NEURALT NETVÆRK TIL TEGNGENKENDELSE

B333

Lau Bech Lauritzen
Jasper Kjersgaard Juhl
Rasmus Jørgensen
Anders Rune Jensen
Ole Laursen
Martin Qvist

Maj 2002

AALBORG UNIVERSITET

Det Teknisk-Naturvidenskabelige Fakultet

Aalborg Universitet

Basisuddannelsen

Titel:

Neuralt netværk til tegnkendelse

Projektperiode:

P2,
4. februar – 27. maj, 2002

Projektgruppe:

B333

Gruppemedlemmer:

Jasper Kjersgaard Juhl
Anders Rune Jensen
Lau Bech Lauritzen
Rasmus Jørgensen
Martin Qvist
Ole Laursen

Vejleder:

Thomas D. Nielsen

Bivejleder:

Claus Monrad Spliid

Antal kopier: 10

Rapport – sideantal: 93

Appendiks – sideantal: 6

Totalt sideantal: 99

Synopsis:

Et af de områder hvor neurale netværk udmærker sig er tegngenkendelse som har fundet anvendelse på håndholdte computere. I rapporten undersøges mulighederne for at anvende et fremadførende neuralt netværk til at genkende enkeltstående håndskrevne tegn. Optrænet med fejlfunktionen sum af kvadrater vil dette netværk maksimere sandsynligheden for rigtig klassifikation. Netværket implementeres i C++, og der indsamles et datasæt til test.

For at kunne udføre testene undersøges den empiriske metode, som bygger på en positivistisk tankegang om at kun det der kan sanses, kan bidrage med viden. Ved hjælp af den empiriske metode kan man udføre eksperimenter og opnå pålidelige konklusioner der dog ikke er garanterede at være korrekte generelt.

To former for forbehandling undersøges, centrering og skalering. Anvendelse af disse viser sig at være væsentlige. Til at optræne netværket undersøges to algoritmer, en simpel gradientnedstigning og konjugeret gradient som er en mere avanceret algoritme – det viser sig dog at gradientnedstigning er hurtigst. Efter at have fundet størrelsen på det optimale netværk, testes der om en eftertræning af netværket på en bestemt testperson øger genkendelsesprocenten. Dette viser sig at være tilfældet; 92,3% uden eftertræning og 99,3% med en eftertræning.

Forord

Denne rapport er udarbejdet som P2-opgave på Aalborg Universitets basisuddannelse og er resultatet af 4 måneders arbejde med emnet neurale netværk og tegngenkendelse. Forudsætningerne for at læse den er kendskab til teorien bag funktioner af flere variable og deres partielle afledede, samt kendskab til lineær algebra.

I rapporten bruges følgende notation:

- Store bogstaver og fed skrift: en matrix $\mathbf{M}$.
- Små bogstaver og fed skrift: en vektor $\mathbf{v}$.
- Transponering af vektor: $\mathbf{v}^T$.
- Det indre produkt (prikproduktet) mellem to vektorer: $\mathbf{u}^T\mathbf{v}$.
- Middelværdien af x: $\langle x \rangle$.

Rapporten må frit distribueres i en hvilken som helst form.

God fornøjelse med rapporten,

Lau Bech Lauritzen	*Jasper Kjersgaard Juhl*
Rasmus Jørgensen	*Anders Rune Jensen*
Ole Laursen	*Martin Qvist*

Indhold

Kapitel 1

Indledning

Efterhånden som du læser denne rapport, genkender du de enkelte elementer i teksten og omsætter dem til noget der kan bruges videre i informationsbehandlingen i din hjerne. Du tænker måske ikke på læseprocessen som en genkendelse fordi den forløber på et ubevidst plan, men ikke desto mindre foretager din hjerne en genkendelse af bogstaverne og ordene i teksten.

