så lille som muligt – den kan godt blive 0. Hvilken tærskelværdi ender du med?

Øvelse 3.4

(*Forudsætter kendskab til naturlige eksponentialfunktion*)

Som nævnt præsenteres sigmoidfunktionerne i hovedsafsnit II, men vi giver her til sidst lige et touch af forskellen fra Relu til sigmoid

Hent filen *her* (link til *3b-træning af en enkelt neuron batch med sigmoid.xls*) kan du træne et netværk til at løse samme opgave, men nu med en neuron der har en *sigmoid aktiveringsfunktion.*

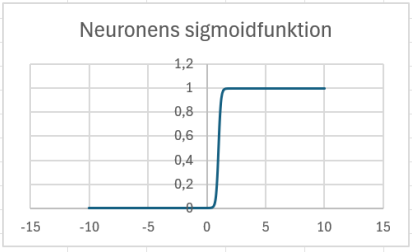
Fejlfunktionen for hvert input beregnes som

$$Fejl(x) = 100 \cdot \left(f_{tærskel}(x) - \frac{1}{1 - e^{10 \cdot (tærskel - x)}} \right)^2,$$

hvor $$f_{tærskel}(x) = \begin{cases} 1 & \text{for } x \geq tærskel \\ 0 & \text{for } x < tærskel \end{cases}$$

Her er faktorerne 100 og 10 bare sat ind for at forstærke fejlen.

	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	træningssæt	input	output	korrekt output	Fejl(x,tærskel)		individuel neuron med sigmoid							
		-5	8,76E-27	0	7,66765E-51									
		-3,2	5,75E-19	0	3,3057E-35		tærskel =	1						
		-1,1	7,58E-10	0	5,74952E-17									
		0,2	0,000335	0	1,1246E-05		Samlet fejl	99,97533						
		1,9	0,999877	0	99,97532261									
		2,2	0,999994	1	3,77509E-09									
		4	1	1	8,73866E-25									
		3,2	1	1	7,78114E-18									
		5	1	1	0									
		7,2	1	1	0									
			samlet fejl =sum af fejl		99,97533386									



a) Juster lidt på tærskelværdien fx +/- 0,1 til du kan se, at fejlen bliver mindre. Skal tærsklen gøres større eller mindre?

b) Fortsæt med at justere tærskelværdien til fejlen bliver så lille som muligt – Hvad er det mindste, fejlen kan blive?

c) Hvilken tærskelværdi ender du med?

d) Hvorfor kan fejlen ikke blive 0?

3.1 Træning af netværk med flere variable

Som vi skal se i det følgende, viser det sig hurtigt at blive meget sværere at træne netværket, når der er flere neuroner og dermed flere variable i spil.

Hent filen om træning af lille netværk i at ræsonnere logisk *her*: (link til: *3c-Logisk 'eller' med 5 variable træning.xls* vender vi tilbage til en af de første opgaver hvor vi skulle træne et lille netværk på 2 inputneuroner og en enkelt outputneuron til at se, om der var mindst et 1-tal i inputlaget:

B	C	D	E	F	G	H	I	J	K	L	M	N	O				
inputlag																	
w11	0,4		OPG 1														
w12	0,4						Træningssæt				output	"logisk eller"	"ELLER"				
b1	0,2		input	1		0	input		lineær 1	sigmo1	lineær 2	sigmo2	korrekt svar	Loss			
			tærskel				0	0	0,2	0,88079708	0,937754	0,999915	0	0,99983			
outputneuron			fyring				0	1	0,6	0,99752738	1,19456	0,999994	1	4,2E-11			
w2	2,2		output	0		0	1	0	0,6	0,99752738	1,19456	0,999994	1	4,2E-11			
b2	-1						1	1	1	0,9999546	1,1999	0,999994	1	3,8E-11			
			input	0								gennemsnitsfejl*10 =		2,49958			
			tærskel														
			fyring														
			output	0													

Neuronerne er her beskrevet "rigtigt" dvs. med lineære funktioner efterfulgt af en sigmoid. Du har mulighed for at justere på vægte og bias i både inputlaget og outputneuronen i de gule felter. Outputtet for træningssættet er vist i de grønne felter efterfulgt af det korrekte svar. Endelig er Loss- eller fejlfunktionen beregnet til højre i lilla felter, og gennemsnitsfejlen er forstørret op med en faktor 10 i det nederste lilla felt.

Øvelse 3.5 Starter

a) Undersøg, hvilken beregning "lineær 1" foretager.

b) Hvilken type sigmoidfunktion er anvendt?

Øvelse 3.6 Træning!

Hvis man ændrer parametrene med 0,1, når man frem til, at de skal ændres i en bestemt retning. Dette er vist lidt længere nede i regnearket:

oprindelig værdi:		skal ændres i retning			
b2	0,4	op	gammel fejl		2,49958
w2	0,4	op	ny fejl efter justering:		
b1	0,2	ned			
w12	2,2	ned			
w11	-1	ned			

a) Prøv at ændre parametrene i de angivne retninger og find den nye fejl.

b) Prøv nu én for én at justere på de nye værdier af parametrene og hold øje med den aktuelle gennemsnitsfejl, så du kan finde ud af, i hvilken retning de nu skal ændres.

c) Fortsæt med denne procedure til du får et samlet loss, der er under 0,01. Det kan godt lade sig gøre, men hvis du er meget utålmodig, kan du gå lidt længere ned i regnearket og se et bud på nogle gode parameterværdier.

Som du sikkert opdagede, er det ikke så let at styre, hvor meget hvilken parameter skal ændres. Vi skal senere se, at der findes metoder med brug af differentialregning til at finde lige præcis, i hvilket forhold parametrene indbyrdes skal ændres.

Vi ser nu på, hvordan man kan implementere neurale netværk i fysiske devices som robotter eller sensorer.

4. Neuromorfe neurale netværk

(Dette afsnit er mest relevant for elever, der har fysik i studieretnngen).

Når man træner et kunstigt neuralt netværk til at løse et bestemt problem, sker det som regel ved at simulere og justere netværket i en computer. Når man har fundet en fornuftig arkitektur for netværket og har trænet det til at løse problemet med en passende succes-procent, kan man begynde at bruge det. Vi antager derfor i det følgende, at man har fundet ud af, hvor mange neuroner der skal bruges, hvordan de er forbundet med hinanden, og hvad de individuelle tærskel- og fyringsværdier er for hver neuron.

Hvis det nu er et netværk, der skal anvendes ude i den virkelige verden f.eks. til at lugte om en plante er en bestemt slags blomst, eller om et skrig kommer fra en bestemt dyreart, vil det ofte være praktisk, hvis man har en *hardware-udgave* af det neurale netværk. Det vil i praksis betyde, at netværket skal implementeres i en chip og forbindes til nogle (sanse)sensorer, der kan give netværket sine input, samt til et output-device som f.eks. nogle lysdioder, en digitalskala eller tilsvarende, der kan fortælle, hvad netværkets output er. Man kalder denne tilgang til neurale netværk for *Neuromorf databehandling.*

Enhver enhed, der bruger fysiske kunstige neuroner til at udføre beregninger, kaldes for en *neuromorf computer eller chip*. I nyere tid er betegnelsen *neuromorf* blevet brugt til at beskrive analoge, digitale, blandede analog/digitale VLSI-systemer samt softwaresystemer, der implementerer modeller af neurale systemer (til perception, motorisk kontrol eller multisensorisk integration). Nyere fremskridt har endda gjort det muligt at detektere lyd ved forskellige bølgelængder gennem flydende kemiske opløsninger! Vi holder os dog her til det mere jordnære, som kan udføres i ethvert fysiklokale:

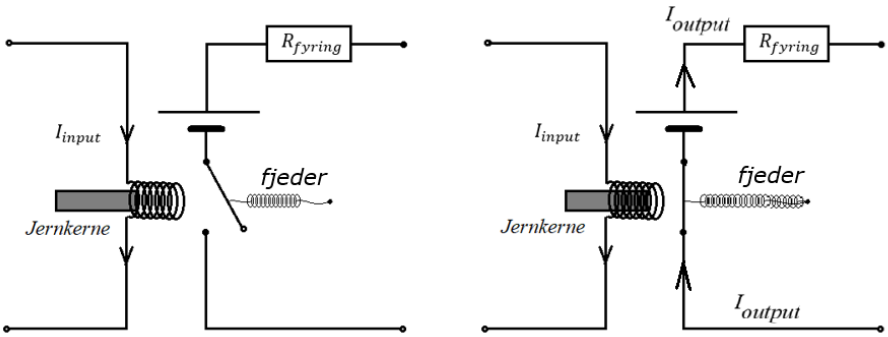
4.1 En elektronisk model af et kunstigt neuralt netværk

Vi gennemgår her en simpel elektromekanisk model af en neuron og viser, hvordan man kan bygge et neuralt netværk af disse.

En elektrisk neuron kan bygges af:

1. En lille spændingskilde, der kan tændes/slukkes af et relæ.
2. Et relæ, der består af en kontakt og en lille spole med en *justerbar jernkerne,* der bruges til at justere *tærskelværdien*
3. En ydre variabel resistor, der bruges til at justere *fyringsværdien*

På figuren herunder ses en model af sådan en elektrisk neuron:



Øvelse 4.1 Hvordan den elektroniske neuron virker

Forklar hvordan neuronen virker, dvs.

a) Hvad får kontakten (relæet) til at tænde for outputstrømmen?

b) Forklar, hvordan justering af jernkernen kan ændre tærskelværdien.

c) Forklar, hvordan justering af resistorens R_{fyring} 's resistans kan justere fyringsværdien.

d) Hvilken type sigmoidfunktion vil beskrive den elektroniske neuron?

Øvelse 4.2 Eksperiment

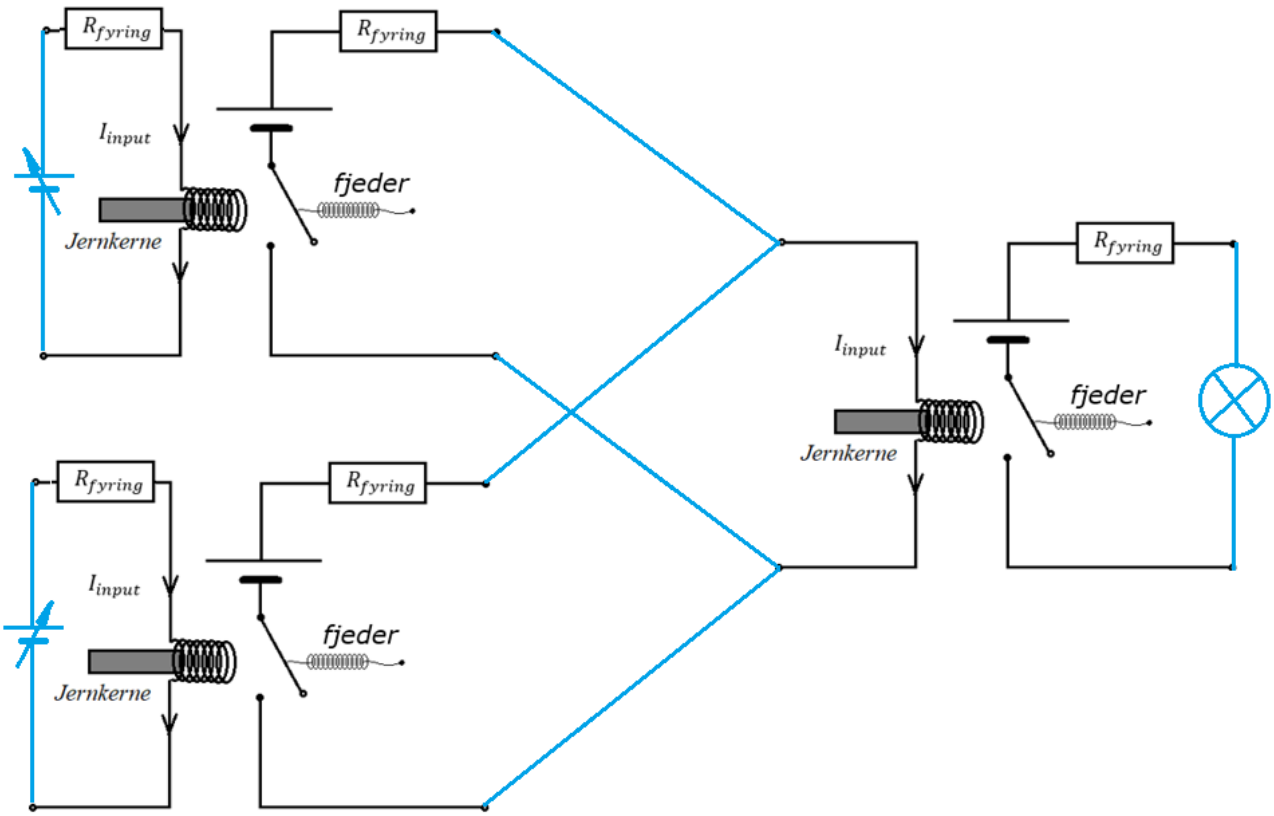
a) Byg en neuron som vist på figuren.

b) Tilslut en spændingskilde og en variabel resistor, så du kan måle tærskelværdien for strømmen med et amperemeter.

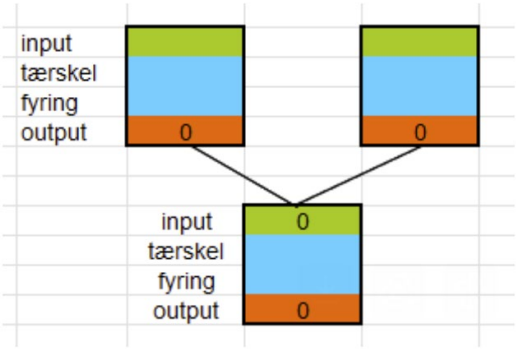
c) Tilslut en pære i serie med et amperemeter på outputsiden og juster fyringsresistoren, så outputstrømmen bliver en værdi du bestemmer. Pæren lyser altså når neuronen fyrer.

Øvelse 4.3 Opgave og eksperiment

Betragt det neuromorfe (elektriske) neurale netværk herunder:



a) Argumenter for, at netværket har samme arkitektur som excel-netværkene fra tidligere:

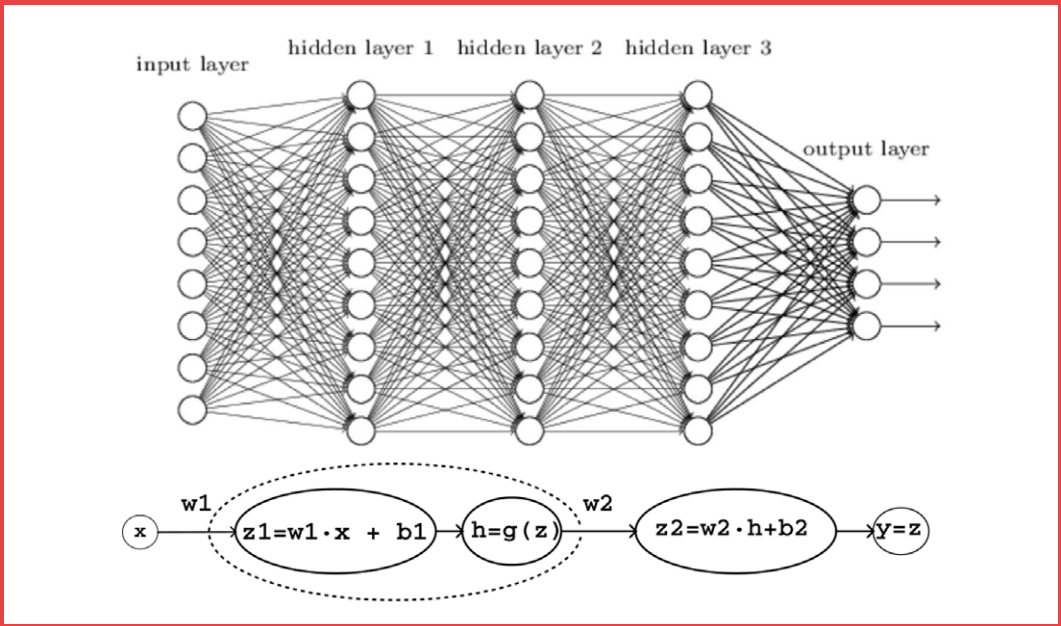


Prøv at træne netværket så det kan løse nogle af opgaverne (hver opgave kræver en ny indstilling af tærskel og fyringsværdier)

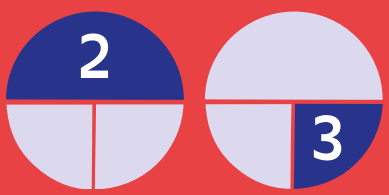
b) Netværket skal have den egenskab, at pæren lyser, hvis bare mindst én af input-strømmene er over 0,5 Ampere

c) Netværket skal have den egenskab, at pæren kun lyser, hvis begge inputstrøm-mene er over 0,5 Ampere.

II.
Neurale netværk.
B- og A-niveau



Neurale netværk er ikke så meget værd, hvis de ikke er trænet på gode data, siger Peter Mølgaard Larsen fra B&O i filmen. Og forudsætningen for, at det kan lade sig gøre er, at alle de funktioner, der er involveret, er differentiable. Illustrationen viser et lille udsnit af et '*fully connected network*', hvor forbindelserne fra input og igennem netværket sker gennem en kombination af lineære og ikke-lineære funktioner. Når netværket trænes, kender vi det ønskede output, og kan derfor beregne, hvor godt eller dårligt det er til at finde det korrekte svar. Træningen går ud på at minimere fejlene, og denne optimering sker med metoder, vi kender fra differentialregning, her blot i mange dimensioner. I denne 2. del bruger vi derfor også kræfter på at generalisere differentialregningens metoder fra én til mange dimensioner. Endelig præsenterer vi også såkaldte '*convolutional networks*', der med metoder fra bl.a. vektorregning er i stand til at genkende mønstre – og ansigter.



matematik
i arbejde

1. Introduktion: Fra hjerne til computer

Neurale netværk er inspireret af den måde, vores hjerne fungerer på. Den menneskelige hjerne består af omkring 86 milliarder neuroner, som er forbundet gennem billioner af synapser. Hver neuron modtager signaler fra tusindvis af andre neuroner, behandler denne information og sender det videre til andre neuroner i netværket.

I den biologiske hjerne er en neuron en specialiseret celle, der kommunikerer gennem elektriske og kemiske signaler. Når en neuron modtager tilstrækkelig stimulering fra andre neuroner, "fyrer" den. Det vil sige, den sender et elektrisk signal gennem sit akson til andre neuroner. Dette signal er binært: enten fyrer neuronen, eller den fyrer ikke.

Kunstige neurale netværk forsøger at efterligne denne struktur, om end i en meget forenklet form. I stedet for milliarder af neuroner arbejder de fleste kunstige netværk med tusinder til millioner af virtuelle neuroner. For eksempel har GPT-3, omkring 175 milliarder parametre, hvilket stadig er betydeligt mindre end antallet af synaptiske forbindelser i den menneskelige hjerne. I kunstige neurale netværk svarer hver vægt til en synaptisk forbindelse i den biologiske hjerne.

Selvom kunstige neurale netværk har færre forbindelser, kan de stadig løse komplekse opgaver, fordi de er designet til specifikke formål, mens hjernen skal håndtere alle livets funktioner.

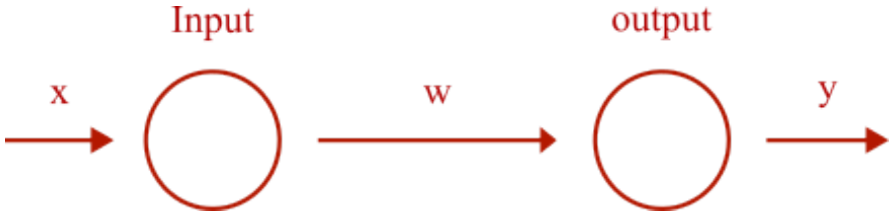
Tabellen viser nogle af forskellene på biologiske og kunstige neurale netværk.

Biologiske neurale netværk	**Kunstige neurale netværk (ML)**
Neuroner: 86 milliarder	**Neuroner:** Tusinder til millioner
Synapser: 100-500 billioner	**Vægte:** Millioner til milliarder
Signal: Elektrisk impuls (binær: fyrer/fyrer ikke)	**Signal:** Numerisk værdi (kontinuerlig)
Hastighed: Millisekunder pr. signal	**Hastighed:** Mikrosekunder pr. beregning
Læring: Ændring i synapsestyrke over tid	**Læring:** Justering af numeriske vægte
Energiforbrug: ~20 watt	**Energiforbrug:** Tusindvis af watt
Kommunikation: Kemiske neurotransmittere	**Kommunikation:** Matematiske operationer
Fejltolerance: Høj (kan kompensere for skadede celler)	**Fejltolerance:** Lav (præcise beregninger)
Plastisk: Kan omstrukturere forbindelser	**Plastisk:** Fast arkitektur under træning

I en virtuel neuron fungerer "signalet" anderledes end i den biologiske version. I stedet for elektriske impulser arbejder vi med tal. En kunstig neuron modtager numeriske input og sender et enkelt tal videre. Hvor en biologisk neuron enten fyrer eller ikke fyrer, kan en kunstig neuron sende et kontinuerligt signal - typisk et tal mellem 0 og 1.

1.1 Én forbindelse, optimering i én dimension.

Vi starter simpelt med én neuron, hvor en x-værdi bliver til en y-værdi. Lige som ved funktioner er y-værdien en funktion af x-værdien. Den simpleste sammenhæng er den *proportionale*, hvor man bruger w for weight i stedet for a til hældningskoefficient, så $y = w \cdot x$. Sammenhængen er illustreret i fig. 1.



Figur 1: Input én neuron

Output fra det simple neurale netværk kan beregnes som

$$\hat{y} = w \cdot x,$$

hvor $\hat{y}$ er den estimerede y-værdi.

Når man træner et netværk, ændrer man på vægtene, w, indtil man får den ønskede sammenhæng mellem input og output.

Eksempel

Lad os forbinde et input $x = 0.8$ med et output $y = 0.4$.

Vi kan relativt let løse ligningen $0.4 = w \cdot 0.8 \Rightarrow w = 0.4/0.8 = 0.5$. Den optimale vægt er altså $w = 0.5$, hvilet giver vores første model,

$$y = 0.5 \cdot x.$$

1.2 Optimering

I ovenstående eksempel løste vi ligningen i stedet for at træne den. Det kan vi kun fordi eksemplet var meget simpelt. Lad os se på, hvordan man træner en model.

Vi gennemgår det med eksemplet ovenfor, hvor vi fandt at hvis $x = 0.8$ og $y = 0.4$ så skulle $w = 0.5$.

Vi starter med en tilfældig w-værdi, eks. $w = 1$, og beregner, hvor langt modellen er fra den rigtige y-værdi. Den estimerede y-værdi kaldet $\hat{y}$ og forskellen δ.

$$\delta = \hat{y} - y = w \cdot x - y = 1 \cdot 0.8 - 0.4 = 0.4.$$

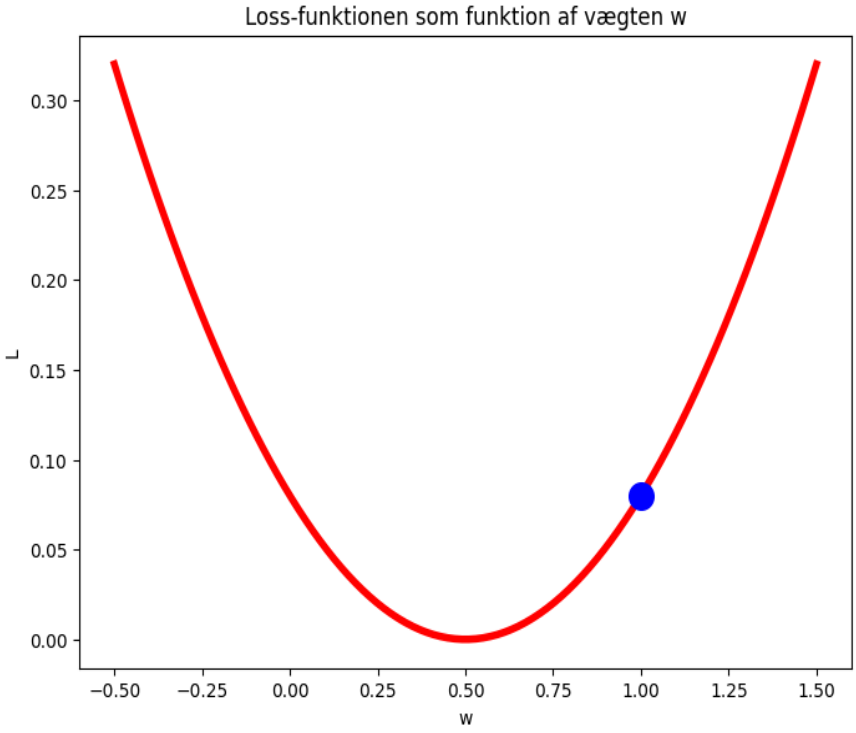
Vi har altså en fejl i vores model på $\delta = 0.4$.

Lige som ved regression beregner vi kvadratet på forskellen og kalder det vores fejl-funktion, eller loss-function på engelsk. Da vi skal find den værdi, der minimerer fejlen, så gør det ingen forskel, om vi ganger en konstant på. Det gør differentiationen en smule lettere at gange 0,5 på, så det gør vi:

$$L = 0{,}5(w \cdot x - y)^2.$$

Opgaven er nu at finde det w som minimerer fejlfunktionen L.

Lad os starte med at tegne en graf med værdier af w ud af 1. aksen og L ud af 2. aksen.

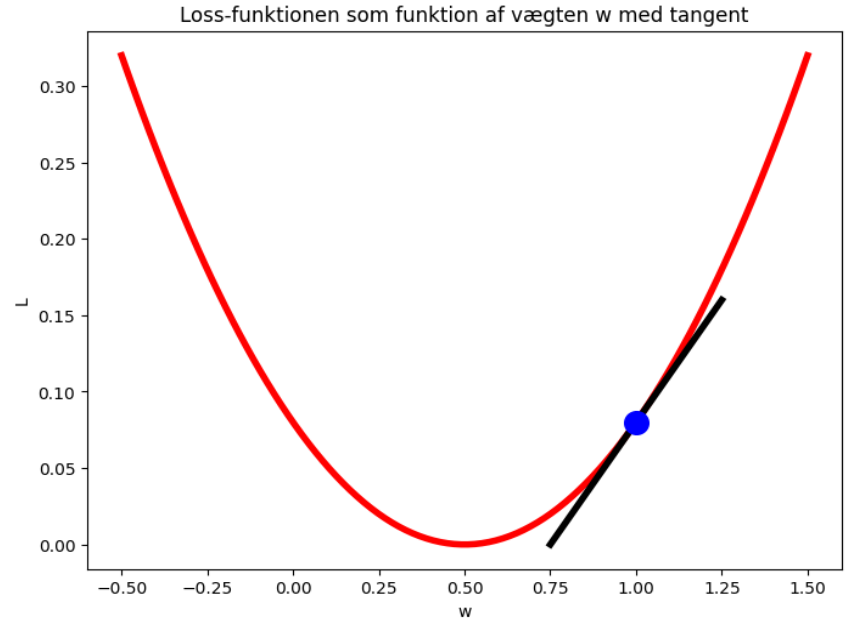


Figur 2: fejlfunktion

Som før kan vi godt se, at den optimale er $w = 0.5$, men lad os finde den ved optimering. Vi finder tangenthældningen ved at differentierer fejlfunktionen med hensyn til w.

$$\frac{dL}{dw} = (w \cdot x - y) \cdot x = \delta \cdot x.$$

Med $x = 0.8$ og $\delta = 0.4$ bliver $\frac{dL}{dw} = 0.4 \cdot 0.8 = 0.32$. Figur fig. 3 viser tangenten



Figur 3: Fejlfunktion med tangent

$\frac{dL}{dw}$ peger i den retning hvor funktionen vokser hurtigst. Hvis vi vil bevæge os i den retning hvor den aftager hurtigst skal vi bruge $-\frac{dL}{dw}$. Nu kender vi retningen og skal

beslutte os for hvor meget vi vil ændre w. Det gøres ved at gange en læringsrate ϵ på. Den nye w-værdi bliver nu beregnet med

$$w_{ny} = w - \epsilon \cdot \frac{dL}{dw}$$

med en læringsrate på $\epsilon = 0.1$ bliver den nye vægt

$$w_{ny} = 1 - 0.1 \cdot 0.32 = 0.968.$$

Med den nye vægt kan man beregne en ny $\hat{y}$ og en ny fejlfunktion og opdaterer vægten, w.

Opskrift på optimering

- Vælg et en startværdi for vægten, w.
- beregn fejlen $\delta = \hat{y} - y$.
- Beregn gradienten for den valgte vægt, $\frac{dL}{dw}$.
- Opdater vægten med formlen $w_{ny} = w - \epsilon \cdot \frac{dL}{dw}$.

Gentag med den nye vægt

Øvelse 1.1

Lad os gentage eksemplet, men denne gang starter vi med en negativ værdi for w, f.eks. $w = -0.2$.

1. Vælg startværdi, $w = -0.2$, $x = 0.8$, $y = 0.4$
2. Beregn $\hat{y}$, og δ.
3. Beregn $\frac{dL}{dw}$ og indtegn tangenten.
4. Opdater w med $w - \epsilon \cdot \frac{dL}{dw}$ og forklar hvilken vej vægten bevæger sig.
5. Gentag trin 2-4 så w nærmer sig 0.5.

Det er noget besværligt at optimere i hånden og vi skal have computeren til at gøre det.

Denne kode gentager proceduren 60 gange. Den er skrevet i Python og kan køres online på https://www.online-python.com/ eller I Jypyter filen, du kan hente *her* (link til filen II.1.a) *# Hvis I kopierer fra PDF, vil indryk ikke komme med, sørg selv for at lave alle indryk i linjerne 8 til 12.*	y = 0.4 w = 1 x = 0.8 epsilon = 0.1 i = 0 *# i er en tælle-variabel og sørger for at vi kører løkken 60 gange.* **while** i < 60: y_hat = w*x d = y_hat-y dL = d*x w = w-epsilon*dL i = i + 1 print('w=',w) print('y_hat=',y_hat)

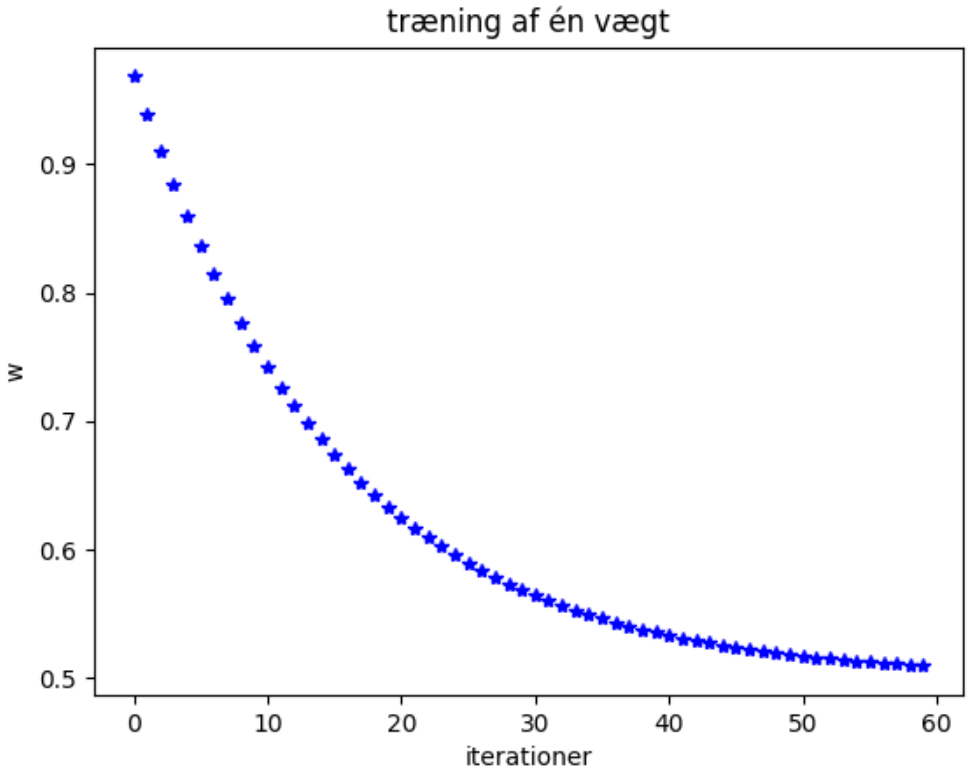
Øvelse 1.2

- Gennemgå linjerne i koden og overbevis jer om, at koden gør præcis det vi gjorde manuelt ovenfor.
- Kør programmet og se hvor godt estimaterne passer.

Antallet af gange vi opdaterer vores vægte kaldes *iterationer*.

- Lav færre iterationer, ved at skrive et lavere tal i while i < 60:.
- Prøv også at lave flere.
- Hvor mange iterationer skal der til for at være inden for 10% af den rigtige y-værdi.
- Prøv at start med en anden vægt.

Hvis man plotter (i, w) , hvor den estimerede parameter w er en funktion af antallet af iterationer i, bliver det:



Figur 4: Optimering af én vægt

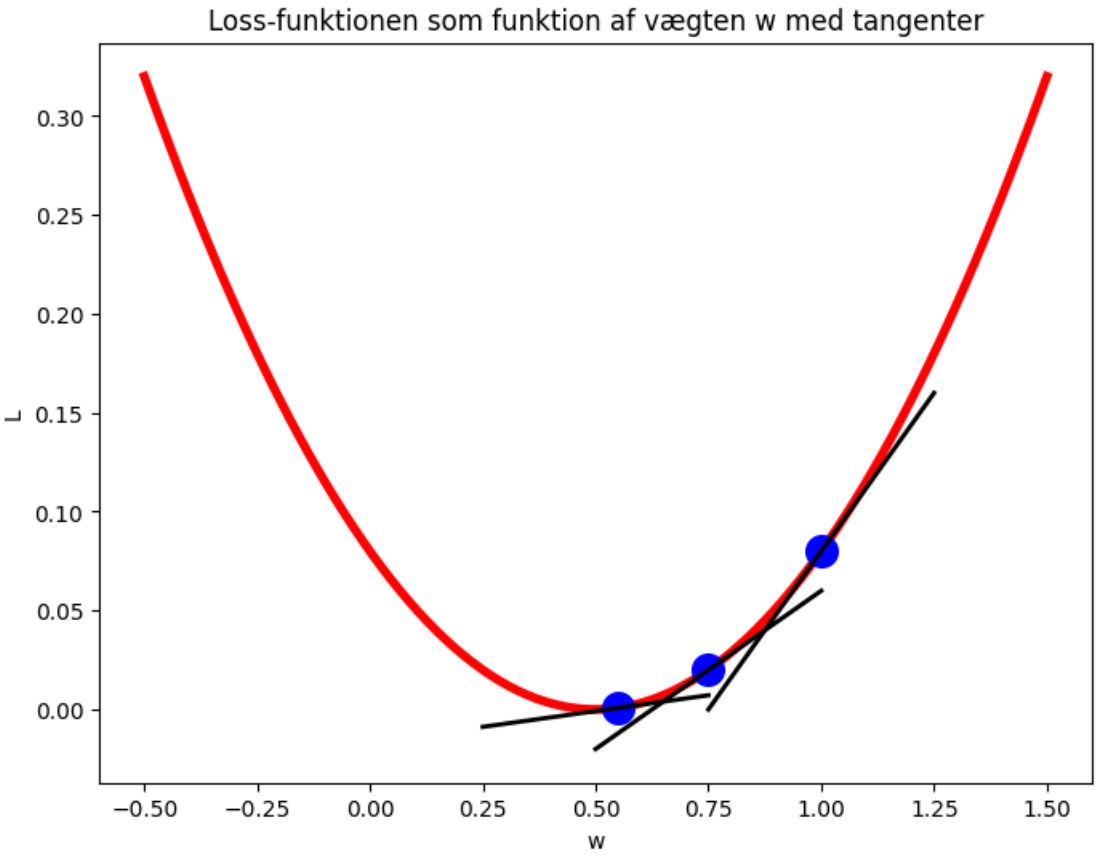
Som det ses, nærmer w-værdien sig $w = 0.5$.

Øvelse 1.3

Figur fig. 5 viser tre forskellige værdier af w og de tilhørende tangenter.

Udregn selv tangentligningerne, og bestem skæringspunktet med 1. aksen.

(*Bemærk, at 1. aksen ikke er den nederste vandrette linje – det er bare grafvinduets kant.*)



Figur 5: Forskellige startværdier for w

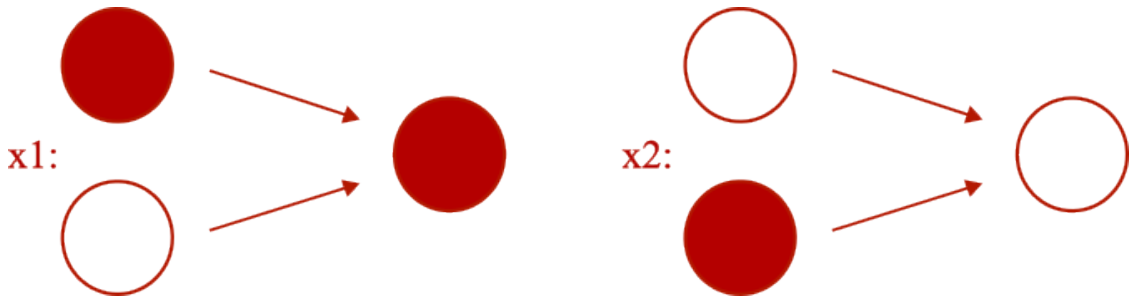
Tangenthældningen er $\frac{dL}{dw}$.

- Brug $w_{ny} = w - \epsilon \cdot \frac{dL}{dw}$ til at forklare, hvorfor træningen går langsommere og langsommere, når man nærmer sig den rigtige w-værdi.

1.3 To input

Vi udvider nu netværket til at have to input som bliver til to output-værdier.

Hvis vi siger at 1 er rød og 0 er hvid kan det illustreres sådan



Figur 6: to input farver

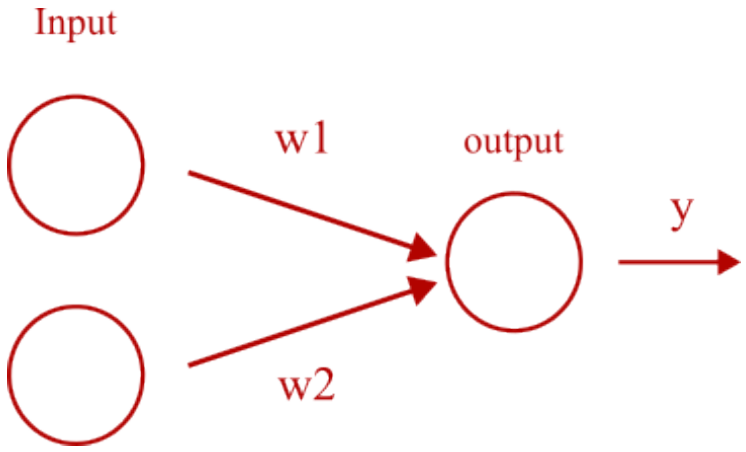
Vi giver input

- $x_1 = (1,0)$ skal give $y = 1$, og: $x_2 = (0,1)$ skal give $y = 0$.

De to vægte er w_1 og w_2. Det kan skrives om med vektorer

$$x_1 = \begin{bmatrix}1\\0\end{bmatrix} \rightarrow 1, x_2 = \begin{bmatrix}0\\1\end{bmatrix} \rightarrow 0, w = \begin{bmatrix}w_1\\w_2\end{bmatrix}$$

Netværket er tegnet i fig. 7. Bemærk at x_1 ikke er den øverste cirkel men både den øverste med værdi 1 og den nederste med værdi 0.



Figur 7: Neuralt netværk med to input og ét output

Output bliver beregnes ved

$$y_1 = w \cdot x_1 = w_1 \cdot 1 + w_2 \cdot 0 = 1$$
$$y_2 = w \cdot x_2 = w_1 \cdot 0 + w_2 \cdot 1 = 0$$

Dette er et ligningssystem med to ligninger og to ubekendte, w_1 og w_2.

Øvelse 1.4

- Løs ligningssystemet og bestem w_1 og w_2.

1.4 Optimering i to dimensioner

Vi vil optimere vægtene, w_1 og w_2. Det er i princippet det samme som ovenfor, men det bliver lidt mere kompliceret med to input.

For at holde styr på input, definerer vi de to input som

$$x_1 = \begin{bmatrix}a_1\\a_2\end{bmatrix}, x_2 = \begin{bmatrix}b_1\\b_2\end{bmatrix}$$

Hvilket i eksemplet svarer til at $a_1 = 1, a_2 = 0$ og $b_1 = 0, b_2 = 1$. De tilhørende output er $y_1 = 1$ og $y_2 = 0$.

Vi starter med tilfældige vægte $w_1 = 0.5$ og $w_2 = 0.5$. De estimerede $\hat{y}$-værdier er:

$$\hat{y}_1 = w_1 \cdot a_1 + w_2 \cdot a_2 = 0.5 \cdot 1 + 0.5 \cdot 0 = 0.5$$
$$\hat{y}_2 = w_1 \cdot b_1 + w_2 \cdot b_2 = 0.5 \cdot 0 + 0.5 \cdot 1 = 0.5$$

Fejlene δ_1 og δ_2 er:

$$\delta_1 = \hat{y}_1 - y_1 = (w_1 \cdot a_1 + w_2 \cdot a_2) - y_1 = 0.5 - 1 = -0.5$$
$$\delta_2 = \hat{y}_2 - y_2 = (w_1 \cdot b_1 + w_2 \cdot b_2) - y_2 = 0.5 - 0 = 0.5$$

Fejlfunktionen L er gennemsnittet af kvadratsummerne af fejlene på hver variabel:

$$L = 0.5(\delta_1^2 + \delta_2^2) = 0.5 \cdot ((w_1 \cdot a_1 + w_2 \cdot a_2) - y_1)^2 + 0.5 \cdot ((w_1 \cdot b_1 + w_2 \cdot b_2) - y_2)^2$$

Vores variable er w_1 og w_2, og vi differentierer L med hensyn til dem hver for sig. Udtrykket ser besværligt ud, men husk at du må differentiere hvert led for sig, og at alt andet end w_1 er konstanter, når man differentierer med hensyn til w_1. Første udtryk bliver:

$$\frac{dL}{dw_1} = ((w_1 \cdot a_1 + w_2 \cdot a_2) - y_1) \cdot a_1 + ((w_1 \cdot b_1 + w_2 \cdot b_2) - y_2) \cdot b_1 = \delta_1 \cdot a_1 + \delta_2 \cdot b_1$$

Øvelse 1.5

Du skal vise at $\frac{dL}{dw_2} = \delta_1 \cdot a_2 + \delta_2 \cdot b_2$, ved at gennemgå følgende skridt:

- Del udtrykket for L op i to led, et som indeholder a og et som indeholder b.
- Omskriv $((w_1 \cdot a_1 + w_2 \cdot a_2) - y_1)^2$ til $(w_1 \cdot a_1 + k)^2$. Hvad er k?
- Hvad er den ydre og den indre funktion i udtrykket?
- Brug kædereglen til at differentierer m.h.t. w_1.
- Indsæt k og vis at $\frac{d}{dw_1}((w_1 \cdot a_1 + w_2 \cdot a_2) - y_1)^2) = 2 \cdot ((w_1 \cdot a_1 + w_2 \cdot a_2) - y_1) \cdot a_1$.
- Gør det samme for w_2.
- Læg resultaterne sammen og husk at gange med 0.5.
- Skriv resultatet med δ_1 og δ_2.

Det samlede resultat er

$$\frac{dL}{dw_1} = \delta_1 \cdot a_1 + \delta_2 \cdot b_1$$
$$\frac{dL}{dw_2} = \delta_1 \cdot a_2 + \delta_2 \cdot b_2$$

Opdateringsreglerne for w_1 og w_2 med en læringsrate ϵ er:

$$w_{1_{ny}} = w_1 - \epsilon \cdot \frac{\partial L}{\partial w_1} = w_1 - \epsilon \cdot (\delta_1 \cdot a_1 + \delta_2 \cdot b_1)$$
$$w_{2_{ny}} = w_2 - \epsilon \cdot \frac{\partial L}{\partial w_2} = w_1 - \epsilon \cdot (\delta_1 \cdot a_2 + \delta_2 \cdot b_2)$$

Eksempel

Vi fortsætter eksemplet med $w_1 = 0.5$ og $w_2 = 0.5$ og en læringsrate $\epsilon = 0.1$. Ovenfor udregnede vi fejlene, $\delta_1 = -0.5$ og $\delta_2 = 0.5$.

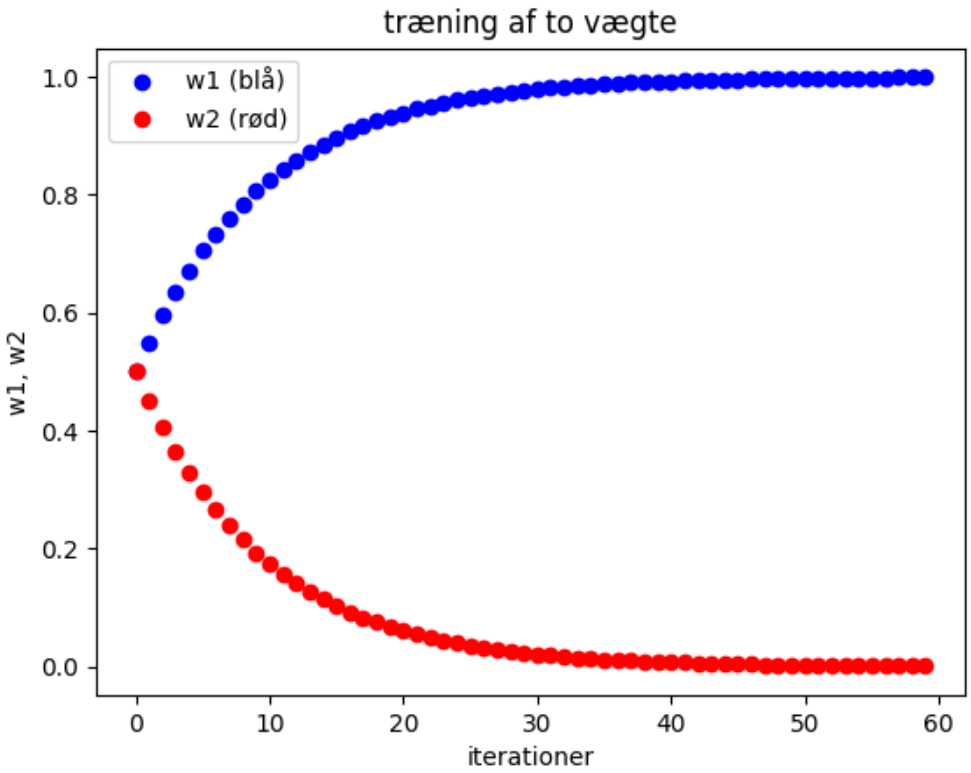
De nye vægte kan derfor beregnes som

$$w_{1_{ny}} = w_1 - \epsilon \cdot (\delta_1 \cdot a_1 + \delta_2 \cdot b_1) = 0.5 - 0.1 \cdot (-0.5 \cdot 1 + 0.5 \cdot 0) = 0.5 + 0.1 \cdot 0.5 = 0.55$$
$$w_{2_{ny}} = w_1 - \epsilon \cdot (\delta_1 \cdot a_2 + \delta_2 \cdot b_2) = 0.5 - 0.1 \cdot (-0.5 \cdot 0 + 0.5 \cdot 1) = 0.5 - 0.1 \cdot 0.5 = 0.45$$

Efter én iteration er w_1 steget til 0.55 og w_2 faldet til 0.45, hvilket bringer dem tættere på de optimale værdier $w_1 = 1$ og $w_2 = 0$.

Med de nye vægte kan vi starte forfra og træne modellen igen.

Figur fig. 8 viser vægtene gennem optimeringen.



Figur 8: Optimering af to vægte

Hvis vi kører algoritmen 60 gange, bliver $w_1 = 0.999$ og $w_2 = 0.001$. Hvis vi vil have mere præcise estimater, skal vi lave flere iterationer, men som I kan se af figuren, går det langsomt med at blive bedre til sidst.

For at træne det neurale netværk er det smart at skrive et program. Nedenstående kode kører 60 iterationer, hvor den beregner nye vægte hver gang. Koden er python og kan køres online på https://www.online-python.com/, eller I Jypyter filen, du kan hente *her* (link til II.1a-machineLearning.ipynb.

```
y1 = 1; y2 = 0; a1 = 1; a2 = 0; b1 = 0; b2 = 1; w1 = 0.5; w2 = 0.5
epsilon = 0.1; i = 0
while i < 60:
    y1_hat = w1*a1+w2*a2
    y2_hat = w1*b1+w2*b2
    d1 = y1_hat-y1
    d2 = y2_hat-y2
    dL1 = d1*a1+d2*b1
    dL2 = d1*a2+d2*b2
    w1 = w1-epsilon*dL1
    w2 = w2-epsilon*dL2
    i += 1
print('y1_hat =',y1_hat)
print('y2_hat =',y2_hat)
```

Øvelse 1.6

- Kør koden.
- Undersøg hvor mange iterationer, der skal til for at $w_1 \approx 0.9$ ved at ændre i linje 12.
- Undersøg om jeres resultater er robuste, hvis I ændrer vægtenes startværdier, linje 4 og 5.

Denne association var jo ret simpel, men hvad med

$$x_1 = \begin{bmatrix} 0.77 \\ -0.92 \end{bmatrix} \rightarrow y = 2.718, x = \begin{bmatrix} 1.22 \\ 0.54 \end{bmatrix} \rightarrow y = 3.141$$

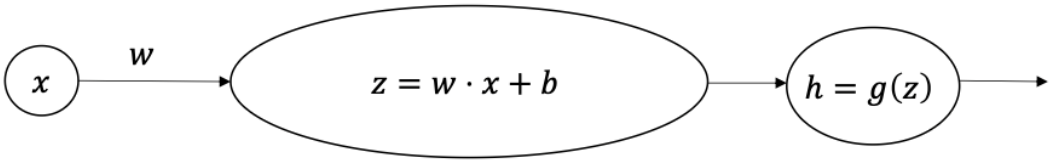
- Indsæt de nye værdier og se om de estimerede $\hat{y}$ passer.
- Lav antallet af iterationer større og se om I kommer tættere på.

2. Aktiveringsfunktioner

I afsnittet om matricer viser vi, at dybe neurale netværk med mange lag kan reduceres til et enkelt, hvis de er lineære. I dette afsnit vil vi se på, hvordan en ikke lineær aktiveringsfunktion kan implementeres i et neuralt netværk.

Vi starter med det simple netværk hvor vi ser på én neuron.

Iinput til en neuron kalder vi *x*, og denne bliver vægtet med en vægt w og en bias b hvilket giver en z-værdi. Denne z-værdi bliver input i den ikke lineære funktion $g(z)$ og output fra neuronen som sendes videre betegnes h.



Figur 9: aktivering af én neuron

Eksempel: Ét input

Lad os se på et simpelt eksempel hvor $x = 0.2$, $w = 2$, $b = -1$ og $g(z) = z^2$. Output fra neuronen bliver

$$z = w \cdot x + b = 2 \cdot 0.2 - 1 = -0.6$$

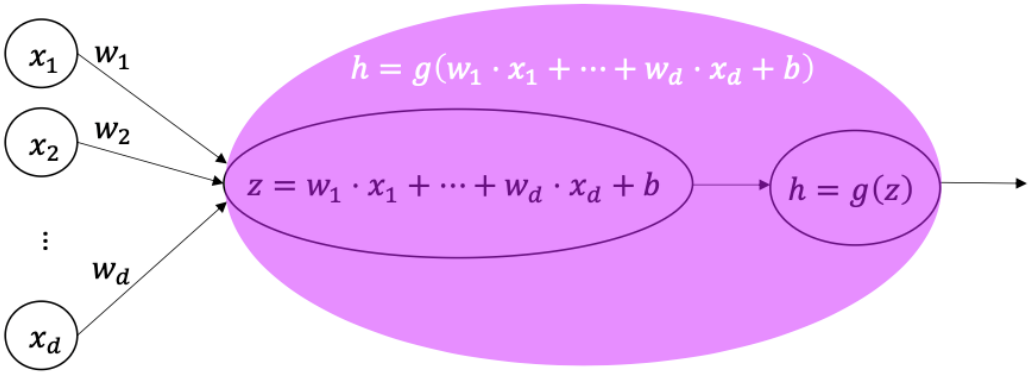
$$h = g(z) = (-0.6)^2 = 0.36$$

Neuronen giver altså et output på $h = 0.36$ ved et input på 0.2. Vi kan samle hele proceduren i

$$h = g(w \cdot x + b).$$

Hvis der er flere input, ser det ud som i figuren nedenfor. *z* er nu en lineær funktion af vægtede input og bias som indsættes i aktiveringsfunktionen $g(z)$ som giver ét output.