Mens processen er ganske ubesværet for dig, er den et meget større problem for computere. Det er ikke indlysende hvordan et program skal bære sig af med at genkende et bogstav ud fra punkterne i et billede. Samtidig er problemet ikke kun teoretisk – et eksempel på en praktisk anvendelse er ved håndholdte computere.

En *håndholdt computer* er en lille lommecomputer til at holde styr på aftaler og lignende. Brugergrænsefladen består af en trykfølsom skærm og et pegeredskab.

På disse små computere er der ikke plads til et tastatur, og derfor må man bruge andre metoder til at kommunikere med computeren. En måde at gøre dette på er at skrive til computeren, dvs. skrive på den trykfølsomme skærm med det medfølgende pegeredskab, men dette kræver at computeren kan genkende den tekst der bliver skrevet.

En metode til at genkende denne tekst kan findes ved at efterligne den måde den menneskelige hjerne processerer information på. Hjernen består af et net af såkaldte neuroner, et *neuralt netværk*. Hvad dette nærmere indebærer gennemgås senere i rapporten; det er blot væsentligt at vide at kunstige neurale netværk, modeller af hjernens netværk, har vist sig brugbare til at løse visse typer problemer som det er svært at opstille fuldstændige modeller for. Et eksempel på et sådant problem er netop håndskriftsgenkendelse.

1.1 Problemformulering

Ønsket er ved hjælp af et kunstigt neuralt netværk at opnå den bedst mulige genkendelse af enkeltstående tegn skrevet på en håndholdt computers skrivefelt. På grund af neurale netværks beskaffenhed forudsætter dette en implementering som kan testes.

En række spørgsmål ønskes besvaret for at give indblik i de metoder og teorier der omgiver problemet:

1. Hvordan og hvorfor fungerer et kunstigt neuralt netværk?
2. Hvilke problemstillinger er der ved anvendelse af kunstige neurale netværk på håndholdte computere? Og hvordan kan disse løses?
3. Hvordan konstrueres et neuralt netværk til at løse tegngenkendelsesproblemet med en høj genkendelsesprocent?
4. Hvordan afgør man om et netværk er bedre end et andet? Kan man være sikker på at man har fundet det bedste netværk? Hvorfor kan man ikke det?
5. Hvordan kan neurale netværk tilpasses individuelle brugere? Kan det betale sig? Kan det lade sig gøre på en håndholdt computer?

1.2 Oversigt over rapporten

Introduktion til neurale netværk beskriver neurale netværks historie. Desuden introduceres de forskellige terminologier der bruges i forbindelse med neurale netværk.

Håndholdte computere beskriver håndholdte computeres grænseflade og specifikationer og hvilken indflydelse det får for et netværk.

Konstruktion af tegngenkendelsessystem beskriver og analyserer forskellige metoder til neurale netværk, herunder forbehandlingsmetoder og træningsmetoder. Desuden beskrives tegngenkendelsesprocessen. Afsnittet danner det teoretiske grundlag for de forskellige test vi senere udfører.

Implementering beskriver netværksprogrammet og de forskellige småprogrammer der er blevet brugt til test. Desuden retfærdiggøres valg af programmeringssprog.

Empirisk metode beskriver og analyserer den empiriske metode videnskabsteoretisk set. Analysen skal bruges til at underbygge pålideligheden og korrektheden af testene.

Testafsnittet opstiller hypoteser på baggrund af de forskellige metoder fra det teoretiske afsnit. Derefter benyttes den empiriske metode til at stille på forskellige parametre med henblik på at efterprøve disse hypoteser.

Konklusion opsummerer resultater fra hele rapporten, konkluderer på anvendeligheden af neurale netværk og opstiller nogle muligheder for forbedringer af en implementering på en håndholdt computer.

Kapitel 2

Introduktion til neurale netværk

Motivationen for forskningen i kunstige neurale netværk som et datalogisk værktøj hænger sammen med at visse problemer som mennesker nemt kan klare, som det at genkende genstande på billeder eller tale og forstå naturlige sprog, har vist sig særdeles svære at løse på computere fordi det er svært at opstille teorier for problemerne direkte.