Parameteren *b* (bias), der svarer til kontantleddet i en lineær funktion, er *ikke* det samme for alle input, men da de er udtryk for det samme fænomen – at der er "støj" i systemet, eller at der er en "rest" vi ikke har styr på i modellen – så vælger vi i det følgende at slå alle disse *bias*-led samme i en parameterværdi, vi kalder *b*.



Figur 10: Aktivering med mange input til én neuron

Input og vægte kan skrives som vektorer og z bliver beregnet med prikproduktet mellem $\vec{w}$ og $\vec{z}$,

$$\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{pmatrix}, \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{pmatrix}, z = \vec{w} \cdot \vec{z}.$$

Eksempel Flere input

Lad os prøve igen med samme aktiveringsfunktion, men med input og vægte,

$$\vec{w} = \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix}, \vec{x} = \begin{pmatrix} 0.2 \\ 1.0 \\ 1 \end{pmatrix}$$

Prikproduktet giver

$$z = \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} 0.2 \\ 1.0 \\ 1 \end{pmatrix} = 2 \cdot 0.2 + 1 \cdot 1 - 1 \cdot 1 = 0.4$$

og aktiveringsfunktionen giver endelig dette resultat:

$$h = g(0.4) = 0.4^2 = 0.16.$$

2.1 Sigmoidfunktioner

Netværk af biologiske neuroner har gennem evolutionen trænet sig op til at kunne håndtere situationer som:

- genkende mønstre
- afkode sammenhænge

Det (og meget mere) har været forudsætningen for at overleve.

Vi vender tilbage til mønstergenkendelse, men vil i det følgende opholde os lidt ved dette begreb, *at afkode sammenhænge*. Bestemte kemiske spor og koncentrationen heraf fortæller historier om føde og om fare. Den mest simple sammenhæng er en proportionalitet, og en anelse mere kompleks så er det lineært. Det er jo også det første matematiske værktøj vi griber til, når vi søger at afkode sammenhænge.

Men som Jes fortæller i filmen, så er verden ikke lineær. På korte distancer og over korte tidsrum kan det være rigtig gode tilnærmelser, men for at få fat i de rigtige

sammenhænge, skal der et twist til. Vi ved at sammensætningen af to funktioner ofte kan give noget ret kompliceret, men sammensætningen af to lineære giver en ny tredje lineær funktion.

Øvelse 2.1 Sammensætning af to lineære funktioner giver noget lineært
Vis dette! Dvs vis, at hvis $f(x) = a \cdot x + b$ og $g(x) = c \cdot x + d$, så er $f(g(x))$ også en lineær funktion.

Skal vi kunne fange noget mere komplekst, skal der et ikke-lineært element ind over. I den matematiske værktøjskasse byder en særlig klasse af funktioner sig til, *sigmoid-funktionerne*.

En sigmoid-funktion er en matematisk funktion, hvis graf er en S-formet kurve, som stiger gradvist og flader ud i toppen og i bunden af kurven.

Definition 2.1: Sigmoid-funktion
En sigmoid-funktion $p(x)$ er en differentiabel funktion, hvorom det gælder, at

1) Definitionsmængden for p er alle reelle tal: $Dm(p) = \mathbb{R}$.
1) Værdimængden er begrænset af funktionens nedre og øvre grænse:
$$\lim_{x \to -\infty} p(x) = m_{nedre} \text{ og } \lim_{x \to \infty} p(x) = m_{øvre}$$
2) Tangenthældningen er ikke negativ i ethvert punkt på kurven, dvs. $p'(x) \geq 0$, for alle x, så p er monotont voksende.
3) Grafen for den første afledede $p'(x)$ er en symmetrisk 'klokkeformet kurve'.
4) Grafen for p, der kaldes en sigmoid-kurve, har netop ét vendepunkt.

Sigmoid-kurver anvendes ofte til at modellere en blød overgang mellem to flader eller overgange mellem forskellige niveauer.

Standardeksemplet: Logistiske funktioner

De mest almindelige sigmoidfunktioner er de *logistiske funktioner*, fx denne:

$$f(x) = \frac{1}{1 + e^{-x}}$$

som er defineret for alle x, $Dm(f) = \mathbb{R}$, og som har en begrænset værdimængden $Vm(f) =]0;1[$, dvs. funktionen kan kun antage positive funktionsværdier. Den første afledede er

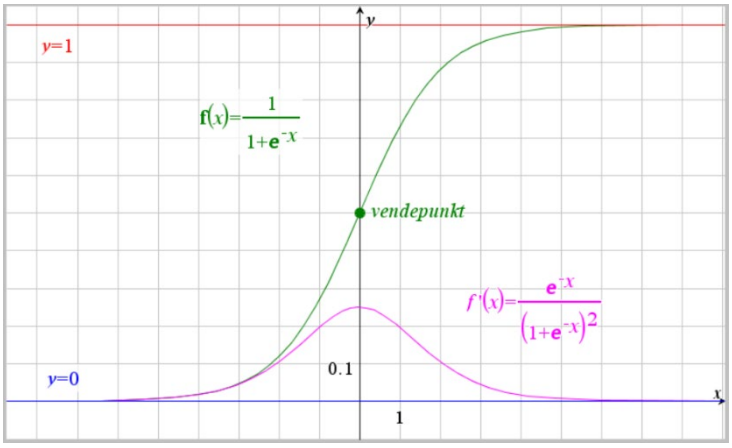
$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

som også er defineret for alle x. Vi ser også, at $f'(x)$ kun antager positive værdier (overvej!) svarende til, at f er voksende. Grafen for f' har netop form som en 'klokkeformet kurve', som vi kan se nedenfor.
Vi kan bestemme vendepunktet ved hjælp af den anden afledede:

$$f''(x) = \frac{e^{-x} - e^{-3x}}{(1 + e^{-x})^4}$$

idet den anden afledede, der beskriver krumningen, netop er nul i vendepunktet. En brøk er nul, når tælleren er nul, så vi løser ligningen

$$\begin{aligned} e^{-x} - e^{-3x} &= 0 \\ e^{-x} &= e^{-3x} \\ \ln(e^{-x}) &= \ln(e^{-3x}) \\ -x &= -3x \\ 2x &= 0 \\ x &= 0 \end{aligned}$$



Altså har grafen for f netop et vendepunkt, og det ligger på y-aksen i

$$y = f(0) = \frac{1}{1+e^0} = \frac{1}{2}$$

Den grafiske repræsentation bekræfter alle vores beregninger.

Der findes en række andre sigmoidfunktioner, og vi angiver nogle af dem her.

Eksempel 1: Den omvendte funktion til tangensfunktionen

Den omvendte funktion til tangensfunktionen kaldes arcus-tangens. Den betegnes således:

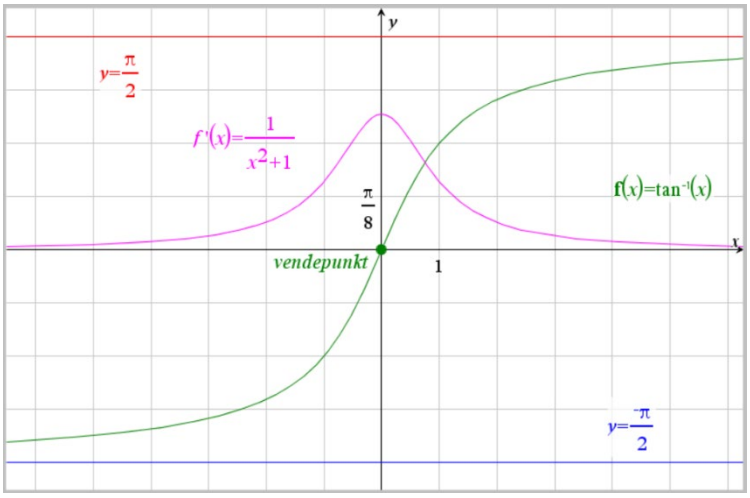
$$f(x) = \arctan(x) = \tan^{-1}(x)$$

Her er igen $Dm(f) = \mathbb{R}$, og værdimængden er begrænset:

$Vm(f) =]-\frac{\pi}{2};\frac{\pi}{2}[$, men her kan funktionen antage både positive og negative funktionsværdier.

Den første afledede er $f'(x) = \frac{1}{1+x^2}$ som også er defineret for alle x.

Vi ser også, at $f'(x)$ kun antager positive værdier svarende til at f er voksende.



Grafen for f' har også her form som en 'klokkeformet kurve'. Vendepunktet kan igen bestemmes med den anden afledede, og det ligger her i origo.

Øvelse 2.2 Bestem vendepunktet for arctan

Bestem den anden afledede af arcustangens-funktionen, og vis, at vendepunktet ligger i origo.

Eksempel 2: Tangens hyperbolsk

Tangens hyperbolske har forskriften

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Her er igen $Dm(f) = \mathbb{R}$.

Grafen viser, at værdimængden er begrænset, $Vm(f) =]-1;1[$.

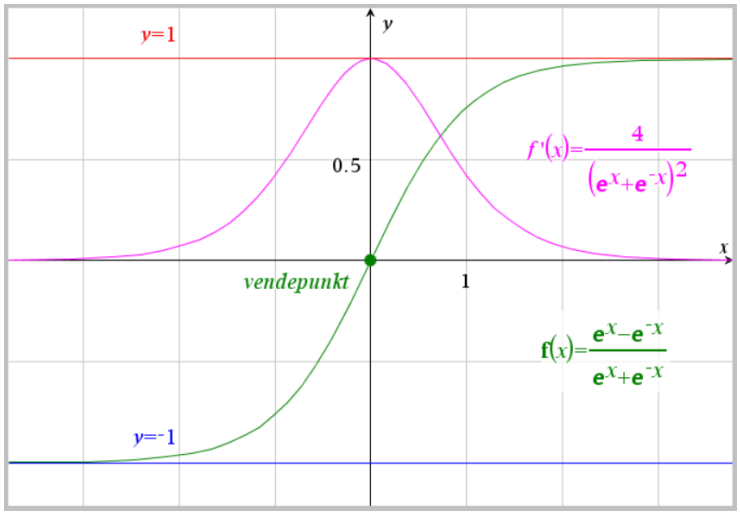
Vi går ikke dybere ind i beviset for det.

Funktionsværdierne være både positive og negative.

Den første afledede er

$$f'(x) = \frac{4}{(e^x + e^{-x})^2}$$

som også er defineret for alle x.



$f'(x)$ antager kun positive værdier svarende til at f er voksende.

Grafen for f' er også her en 'klokkeformet kurve'.

Øvelse 2.3 Bestem vendepunktet for tangens hyperbolsk.

Bestem den anden afledede af *tanh*-funktionen, og vis, at vendepunktet ligger i origo.

Eksempel 3: Reciprokfunktion med rodfunktion

Følgende funktion er også en sigmoid-funktion: $f(x) = \frac{1}{\sqrt{1+x^2}}$.

Øvelse 2.4 Bevis dette ved at følge opskriften:

a) Gør rede for, at f opfylder betingelserne for at være en sigmoid-funktion, ved at besvare følgende hjælpespørgsmål:
b) Tegn grafen for f.
c) Hvad er definitionsmængde og værdimængde for f?
d) Bestem $f'(x)$, og benyt denne til at argumentere for, at f er monotont voksende.
e) Bestem $f''(x)$, og benyt denne til at påvise, at grafen for f har netop et vendepunkt, og bestem koordinatsættet til vendepunktet.

Eksempel 4: Normalfordelingens fordelingsfunktion

Tæthedsfunktionen for standardnormalfordelingen har forskriften

$$f(x) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot e^{-\frac{1}{2} \cdot x^2}$$

og grafen for f er som bekendt en 'klokkeformet kurve'.

Øvelse 2.5. Undersøg standardnormalfordelingens fordelingsfunktion
Undersøg, om stamfunktionen til *f*, dvs. standardnormalfordelingens fordelingsfunktion, opfylder betingelserne for at være en sigmoid-funktion.
Bemærk: *Begge funktioner er indbygget i dit matematiske værktøjsprogram – her hedder de typisk noget i retning af: normpdf(…) og normcdf(…).*

2.2 Sammensætning af lineær med sigmoid

I filmen fortæller Jes Frellsen, at en neuron typisk fungerer som en *funktion* der er sammensat af en lineær funktion $f(x) = w \cdot x + b$ efterfulgt af en *sigmoid*-funktion. Men hvordan svarer det egentlig til den måde, vi lige har beskrevet en neuron på, med en *tærskelværdi* i kombination med en *fyringsværdi*? Det vil vi undersøge i de følgende opgaver!

Vi starter med at kigge på hvordan en neurons virkemåde kan beskrives med en simpel sigmoidfunktion med forskriften $s(x) = \frac{1}{1+e^{-x}}$

Øvelse 2.6 Parallelforskydning af en graf

a) Plot grafen for funktionen $s_1(x) = \frac{1}{1+e^{-x}}$

b) Plot grafen for funktionen $s_2(x) = \frac{1}{1+e^{-x}} - 1$ - hvad er fordelen ved en aktiveringsfunktion af denne type? (*Vink: Tænk på tidligere - i nogle opgaver havde vi brug for en negativ fyringsværdi!*)

Vi prøver nu at sammensætte sigmoidfunktionen $s(x)$ med en simpel lineær funktion $f(x) = w \cdot x + b$

Øvelse 2.7 Undersøg sammensætningen af sigmoid med lineær

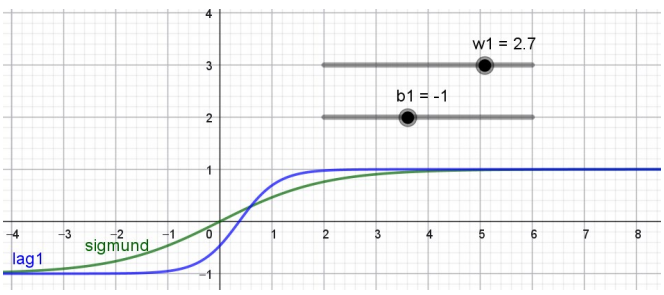
a) Opret skydere for parametrene *w* og *b*.

b) Tegn grafen for den sammensatte funktion $h(x) = s_1(f(x)) = \frac{1}{1+e^{w \cdot x + b}}$

Sammenlign med den oprindelige Heaviside aktiveringsfunktion med tærskelværdi *t* og fyringsværdi *f* (se afsnit 1.3 i del I):

c) Prøv at ændre på parameteren *b*. Er det tærskelværdien eller fyringsværdien denne parameter styrer?

d) Prøv nu at ændre på parameteren *w*. Er det tærskelværdien eller fyringsværdien denne parameter styrer?



2.3 Relu-funktionen

Definition: Relu-funktionen

Relu-funktionen kan defineres som: $r(x)=\begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{for } x<0 \end{cases}$

Eller den kan defineres som: $r(x)=\max(0,x)$

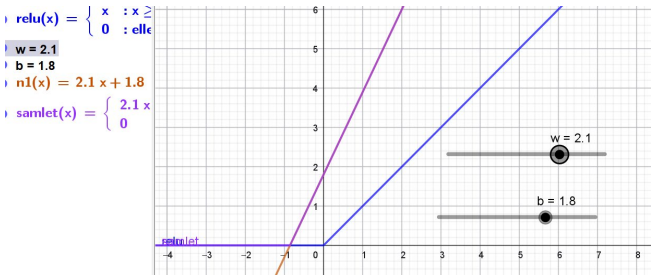
Øvelse 2.8 De to definitioner er ækvivalente

Argumenter for, at de to funktionsudtryk i definitionen giver samme funktion

Øvelse 2.9 Grafen for relu og for relu sammensat med en lineær.

a) Tegn grafen for relufunktionen.

b) Sammensæt relu-funktionen med den lineære $f(x)=w\cdot x+b$ og tegn grafen for den samlede funktion $samlet(x)=relu(w\cdot x+b)$ med skydere for w og b

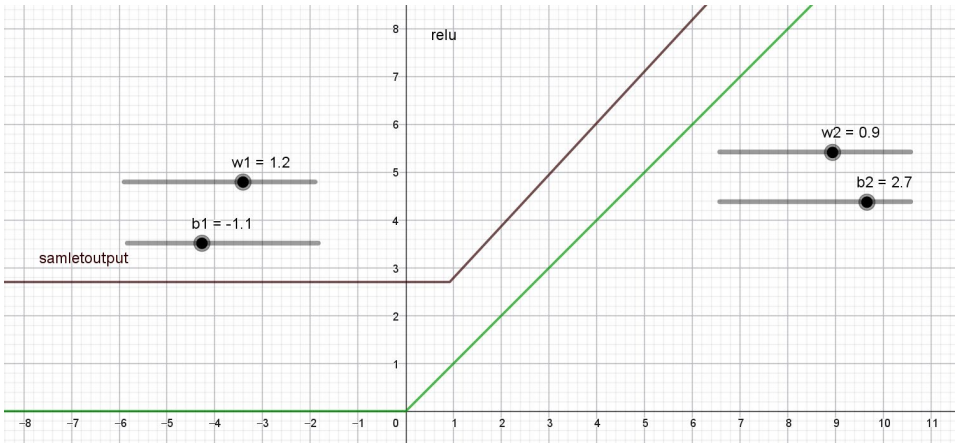


c) Hvilke af parametrene w og b styrer henholdsvis *tærskelværdien* og *fyringsværdien*?

I filmen nævner Jes Frellsen, at man i et dybt neuralt netværk godt kan klare sig med simple relufunktioner som ikke-lineariteter, hvis bare netværket er stort nok. Vi antyder sammenhængen i næste øvelse.

Øvelse 2.10 Sammensætning af to relu'er giver en sigmoid-lignende funktion!

Hent filen *2a (2a-samensætning af to relu giver sigmoid simpel.ggb)*.



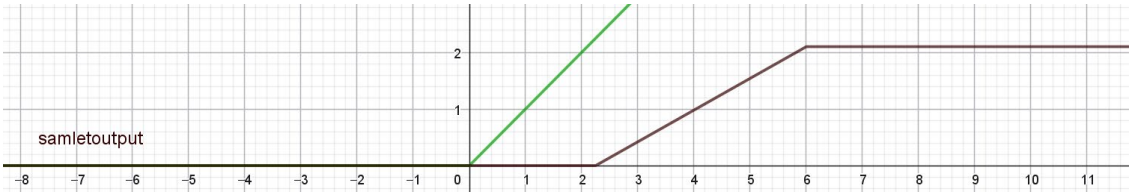
Vi har givet to lineære funktioner:
$f_1(x)=w_1\cdot x+b_1$ og
$f_2(x)=w_2\cdot x+b_2$.

Sammensæt nu de to lineære med to standard-reluer efter hinanden, således:

$$samletoutput(x)=r(f_2(r(f_1(x))))=r\left(w_2\cdot r\left(w_1x+b_1\right)+b_2\right)$$

Grafen for $samletoutput(x)$, hvor parametrene er angivet med skydere, finder du på næste side:

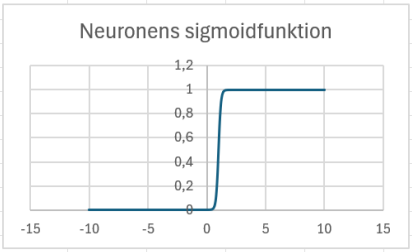
a) Leg med tærskel og fyringsværdierne w_1, b_1, w_2, b_2 til du får sammensætningen til at ligne en sigmoidfunktion dvs. se således ud:



b) Hvilke(n) parameter styrer tærskelværdien for den fremkomne sigmoidfunktion?

c) Hvilke(n) parameter styrer fyringsværdien for den fremkomne sigmoidfunktion?

træningssæt	input	output	korrekt output	Fejl(x,tærskel)		individuel neuron med sigmoid	
	-5	8,76E-27	0	7,66765E-51			
	-3,2	5,75E-19	0	3,3057E-35		tærskel =	1
	-1,1	7,58E-10	0	5,74952E-17			
	0,2	0,000335	0	1,1246E-05		Samlet fejl	99,97533
	1,9	0,999877	0	99,97532261			
	2,2	0,999994	1	3,77509E-09			
	4	1	1	8,73866E-25			
	3,2	1	1	7,78114E-18			
	5	1	1	0			
	7,2	1	1	0			
		samlet fejl =sum af fejl		99,97533386			



e) Juster lidt på tærskelværdien fx +/- 0,1 til du kan se at fejlen bliver mindre. Skal tærsklen gøres større eller mindre?

f) Fortsæt med at justere tærskelværdien til fejlen bliver så lille som muligt – Hvad er det mindste fejlen kan blive?

g) Hvilken tærskelværdi ender du med?

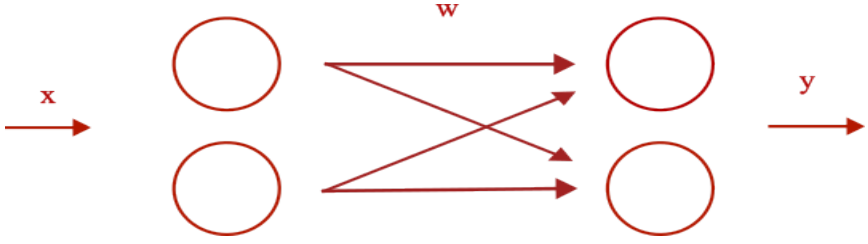
h) Hvorfor kan fejlen ikke blive 0?

3. Matricer

I afsnit 1 viste vi, hvordan simple lineære neurale netværk kan opstilles og trænes. Vi viste, hvordan træningen af én neuron svarede til at løse en ligning med én ubekendt, og to neuroner til to ligninger med to ubekendte. Når vi skal op i større netværk, har vi brug for en effektiv metode til at holde styr på ligningerne og optimeringen. Vektorer og matricer er som skabt til det formål, og vi vil vise, hvordan neurale netværk kan trænes med matrix-regning.

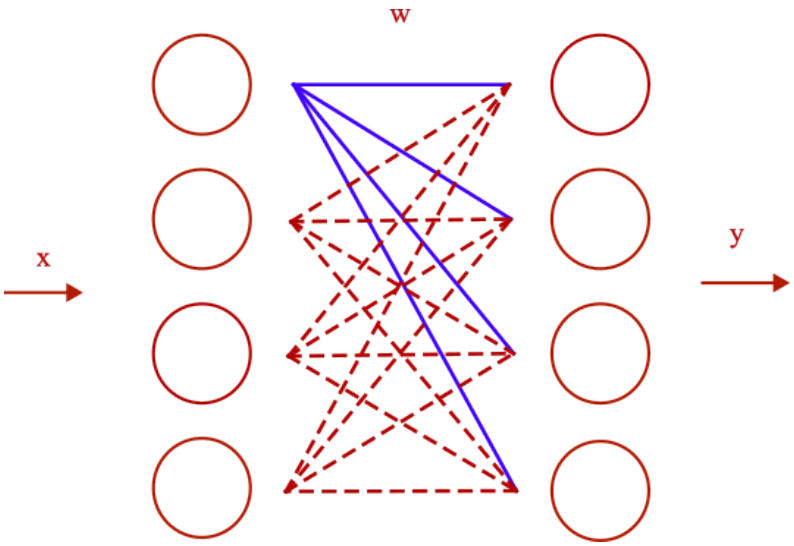
3.1 Matrix-beskrivelse af netværk

Figur 11 viser et neuralt netværk med to input og to output. Alle input er forbundet til alle output, og det samlede antal vægte er $2 \cdot 2 = 4$



Figur 11: neuralt netværk med to input og to output

Figur 12 viser et neuralt netværk med fire input og fire output. Alle input er forbundet til alle output, og de blå streger illustrerer, at input x_1 er forbundet til output y_1, y_2, y_3, y_4, og de stiplede linjer er de andre forbindelser. Det samlede antal vægte er $4 \cdot 4 = 16$



Figur 12: neuralt netværk med fire input og fire output

Hvis man har I input og O output i et netværk er antallet af vægte $I \cdot O$. Hvis man skal til at genkende billeder giver selv et lille billede på 250 pixels et antal vægte på $250 \cdot 250 = 62500$.

For at holde styr på det, skal vi bruge vektorer og matricer. Vi starter med det lille netværk med to input og to output, men resultaterne kan bruges ved alle størrelser af netværk.

3.1.1 Vektor og matrix-notation

Input kan skrives som en vektor $\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ og output som en vektor $\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$. Vektoren $\vec{x}$ indeholder netop de to input (x_1, x_2), og vektoren $\vec{y}$ indeholder de to output (y_1, y_2). Generelt for vektorer skrives de med en pil over og som en søjle.

For at forbinde input og output skal vi bruge en matrix med vægtene. En matrix er en udvidelse af vektorer, og vægtmatricen skrives som

$$W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}.$$

Den indeholder altså alle de fire vægte, vi skal optimere i det neurale netværk. Måden man finder output på, er ved at gange matricen og inputvektoren.

3.1.2 Matrix virkende på en vektor

En matrix, W, virker på vektoren $\vec{x}$ fra venstre ved at prikke rækkerne i matricen med søjlen i vektoren,

$$W \cdot \vec{x} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{pmatrix} = \vec{y}.$$

Derved kommer der en ny vektor med samme størrelse som $\vec{x}$. Bemærk, at der ikke er en regel, hvis rækkefølgen er omvendt $\vec{x} \cdot W$.

$$\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{pmatrix}$$

Figur 13: matrix på vektor

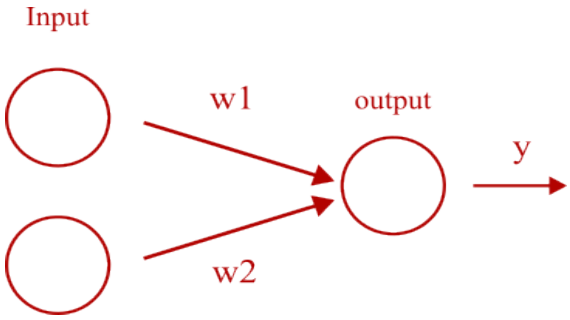
Figur 13 viser, hvordan den øverste del af outputvektoren bliver beregnet. Man tager elementet i 1. række og 1. søjle, w_{11}, og ganger det med første komponent i vektoren, x_1. Det lægger man sammen med elementet fra 1. række og 2. søjle, som man ganger med 2. komponent i vektoren, x_2.

Øvelse 3.1

Vi har trænet et netværk og har en vægtmatrice $W = \begin{pmatrix} 0.5 & 0.5 \\ 1 & 1 \end{pmatrix}$.

- Beregn output $\vec{y}$ med en inputvektor på $\vec{x} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$.
- Beregn output $\vec{y}$ med en inputvektor på $\vec{x} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$.
- Argumentér for, at i outputvektoren $\vec{y} = (y_1, y_2)$ er y_1 gennemsnittet, mens y_2 er summen.
- Hvordan skal vægtmatricen se ud, hvis vi vil have differensen i stedet for summen?