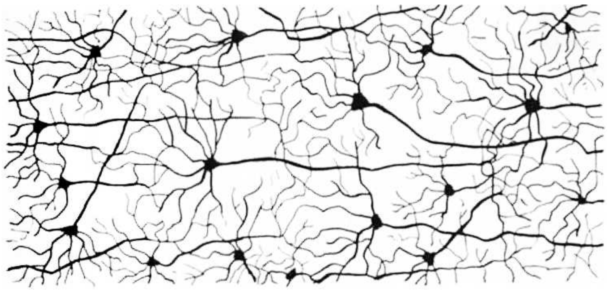
I stedet har man prøvet at efterligne hjernens funktion ved hjælp af et kunstigt neuralt netværk. Netværket skulle da kunne oplæres i en overvåget træningsfase hvor de rigtige resultater var til rådighed, ligesom ved et menneske, hvorefter det skulle være i stand til at *generalisere* ud fra disse og løse det givne problem ved fremtidige, ukendte data.

Overraskende nok er dette faktisk muligt, selv med en simpel model af hjernen. På denne måde undgås problemet med at skulle opstille en kompliceret model for det specifikke problemområde.

2.1 Biologiske neurale netværk

Udtrykket neuralt netværk dækker generelt over forskellige former for netværk som er opbygget som eller i det mindste inspireret af det biologiske neurale netværk i den menneskelige hjerne. Hjernen består af omkring 100 mia. *neuroner* [Brunak og Lautrup, 1988] som suppleres af ti gange så mange ikke-neurale støtteceller (f.eks. til energiforsyning) i kraniet. Neuronerne, der er en specialiseret form for nerveceller, sidder yderst på den stærkt foldede, ca. 3 mm tykke hjerneoverflade, hjernebarken, og er arrangeret i omkring seks lag.

Hver neuron er typisk forbundet til 1.000-10.000 andre neuroner (men tallet kan være helt op over 200.000) i netværket, og tankeprocesserne i hjernen foregår ved at elektriske impulser bevæger sig fra neuron til neuron ved hjælp af kemiske reaktioner [Harvey, 1994]. Påvirkningen fra en bestemt neuron til en anden kan enten være svækkende eller forstærkende og afhænger af styrken og frekvensen af de elektriske impulser som den første neuron udsender. Hvis den samlede



Figur 2.1: Neuralt netværk fra hjernebarken – kun 5% af de faktiske neuroner i området er synlige.

signalstyrke fra alle de forskellige indkommende signaler ved en bestemt neuron overskrider dens tærskel, „fyrer“ neuronen og sender et signal videre ad de udgående forbindelser.

Effektiviteten af kontaktfladerne mellem neuronerne, *synapserne*, kan ændres med tiden af det signal der passerer gennem dem. Denne ændring af overføringseffektiviteten formodes at være årsagen til at det er muligt for mennesker at huske og at lære [Brunak og Lautrup, 1988].

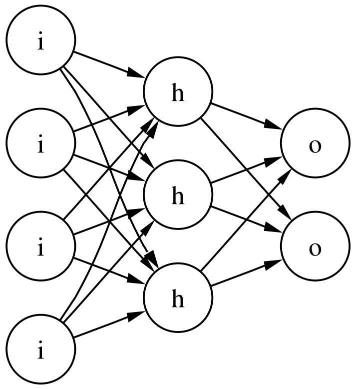
Et skønsmæssigt overslag på antallet af aritmetiske beregninger som en computer skulle foretage for at udføre den samme beregning som alle neuronerne i hjernen til sammen udfører analogt, er af størrelsesordenen 10^{18} operationer pr. sekund [Brunak og Lautrup, 1988]. Til sammenligning kan den hurtigste computer i verden per november 2001 teoretisk klare $1{,}23 \cdot 10^{13}$ operationer pr. sekund [Top500, 2001]. De to tal kan naturligvis ikke direkte sammenlignes idet hjernens parallellitet fungerer grundlæggende anderledes end en elektronisk processor, men illustrerer dog at hjernen besidder en meget stor beregningskapacitet.

2.2 Kunstige neurale netværk

Den simpleste model af hjernens komplicerede netværk med forbindelser på kryds og tværs er et *fremadførende* netværk, se figur 2.2 på den følgende side. En neuron repræsenteres nu blot af et tal under beregningerne i nettet, og de illustrerede forbindelser mellem neuronerne stammer fra den måde værdier føres fremad i netværket. Hver forbindelse har en *vægt* i form af et tal associeret med forbindelsen.

Dataene indsættes som tal i den ene ende af netværket (ved i’erne på figur 2.2) og sendes fremad lag for lag ved at den vægtede sum af de tilgående forbindelser til en neuron indsættes i en funktion hvis resultatet er værdien for neuronen. Værdien for en tilgående forbindelse beregnes som produktet mellem værdien x_i for den neuron den udgår fra, og forbindelsens vægt w_i, så beregningen af en neurons værdi inde i netværket foregår ved

$$y = g\left(\sum_{i=1}^{n} w_i x_i\right) = g(\mathbf{v}^T\mathbf{x}),$$



Figur 2.2: Et eksempel på et kunstigt fremadførende neuralt netværk

hvor g, *aktiveringsfunktionen*, har til formål at efterligne de menneskelige neuroners tærskeleffekt og f.eks. kunne defineres ved hjælp af en diskret model som $g(x) = 0$ for $x \leq t_i$ og $g(x) = 1$ for $x > t_i$ hvor t_i er tærsklen for neuronen. Valget af aktiveringsfunktion har stor indflydelse på netværkets evner og behandles mere indgående i afsnit 4.3.3.

Til slut fås ved det sidste neuronlag en vektor (i figur 2.2 ved o'erne) som repræsenterer resultatet af netværkets samlede beregning.

2.3 Træning og anvendelse

Man kan betragte det fremadførende netværk som en funktion der afbilder fra en vektor $\mathbf{x}$ i $\mathbb{R}^n$ til en vektor $\mathbf{y}$ i $\mathbb{R}^m$ hvor n er antallet af neuroner i første lag, og m er antallet i sidste lag. Det der er til rådighed for et givet problem, er allerede kendte datasæt med samhørende værdier af $\mathbf{x}$ og ønskede værdier $\mathbf{t}$, *træningssættet*, og funktionen skal så tilpasses efter disse så den returnerer en rigtig vektor $\mathbf{y}$ for en given fremtidig værdi $\mathbf{x}$. På denne måde approksimerer netværket den ukendte klassifikationsfunktion som det var for kompliceret at opstille en eksplicit forskrift for.

Tilpasningen foregår ved at justere vægtene i netværket så forskellen mellem netværkets faktiske resultat $\mathbf{y}$ og det ønskede resultat $\mathbf{t}$ minimeres for hele træningssættet – denne proces benævnes *træning* eller *indlæring*, og algoritmer til at foretage den er beskrevet i afsnit 4.4.

Ideen med netværket er at det efter træningsfasen skal kunne *generalisere* ud fra træningseksemplerne og give korrekte svar på værdier der ligger „i nærheden" af dem. Dette kan dels lade sig gøre fordi et neuralt netværk kan vises at kunne approksimere enhver funktion vilkårligt præcist, og dermed også klassifikationsfunktionen; dels kan det lade sig gøre fordi netværkets struktur bevirker at det kan være modstandsdygtigt over for mindre afvigelser eller fejl i dataene – begge egenskaber har visse forudsætninger som behandles i afsnit 4.3. Generaliseringsevnen kommer dog ikke af sig selv, og et fænomen som benævnes *overtræning*, kan ødelægge den. Overtræning er et problem der opstår når netværket kun kan genkende de bogstaver det er blevet trænet på. Vi kommer nærmere ind på overtræning og hvordan man kan formindske dette fænomen i 4.4.5.