I ovenstående var antallet af input det samme som antallet af output. Det er også muligt at få færre output end input. Lad os se på netværket, vi arbejdede med i afsnit 1.3 med to input og et output.



Figur 14: To input, et output

Vægtmatricen for dette netværk er $W = (w_1, w_2)$, og multiplikationen med inputvektor giver

$$(w_1, w_2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = w_1x_1 + w_2x_2.$$

Ved at gange række og søjle kommer der altså kun ét output.

Øvelse 3.2

Et netværk med 3 input og 2 output har vægtmatricen

$$W = \begin{pmatrix} 1 & 0.5 & 2 \\ 0.5 & 2 & 1 \end{pmatrix}$$

- Beregn output $\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ med en inputvektor på $\vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.

3.1.3 Fejlfunktionen

Vi kan altså bruge matricer og vektorer til at skrive output af et neuralt netværk meget simpelt,

$$\vec{y} = W \cdot \vec{x}$$

Vi skal nu sammenligne med de rigtige y-værdier i vektoren $\vec{y}_0$. Det kan skrives som

$$\vec{\delta} = \vec{y} - \vec{y}_0$$

Fejlfunktionen er ikke en vektor, men summen af de kvadrerede bidrag. Ved et netværk med to output bliver det

$$E = (\delta_1)^2 + (\delta_2)^2$$

Eksempel Udregning af fejl

Vi starter med $\vec{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \vec{y}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, W = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$

Fejlen udregnes som $\vec{\delta} = W \cdot \vec{x} - \vec{y}_0$, hvilket skrevet ud giver

$$\begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.5 \cdot 1 + 0.5 \cdot 0 \\ 0.5 \cdot 1 + 0.5 \cdot 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 0.5 \end{pmatrix}$$

Fejlfunktionen giver $L = (-0.5)^2 + 0.5^2 = 0.25 + 0.25 = 0.5$.

I større neurale netværk med n output skrives fejlfunktionen med sum tegnet

$$E = \sum_{t=1}^{n} \delta_t^2$$

I vores eksempel er $n = 2$, og summen bliver

$$E = \sum_{t=1}^{2} \delta_t^2 = \delta_1^2 + \delta_2^2$$

3.2 Gradienten og optimering af neurale netværk

Vi optimerer vores netværk ved at minimere fejlfunktionen L med hensyn til vægtene w_t. Fejlfunktionen er en funktion af flere variable, $L(w_1, w_2, w_3, \dots)$. Optimeringen sker ved at beregne gradienten af L og opdatere vægtene.

3.2.1 Gradienten

I et neuralt netværk med to vægte svarer fejlfunktionen til en funktion af to variable. Vi gennemgår det generelt som en funktion af variablene x og y, men generaliseringen til n vægte er lige til. Gradienten til en funktion $f(x, y)$ af variablene x og y defineres som

$$\nabla f(x, y) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}.$$

Det krøllede ∂ betyder, at man differentierer med hensyn til én variabel og holder de andre konstante. Det kaldes den partielt afledte. Hvis funktionen er $f(x,y) = a \cdot x + y$, så er $\frac{\partial f}{\partial x} = a$, mens $\frac{\partial f}{\partial y} = 1$.

Sætning
Gradienten er en vektor, som peger i den retning, hvor funktionen vokser hurtigst.

Bevis (*se også afsnit III*)
Lad $f'_u(x_0, y_0)$ være funktionstilvæksten i retningen u i punktet (x_0, y_0). Det angiver altså, hvor meget funktionen vokser i en bestemt retning, bestemt af vektoren $\vec{u} = \binom{a}{b}$. Som generelt ved vektorer giver a, hvor langt $\vec{u}$ bevæger sig langs x-aksen, og b, hvor langt $\vec{u}$ bevæger sig langs y-aksen. De afledte langs x- og y-aksen er netop de partielt afledte

$$f'_x(x_0, y_0) = \frac{\partial f}{\partial x},$$
$$f'_y(x_0, y_0) = \frac{\partial f}{\partial y}.$$

Funktionstilvæksten i en vilkårlig retning er

$$f'_u(x_0, y_0) = \frac{\partial f}{\partial x} \cdot a + \frac{\partial f}{\partial y} \cdot b = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \cdot \binom{a}{b} = \nabla f(x_0, y_0) \cdot \binom{a}{b}$$

f'_u er størst, når vektor $\vec{u}$ og gradienten peger i samme retning. I 2 dimensioner kan vi se dette ved omskrive skalarproduktet med brug af cosinus til vinklen mellem de to vektorer: $\vec{a} \cdot \vec{b} = |a| \cdot |b| \cdot \cos(v)$, hvor || angiver længderne, og v vinklen mellem dem. Cosinusfunktionen har sit maksimum, når $v = 0$, hvilket viser, at f'_u har sit maksimum i samme retning som gradienten.

For et *n*-dimensionalt rum vil man kunne indføre vinklen mellem vektorerne som i 3D, og dernæst argumentere for at samme sætning gælder. Men føler man sig lidt på gyngende grund mht. argumentationer i rum, vi ikke kan forestille os, så er der en meget elegant sætning om skalarprodukt og længder af vektorer, den såkaldte Cauchy-Schwartz sætning, der giver os resultatet. Dette er gennemført i del III.

Øvelse 3.3
En funktion af to variable er givet ved $f(x,y) = 2 \cdot x^2 + (y-1)^2$

- Tegn grafen for funktionen, og vurder om den har et minimum, og hvor det er.
- Beregn gradienten i punktet $(1{,}1)$. Hvilken vej peger vektoren?
- Beregn gradienten i punkterne $(1{,}3)$ og $(-1{,}1)$. Hvilken vej peger vektoren?
- Beregn gradienten i punktet $(0{,}1)$. Hvor ligger det på grafen, og er der noget specielt ved det punkt?

For optimering er vi mere interesserede i, hvornår f'_u er mindst, og det sker, når $v = 180°$, hvor $\cos(180°) = -1$. Vektor $\vec{u}$ giver bare retningen, og vi kan vælge, at den har en længde på 1. Hvis vi ser på fejlfunktionen og vil bevæge os ned, svarer det til

$$f'_u(x_0, y_0) = |\nabla E(x_0, y_0)| \cdot |u| \cdot \cos(180) = -|\nabla E(x_0, y_0)|.$$

3.2.2 Optimering af neuralt netværk

Vi kigger igen på netværket med to input og to output i fig. 11 med vægtmatrix $W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$ og fejl $\vec{\delta} = \vec{y} - \vec{y}_0 = W \cdot \vec{x} - \vec{y}_0 = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$.

Fejlfunktionen er $L = 0.5 \cdot (\delta_1^2 + \delta_2^2) = 0.5 \cdot ((w_{11}x_1 + w_{12}x_2 - y_1)^2 + (w_{21}x_1 + w_{22}x_2 - y_2)^2)$.

De partielt afledte er

$$\frac{\partial L}{\partial w_{11}} = (w_{11}x_1 + w_{12}x_2 - y_1) \cdot x_1 = \delta_1 \cdot x_1$$
$$\frac{\partial L}{\partial w_{12}} = (w_{11}x_1 + w_{12}x_2 - y_1) \cdot x_2 = \delta_1 \cdot x_2$$
$$\frac{\partial L}{\partial w_{21}} = (w_{21}x_1 + w_{22}x_2 - y_2) \cdot x_1 = \delta_2 \cdot x_1$$
$$\frac{\partial L}{\partial w_{22}} = (w_{21}x_1 + w_{22}x_2 - y_2) \cdot x_2 = \delta_2 \cdot x_2.$$

De to øverste udtryk er altså fejlen ved output y_1 ganget med inputværdierne. Det samme for de to nederste er output y_2.

Fejlfunktionen er en funktion af 4 variable, og vægtene bliver opdateret ved

$$\begin{pmatrix} w_{11_{ny}} \\ w_{12_{ny}} \\ w_{21_{ny}} \\ w_{22_{ny}} \end{pmatrix} = \begin{pmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{pmatrix} - \epsilon \begin{pmatrix} \delta_1 \cdot x_1 \\ \delta_1 \cdot x_2 \\ \delta_2 \cdot x_1 \\ \delta_2 \cdot x_2 \end{pmatrix}$$

3.2.3 Ydre produkt (outer product) i vægtopdatering

Man kan nu vælge at lave en løkke, hvor man kører alle vægte igennem. Hver vægt skal opdateres i forhold til

$$w_{ij,ny} = w_{ij} - \epsilon\delta_i x_j,$$

hvor $i = \{1,2\}, j = \{1,2\}$. F.eks. $i = 1, j = 2 \rightarrow w_{12,ny} = w_{12} - \epsilon\delta_1 x_2$.

Det kan også gøres med matricer, hvor vi opdaterer vægtmatricen i et neuralt netværk. Vi kan udtrykke opdateringen som *et ydre produkt* mellem fejlvektoren $\vec{\delta}$ og inputvektoren $\vec{x}$. Det ydre produkt mellem to vektorer $\vec{\delta}$ og $\vec{x}$ skrives som:

$$\vec{\delta} \otimes \vec{x} = \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} (x_1 \quad x_2) = \begin{pmatrix} \delta_1 x_1 & \delta_1 x_2 \\ \delta_2 x_1 & \delta_2 x_2 \end{pmatrix}$$

I opdateringsreglen for vægtmatricen bruges dette ydre produkt:

$$\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}_{ny} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} - \epsilon \begin{pmatrix} \delta_1 x_1 & \delta_1 x_2 \\ \delta_2 x_1 & \delta_2 x_2 \end{pmatrix}$$

Øvelse 3.4

Et neuralt netværk med to input og to output er givet ved

$$W = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}, W_{optimal} = \begin{pmatrix} 0.5 & 0.5 \\ 1.0 & -1.0 \end{pmatrix}, \vec{x} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

- Beregn det optimale output $\vec{y}_0 = W_{optimal} \cdot \vec{x}$ og det estimerede $\vec{y} = W \cdot \vec{x}$.
- Beregn fejlvektoren $\vec{\delta} = \vec{y} - \vec{y}_0$.

- Beregn det ydre produkt $\vec{\delta} \otimes \vec{x}$.
- Find den nye vægtmatrice $W_{ny} = W - \epsilon \cdot \vec{\delta} \otimes \vec{x}$

Eksempel: Optimering i Python
Nedenstående Python-kode optimerer det samme netværk.

```
import numpy as np
W = np.array([[0.5,0.5],
              [0.5,0.5]]) # startvægtene
W_optimal = np.array([[0.5,0.5],
                      [1.0,-1.0]]) # den optimale efter træning
x = np.array([2,3]) # x-værdier
y = np.dot(W_optimal, x) # y-værdier optimalt
epsilon = 0.1
for i in range(60):
    d2 = np.dot(W, x) - y # fejlen
    W = W - epsilon *  np.outer(d2, x) # opdatering af vægte
print('y estimeret', np.dot(W, x))
```

Øvelse 3.5

- Gå ind på https://www.online-python.com/, kopiér koden ind, og kør den.
- Hvor godt passer de estimerede y-værdier med de beregnede i øvelsen ovenfor?

Python bruger **biblioteket numpy** til at lave matrixberegninger. Hver gang man skriver np, bruges dette bibliotek. De tre funktioner er np.array() til at lave matricer og vektorer, np.dot() til at lave matrixprodukt med vektor og np.outer() til at lave det ydre produkt.

- Gennemgå koden, og overbevis jer om, at den gør det samme som blev gennemgået matematisk.
- Lav om i antallet af iterationer fra 60 til 20, og tjek resultatet.

3.3 Klassifikation af iris-blomster

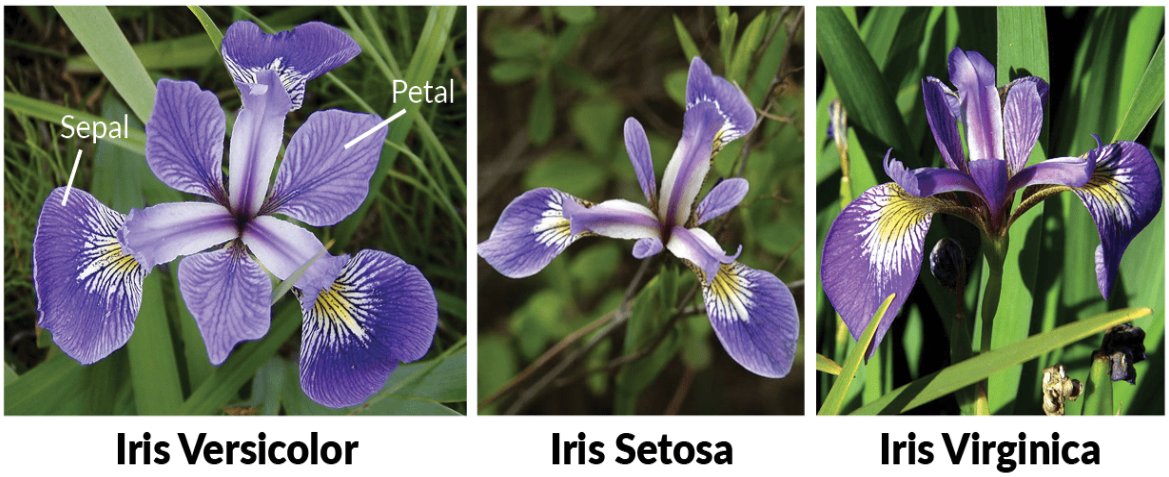
Nu sætter vi fokus på mønstergenkendelse. I første omgang vil vi prøve at træne et neuralt netværk til at klare klassifikationen af forskellige iris-blomster.

Der findes tre slags iris-blomster kaldet Iris Versicolor, Iris Setosa og Iris Virginica. Deres blomster er lidt forskellige, og her skal vi arbejde med et datasæt, hvor længden og bredden af bladene er blevet målt. Vi vil prøve manuelt at lave klassifikationen og se, hvordan den passer med en simpel optimeret maskinlæring.

Vores kunstige intelligens skal kunne bestemme arten (target) af iris-blomster baseret på deres karakteristika (features). Det er ikke et datasæt med billeder af blomsterne, men data om målene af bladene.

- Feature: længden og bredden af hhv. kronblade (petal) og bægerblade (sepal).
- Target: arten.

De tre arter er vist på billedet her:



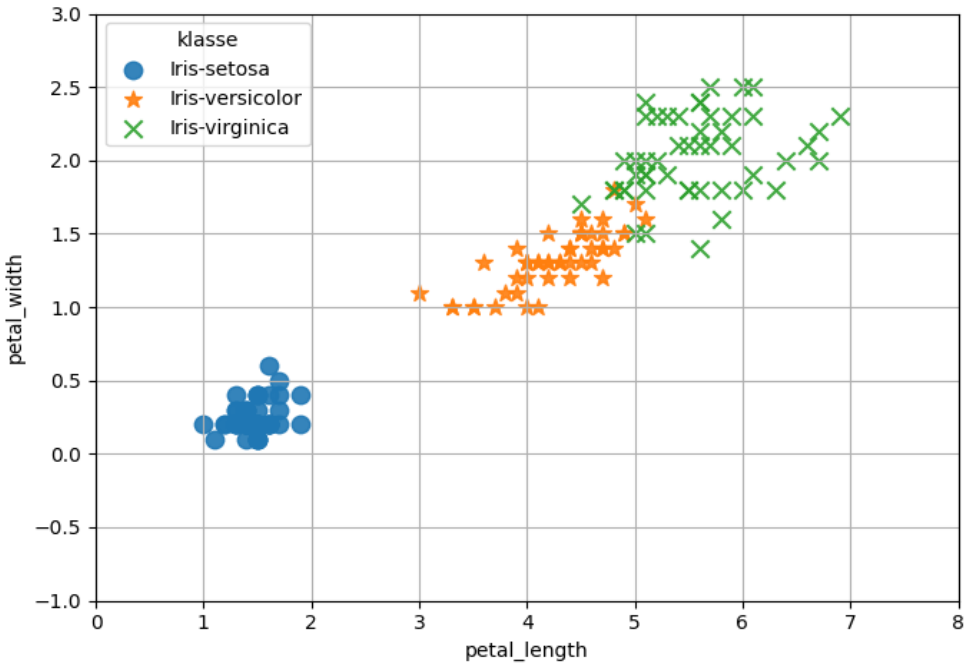
Figur 15: Iris-blomster

Kronbladene har den største variation. Dem bruger vi til klassifikationen.

3.3.1 Manuel sortering

Der er et datasæt tilgængeligt om netop de tre typer af Iris blomster. Du kan hente datasætter *her* (link til filen: II.3a - Iris data)

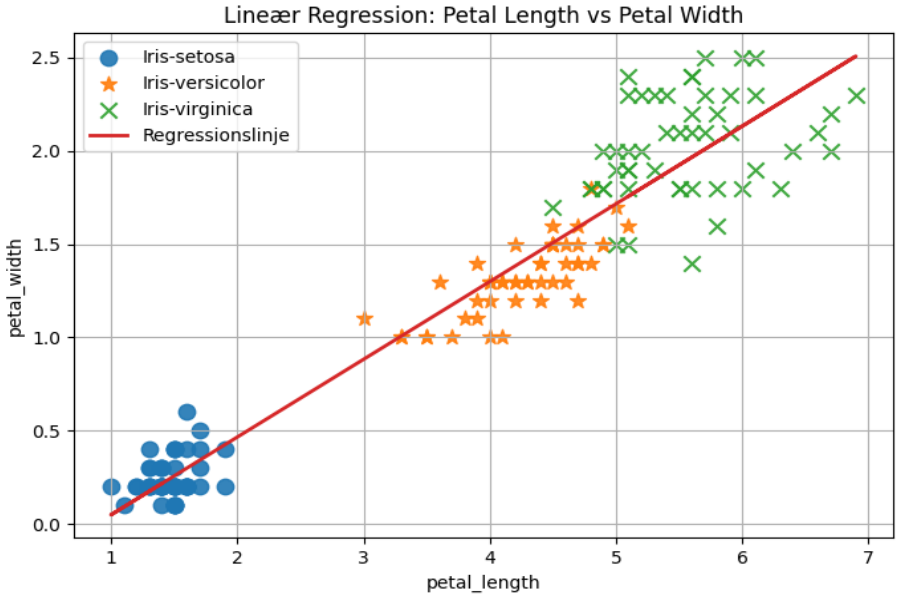
For at få et overblik over data skal du fremstille et punktplot. Du skal få noget der ligner figur 16 her:



Figur 16: Iris petal længde og bredde

I de lineære modeller, vi har arbejdet med, klassificerer / eller sorterer vi ved at trække den bedste rette linje gennem datapunkterne og dernæst lave intervaller langs den. Hvis man har mange variable, kan det være en hyperplan i mange dimensioner. Vi holder os til to dimensioner, som vi kan tegne.

Figur 17 viser en lineær regressionslinje gennem datapunkterne.



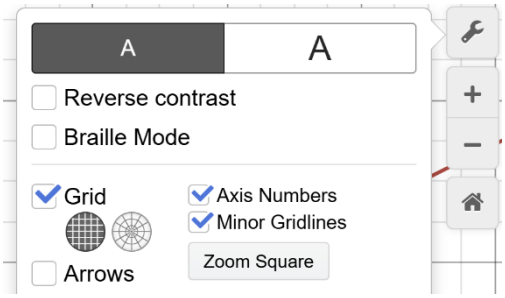
Figur 17: Iris petal længde og bredde med regressionslinje

Vi vil nu *manuelt* sortere i tre grupper. Vi tager udgangspunkt i regressionslinen, fordi den tager hensyn til både længde og bredde.

Vi vælger at sortere data vha. linjer vinkelret på regressionslinjen. Vi kan se, at det er en simpel sag at skille de blå fra, men vi kan ikke indgå et overlap mellem de orange og de grønne.

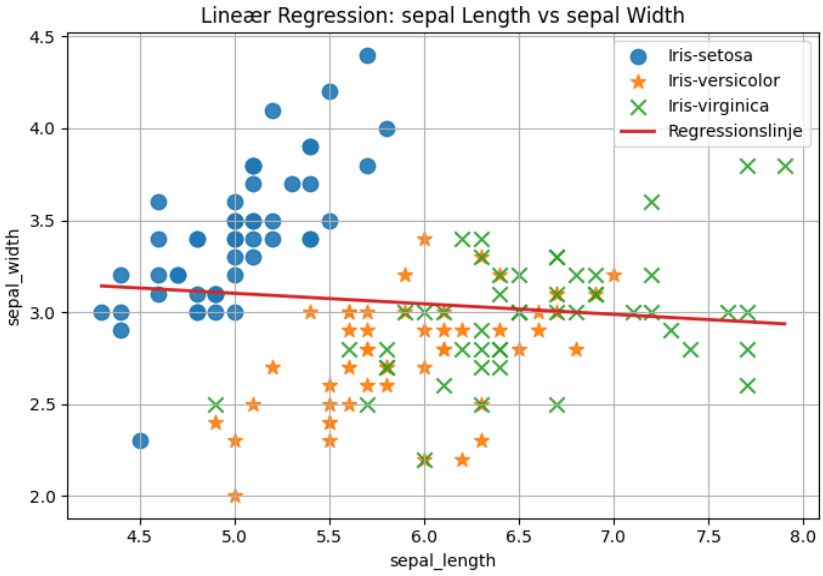
Øvelse 3.6

Data og skydere til to vinkelrette linjer er her: [Desmos Iris Petal](#).



- Lav om i størrelsen af browservinduet, til felterne i koordinatsystemet har samme størrelse, ellers ser det ikke ud til, at linjerne er vinkelrette. (*Gå ind i værktøjer og tryk på zoom square*)
- Zoom ud så meget du kan, hvor du bevarer alle datapunkterne
- De vinkelrette linjer hedder *a* og *b* (det er ikke hældning og skæring). Flyt dem ved at trække i skyderne, indtil du har fået den opdeling, du vurderer, er den skarpeste.
- Hvis en grøn er i den orange gruppe er det en fejl, og tilsvarende omvendt. Optæl antallet af fejl, og angiv det i procent af totalen på 150 datapunkter.

Man kunne også vælge at sortere efter bægerbladene kaldet sepal. De er vist i fig. 18.



Figur 18: Iris sepal regression

Øvelse 3.7

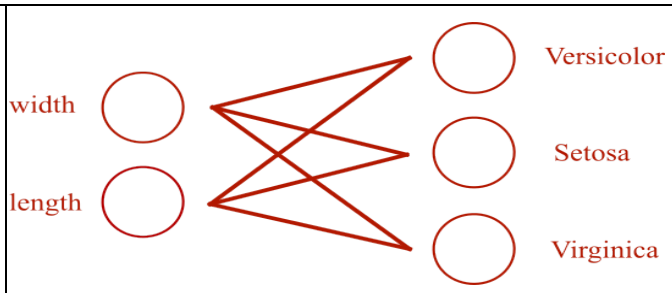
Data og skydere til to vinkelrette linjer er her: Desmos Iris Sepal.

- Lav om i størrelsen af browservinduet som i øvelse 3.6
- Flyt de vinkelrette linjer som i øvelse 3.6, indtil du har fået den opdeling, du vurderer, er den skarpeste.
- Hvorfor tror du, at det er meget sværere at klassificere ud fra bægerbladene?

3.3.2 Klassifikation vha. et neuralt netværk

Hvis vi skal lave en model for iris-datasættet, ser det ud som i fig. 19.

$$W = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \end{pmatrix}, \vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$



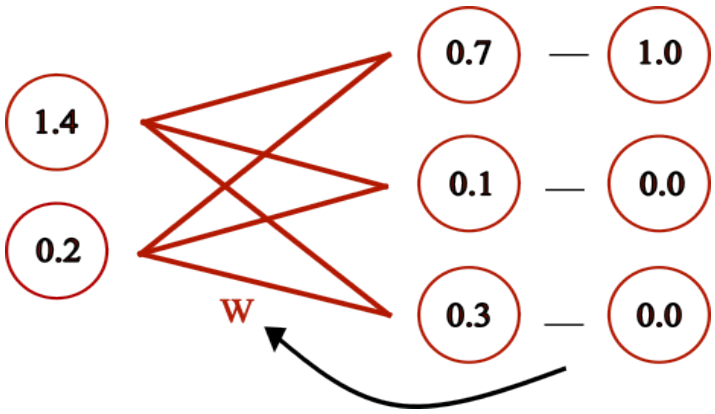
Figur 19: Iris-model

Vi har to input og tre output.

Det første datapunkt giver

$$\vec{x} = \begin{pmatrix} 1.4 \\ 0.2 \end{pmatrix}, \vec{y} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

Optimeringen sker ved at gange inputvektorerne på vægtmatricen, hvilket giver et estimeret output, som er omregnet til en sandsynlighed. Forskellen på den estimerede sandsynlighed og den kendte beregnes, og vægtene opdateres, hvilket kan ses i fig. 20.



Figur 20: forward propagation

Optimeringen kører nu over to løkker: den yderste, som angiver, hvor mange gange netværket skal trænes, og den inderste, som kører over alle 150 datapunkter.

```
for i in range(iterationer):
  for j in range(n):
    y0 = np.zeros(3)                    # sætter alle tre output til 0
    y0[y[j]] = 1                        # sætter den ønskede til 1 (100%)
    data = np.array([x1[j], x2[j], 1.0]) # loader data + bias 1.0
    probs = softmax(W.dot(data))        # laver de estimerede y om til et tal
                                        mellem 0-1
    d = probs-y0                        # beregner fejlen
    W = W - epsilon*np.outer(d, data)   # opdaterer wægtene
```

Resultatet af det neurale netværk efter 100 iterationer er 146 rigtige og 4 forkerte. Hele programmet med data ligger på hjemmesiden, iris_model_data.py, og kan køres på https://www.online-python.com/.

Øvelse 3.8

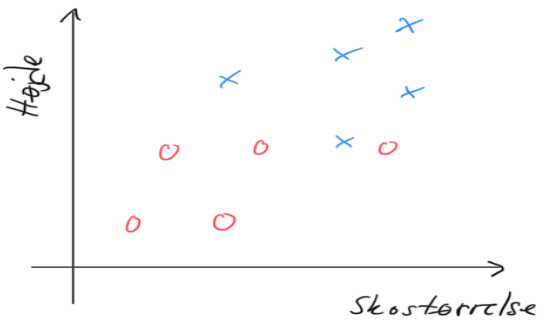
a) Sammenlign med jeres egen klassificering. Hvor mange rigtige og forkerte fik I?

b) Beregn procenten af rigtige og af forkerte. Er det ok performance af modellen?

Øvelse 3.9 Sko, højde og køn

Vi vil lave en model, som kan afgøre folks køn med input af skostørrelse og højde.

Figuren viser et koordinatsystem med sko-størrelse ud af x-aksen og højde ud af y-ak-sen. Krydser repræsenterer drenge mens cirkler repræsenterer piger.