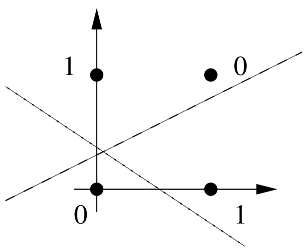
For at anvende et neuralt netværk på et praktisk problem skal dataene først omsættes til noget der kan sendes ind som en reel vektor i netværket. Desuden skal en række træningseksempler være klassificeret på forhånd – hver klasse tildeles en bestemt kodning i den resulterende vektor, f.eks. kan én af indgangene i vektoren have høj værdi, mens resten er lave.

På denne måde er det komplicerede genkendelsesproblem transformeret om til en række problemer i forbindelse med design af et neuralt netværk.

2.4 Historisk baggrund

Traditionelt set har der været to hovedgrupper inden for forskning i neurale netværk: en gruppe der var interesseret i netværkene som et middel til at løse komplicerede problemer som billed- og talegenkendelse, og en gruppe der var interesseret i netværkene som en måde at skaffe en større indsigt i hjernens funktion på [Madsen, 1993]. Udgangspunktet for dette projekt er den førstnævnte gruppe.

Forskning i kunstige neurale netværk startede i begyndelsen af 1940'erne hvor McCulloch og Pitts beskrev den såkaldte neuronmodel [Abe, 1995] inspireret af hjernens virkemåde. Neuronmodellen benytter en diskontinuert model for aktiveringsfunktionerne (som den diskrete aktiveringsfunktion nævnt i afsnit 2.2) der efterligner biologiske neuroner, heraf navnet. I slutningen af 50'erne og starten af 60'erne udvikledes en model med et enkelt lag kaldet *perceptronen* som kunne optrænes til at lære simple funktioner, f.eks. AND og OR. Men Minsky og Papert viste i 1969 at enkeltlagsnetværk kun kunne adskille lineært opdelelige data og f.eks. ikke engang kunne bruges til at simulere en simpel funktion som XOR [Mitchell, 1997, s. 89], se figur 2.3. Dette kombineret med det faktum at man ingen algoritmer havde til at træne flerlagsperceptroner op, betød at interessen for feltet og resultaterne for en tid stod stille inden for den datalogisk interesserede gruppe.



Figur 2.3: XOR-problemet – figuren viser resultatet fra en XOR-funktion (punkterne) hvor akserne angiver de to variable. Da punkterne med værdien 1 ikke kan adskilles fra dem med værdien 0 med en linje (to forsøg er vist med stiplede linjer), kan en perceptron ikke trænes til at efterligne denne funktion.

Der gik 16 år før den næste store udvikling satte ind [Madsen, 1993]. I 1982 udviklede Hopfield en yderst effektiv model baseret på neurale netværk til løsning af kombinatoriske problemstillinger. *Tilbageførselsalgoritmen*, backpropagation eller blot backprop, til træning af flerlagsnetværk blev alment kendt i 1986 gennem en artikel af bla. Rumelhart – algoritmen var dog allerede opfundet af Werbos (1975) og Parker (1985). Metoden forudsætter differentiable funktioner i neuronerne til forskel fra McCulloch og Pitts diskontinuerte neuronmodel.

Udviklingen af en træningsmetode og den forøgede tilgængelige regnekraft har sat kraftigt gang i forskningen på det datalogiske område hvilket har medført at der i dag kendes et utal af indlæringsalgoritmer og netværksarkitekturer. En del teoretiske problemer, f.eks. vedrørende hvordan man kan forudsige hvor mange neuroner der er brug for i netværket, er dog endnu uløste. Man er derfor henvist til at benytte empiriske eksperimenter. Dette og lignende problemer behandles mere indgående i kapitel 4 hvor de bliver aktuelle.