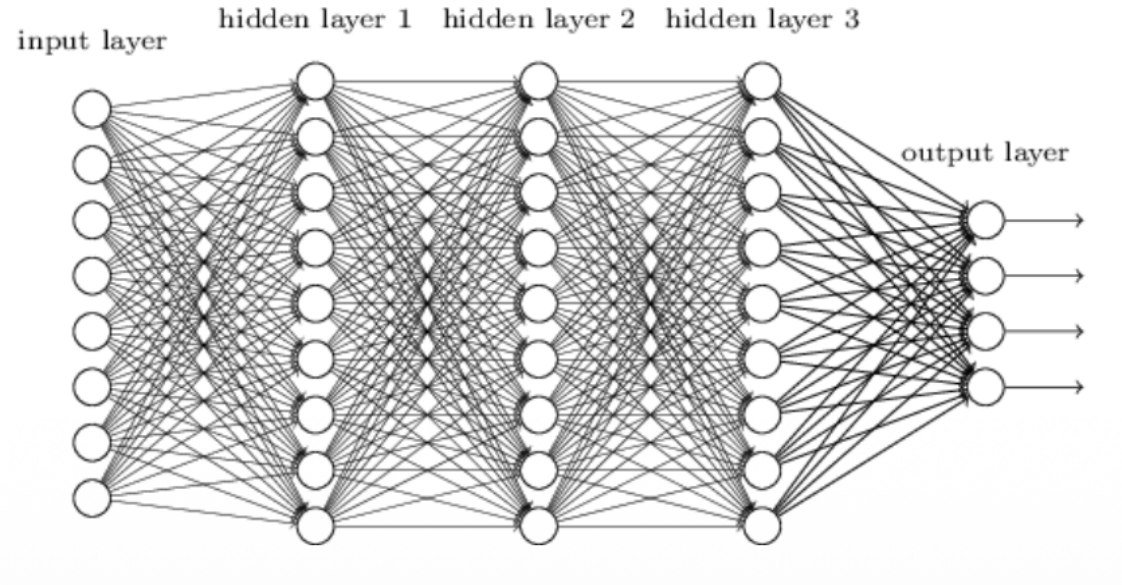
Anvend klassens egne skostørrelser og højde.

- Skriv jeres egne data i et regneark.
- Lav lineær regression.
- Indtegn en decision line vinkelret på.

Optæl antal rigtige og forkerte, og beregn, hvor korrekt modellen er.

4. Dybe neurale netværk

Når man taler om dybe neurale netværk, menes der netværk med mange lag. fig. 22 viser et neuralt netværk med et inputlag med 8 input, 3 skjulte lag med hver 9 knuder og et outputlag med 4 output. De 8 input aktiverer de 9 knuder i lag 1, som aktiverer de 9 knuder i lag 2 osv.



Figur 22: dybt neurale netværk

Det *samlede antal vægte* kan beregnes som

$$8 \cdot 9 + 9 \cdot 9 + 9 \cdot 9 + 9 \cdot 4 = 270.$$

For hvert lag er der en *vægtmatrix*, og det samlede netværk kan skrives som

$$W_1 = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \dots \\ w_{21} & w_{22} & w_{23} & \dots \\ \vdots & \vdots & & \vdots \end{pmatrix}_{9\times 8}, W_2 = \begin{pmatrix} v_{11} & v_{12} & v_{13} & \dots \\ v_{21} & v_{22} & v_{23} & \dots \\ \vdots & \vdots & & \vdots \end{pmatrix}_{9\times 9}$$

$$W_3 = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots \\ u_{21} & u_{22} & u_{23} & \dots \\ \vdots & \vdots & & \vdots \end{pmatrix}_{9\times 9}, W_4 = \begin{pmatrix} k_{11} & k_{12} & k_{13} & \dots \\ k_{21} & k_{22} & k_{23} & \dots \\ \vdots & \vdots & & \vdots \end{pmatrix}_{4\times 9},$$

hvor 9×8 viser, at W_1 har 9 rækker og 8 søjler, mens 4×9 har 4 rækker og 9 søjler.

Inputvektoren er

$$\vec{x}_1 = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix}_{8\times 1}$$

Den nye vektor i første skjulte lag beregnes som

$$\vec{x}_2 = W_1 \cdot \vec{x}_1.$$

Størrelsen af $\vec{x}_2$ kan ses ud fra størrelsen af W_1 og $\vec{x}_1$:

$$[9 \times 8] \cdot [8 \times 1] \rightarrow [9 \times 1].$$

Reglen er, at de to inderste (her 8) skal være de samme, og dimensionen er de to yderste $[9 \times 1]$.

Øvelse 4.1

Givet matricen $A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \end{pmatrix}$ og vektoren $\vec{b} = \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$

- Bestem dimensionerne som (antal rækker, antal søjler) for matricen og vektoren.
- Bestem dimensionen af produktet $A\vec{b}$.
- Beregn produktet, og tjek om dimensionen passer til ovenfor.
- Forklar, hvorfor antallet af søjler i matricen skal være det samme som antallet af rækker i vektoren.

Øvelse 4.2

Det neurale netværk ser kompliceret ud, men vi kan starte med at se, om dimensionerne passer. Vi viste ovenfor, at $W_1 \cdot \vec{x}_1$ gav en $[9 \times 1]$ vektor.

- Vis, at det næste lag neuroner også er en $[9 \times 1]$ vektor.
- Vis, at outputlaget netop er en $[4 \times 1]$ vektor.

Lad os se på netværket igen *uden at beregne det*:

$$\vec{x}_2 = W_1 \cdot \vec{x}_1, \text{ og } \vec{x}_3 = W_2 \cdot \vec{x}_2 = W_2 \cdot W_1 \cdot \vec{x}_1.$$

Vi kan altså få neuron nr. 3 direkte uden lag 2 ved $\vec{x}_3 = W_2 \cdot W_1 \cdot \vec{x}_1$. Der gælder samme regler for at gange matricer, nemlig at de midterste dimensioner skal være ens. Lad os tjekke med dimensionerne:

$$[9 \times 9][9 \times 8] \rightarrow [9 \times 8].$$

Vi kan altså godt gange dem sammen direkte, og dimensionen af den matrix, der kommer ud af det, passer med inputvektoren.

Øvelse 4.3

Vis, at outputvektoren med dimension $[4 \times 1]$ kan laves ud fra inputvektoren ved $W_4 \cdot W_3 \cdot W_2 \cdot W_1 \cdot \vec{x}_1$ ved at tjekke dimensionerne.

Dette er et problem for neurale netværk, for det betyder, at et dybt neuralt netværk kan reduceres til et neuralt netværk med kun ét lag.

Problemet opstår, fordi produktet af to matricer også er en matrix. For at lave dybe neurale netværk har man brug for noget ikke-lineært, hvilket kommer gennem den ikke-lineære aktiveringsfunktion.

5. Universal approksimator

Jes Frellsen forklarer, hvordan et neuralt netværk kan være en universel approksimator for kontinuerte funktioner. Det betyder, at vi kan anvende neurale netværk i stedet for funktioner. Hvis vi kender funktionen, er det både nemmere og mere præcist at bruge den, men neurale netværk kan bruges, når funktionen ikke er kendt. Det kan være, at data følger en af vores standardfunktioner, men at vi bare ikke ved hvilken eller hvilken

sammensætning. Det kan også være, at vi slet ikke har en funktion til at beskrive vores data. I begge tilfælde kan vi bruge neurale netværk til at approksimere en funktion.

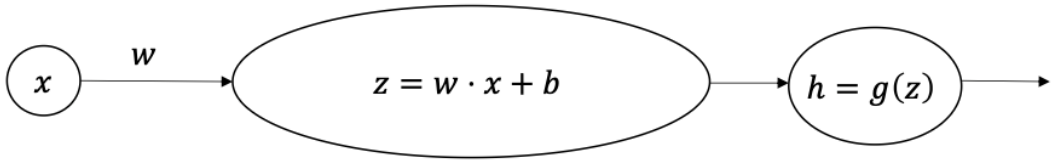
Det kan vises, at neurale netværk med kun et enkelt skjult lag kan virke som en universel approksimator. I dette afsnit skal vi se på hvordan backpropagation med et skjult lag virker og bruge det på flydata.

5.1 Neuralt netværk med ét input og ét output

Vi vil se på funktioner af én variabel som vi kender som $y = f(x)$, med de to variable x og y

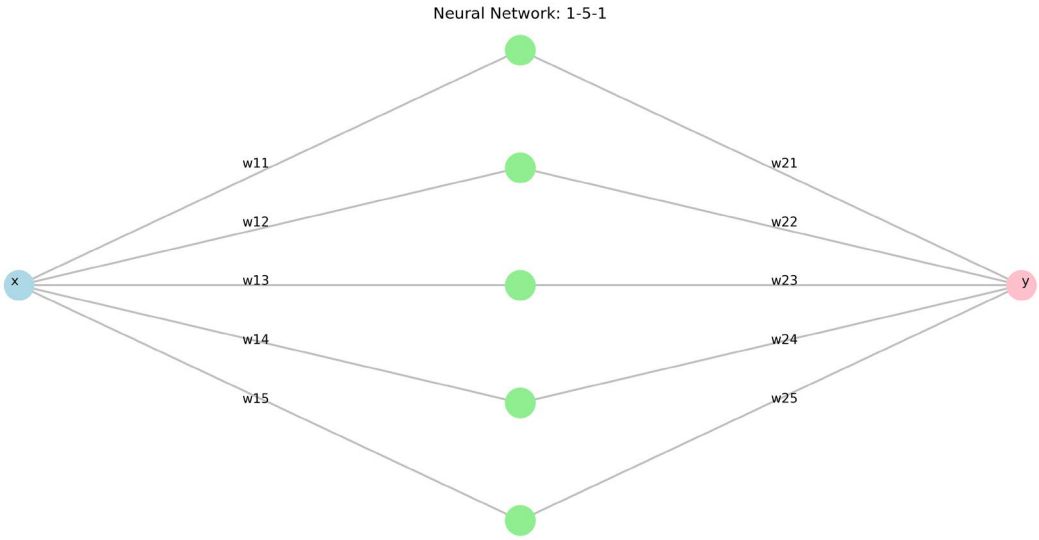
Vi kommer til at bruge sigmoid funktionen $g(z) = \frac{1}{1+e^{-z}}$ som aktiveringsfunktion og tilføje en bias til vores data.

Figur fig. 23 viser setuppet for den enkelte neuron



Figur 23: Aktivering af én neuron

mens fig. 24 viser hele netværket



Figur 24: 1-5-1 netværk

med 5 neuroner i det skjulte lag bliver antallet af vægte der skal estimeres 10.

De to vægtmatricer kan skrives som

$$W_1 = \begin{pmatrix} w_{11} \\ w_{12} \\ w_{13} \\ w_{14} \\ w_{15} \end{pmatrix}, W_2 = (w_{21}, w_{22}, w_{23}, w_{24}, w_{25})$$

Det skjulte lag h findes ved

$$h = g(z) = g(W_1 \cdot x + b) = g\begin{pmatrix} w_{11} \cdot x + b_1 \\ w_{12} \cdot x + b_2 \\ w_{13} \cdot x + b_3 \\ w_{14} \cdot x + b_4 \\ w_{15} \cdot x + b_5 \end{pmatrix} = \begin{pmatrix} g(w_{11} \cdot x + b_1) \\ g(w_{12} \cdot x + b_2) \\ g(w_{13} \cdot x + b_3) \\ g(w_{14} \cdot x + b_4) \\ g(w_{15} \cdot x + b_5) \end{pmatrix}$$

Den ikke lineære $g(z) = \frac{1}{1+e^{-z}}$ bruges på hver række i matricen. Til outputtet bruges den lineære funktion $g(z) = z$ og output beregnes som

$$y = (w_{21}, w_{22}, w_{23}, w_{24}, w_{25}) \cdot h = w_{21} \cdot h1 + w_{22} \cdot h2 + w_{23} \cdot h3 + w_{24} \cdot h4 + w_{25} \cdot h5 + b.$$

For overskuelighedens skyld har jeg lagt alle bias sammen $b = b1 + b2 + \cdots$.

Hvis vi skal se på dimensionerne i vores netværk, er de $W_1: [5 \times 1]$, $x: [1 \times 1]$, $h: [5 \times 1]$, $W_2: [1 \times 5]$ og $y: [1 \times 1]$

Øvelse 5.1

Matricer kan ganges sammen hvis dimensionen mellem dem er ens.

$[k \times m][m \times j]$ kan ganges da $m = m$ og den matrix der kommer ud af det er en $[k \times j]$

a) Brug det til at tjekke dimensionerne i netværket.

5.2 Backpropagation

I det indledende afsnit 1.2 om optimering i neurale netværk viste vi, at hvis vi har en simpel lineær model med loss-funktionen

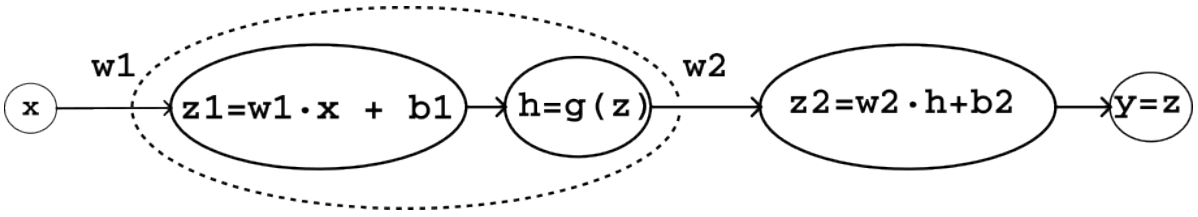
$$L = 1/2\,(w \cdot x - y)^2,$$

så fås opdateringen af vægten som

$$w_{ny} = w - \epsilon \cdot \frac{\partial L}{\partial w} = w - \epsilon \cdot (\hat{y} - y)x = w - \epsilon \cdot \delta x,$$

hvor $\delta = \hat{y} - y$ beskriver fejlens størrelse i outputlaget.

Hvis vi nu udvider modellen til et netværk med et inputlag, ét skjult lag og et outputlag, kan det skitseres som vist i figur fig. 25.



Figur 25: Skematisk hidden layer

Den stiplede ellipse markerer det skjulte lag med input z_1 og output $h = g(z_1)$, hvor g er en aktiveringsfunktion (for eksempel en sigmoid). Outputfunktionen i dette eksempel er blot identiteten, så $\hat{y} = z_2$.

Vi kan beskrive netværket matematisk som:

$$z_1 = w_1 \cdot x + b_1, \qquad h = g(z_1), \qquad z_2 = w_2 \cdot h + b_2, \qquad \hat{y} = z_2.$$

resultatet for opdatering af vægtene w_1 viser vi nedenfor bliver

$$w_{1,ny} = w_1 - \epsilon\, \delta_1\, x = w_1 - \epsilon\, g'(z_1)\, w_2\, \delta_2\, x,$$

5.2.1 Udledning af opdatering af vægtene

For outputlaget følger opdateringen samme princip som i det lineære eksempel:

$$w_{2,ny} = w_2 - \epsilon \cdot \frac{\partial L}{\partial w_2}.$$

Da

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial w_2} = (\hat{y} - y)h = \delta_2 h,$$

får vi:

$$w_{2,ny} = w_2 - \epsilon\, \delta_2\, h,$$

hvor $\delta_2 = \hat{y} - y$ er fejlens størrelse i outputlaget.

Fejlen i det skjulte lag

For at opdatere vægten w_1 i det skjulte lag skal vi bruge kædereglen:

$$w_{1,ny} = w_1 - \epsilon \cdot \frac{\partial L}{\partial w_1}.$$

Vi finder først:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial w_1}.$$

Den anden faktor er simpel:

$$\frac{\partial z_1}{\partial w_1} = x.$$

Til gengæld kræver den første faktor $\frac{\partial L}{\partial z_1}$, at vi følger fejlen baglæns gennem netværket. Ved at bruge kædereglen igen får vi:

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial z_1}.$$

Her gælder:

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial h} = \delta_2\, w_2,$$

og

$$\frac{\partial h}{\partial z_1} = g'(z_1).$$

Dermed får vi:

$$\frac{\partial L}{\partial z_1} = g'(z_1)\, w_2\, \delta_2.$$

Vi definerer denne størrelse som:

$$\delta_1 = g'(z_1)\, w_2\, \delta_2,$$

som er den fejl, der *sendes tilbage* til det skjulte lag.

Til sidst kan vi skrive:

$$\frac{\partial L}{\partial w_1} = \delta_1 \, x,$$

og dermed bliver opdateringen:

$$w_{1,ny} = w_1 - \epsilon \, \delta_1 \, x = w_1 - \epsilon \, g'(z_1) \, w_2 \, \delta_2 \, x.$$

Fortolkning

Udtrykket viser, hvordan fejlen i outputlaget spredes tilbage gennem netværket og justerer vægtene i det skjulte lag.
Funktionen $g'(z_1)$ bestemmer, hvor følsomt dent skjulte neuron reagerer på fejlen, mens w_2 vægter, hvor stærkt denne fejl "sendes tilbage".

Øvelse 5.2

Sigmoid funktinen er givet ved $g(x) = \frac{1}{1+e^{-x}}$

- Vis at $g'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$.
- Vis at det også kan skrives som $g'(x) = g(x) \cdot (1 - g(x))$.

```
import numpy as np
import matplotlib.pyplot as plt

# laver en liste med x og y værdier
x = np.arange(0,10,0.1)
y = np.sin(x)+x

# generer tilfældige startvægte
np.random.seed(10)
W1 = np.random.rand(5,1)*0.01   # (5,1)
W2 = np.random.rand(1,5)*0.01   # (1,5)

# sigmoid funktionen
def f(x):
    return 1/(1+np.exp(-x))

# den afledte sigmoid
def fm(x):
    s = f(x)
    return s*(1-s)

# løkken som kører 200 gange gennem alle datapunkter
epsilon = 0.1
for epoch in range(200):
    y_list = []
    for j in range(len(x)):
        z1 = W1 * x[j]                  # (5,1)
        h = f(z1)                       # (5,1)
        z2 = W2 @ h                     # (1,1)
        y_hat = z2[0,0]                 # skala forudsigelse
        d2 = y_hat - y[j]               # loss w1
```

```
        d1 = (W2.T * d2) * fm(z1)                # (5,1) loss w2

        # gradient updates (shapes must match weights)
        W1 = W1 - epsilon * (d1 * x[j])          # (5,1) <- (5,1) * scalar
        W2 = W2 - epsilon * (d2 * h.T)           # (1,5) <- scalar * (1,5)
        y_list.append(y_hat)

plt.plot(x,y_list,color='red')
plt.scatter(x,y)
plt.show()
print(W1)
print(W2)
```

Filen kan findes på hjemmesiden som fit_nonlinear.py eller køres gennem jupyter filen.

Øvelse 5.3

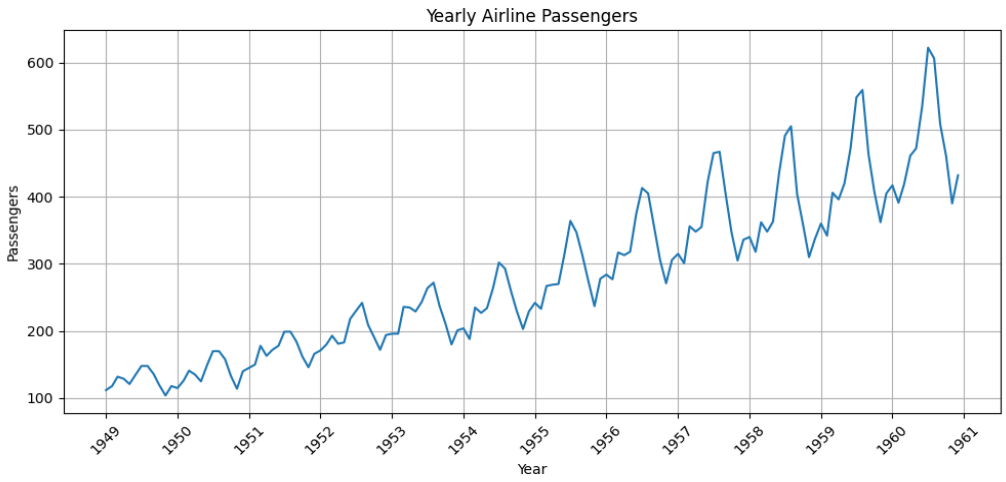
Koden ser lidt lang ud, men følger resultaterne ovenfor.

- Gennemgå linjerne 27-36 og overbevis jer om, at det netop er, hvad vi matematisk har udledt (@ betyder prikprodukt).
- Find de steder i koden, hvor sigmoid funktionen bliver defineret, og hvor den afledte bliver beregnet. I linje 4 defineres den ikke lineære $y = sin(x) + x$
- Prøv med andre funktioner, og se hvor godt modellen passer.
- I linje 10 og 11 bestemmes størrelsen af det skjulte lag til 5. Prøv at gøre det større.

Flydata

Jes Frellsen kommer med et eksempel med flydata. Her tager lineær regression ikke højde for sæsonudsving og det er måske ikke helt lige til at gætte på en funktion til at lave regression.

Figur 26 viser udsvingene.



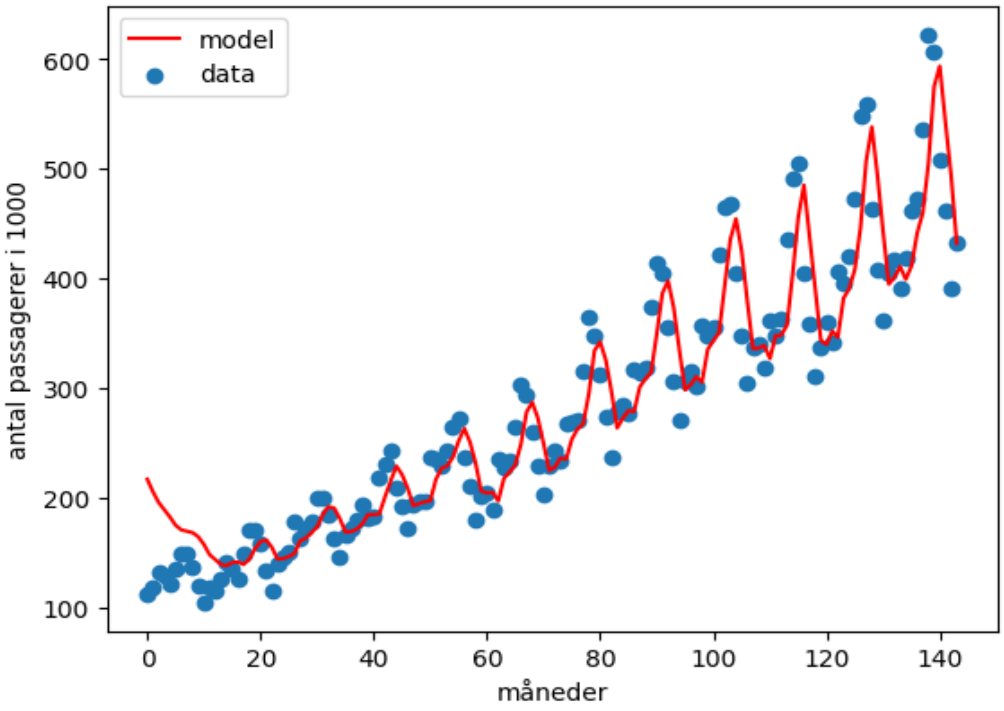
Figur 26: Flypassagerer måneder

Vi vil igen prøve med et simpelt neuralt netværk med ét skjult lag på 60 neuroner.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv('AirPassengers.csv', parse_dates=['Month'])
data.set_index('Month', inplace=True)

y = data['Passengers']
x = np.arange(len(y)).astype(float)
x = x / x.max()

np.random.seed(42)
W1 = np.random.randn(60,1) * 0.05   # (10,1)
W2 = np.random.randn(1,60) * 0.05   # (1,10)

def f(x):
    x = np.clip(x, -500, 500) # undgår meget store værdier
    return 1.0 / (1.0 + np.exp(-x))

def fm(x):
    s = f(x)
    return s*(1-s)

epsilon = 0.01
for epoch in range(400):
    y_list = []
    for j in range(len(x)):

        z1 = W1 * x[j]                  # (10,1)
        h = f(z1)                      # (10,1)
        z2 = W2 @ h                     # (1,1)
        y_hat = z2[0,0]                 # skalar prediction

        d2 = y_hat - y.iloc[j]             # skalar, finder værdien med ilco
        d1 = (W2.T * d2) * fm(z1)          # (10,1)