Kapitel 3

Håndholdte computere

En håndholdt computer er en computer som er så meget mindre end en bærbar computer at den passer i en lomme eller i håndfladen. Der er ikke plads til et tastatur, og det meste af overfladen er i stedet dækket af en trykfølsom skærm med noget kraftigt hardware bag i forhold til f.eks. lommeregnere af samme størrelse. Computeren betjenes næsten udelukkende ved at berøre den trykfølsomme skærm med en tilhørende pen.

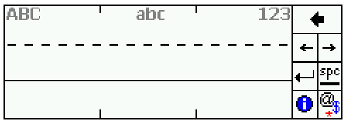
Salget af håndholdte computere har i de sidste år været kraftigt stigende. I USA er det steget fra 1,4 mio. enheder i 1999 til 4,9 mio. i 2001 [Bush, 2001]. Håndholdte computere er altså blevet interessante for mange.

3.1 Eksisterende grænseflade

Den mest almindelige måde at kommunikere med en håndholdt computer når det gælder indtastning af bogstaver og tal, er ved hjælp af skriftgenkendelse. Nederst på skærmen er der indrettet to eller tre felter – f.eks. et til store bogstaver, et til små og et til tal – og brugeren skriver så med pennen de ønskede tegn, ét ad gangen, i det rigtige felt på skærmen. De tegn det er muligt at få frem på denne måde, er

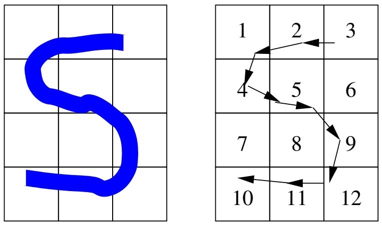
Figur 3.1: Fire af de mest udbredte, nyere håndholdte computere.

a til z samt nogle få ekstra som mellemrum og punktum. Specialtegn som % og & eller æ, ø og å skrives ikke direkte, men kan findes i en menu, se figur 3.2.



Figur 3.2: Et eksempel på en grænseflade til at skrive tal og bogstaver. Bogstaver skrives i de to første felter, tal i det sidste og en menu i højre side giver adgang til nogle ofte benyttede tegn som f.eks. mellemrum og linjeskift samt en pop op-menu som indeholder ekstra tegn, herunder æ, ø og å.

Den mest almindelige fremgangsmåde for computeren efter brugeren har skrevet et tegn er tilsyneladende ikke at fortolke det resulterende billede, men i stedet at inddele skrivefeltet i mindre felter [Willey, 1997], finde ud af hvilken vej pennen passerede gennem disse felter, se figur 3.3, og så sammenligne det med en database over baner.



Figur 3.3: Bogstavet s kan genkendes som den bane der går igennem felterne 3, 2, 1, 4, 5, 8, 9, 12, 11, 10 i den rækkefølge

Det bevirker bl.a. at der er en begrænsning på hvor mange forskellige tegn der kan genkendes fordi banerne kommer til at ligne hinanden. Desuden kræver det at brugerne lærer at tegne den rigtige slags baner uden f.eks. at løfte pennen, og det kan være nødvendigt at ændre banerne i forhold til almindelig praksis ved håndskrift (i'er skrives f.eks. nedefra og op for at undgå forveksling med l'er).

3.2 Forslag til forbedret grænseflade

Udgangspunktet for at bruge et neuralt netværk til tegnkendelsen i dette projekt er at der arbejdes med det færdige billede, selvom det også ville være muligt at arbejde med andre oplysninger, f.eks. om pennens bevægelse og hastighed. En fordel ved dette er at genkendelsen ikke bliver afhængig af hvilken rækkefølge de forskellige dele af bogstaverne tegnes i. Til gengæld forkastes nogle informationer som ellers kunne medvirke til en højere genkendelsesprocent.

Men det neurale netværk giver nogle muligheder for forbedringer af den håndholdtes grænseflade. Dels er der mulighed for at netværket kan trænes op på de data det faktisk klassificerer, og dermed bringes til at tilpasse sig de individuelle brugere, dels kan der benyttes en mere nuanceret klassificering der er i stand til at give mere end ét forslag hvis netværket giver et tvetydigt resultat.