        # gradient updates (shapes must match weights)
        W1 = W1 - epsilon * (d1 * x[j])      # (10,1) <- (10,1) * skalar
        W2 = W2 - epsilon * (d2 * a1.T)      # (1,10) <- skalar * (1,10)
        y_list.append(y_hat)


plt.Figur()
plt.plot(np.arange(len(y)), y_list, color='red', label='data (original index)')
plt.scatter(np.arange(len(y)), y, label='model'); plt.legend()
plt.savefig("fly_nn.png",bbox_inches='tight')
plt.show()
```

Filen kan findes på hjemmesiden som fly2.py og datasættet AirPassengers.csv eller køres gennem jupyter filen.

Resultatet kan ses i fig. 27, hvor de blå er datapunkter, og den røde kurve forudsigelsen fra det neurale netværk



Figur 27: flypassagerer

Det neurale netværk fanger både, at der er udsving, og at de vokser i styrke og deres middelværdi. Det ser også ud til, at der i starten er noget, hvor den ikke fitter særligt godt. Det kan skyldes, at vi prøver at fitte 120 vægte til et datasæt på 144 datapunkter.

Øvelse 5.4

- Prøv at ændre på størrelsen af netværket og se, hvor godt den fitter data.
- Prøv at loade data ind i Maple og forsøg jer med en kombination af lineære og trigonometriske funktioner.

2

6. Billed og mønster-genkendelse

Billed- og mønster-genkendelse er noget *mennesker* er gode til, og er også en vigtig del af en *kunstig* intelligens. Mennesket har ca. 126 millioner fotoreceptorer i øjet, der opfanger lys og farver, og ca 1 million nerveceller, der sender informationen videre til hjernen. Det er altså under 1% af informationen fra fotoreceptorerne som kommer op i hjernen. Der sker en første billedbehandling i selve øjet, hvor signaler fra fotoreceptorerne bliver komprimeret og bearbejdet, inden det sendes videre til hjernen. Formålet er givetvis at udpege vigtige detaljer og overordnede strukturer i billedet, som hjernen så kan arbejde videre med.

Den billed- og mønstergenkendelse, der foretages af kunstig intelligens, forsøger at efterligne dette. Ønsker man at finde ud af, hvem der er på et givet billede, så foretager programmet ikke en udtømmende søgning på nettets milliarder af sider og svarer, at vedkommende ikke findes, men starter med at udpege vigtige detaljer i billedet. Som feks lodrette, vandrette og skrå linjer i billedet.

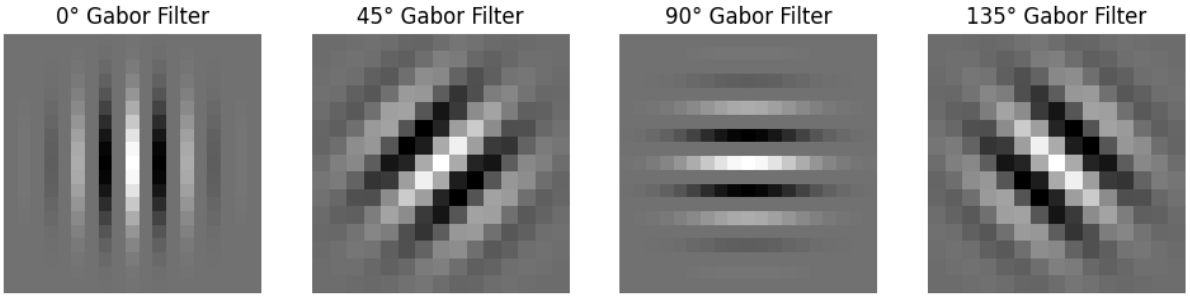
Metoderne bag en sådan "edge-detection" bygger på den matematiske "foldnings-teori", eller på engelsk: convolutional theory. Det er blevet et selvstændigt område af machine learning, hvor der arbejdes med udvikling af convolutional neural networks (CNN). Da det danske ord 'foldning' ikke har meget med det at gøre, som vi foretager os her, så vil vi normalt anvende begrebet *'convolution'*..

Når telefonen genkender ansigter i billeder eller når læger diagnosticerer kræft ud fra røntgenbilleder, er det eksempler på machine learning, der bruger CNN.

Vi kan illustrere den grundlæggende ide via nogle billeder fremstillet af Andreas Aeschlimann og hentet fra hans hjemmeside, Gabor filters.

Originalen er et fotografi af en lampe. Billedet til højre er lavet som summen af fire filtre, lodret, $45°$, vandret og $-45°$. Det er ret tydeligt at se, at "kanterne" bliver fremhævet. Vi kan også få en fornemmelse af, at ved at fortsætte den vej, må vi kunne frembringe et vellignende billede.	original  *figur 28- foto af lampe*	4 filtre  *–og simpel CNN detection*

De fire filtre er illustreret i fig. 29. Figuren viser lyse og mørke striber. Ideen i filtrene er, at hvid svarer til at gange pixelværdien i den originale med 1 og sort med -1.



Figur 29: Gabor filters

Øvelse 6.1 Gabor-filtre

Andreas Aeschlimann har i forbindelse med sit speciale på University of Basel arbejdet med Gabor filtre og har lavet hjemmesiden Gabor filters. På siden kan man loade billeder og lave foldninger med gabor filtre. Der er flere parametre, man kan stille på, men vi holder os til at ændre på vinklen θ. De andre ændrer på styrken og du kan jo prøve dig frem.

- Gå ind på hjemmesiden https://andreas-aeschlimann.github.io/gabor/#/demo.
- Load billedet Church st. Margarethen og generer output.
- Forklar hvorfor ledningerne forsvinder med et lodret filter.
- Lav om på filtret så ledningerne bliver fremhævet.
- Hvad skal vinklen være for at taget bliver fremhævet?

Det er også muligt at uploade egne billeder

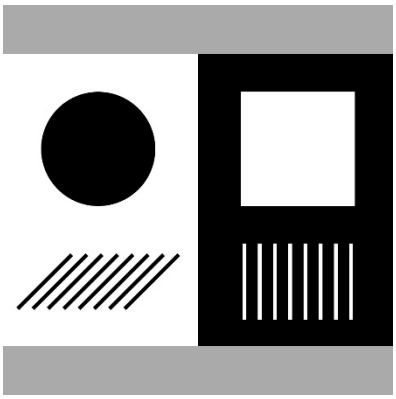
Figur 30: Gabor filter på safari

Øvelse 6.2

- Upload eget billede og leg med filtrene.

Øvelse 6.3 Syntetisk billede

Lad os se på et syntetisk billede, hvor det er lettere at se, hvad filtrene gør.



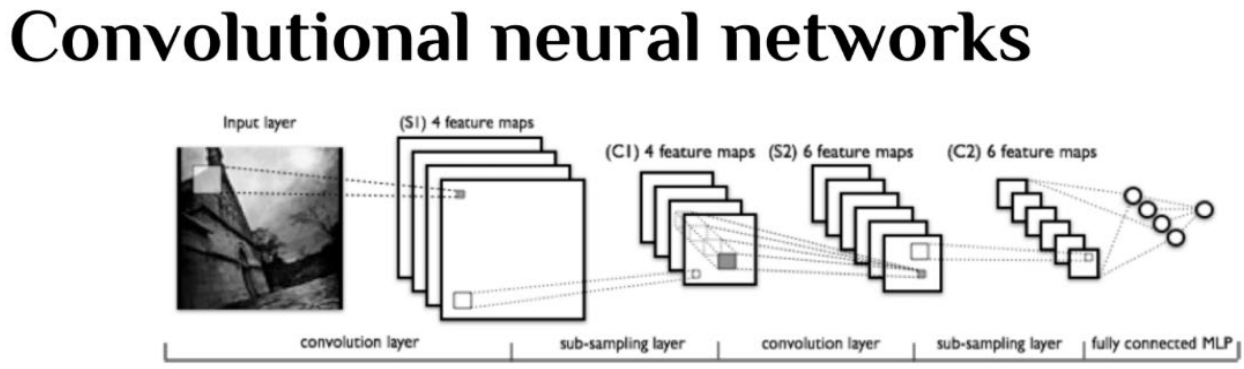
- I Image Selection i højre spalte kan du finde billedet Synthetic image. Det er et geometrisk konstrueret billede. Upload det.
- Lav et filter med $\theta = 0$ og forklar, hvorfor Gabor filtre kan bruges til kantedetektion.
- Lav om på vinklen så de lodrette hvide streger forsvinder.

Når vi skal træne vores neurale netværk, har vi netop brug for at udpege features i billedet, som netværket kan arbejde videre med. Gabor filtrene er et godt udgangspunkt til at træne et neuralt netværk.

I andre typer neurale neværk er alle neuroner i et lag forbundet med alle neuroner i det næste lag. Det giver mange forbindelser og mange vægte, som skal trænes.

I et CNN er der kun forbindelse mellem neuroner, der ligger tæt på hinanden i billedet. Det betyder at hver neuron har færre forbindelser og færre vægte, som skal trænes. Det gør at CNN'er kan trænes hurtigere end andre typer neurale netværk.

fig. 31 viser et CNN hvor et billede kommer gennem to foldninger og to sub-samplings lag. Til sidst køres de 6 feature maps gennem et neuralt netværk. Sub-samling er en proces hvor størrelsen af billederne reduceres. Det svarer til at reducere opløsningen på billedet. Det kan man gøre uden at miste for meget information, fordi de væsentligste features netop er blevet fremhævet.



Figur 31: Convolutional neural network, Jes Frellsen

Men hvad foregår der egentlig, når der foretages foldninger eller convulutions?

6.1 Foldning (convolution)

Foldning er grundigt beskrevet i bogen om 3shape. Der gennemgås også foldning af kontinuerte funktioner. Her vil vi holde os til den diskrete version, som er mere relevant for computerbehandling af billeder.

6.1.1 2D foldning

Da billeder er to-dimensionelle, er det naturligt at bruge to-dimensionel foldning ved hjælp af en 2D foldningsmatrix. Matricer optræder anderledes her end i afsnit 3 om matricer og matrixregning. Dér lærte vi særlige regneregler for matrix-multiplikation.

Her under "foldning" betragtes en matrix som en *vektor*, der er skrevet som en *tabel* af tal. Og multiplikation af matricer svarer her til *skalarprodukt* af vektorer.

Prøv selv at kontrollere dette eksempel:

$$\begin{bmatrix} 1 & -7 & 0 \\ 0 & 2 & 0 \\ 5 & 0 & 4 \end{bmatrix} * \begin{bmatrix} 3 & -1 & 1 \\ 8 & -13 & 8 \\ 0 & -5 & 6 \end{bmatrix} = 1\cdot 3 + (-7)\cdot(-1) + 0\cdot 1 + 0\cdot 8 + 2\cdot(-13) + 0\cdot 8 + 5\cdot 0 + 0\cdot(-5) + 4\cdot 6 = 3 + 7 - 26 + 24 = 8$$

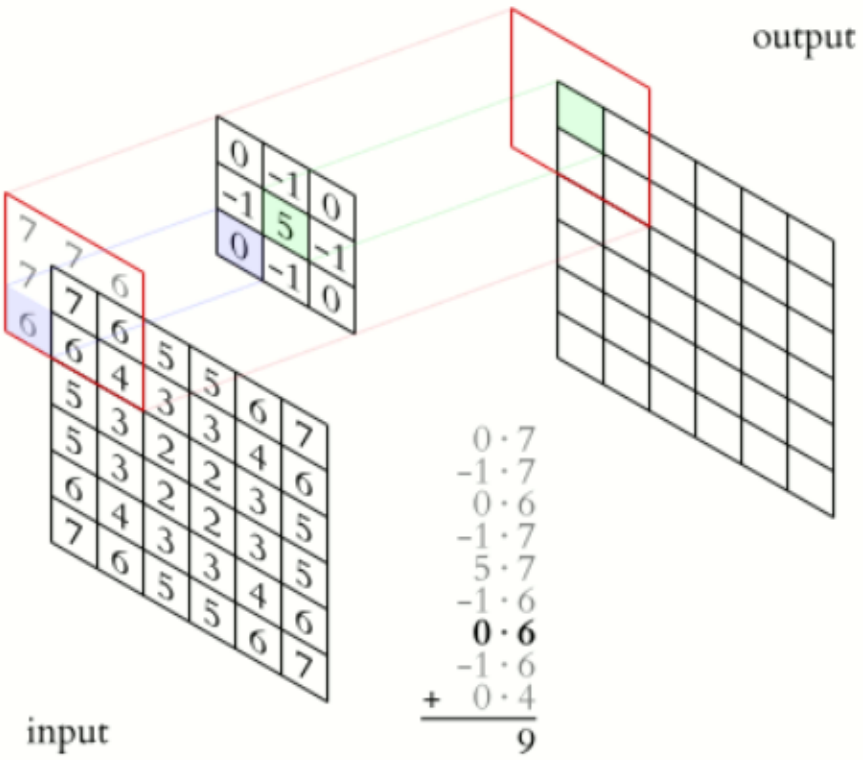
Det billede, vi vil lave en foldning af, er pixeleret, og hver pixel har fået tilordnet et tal, der angiver kontrastværdien, så billedet fremstår som en stor matrix.

Foldning med en 2D matrix foregår ved at placere matricen oven på billedet og udføre skalarmultiplikationen som beskrevet. Resultatet skrives i en ny matrix. Derefter flyttes matricen en plads og processen gentages indtil hele billedet er foldet (*se næste side*).

Læg mærke til, at de yderste pixel-værdier beregnes ved at kopiere kendte værdier ind der hvor de mangler i foldningen.

Der er mange metoder til at behandle siderne og hjørnerne, men for billeder med mange pixels er det simpleste at smide de yderste væk

Figuren viser en stillbillede fra en animation af processen, der kan findes her, [wikipedia convolution].



Figur 32: 2d foldning.

Den foldningsmatrix der er blevet brugt er: $W = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$.

Foldningen kan skrives med asterix $*$ som

$$(f * W)[x, y] = \sum_{i=-[N/2]}^{[N/2]} \sum_{j=-[M/2]}^{[M/2]} f\,(x + i, y + j) \cdot W(i, j).$$

N angiver hvor meget vi skal bevæge os lodret ved foldningen og M hvor meget vandret. Med en $[3 \times 3]$ foldningsmatrice som ovenfor er $N = M = 1$.

Værdien i venstre hjørne af illustrationen ovenfor beregnes som

$$(f * w)[0,0] = f(-1,0) \cdot w(-1,0) + f(0,-1) \cdot w(0,-1) + f(0,0) \cdot w(0,0) + f(0,1) \cdot w(0,1) + f(1,0) \cdot w(1,0)$$

$$(f * w)[0,0] = 7 \cdot (-1) + 7 \cdot (-1) + 7 \cdot 5 + 6 \cdot (-1) + 6 \cdot (-1) = 9$$

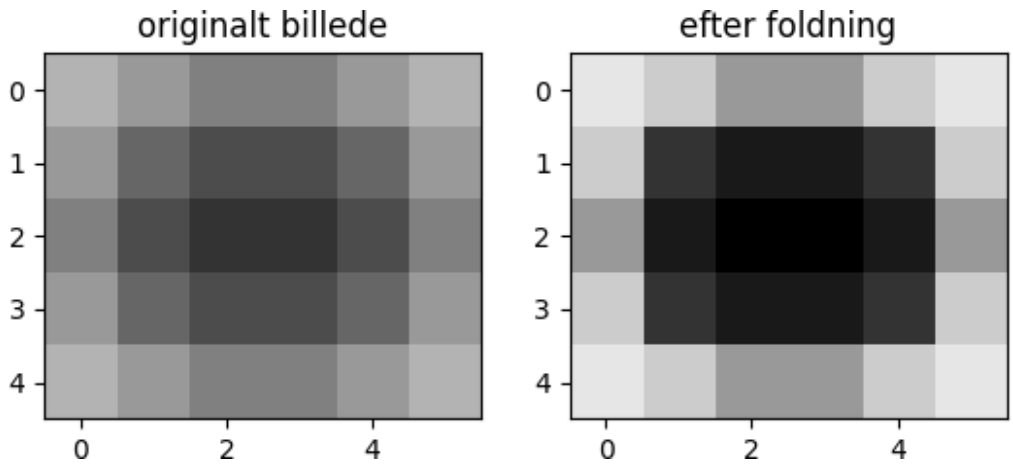
.

Øvelse 6.4

$[x, y]$ angiver koordinaterne til de celle, i den store matrix, der er centrum for den aktuelle beregning. Som normalt lader vi *x* tælle vandret og *y* tælle lodret. $[1,0]$ er således cellen til højre for øverste hjørne, der indeholder tallet 6.

- Beregn værdien for $(f * w)[1,0]$

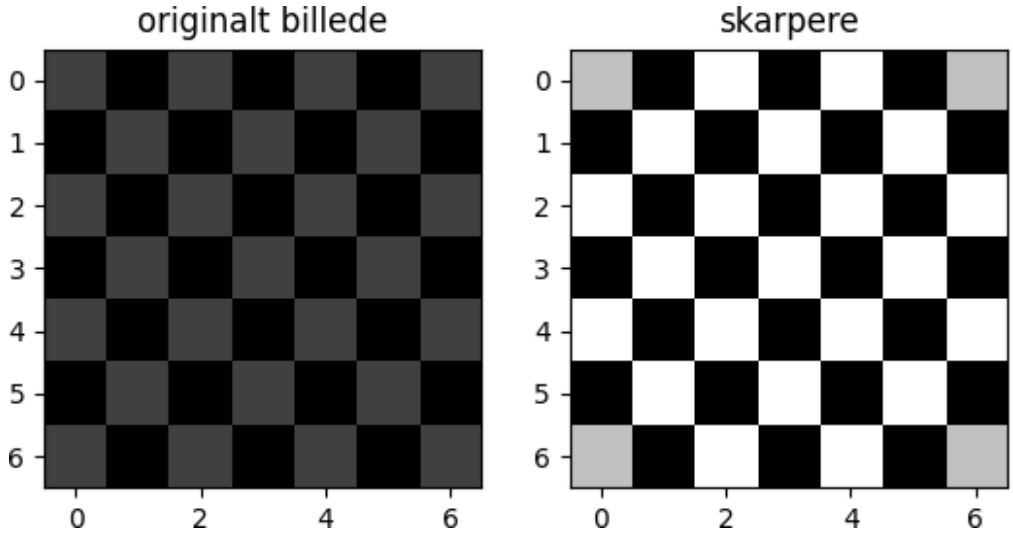
Nedenfor er lavet en foldning med et gråtonebillede. Her er lave værdier mørke og høje værdier lyse. Billedet fig. 33 er efter foldning blevet skarpere i den forstand, at mørke regioner er blevet mørkere mens lyse er blevet lysere.



Figur 33: 2d-foldning_sharp

Nedenfor viser vi udregningen for et billede af et skakternet mønster som er foldet med samme vægt-matrice, w. Her er det tydeligt at foldningen gør billedet skarpere.

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 & -3 & 4 & -3 & 4 & -3 & 3 \\ -3 & 5 & -4 & 5 & -4 & 5 & -3 \\ 4 & -4 & 5 & -4 & 5 & -4 & 4 \\ -3 & 5 & -4 & 5 & -4 & 5 & -3 \\ 4 & -4 & 5 & -4 & 5 & -4 & 4 \\ -3 & 5 & -4 & 5 & -4 & 5 & -3 \\ 3 & -3 & 4 & -3 & 4 & -3 & 3 \end{bmatrix}$$



Figur 34: Foldning af skaktern

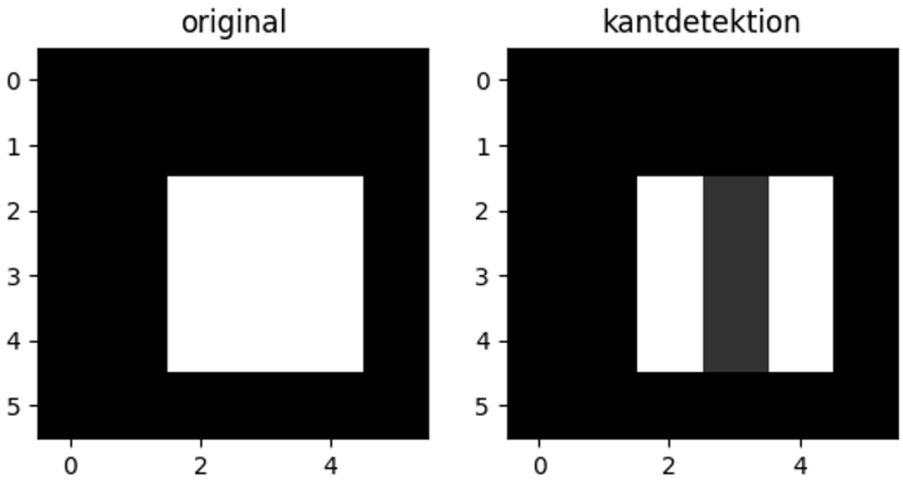
Igen ses uønskede effekter ved hjørnerne.

Øvelse 6.5

- Udfør beregningen af pixelværdierne de fire midterste pixels.
- Overvej hvad der vil ske hvis vægtmatricen ikke summerer til 1.

6.1.2 Kantdetektion

I fig. 35 er vist et simpelt billede med en hvid firkant og en sort baggrund. Vi vil forsøge at finde en metode til at foretage en kantdetektion, der kan give et billede som dette:



Figur 35: Simpelt billede

Hvis vi vil finde de lodrette kanter i billedet, kan vi forsøge med en vægt-matrice, der kun har én vandret række med tal forskellig fra 0 og som summerer til 0, fx denne:

$$,W = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Øvelse 6.6 En matrixbeskrivelse af et sort-hvid billede-1

Værdierne i matricerne angiver gråtoner, hvor 0 er sort og 1 er hvid.

- Argumenter for, at matricen *B* giver en beskrivelse af originalen
- Argumenter for at de to øverste rækker i 'Kantdetektionen' fremkommer ved at gennemføre en foldning af *B* med matricen *W*

$$B = \begin{bmatrix} 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 1. & 1. & 0. \\ 0. & 0. & 1. & 1. & 1. & 0. \\ 0. & 0. & 1. & 1. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix}, W = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Øvelse 6.7 En matrixbeskrivelse af et sort-hvid billede-2

- Vis at den røde kasse giver 0.
- Vis at den grønne kasse giver 1.
- Vis at den blå kasse giver 0.
- Forklar hvorfor vægt matricen netop fremhæver de lodrette kanter.
- Lav en vægrt-matrice der fremhæver de vandrette kanter.

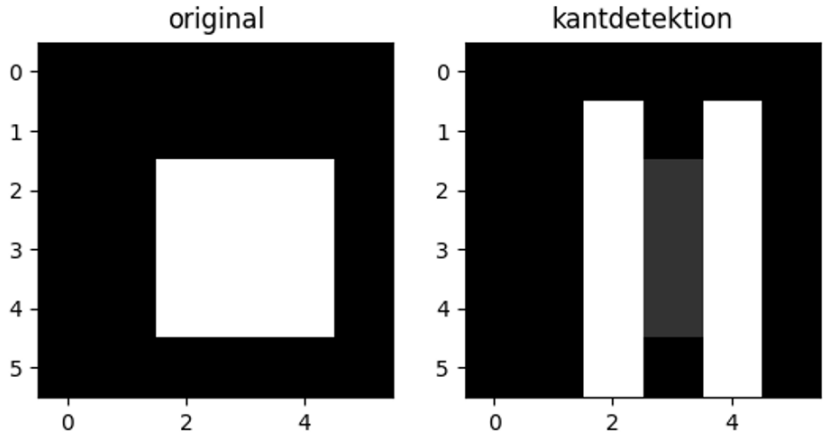
$$B = \begin{bmatrix} 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 1. & 1. & 0. \\ 0. & 0. & 1. & 1. & 1. & 0. \\ 0. & 0. & 1. & 1. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix}$$

Figur 36: Kantdetektion

Vægtmatricer til vandret detektion vil ofte se sådan ud,

$$W = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}.$$

Resultatet kan ses her:



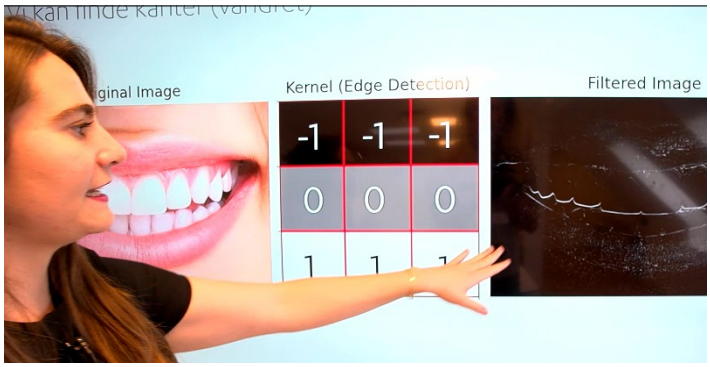
Figur 37: kantdetektion

Øvelse 6.8 Kantdetektion af lodrette linjer

- Beskriv ud fra figuren hvorfor den nye foldningsmatrix fremhæver kanterne i forhold til den ovenfor.
- Lav foldningen for position $(2,3)$ og $(3,3)$ og argumenter for hvorfor kanterne bliver fremhævede.

På denne måde kan vi fremhæve features i billederne før de bliver kørt gennem et neuralt netværk.

6.1.3 3Shape



3Shape, der er globalt førende indenfor Dental Teknologi, og som har udviklet avancerede metoder til scanning af tandsæt, har nu også udviklet algoritmer, der kan vise hvordan folks tandsæt kommer til at se ud efter en tandretning. For at gøre det, skal de identificere hver tand og dens placering. Det gør de med kunstig intelligens, hvor de bruger foldning til at fremhæve kanter mellem tænderne og mellem tænder og tandkød. (Orienter dig om virksomheden 3Shape her: Træk virksomhederne ind i undervisningen)

Billedet, som er hentet fra filmen, hvor 3Shape er en af de tre virksomhedscases, viser Laura Fenoy forklare, hvordan de ved brug af den matematiske foldningsteknik kan få taget billeder, der fremhæver kanter i den retning de ønsker. Ved at lægge lag på lag af disse optagelser, kan de frembringe billeder af de enkelte tænder, som giver mulighed for en individuel manipulation heraf, som grundlag for de ønskede "fremtidsbilleder".

På billedet ses et filter (på engelsk kernel) der finder vandrette linjer. Men hvilke vandrette linjer bliver fremhævet?

Øvelse 6.9 Vægtmatricen fremhæver øverste, ikke nederste kant!

Vi undersøger, hvad denne vægtmatricen gør, $W = \begin{bmatrix} -1.0 & -1.0 & -1.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$.

ved at anvende den på billedmatricen *B* fra øvelse 6.6

- Udregn et par foldninger og overbevis jer om at denne vægtmatrice faktisk giver det modsatte, altså fremhæver *øverste kant*.

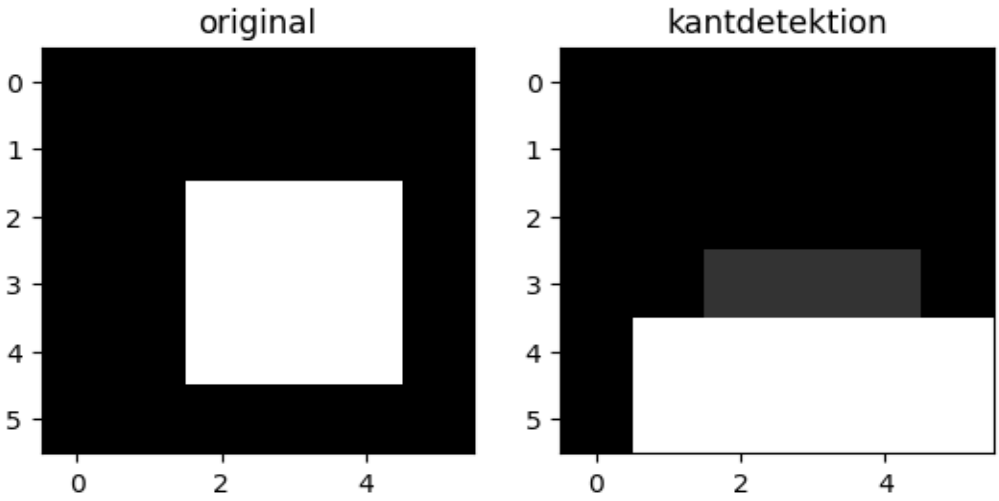
Problemet skyldes en egenskab ved foldning, vi ikke har kigget på endnu. Den er ret grundigt beskrevet i undervisningsmaterialet om 3Shape, så her tager vi bare konklusionen. Når der sker en foldning, bliver vægtmatricen flippet, både vertikalt og horisontalt. Det har ikke betydet noget i de første eksempler hvor matricen var symmetrisk men her skal den flippes, hvilket betyder *W* erstattes af *W'*:

$$W = \begin{bmatrix} -1.0 & -1.0 & -1.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \Rightarrow W^* = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix}$$

Øvelse 6.10 Flipmatricen fremhæver nederste vandrette kant!

Brug W^* og vis at foldningen netop forstærker nederste kant i kvadratet.

Resultatet ses i fig. 39



Figur 39: Kantdetektion, flipped matrix

Nu hvor vi har en erfaring med, hvordan vægtmatricen skal opbygges for at fremhæve vandrette kanter, kan vi så ræsonnere os til, hvordan vi fremhæver lodrette kanter?

Øvelse 6.11 Konstruer vægtmatricen, så den fremhæver lodrette linjer

a) Indsæt tal i vægtmatricen, så den fremhæver lodrette kanter. Forklar *uden at regne* at dette vil være tilfældet

b) Gennemfør nogle få udregninger og konkluder: Hvilke kanter fremhæves?

Det er også muligt at selv at arbejde med billederne. Nedenstående python kode kan du køre online med jyputer notebook her colab.research.google.com.

- Hent 3Shape_kantdetektion_2025.ipynb fra hjemmesiden.
- Hent billedet teeth.png fra hjemmesiden.
- Upload begge filer på colabs.
- Kør ved at trykke play i venstre side inden for hvert felt.

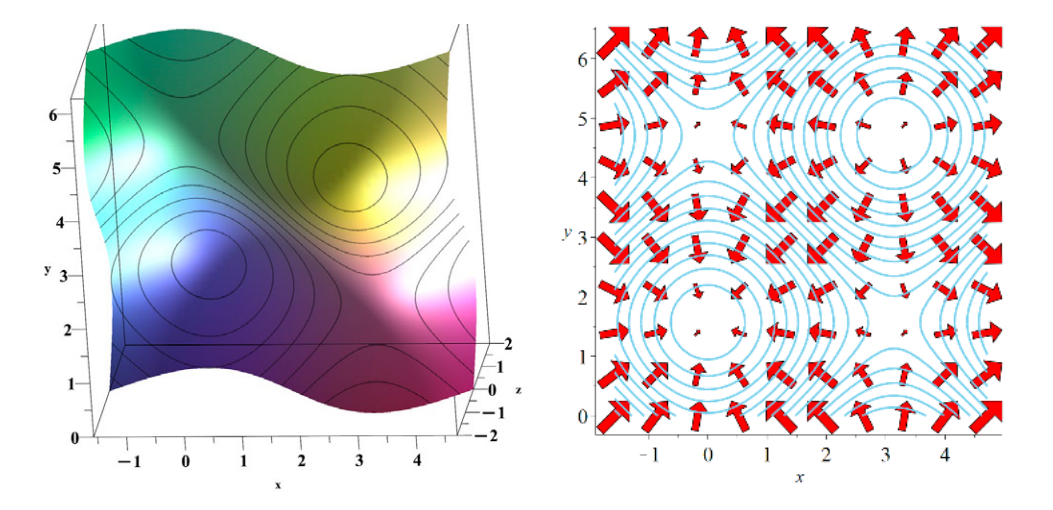
NB vær opmærksom på, at indryk ikke kommer med, når der kopieres fra PDF. Det er vigtigt i python og giver fejl, hvis du ikke retter det i koden.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d
from PIL import Image

img = Image.open('teeth.png').convert('L') # laver billedet om til gråtoner
B = np.array(img)

K = np.array([[-1,-1,-1],
              [0,0,0],
              [1,1,1]])

Z = convolve2d(B,K, mode='same')

plt.imshow(Z,cmap='gray')
plt.show()
```

Øvelse 6.12 Eksperimenter med vægtmatricen

- Prøv at lav om i vægtmatricen.
- Prøv med jeres eget billede.

III.
Et materiale til en srp om neurale netværk. Gradientmetoden i *n* dimensioner



Når vi træner neurale netværk, handler det om at bestemme millioner af parametre – dem vi kalder vægte. Hver ny parameter svare til én dimension. Det er selvfølgelig umuligt at visualisere det, men 2D kan godt give et indtryk af den metode, vi anvender til at optimere parametrene. Denne metode, *gradient descent*, går ud på at man minimerer en funktions værdi, ved at man bevæger sig ortogonalt på højdekurverne. På figuren til højre angiver pilene en bevægelse hen mod det sted, hvor der findes maksimum. Minimum er naturligvis den modsatte vej. En tegning er ikke et bevis, men den kan lede os på vej. I denne tredje del gennemføres beviser for gradientmetoden i 2 dimensioner. Samtidig gives en indføring i emnet differentiation af funktioner af flere variable. At finde minimum oversættes normalt til at finde nulpunkter de afledede. Vi er i en verden, hvor man ikke kan løse ligninger eksakt, alt er numerisk, og vi demonstrerer i et bilag, hvordan vi i én dimension bestemmer nulpunkter til vilkårlige ligninger, den såkaldte Newton Raphson metode.

matematik i arbejde

I dette projekt skal du argumentere for, at metoden steepest descent også virker i *n* dimensioner, i den forstand, at gradienten $\nabla \boldsymbol{f}(\mathbf{x}_0)$ for en differentiabel funktion af *n* variable peger i den retning, hvor funktionen *f* vokser hurtigst, samt at gradienten generelt står vinkelret på niveaufladen $\boldsymbol{f}(\mathbf{x}) = \boldsymbol{f}(\mathbf{x}_0) = \boldsymbol{k}$

Vi går frem i en række steps:

III.1 Skalarprodukt, længde af vektorer og ortogonalitet

Definition: Skalarproduktet af vektorer med *n* variable

Som for vektorer med to og tre koordinater definerer vi skalarproduktet eller prikproduktet af vektorer med *n* koordinater som summen af koordinatprodukterne:

$$\mathbf{u} \cdot \mathbf{v} = \begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_n \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_n \end{pmatrix} = \boldsymbol{u}_1 \cdot \boldsymbol{v}_1 + \boldsymbol{u}_2 \cdot \boldsymbol{v}_2 + \ldots + \boldsymbol{u}_n \cdot \boldsymbol{v}_n$$

Definition: Skalarproduktet og ortogonalitet af vektorer med *n* variable

Som for vektorer med to og tre koordinater definerer vi *skalarproduktet* eller prikproduktet af vektorer med *n* koordinater som summen af koordinatprodukterne:

$$\mathbf{u} \cdot \mathbf{v} = \begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_n \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_n \end{pmatrix} = \boldsymbol{u}_1 \cdot \boldsymbol{v}_1 + \boldsymbol{u}_2 \cdot \boldsymbol{v}_2 + \ldots + \boldsymbol{u}_n \cdot \boldsymbol{v}_n$$

To vektorer $\mathbf{u}$ og $\mathbf{v}$ kaldes *ortogonale* hvis deres skalarprodukt giver 0. Dette betegnes $\mathbf{u} \perp \mathbf{v}$

Opgave 1

Vis, at man må "prikke ind i parentes" dvs. at der gælder

$$\mathbf{w} \cdot (\mathbf{u} + \mathbf{v}) = \mathbf{w} \cdot \mathbf{u} + \mathbf{w} \cdot \mathbf{v}$$

Definition: Længden af en vektor

Ved *længden* af en vektor forstås $\|\mathbf{u}\| = \sqrt{\mathbf{u} \cdot \mathbf{u}} = \sqrt{\boldsymbol{u}_1^{\,2} + \boldsymbol{u}_2^{\,2} + \ldots + \boldsymbol{u}_n^{\,2}}$

Opgave 2

Givet vektorerne $\mathbf{u} = \begin{pmatrix} -1 \\ 2 \\ 3 \\ 1\,/\,2 \end{pmatrix}$ og $\mathbf{v} = \begin{pmatrix} 2 \\ 1 \\ -1 \\ 0 \end{pmatrix}$ samt vektoren $\mathbf{w} = \begin{pmatrix} -1 \\ 1 \\ -1 \\ \boldsymbol{k} \end{pmatrix}$

a) Beregn skalarproduktet $\mathbf{u}\cdot\mathbf{v}$ - er ***u*** og ***v*** ortogonale?

b) Beregn skalarproduktet $\mathbf{u}\cdot\mathbf{w}$.

c) Bestem tallet *k* så vektorerne ***u*** **og** ***w*** bliver ortogonale

d) Findes der et *k* så ***v*** og ***w*** er ortogonale?

Opgave 3

Vis, at for alle vektorer ***u*** og for alle tal *k* gælder

$$\|\boldsymbol{k}\cdot\mathbf{u}\| = |\boldsymbol{k}|\cdot\|\mathbf{u}\|$$

Opgave 4

Vis, at hvis to vektorer $\mathbf{u}$ og $\mathbf{v}$ er ortogonale $\mathbf{u}\perp\mathbf{v}$, så gælder *Pythagoras sætning*:

$$\|\mathbf{u}+\mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2$$

III.2 Differentiabilitet og retningsafledet

Differentiabilitet af en funktion af *n* variable

En reel funktion *f* af *n* variable, dvs. en funktion, hvis værdier er reelle tal, siges at være *differentiabel* i punktet $\mathbf{x}_0$, hvis der findes en vektor $\mathbf{c}=(\boldsymbol{c_1},\boldsymbol{c_2},\ldots,\boldsymbol{c_n})$, således at

$$\Delta\boldsymbol{f}(\mathbf{x}_0,\Delta\mathbf{x}) = \boldsymbol{f}(\mathbf{x}_0+\Delta\mathbf{x}) - \boldsymbol{f}(\mathbf{x}_0) \approx \boldsymbol{c_1}\cdot\Delta\boldsymbol{x_1} + \boldsymbol{c_2}\cdot\Delta\boldsymbol{x_2}\ldots + \boldsymbol{c_n}\cdot\Delta\boldsymbol{x_n} = \mathbf{c}\cdot\Delta\mathbf{x}$$

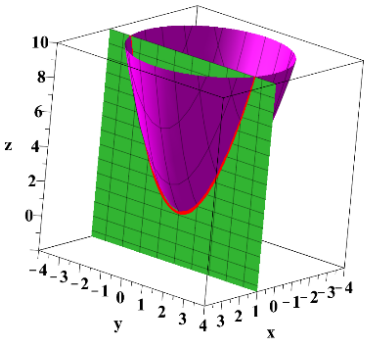
for enhver tilvækst: $\Delta\mathbf{x}=(\Delta\boldsymbol{x_1},\Delta\boldsymbol{x_2},\ldots,\Delta\boldsymbol{x_n})$. Med $\approx$ ("tilnærmet") mener vi mere præcist:

$$\frac{\Delta\boldsymbol{f}(\mathbf{x}_0,\Delta\mathbf{x})-\mathbf{c}\cdot\Delta\mathbf{x}}{\|\Delta\mathbf{x}\|}\to 0 \text{ for } \|\Delta\mathbf{x}\|\to 0\,.$$

Bemærk: Vi kan ikke anvende "tretrinsreglen" når vi går op i dimensionerne – allerede fra dimension 2 er situationen helt anderledes. Det skyldes, at $\Delta\mathbf{x}$ *kan nærme sig 0 på "alle mulige måder", ikke kun langs akserne, men også langs skrå linjer eller gennem spiraler eller ad endnu mere komplicerede veje. Her nøjes vi med at betragte bevægelser langs rette linjer. Og det giver anledning til at definere den afledede, når vi bevæger os langs en ret linje:*

2.1 Retningsafledede, snitplaner og snitfunktioner

En given egentlig vektor $\vec{\boldsymbol{r}}=\begin{pmatrix}\boldsymbol{h}\\ \boldsymbol{k}\end{pmatrix}$ fastlægger sammen med punktet $(\boldsymbol{x_0},\boldsymbol{y_0},\boldsymbol{f(x_0,y_0)})$ en *plan* (den grønne), der ligger *lodret* over linjen $\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{pmatrix}=\begin{pmatrix}\boldsymbol{x_0}\\ \boldsymbol{y_0}\end{pmatrix}+\boldsymbol{t}\cdot\begin{pmatrix}\boldsymbol{h}\\ \boldsymbol{k}\end{pmatrix}$, og som snitter grafen for *f*. Denne *snitplan* frembringer en graf (den røde kurve) for en *snitfunktion*, der er en funktion af én variabel, *t*. Og denne kan vi derfor differentiere "klassisk":



Definition: Den retningsafledede i vektor $\vec{\boldsymbol{r}}$'s retning

f siges at være differentiabel i $\vec{\boldsymbol{r}}$**'s** retning i punktet $(\boldsymbol{x_0},\boldsymbol{y_0})$, hvis den tilsvarende snitfunktion er differentiabel, dvs hvis den sammensatte funktion:
$\boldsymbol{f_{\vec{r}}(t)=f(x_o+t\cdot h, y_0+t\cdot k)}$ er differentiabel.

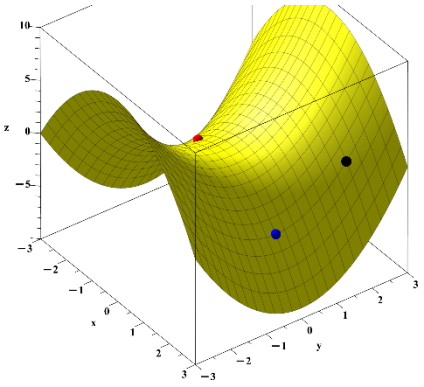
I så fald kaldes differentialkvotienten af $f_{\vec{r}}(t)$ i $t=0$ for den *retningsafledede* af $f(x,y)$ i (x_0,y_0) og i $\vec{r}$'s retning. Denne retningsafledede betegnes $f_{\vec{r}}'(x_0,y_0)$ eller $f'(x_0,y_0,\vec{r})$.

Bemærk: Af definitionen fremgår, at den retningsafledede er afhængig af, hvilken vektor vi vælger som retningsvektor. Det kan forekomme lidt underlig, men skyldes, at teorien er opstået i nær tilknytning til fysik, hvor parameterkurver opfattes som kurver der gennemløbes med en vis hastighed. Bliver hastighedsvektoren dobbelt så lang, er det i fysik et andet gennemløb, men i matematik den samme kurve.

Vælges en *enhedsvektor* $\vec{e}$ som retningsvektor, så kan den retningsafledede $f'(x_0,y_0,\vec{e})$ tolkes som ved funktioner af én variabel:

$f'(x_0,y_0,\vec{e})$ angiver hvor meget vi går op (eller ned) på grafen, når vi går stykket 1 frem i den givne retning, altså hvor stejl grafen er pågældende sted og i pågældende retning.

Øvelse 5.14: Beregn retningsafledet



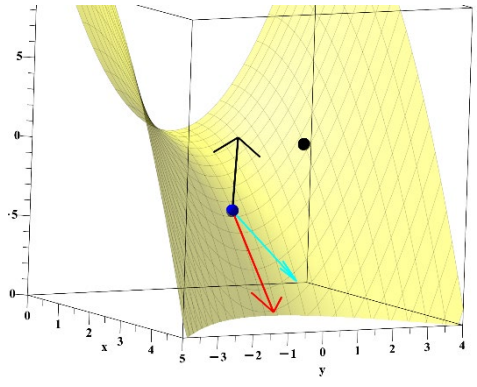
Betragt funktionen $g(x,y)=-x^2+y^2$.

a) Fremstil et grafisk billede af funktionen i området $-3\le x\le 3$ og $-3\le y\le 3$.

b) Udregn funktionsværdierne og afsæt punkterne på grafen: $A(-1,0)$ (rød), $B(2,0)$ (blå) og $C(2,2)$ (sort)

Vi ønsker at udregne de retningsafledede i *B* i retningerne bestemt af

$$\vec{r}=\begin{pmatrix}1\\1\end{pmatrix},\ \vec{s}=\begin{pmatrix}-1\\1\end{pmatrix} \text{ og } \vec{t}=\begin{pmatrix}\sqrt{2}\\0\end{pmatrix}.$$



c) Kontroller, at retningsvektorerne har samme længde.

d) Vis, at den funktion, vi skal differentiere for at bestemme den retningsafledede i $\vec{r}$'s retning er:

$$g_{\vec{r}}(2+x,0+x)=-4x+4$$

e) Bestem de retningsafledede i de tre retninger, dvs de afledede $g_{\vec{r}}'(0)$, $g_{\vec{s}}'(0)$ og $g_{\vec{t}}'(0)$. Du skal få:

$$-4,\ 4 \text{ og } -4\cdot\sqrt{2}\approx -5{,}64 \quad .$$

f) Hvordan stemmer det med det grafiske billede?

Sætning 3: Differentiable funktioner har retningsafledet i alle retninger

Antag $f(x,y)$ er differentiabel, (x_0,y_0) er et vilkårligt punkt, og $\vec{r}$ er en egentlig vektor. Så er *f* også differentiable i $\vec{r}$'s retning i punktet (x_0,y_0).

Bevis

Da *f* er differentiabel, er grafen lokalt plan. Punktet (x_0, y_0) og vektoren $\vec{r}$ bestemmer en lodret plan gennem punktet. Snittet mellem denne lodrette *plan*, og den *plan*, der ligger tæt sammen med grafen i nærheden af punktet (x_0, y_0), er en ret linje, der følger grafen tæt, dvs. følger snitkurven tæt. Men så er denne snitkurve jo lokalt lineær, dvs. differentiabel.

III.3. Partielle afledte og gradient for en funktion af *n* variable.

Der gælder desværre ikke det modsatte: En funktion behøver ikke være differentiabel, fordi den har alle retningsafledede. Men to af de retningsafledede spiller en særlig rolle, nemlig de, som er bestemt af henholdsvis *x*-aksens og *y* -aksens retninger, og med anvendelse af *enhedsvektorerne* som retningsvektorer. Enhedsvektorerne er henholdsvis $\vec{e}_x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ og $\vec{e}_y = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Når vi går i en af disse to retninger, varierer kun den ene af de variable. Disse to afledede funktioner kaldes de *partielt afledede*.

Definition: Partielt afledede

Den partielt afledede $f'_x(x,y)$ af $f(x,y)$ med hensyn til den uafhængige variabel *x* bestemmes ved at differentiere snitfunktionen $g(x) = f(x,y)$, mens *y* opfattes som en konstant. Der gælder altså $f'_x(x,y) = g'(x)$.

Den partielt afledede $f'_y(x,y)$ af $f(x,y)$ med hensyn til den uafhængige variabel *y* bestemmes ved at differentiere snitfunktionen $g(y) = f(x,y)$, mens *x* opfattes som en konstant. Der gælder altså $f'_y(x,y) = g'(y)$.

Bemærk, at de partielt afledede igen er funktioner af to variable, og at funktionsværdierne for disse funktioner i et bestemt punkt $P(x_0, y_0)$ kan fortolkes som hældningskoefficienter for de tangenter til grafen for *f*, der er parallelle med henholdsvis *x*-aksen og *y*-aksen. Dvs. de partielt afledede fortæller noget om, hvordan funktionen selv opfører sig, når vi i *xy*-planen bevæger os i et lille område omkring punktet $P(x_0, y_0)$ langs henholdsvis en ret linje parallel med *x*-aksen og en ret linje parallel med *y*-aksen.

Man kan vise, at hvis de partielt afledede findes, og hvis disse er kontinuerte, så er funktionen selv differentiabel. Det er langt det mest normale, men der findes alligevel mange eksempler, hvor en funktion i nogle få punkter ikke opfylder denne betingelse. En sådan funktion er så differentiabel, bortset fra i disse enkelte punkter.

Partielle afledede af funktioner af *n* variable

Hvis vi lader alle koordinater i Δx på nær én være 0 kan vi se også her, at konstanterne kan bestemmes ved almindelig differentialregning! Hvis vi fx holder den i'te koordinat $\Delta x_i \neq 0$ får vi:

$$\frac{\Delta f(\mathbf{x}_0, \Delta\mathbf{x}) - c_i \cdot \Delta x_i}{\Delta x_i} \to 0 \text{ for } \|\Delta\mathbf{x}\| \to 0$$

Dvs. $$\lim_{\Delta x_i \to 0} \frac{\Delta f(\mathbf{x}_0, \Delta\mathbf{x})}{\Delta x_i} = c_i$$

Og dermed: $$\lim_{\Delta x_i \to 0} \frac{f(\mathbf{x}_0 + \Delta\mathbf{x}) - f(\mathbf{x}_0)}{\Delta x_i} = c_i$$

Så den i'te koordinat af c_i er altså *differentialkvotienten* af funktionen *f* opfattet som om den alene er en funktion af variablen på den i'te koordinats plads, mens alle andre variable holdes konstant. Vi kalder denne type af differentiation for *partiel differentiation* og noterer det på følgende måde:

$f_i'(\mathbf{x}_0) = c_i$, eller: $$\frac{\partial f(\mathbf{x})}{\partial x_i}_{\mathbf{x}=\mathbf{x}_0} = c_i$$

Opgave 5

Betragt funktionen $f(x_1, x_2) = \sin(x_1) + {x_2}^3$

Beregn $f_1'(x_1, x_2)$ og $f_2'(x_1, x_2)$

Ligesom ved funktioner af to variable, kan vi helt generelt definere

Gradienten af en funktion af *n* variable

For en funktion af *n* variable, der er differentiabel, definerer vi gradienten $\nabla f(\mathbf{x}_0)$ som den *n*-dimensionale vektor, der består af de partielle afledede i punktet $\mathbf{x}_0$

$$\nabla f(\mathbf{x}_0) = (f_1'(\mathbf{x}_0), f_2'(\mathbf{x}_0), \ldots, f_n'(\mathbf{x}_0))$$

Dvs. gradienten svarer bare til vores $\mathbf{c}$ vektor i definitionen af differentiabilitet.

Opgave 6

Beregn gradienten af funktionen $f(x, y, z) = e^{x^2 + 2y - z^2}$

Niveaufladen med niveauet *k* for en funktion af *n* variable

Defineres som den delmængde af definitionsmængden, hvis funktionsværdier er lig med konstanten *k*, dvs. mængden af alle punkter hvor $f(\mathbf{x}) = k$

Opgave 7

Vis, at niveaukurverne for funktionen $f(x, y) = e^{x^2 + y^2}$ er cirkler med centrum i (0,0)

Opgave 8 Gradienten er ortogonal på niveaufladen

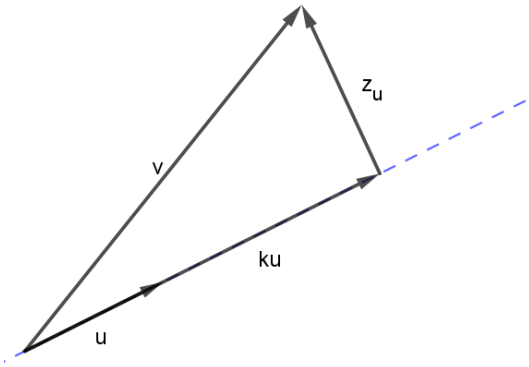
a) Vis, at funktionstilvæksten $\Delta f(\mathbf{x}_0, \Delta\mathbf{x})$ kan skrives som et skalarprodukt mellem gradienten af funktionen og $\Delta\mathbf{x}$

b) Hvad vil funktionstilvæksten være hvis $\Delta\mathbf{x}$ fører os til et $\mathbf{x}$ der ligger på niveaukurven med niveauet $f(\mathbf{x}_0)$?

c) Argumenter for, at gradienten $\nabla f(\mathbf{x}_0)$ er ortogonal på niveaukurven for f med niveauet $f(\mathbf{x}_0)$

III.4 Projektion af vektor på vektor og Cauchy-Schwarz ulighed

Givet en bestemt vektor $\mathbf{u} \neq \mathbf{0}$ og en vilkårlig vektor $\mathbf{v}$. Vi ønsker at bestemme et tal k så vektoren $\mathbf{z}_u = \mathbf{v} - k \cdot \mathbf{u}$ er ortogonal på $\mathbf{u}$ - se figuren her:



Vektoren $k \cdot \mathbf{u}$ vil vi så kalde ortogonalprojektionen af $\mathbf{v}$ på $\mathbf{u}$ og vi noterer det $\mathbf{proj}_{\mathbf{u}}\mathbf{v}$

Opgave 9 Bestemmelse af tallet *k*

a) Analyser kravet: $\mathbf{z}_u = \mathbf{v} - k \cdot \mathbf{u}$ skal være ortogonal til $\mathbf{u}$ ved at anvende definitionen:

$$(\mathbf{v} - k \cdot \mathbf{u}) \cdot \mathbf{u} = 0$$

b) Udnyt regnereglerne og vis, at $k = \dfrac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|^2}$

c) Konkluder, at $\mathbf{proj}_{\mathbf{u}}\mathbf{v} = \dfrac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|^2} \cdot \mathbf{u}$

Sammenlign med formlen for projektion af vektor på vektor fra din formelsamling.

Opgave 10

Givet vektorerne $\mathbf{u} = \begin{pmatrix} -1 \\ 2 \\ 3 \\ 1/2 \end{pmatrix}$ og $\mathbf{v} = \begin{pmatrix} 2 \\ 1 \\ -1 \\ 0 \end{pmatrix}$.

Bestem projektionen $\mathbf{proj}_{\mathbf{u}}\mathbf{v} = \dfrac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|^2} \cdot \mathbf{u}$

Opgave 11

Hvad kan du slutte om vektorerne $\mathbf{u}$ og $\mathbf{v}$ hvis $\mathbf{proj}_{\mathbf{u}}\mathbf{v} = \mathbf{0}$?

Opgave 12 Projektionen $\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}$ er den vektor på linjen, der ligger tættest på $\boldsymbol{v}$

Opstil funktionen $\boldsymbol{f}(\boldsymbol{x}) = \|\mathbf{v} - \boldsymbol{x} \cdot \mathbf{u}\|^2$, hvor *x* nu bare er en parameter.

a) Omskriv udtrykket til et polynomium i *x*.

b) Hvilke egenskaber kender du ved dette polynomium?

c) Differentier og bestem førstekoordinaten $\boldsymbol{x_{top}}$ til toppunktet for *f* udtrykt ved vektorerne $\boldsymbol{u}$ og $\boldsymbol{v}$. Sammenlign den fundne $\boldsymbol{x_{top}}$ med *k* fra opgaven før.

d) Konkluder hvad du har fundet ud af, ved også at inddrage opgave 9

Opgave 13 Længden af projektionen $\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}$ er altid kortere end længden af vektoren $\boldsymbol{v}$

a) Gør rede for at vektoren $\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}$ og vektoren $\mathbf{z}_{\boldsymbol{u}}$ er ortogonale

b) Vis, at der så må gælde Pythagoras: $\|\mathbf{v}\|^2 = \|\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}\|^2 + \|\mathbf{z}_{\boldsymbol{u}}\|^2$ (vink: *Udnyt at længden i anden kan udregnes som et skalarprodukt*)

c) Konkluder, at $\|\mathbf{v}\| \geq \|\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}\|$

Opgave 14 Cauchy-Schwarz ulighed

a) Beregn $\|\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}\|^2$ ved at prikke projektionsvektoren med sig selv

b) Argumenter for, at $\|\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}\| = \dfrac{|\mathbf{u} \cdot \mathbf{v}|}{\|\mathbf{u}\|}$

c) Brug b) samt resultatet fra c) i opgaven ovenfor: $\|\mathbf{v}\| \geq \|\boldsymbol{proj_{\mathbf{u}}\mathbf{v}}\|$ til at udlede *Cauchy-Schwarz ulighed:* $|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\| \cdot \|\mathbf{v}\|$

Opgave 15 Cauchy-Schwarz i to dimensioner

Udnyt din viden om skalarproduktet i to dimensioner til at argumentere for **Cauchy-Schwarz i to dimensioner**, dvs:

$$\vec{\boldsymbol{a}} \cdot \vec{\boldsymbol{b}} \leq \left|\vec{\boldsymbol{a}}\right| \cdot \left|\vec{\boldsymbol{b}}\right|,$$

hvor den første prik er skalarprodukt, den næste almindeligt gangetegn.