Tilpasning til den individuelle bruger fordrer at der på den håndholdte indføres understøttelse af flere brugere for at undgå at genkendelsesevnen mistes hvis forskellige personer benytter computeren. Der er altså behov for en facilitet til at skifte bruger.

Desuden kræver tilpasningen at det korrekte resultat for de skrevne tegn er kendt. Hvis brugeren tegner et a, skal netværket vide at tegnet faktisk er tænkt som et a for at kunne træne sig på det. Dette kan imidlertid løses ved som udgangspunkt at antage at netværket altid gætter rigtigt og så bemærke om brugeren sletter tegnet efter det er skrevet. I så fald kan man undlade at træne netværket på netop det specifikke tegn (hvis man tillod sig at træne netværket på det bogstav der blev skrevet i stedet, var der risiko for at træne netværket forkert idet det ikke er muligt at vide om bogstavet blev slettet fordi det var forkert genkendt).

Endelig kan det neurale netværk indrettes sådan at det dels kan genkende flere forskellige tegn end hvad der er muligt med en eksisterende løsning som kræver tre forskellige felter til blot A–Z, a–z og 0–9, dels kan give et valg mellem flere mulige tegn hvis resultatet er tvetydigt. På denne måde bliver det hurtigere at rette fejl, og der kan spares plads i grænsefladen. Det stiller dog følgende krav til netværkets arkitektur:

- Det skal være muligt at afgøre om der er tvivl om hvilket tegn der er blevet skrevet, og hvilke tvivlen i givet fald står mellem.
- Netværksarkitekturen skal være tilstrækkeligt skalérbar til at kunne skelne alle de forskellige tegn.

3.3 Ydelsesmæssige krav til netværket

En håndholdt computer købes som en samlet enhed, og specifikationerne er derfor forudbestemt af producenten. De kan med få undtagelser ikke senere ændres ved en opgradering som ved stationære computere. Hardwaren sætter derfor nogle begrænsninger for hvad der er muligt.

De to mest begrænsende forhold er processorens hastighed og størrelsen af arbejdslageret. En typisk processor er en 206 MHz Intel StrongARM, og den samlede lagerplads til både permanente data (inkl. programmer) og arbejdshukommelse svinger mellem 16 og 64 Mb, med en standardkonfiguration på 32 Mb. Dette stiller altså følgende krav til netværket:

- Genkendelsesprocessen skal være forholdsvis hurtig, dvs. hurtig nok til at kunne afvikles i realtid på den håndholdte.
- Netværket må ikke bruge for meget hukommelse, heller ikke midlertidigt.

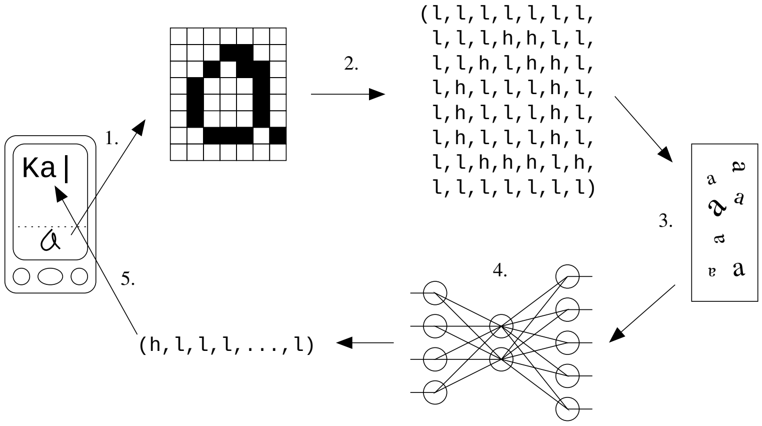
Hvor grænserne for hvad der er acceptabelt ydelsesmæssigt, helt præcist ligger, kan ikke præciseres uden nærmere kendskab til implementeringsforholdene på de forskellige håndholdte, men med hensyn til hastighed er en tommelfingerregel [Bickford, 1999] at processeringstiden højst må være 200 ms for at programmet skal virke flydende, og med hensyn til hukommelse skal der naturligvis være plads til andet end det neurale netværk på computeren.