Hvornår gælder der lighedstegn?

Vi er nu klar til at bevise metoden om *steepest descent* via Cauchy Schwarz ulighed.

Kig på udtrykket for funktionstilvæksten: $\Delta \boldsymbol{f}(\mathbf{x}_0, \Delta\mathbf{x}) \approx \nabla \boldsymbol{f}(\mathbf{x}_0) \cdot \Delta\mathbf{x}$. For en *differentiabel* funktion *f* bliver udtrykket uendeligt lille i grænsen hvor $\|\Delta\mathbf{x}\| \to 0$ i den forstand, at:

$\frac{\Delta \boldsymbol{f}(\mathbf{x}_0, \Delta\mathbf{x}) - \nabla \boldsymbol{f}(\mathbf{x}_0)\cdot\Delta\mathbf{x}}{\|\Delta\mathbf{x}\|} \to 0$ for $\|\Delta\mathbf{x}\| \to 0$. **Dette udnyttes i følgende opgave**:

4.1 Gradienten peger i den retning hvor funktionsværdien ændres mest

a) Vis, at definitionen på at *f* er differentiabel medfører at

$$\lim_{\|\Delta\mathbf{x}\|\to 0} \frac{\Delta \boldsymbol{f}(\mathbf{x}_0, \Delta\mathbf{x})}{\|\Delta\mathbf{x}\|} = \lim_{\|\Delta\mathbf{x}\|\to 0} \frac{\nabla \boldsymbol{f}(\mathbf{x}_0)\cdot\Delta\mathbf{x}}{\|\Delta\mathbf{x}\|} = \nabla \boldsymbol{f}(\mathbf{x}_0)\cdot\mathbf{e},$$

hvor $\mathbf{e} = \frac{\Delta\mathbf{x}}{\|\Delta\mathbf{x}\|}$ er en enhedsvektor.

b) Brug Cauchy-Schwarz ulighed til at vise, at den *højst mulige værdi* af funktionstilvæksten er: $\lim_{\|\Delta\mathbf{x}\|\to 0} \frac{\Delta \boldsymbol{f}(\mathbf{x}_0, \Delta\mathbf{x})}{\|\Delta\mathbf{x}\|} \le \|\nabla \boldsymbol{f}(\mathbf{x}_0)\|$

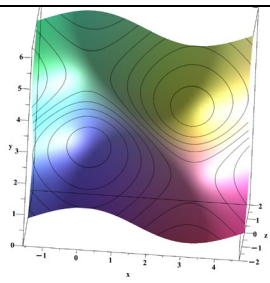
c) Vis, at vektoren $\mathbf{u} = \frac{\nabla \boldsymbol{f}(\mathbf{x}_0)}{\|\nabla \boldsymbol{f}(\mathbf{x}_0)\|}$ er en enhedsvektor.

d) Hvilken *retning* har vektoren $\mathbf{u}$?

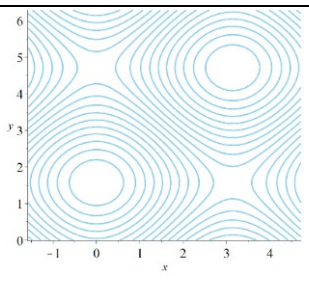
e) Vis, at i retningen af $\mathbf{u}$ bliver funktionstilvæksten $\lim_{t\to 0} \frac{\Delta \boldsymbol{f}(\mathbf{x}_0, \Delta\mathbf{x})}{\|\Delta\mathbf{x}\|} = \|\nabla \boldsymbol{f}(\mathbf{x}_0)\|$

f) Konkluder, at funktionen vokser hurtigst ved at gå fra $\mathbf{x}_0$ i retning af $\nabla \boldsymbol{f}(\mathbf{x}_0)$

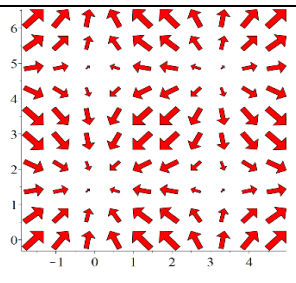
Vi har altså vist, at gradienten er ortogonal på niveaukurverne og at funktionen vokser hurtigst (eller aftager hurtigst) i gradientens retning. Dette fænomen kan illustreres grafisk med funktionen $f(x,y) = \cos(x) + \sin(y)$



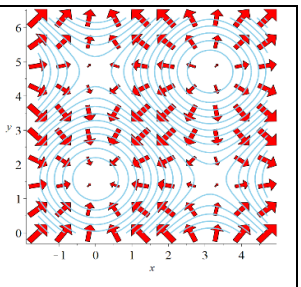
Graf med højdekurver



Niveaukurver



Gradientfelt



… med niveaukurver

Illustrationen virker overbevisende: Gradienten peger i ethvert punkt i den retning, hvor funktionsværdierne $z = f(x,y)$ vokser stærkest, eller aftager mest. Altså, når vi bevæger os langs gradienten i *xy*-planen, så svarer det til, at landskabet går stejlt opad eller nedad i netop den retning. Det betyder med andre ord, at hvis vi står i et punkt på grafen for *f* og kigger ned i *xy*-planen, så vil gradienten angive den retning, hvor grafen ændrer sig mest, opad eller nedad.

Bilag om at bestemme nulpunkter numerisk.

Både Jes Frellsen fra DTU og Peter Mølgaard fra B&O præsenterer grafisk ideen i at finde minimum for en loss-funktion. Da funktionerne er differentiable, vil første trin altid være at differentiere Loss-funktionen og så søge at løse den ligning $Loss'(x) = 0$ der fremkommer, når vi differentierer. Men en sådan ligning vil vi stort set aldrig kunne løse med en formel. Derfor er der udviklet andre, såkaldte "numeriske metoder", der kan give os en tilnærmet værdi, og med det antal decimaler vi ønsker! Dette bilag, der kan danne grundlag for et projekt, er et eksempel på en metode i 2D, der anvender samme metoder som '*gradient descent*" i n dimensioner.

Newton Raphsons metode og hvorfor den virker

(*Første del af dette afsnit er en lettere redigeret version af afsnittet fra Hvad er matematik? 2, kapitel 8, 'Numeriske metoder og algoritmer', s. 329-331. Dette giver os en stærk intuition om, at metoden må virke – og at den er effektiv. Anden del giver en begrundelse for at den virker, ved at oversætte problemet til et, der kan behandles i teorien for iteration og kaos.*)

1. Newton Raphsons metode

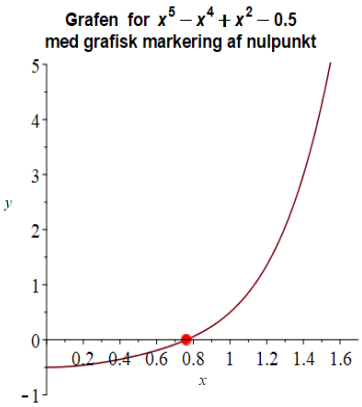
Udvikling af metoder til ligningsløsning har været en vigtig del af matematikken siden oldtiden. Med udviklingen af differentialregningen opstod nye muligheder. Den metode, vi her omtaler, blev udviklet nogenlunde samtidig og uafhængigt af hinanden af Newton og hans samtidige Joseph Raphson (1648-1712). Den er både hurtig og præcis.

Lad os illustrere metoden med et eksempel. Man vil undervejs se, at metoden har generel karakter.

Betragt polynomiet $p(x) = x^5 - x^4 + x^2 - 0.5$. Vi tegner grafen for at få en fornemmelse af funktionens nulpunkter.

Ideen i Newton-Raphsons metode er nu, at vi starter med et "gæt" på et nulpunkt relativ tæt på det røde nulpunkt. Vi vælger $x = 1.5$, hvorved vi bedre kan få en tegning der viser metoden.

1. trin: Vi bestemmer punktet $(1.5, f(1.5))$, og lægger en tangent $t_1(x)$ til grafen i dette punkt. Denne tangent vil skære x-aksen.



2. trin: Vi bestemmer tangentens skæring med x-aksen.

Fra din lærebog finder du en forskrift for tangenten:

$$t_1(x) = p(x_0) + p'(x_0) \cdot (x - x_0)$$

Denne anvendes nu til at bestemme en formel for tangentens skæring med x-aksen. Vi anvender blot de almindelige regler for ligningsløsning:

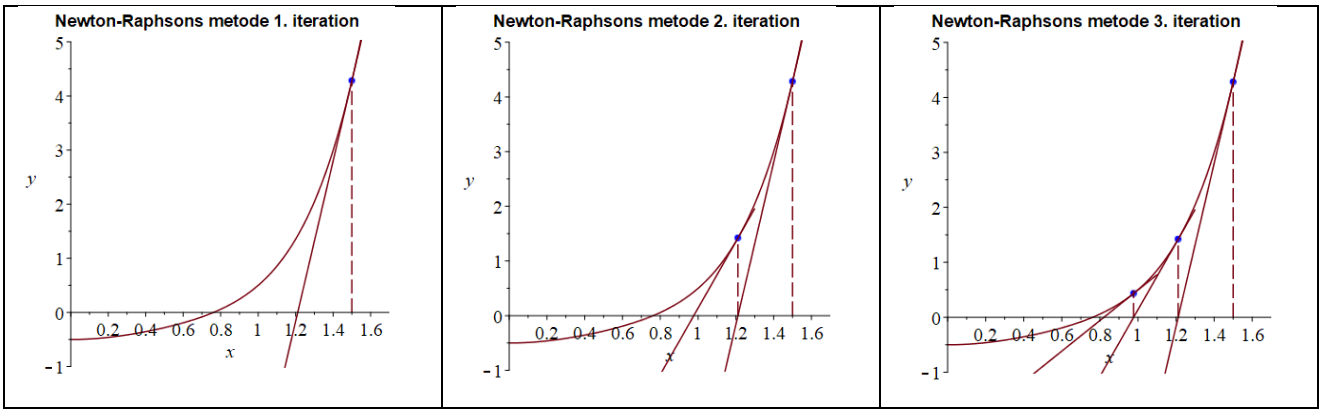
$$
\begin{aligned}
0 &= p(x_0) + p'(x_0)\cdot(x - x_0) \\
-p(x_0) &= p'(x_0)\cdot(x - x_0) \\
-\frac{p(x_0)}{p'(x_0)} &= (x - x_0) \\
x &= x_0 - \frac{p(x_0)}{p'(x_0)}
\end{aligned}
$$

Vi må naturligvis ikke dividere med 0, så udregningerne kan ikke altid gennemføres.

Hvad svarer situationen $p'(x_0) = 0$ til rent grafisk?

Dette betyder, at der er tilfælde, hvor metoden ikke kan anvendes.

3. trin: Vi har nu en formel for en tangents skæring med x-aksen. Ved at indsætte startværdien x_0 får vi den næste værdi: $x_1 = 1.210970$. Med x_1 kan vi gentage processen, bestemme en tangent og opsøge skæringspunktet med x-aksen:



Øvelse 1. Kør Newton-Raphson

a) Bestem tangenten $t_1(x) = p(x_0) + p'(x_0)\cdot(x - x_0)$ og beregn selv $x_1 = x_0 - \frac{p(x_0)}{p'(x_0)}$

b) Udnyt punktet x_1 til at bestemme x_2 osv. Du skal få følgende række af tal:

$x_0 = 1.5$	$x_1 = 1.210970$	$x_2 = .9770493$	$x_3 = .8210262$	$x_4 = .7664689$	$x_5 = .7617505$

c) Hvis vi bestemmer nulpunktet med en solve-facilitet, får vi $x_* = 0.7617200$. Hvor mange gange skal vi køre NR, før det tilnærmede nulpunkt er korrekt med 7 decimaler.

Øvelse 2. Funktioner med flere nulpunkter

Lad $p(x) = x^3 - 3x + 1$.

a) Tegn grafen i intervallet $[-2.5; 2.5]$.

Du skal fået grafisk billede, der viser, at $p(x)$ har tre nulpunkter.

b) Definer NR-funktionen: $p_{NR}(x) := x - \frac{p(x)}{p'(x)}$.

c) For hvert af de tre nulpunkter skal du som begyndelsesværdi vælge en x-værdi tæt ved nulpunktet, og så køre Newton-Raphson ud fra dette og med $p_{NR}(x)$. (brug det grafiske bilolede til at foretage dette valg)

(*Du skal få nulpunkterne: 0.347296, 1.532089, -1.879385*)

Øvelse 3 Når Newton Raphson ikke fungerer
Der er tilfælde, hvor NR-algoritmen ikke virker, skrev vi ovenfor. Giv grafiske illustrationer af sådanne situationer.

Newton-Raphsons algoritme er let at implementere, da det er en simpel iterationsproces, dvs en proces hvor det samme gentages igen og igen:

- Tegn en tangent i et punkt på grafen

- Opsøg tangentens skæring med 1.-aksen

- Bestem punktet på grafen lodret over denne x-værdi, tegn en tangent i det nye punkt på grafen, opsøg ...

Der findes naturligvis en række andre algoritmer til at bestemme nulpunkter, også nogle der er betydeligt mere effektive. Men de er til gengæld også ganske komplicerede. Newton-Raphson er forbløffende effektiv i betragtning af, hvor simpel den er.

2. Hvorfor Newton Raphsons algoritme er så effektiv

Vi kan indse, hvorfor Newtons nulpunktsmetode er så effektiv, ved hjælp af resultater fra *Teorien om iteration og kaos*! Dette er udførligt behandlet i **projekt 0.4: Iteration og Kaos,** som du kan finde *her* (link til projekt 0.4: Iteration og Kaos)

Vi har brug for enkelte begreber herfra:

Definition 1: Fixpunkt

Givet en vilkårlig funktion *f*. Et punkt $x*$, der opfylder:

$$f(x*) = x*$$

kaldes for et fixpunkt for funktionen.

Definition 2. Tiltrækkende og frastødende fixpunkter

Et fixpunkt x* kaldes et *tiltrækkende fixpunkt*, hvis der findes et interval I om x*, så det for enhver følge $\{x_{n+1} = g(x_n)\}$ gælder, at lander bare èt af x_n'erne i intervallet, så vil følgen konvergere mod x*. (Billedligt suges følgen ind til x* når vi kommer for tæt på - som et sort hul!).

Et fixpunkt kaldes *frastødende*, hvis der findes et interval I, så hver gang et x_n fra en sådan følge falder inde i intervallet, så vil det næste i rækken blive skubbet længere bort fra x*. (Billedligt som når vi nærmer to nordpoler til hinanden).

Overbevis dig selv om, at den geometriske betydning af fixpunkter er, at disse optræder, hvor grafen skærer linjen $y = x$.

Vi beviser i projekt 0.4, at der gælder:

Sætning 1. Betingelsen for om et punkt er frastødende eller tiltrækkende

Lad x^* være et fixpunkt for den differentiable funktion g. Så gælder:

x^* er et tiltrækkende fixpunkt for g, hvis $|g'(x^*)| < 1$

x^* er et frastødende fixpunkt for g, hvis $|g'(x^*)| > 1$.

Vi indfører følgende:

Betegnelser:

Lad x^* være et fixpunkt for den differentiable funktion g.

Hvis $|g'(x^*)| = 1$ kaldes x^* for et neutralt fixpunkt.

Hvis $|g'(x^*)| = 0$ kaldes x^* for et supertiltrækkende fixpunkt.

Prøv selv at tegne situationerne, for at se, at betegnelserne er rimelige. Tjek evt ved at se i projekt 0.4

Vi ser nu, hvorledes kaosteorien kan hjælpe os til at forstå, hvor Newton Raphsons metode er så effektiv.

Vi tager det i to skridt:

Sætning 2. Nulpunkter for *f* svarer til fixpunkter for Newton Raphsons iterationsfunktion.

1. Givet en differentiabel funktion *f,* og et tal x^*, hvorom der gælder, at $f'(x^*) \neq 0$, og at x^* er fixpunkt for funktionen $g(x) = x - \frac{f(x)}{f'(x)}$. Så gælder, at x^* er et nulpunkt for *f*.

2. Givet en differentiabel funktion *f*. Antag, at *f* har nulpunkt i x^*, og at $f'(x^*) \neq 0$

Så gælder, at x^* er et fixpunkt for funktionen $g(x) = x - \frac{f(x)}{f'(x)}$.

Bevis.

1. Antag, at x^* er fixpunkt for funktionen $g(x) = x - \frac{f(x)}{f'(x)}$. Så får vi:

$$\begin{aligned} g(x^*) &= x^* \\ x^* - \frac{f(x^*)}{f'(x^*)} &= x^* \\ -\frac{f(x^*)}{f'(x^*)} &= 0 \\ f(x^*) &= 0 \end{aligned}$$

Altså er x^* nulpunkt for *f*.

2. Antag *f* har nulpunkt i x^*, og at $f'(x^*) \neq 0$.

Vi ved altså, at $f(x^*) = 0$. Inspireret af punkt 1 foretager vi nu følgende omskrivning, der er den samme som ovenfor, blot den modsatte vej:

$$\begin{aligned} f(x^*) &= 0 \\ \frac{f(x^*)}{f'(x^*)} &= 0 \quad \text{(divisionen er tilladt, da } f'(x^*) \neq 0) \\ x^* - \frac{f(x^*)}{f'(x^*)} &= x^* \\ g(x^*) &= x^* \end{aligned}$$

.

Altså er x^* et fixpunkt for *g*.

Ud fra sætning 2 får vi nu selve sætningen om effektiviteten i Newtons metode:

Sætning 3. Nulpunkter for *f* svarer til supertiltrækkende fixpunkter for Newton Raphsons iterationsfunktion.

Givet en flere gange differentiabel funktion *f*. Antag, at x^* er nulpunkt for *f*, og at $f'(x^*) \neq 0$.

Så gælder, at x^* er et supertiltrækkende fixpunkt for funktionen $g(x) = x - \frac{f(x)}{f'(x)}$.

Øvelse 3. En grafisk fremstilling af et supertiltrækkende fixpunkt

Tegn med fri hånd på et papir grafen for en funktion $h(x)$, som har et fixpunkt, dvs. grafen skærer linjen $y = x$ i et punkt. Tegn et eksempel, hvor tangenthældningen i dette punkt er 0. Hent projekt 0.4, og anvend den metode der er beskrevet i afsnit 6 heri om *grafisk iteration*. Så vil du se, at vælges startpunktet ikke for langt væk fra dette fixpunkt, så er det som om iterationsfølgen øjeblikkeligt suges ind i fixpunktet.

Bevis for sætning 3.

Vi skal vise, at $g'(x^*) = 0$. Så derfor udregner vi simpelthen $g'(x^*)$. Hvis man kender brøkreglen for differentiation, kan man anvende den. Ellers kan man omskrive til et produkt og bruge produktreglen. Vi gør det sidste:

Omskrivning:

$$g(x) = x - \frac{f(x)}{f'(x)} = x - f(x) \cdot \frac{1}{f'(x)} = x - f(x) \cdot \left(f'(x)\right)^{-1}$$

Øvelse 4. Gør rede for omskrivningerne

Du skal nøje gøre rede for hvert trin i følgende omskrivninger, specielt hvilke regneregler fra differentialregningen vi anvender. Udregning af den afledede funktion:

Udregningen trin for trin		Hvilke regler anvendes i de enkelte trin
$\left(g(x)\right)' = \left(x - f(x) \cdot \left(f'(x)\right)^{-1}\right)'$	(1)	
$= (x)' - \left(f(x) \cdot \left(f'(x)\right)^{-1}\right)'$	(2)	
$= 1 - \left(f'(x) \cdot \left(f'(x)\right)^{-1} + f(x) \cdot (-1) \cdot \left(f'(x)\right)^{-2} \cdot \left(f''(x)\right)\right)$	(3)	
$= 1 - 1 + f(x) \cdot \left(f'(x)\right)^{-2} \cdot f''(x)$	(4)	
$= f(x) \cdot \frac{1}{\left(f'(x)\right)^2} \cdot f''(x)$	(5)	
$= \frac{f(x) \cdot f''(x)}{\left(f'(x)\right)^2}$	(6)	

Heraf kan vi nu let se: Hvis x^* er et nulpunkt for *f*, dvs $f(x^*) = 0$ så er $g'(x^*) = 0$, dvs, at så er x* et supertiltrækkende fixpunkt.

Bemærkning. Vi har dermed *bevist*, at Newtons nulpunktsbestemmelse virker. Den iterationsfølge vi laver vil konvergere (hurtigt) mod nulpunktet x^*, hvis vi starter rigtigt. Af beviset fremgik, at vi startede med at vælge et bestemt interval *J* om x^*. Derefter så vi, at hvis blot ét punkt landede i dette interval, ville følgen hurtigt zoome ind i nulpunktet. Det betyder jo også, at hvis vi vælger vores startværdi indenfor intervallet, så konvergerer følgen (hurtigt!).

Vi kunne godt give os til at regne på, hvordan dette interval *J* skal være, når vi har en given funktion. Men det er alt for besværligt. Vi er tilfredse med den teoretiske indsigt, vores sætning giver. I praksis vil vi blot vælge en startværdi "tæt" på x^* og således, at $f'(x^*)$ ikke bliver 0 i nærheden.

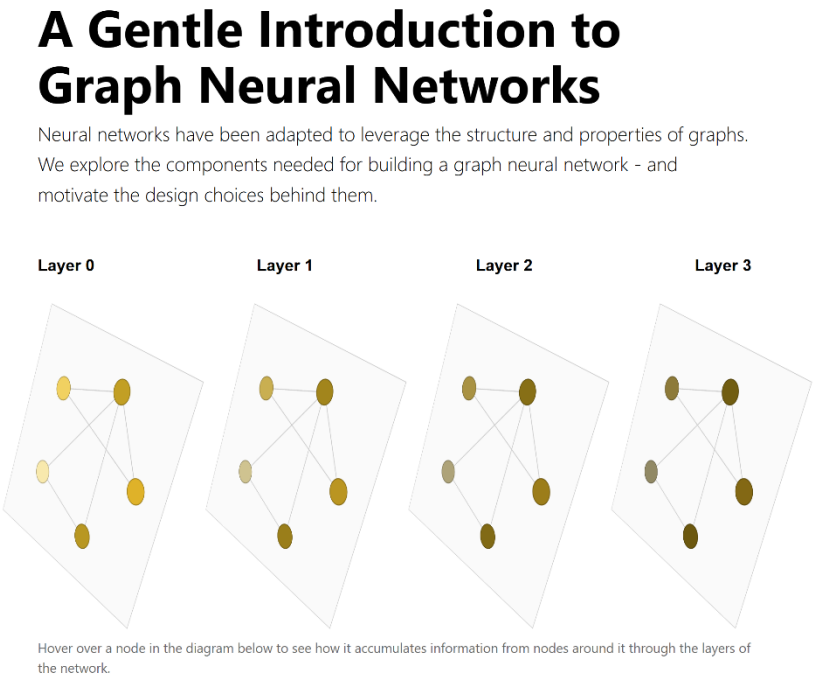
IV Graf neurale netværk

På website kan man finde denne lille film om DMI's anvendelse af Machine Learning til at frembringe vejrudsigter med meget kort varsel – især når det trækker op til alvorlige hændelser.

Med adgang til supercomputeren *Gefion* og med anvendelsen af en ny gren indenfor Machine Learning, som kaldes *Graf neurale netværk* er det ambitionen at nedsætte disse varslingstider fra ca 3 timer til et kvarter. Når det føres ud i livet, vil det have meget store samfundsmæssige gevinster, både økonomisk og menneskeligt

Udviklingen af de nye modeller er i sin vorden, og i filmen præsenterer Simon Kamuk Christiansen de grundlæggende ideer bag dette: Danmark inddeles i små kvadrater, måske i størrelsesorden 1 km^2, eller mindre når modellerne er udviklede. I hvert kvadrat opstilles et neuralt netværk over de mange parametre, der styrer vejret, og alle disse kvadrater er samtidig indbyrdes forbundet.

Selv med *Gefion* til rådighed er det urealistisk at arbejde med ét samlet dybt neuralt netværk for hele Danmark. Men med opdelingen som beskrevet, hvor de enkelte felter er knyttet sammen med de nærmeste naboer, så kan det håndteres.



Graf neurale netværk bygger naturligvis på teorien, vi har behandlet i de første dele, men har også sine særlige karakteristika, der gør at det er skilt ud. Et af de bedste steder at hente information om, og fordybe sig i dette nye fascinerende område er på bloggen https://distill.pub/2021/gnn-intro/.

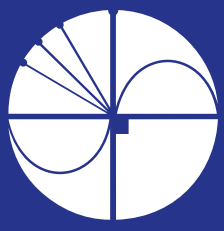
Et af områderne, men kan dykke ned i, er det som er vist her med sin forside.

Projektet **Træk virksomhederne ind i undervisningen** fortæller, hvordan matematik er uundværlig for avancerede virksomheder i et moderne samfund. I filmene besøger vi 12 forskellige virksomheder og institutioner, og lærer noget om den matematik, de anvender.

Denne afsluttende film er bygget op om et samspil mellem en præsentation af emnet Machine Learning hos DTU Compute, og besøg på tre virksomheder, henh. DMI, B&O og 3Shape, hvor teorierne føres ud i livet. Her møder vi engagerede unge matematikere, der er med helt i front af det matematiske arbejde. Alle har uddannelser med et betydelig matematisk indhold.

Til hver film kan du downloade undervisningsmaterialer. Her finder du øvelser, opgaveforløb, projekter og oplæg til studieretningsprojekter. Materialerne er opdelt i tre niveauer af sværhedsgrad fra 9 kl til 3 g. Emnerne er inspireret af filmen om virksomheden og den anvendte matematik, men man kan arbejde med materialet uden at have filmen kørende.

Hæftet om Matematikken bag Machine Learning knytter sig til filmen af samme navn.



matematik
i arbejde