Kapitel 4

Konstruktion af tegngenkendelsessystem

4.1 Oversigt over proces

Den følgende oversigt giver et hurtigt overblik over processen i tegnkendelsen, se figur 4.1, fra tegnet billede til færdigt bogstav.



Figur 4.1: Genkendelsesprocessen fra håndholdt til genkendt bogstav; h og l står for henh. høje og lave værdier.

4.1.1 Fra tegning til billede

Genkendelsen starter med at en bruger skriver på den trykfølsomme skærm. Information om hvilke punkter pennen har passeret leveres derefter videre i form af et færdigt billede hvor de berørte punkter er sorte og resten af punkterne er hvide. Det er også muligt at tage oplysninger om pennens hastighed og bane med i den efterfølgende behandling, men som nævnt i foregående kapitel forsøges det ikke i dette projekt.

4.1.2 Fra billede til vektor

For at muliggøre yderligere behandlinger på dataene omsættes de fra denne repræsentation til en n-dimensional vektor, hvor $n = 2880$ er produktet mellem højden og bredden af billedet, henholdsvis 60 og 48, og hvor indgangene i vektoren er 1 for de sorte punkter og -1 for de hvide. Valget af de specifikke værdier hænger sammen med det neurale netværks opbygning og begrundes senere.

4.1.3 Fra vektor til forbehandlede netværksdata

I stedet for at benytte de rå data direkte i netværket kan den n-dimensionale vektor sendes videre til en forbehandlingsproces som transformerer den om til en form som er bedre egnet til at indgå i netværket. Resultatet af forbehandlingen bruges så som data til neuronerne i inputdelen af netværket.

4.1.4 Fra netværksdata til resulterende vektor

Ved at sende disse værdier fremad i det neurale netværk gennem de vægtede summer fås en vektor af værdier i det sidste lag af neuroner. Antallet af neuroner i det sidste lag afgør naturligvis dimensionen af vektoren og kan vælges efter hvordan fortolkningen af vektoren fastlægges. En mulighed for valg af fortolkning er at benytte lige så mange neuroner som bogstaver så hver indgang i vektoren svarer til et bogstav – en positiv værdi tæt på 1 i en bestemt indgang angiver at netværket har klassificeret billedet som indgangens tilknyttede bogstav, mens negative værdier tæt på -1 angiver det modsatte. Hvis flere indgange er positive, angiver det et tvivlstilfælde hvor netværket ikke kan afgøre om billedet tilhører den ene eller den anden klasse.

En anden mulighed for fortolkning af netværkets resultat er at benytte en enkelt neuron i det sidste tal. På den måde fås et reelt tal som kan fortolkes ved at dele intervallet $[-1; 1]$ op i lige så mange underintervaller som antallet af bogstaver. Et enkelt interval repræsenterer da et bogstav – fordelen ved dette er at der spares nogle vægte fordi der er færre neuronerne i det sidste lag. Der er imidlertid to problemer ved denne fremgangsmåde, hvorfor den ovenfor beskrevne kodning benyttes i stedet.

Dels bliver det sværere for netværket at adskille tegnene fra hinanden så der skal flere mellemlagsneuroner til at klare den samme klassifikation [Bishop, 1995, s. 225-228] – og antallet af forbindelser vokser hurtigere ved tilføjelse af mellemlagsneuroner end ved tilføjelse af neuroner i det sidste lag (dette diskuteres i afsnit 4.3.2).

Figur 4.2: Tankeeksperiment der viser problemet med kun at bruge én outputneuron. Til venstre har netværket foretaget en sikker klassifikation af et d, men til højre er der tvivl om det er et b eller et d, og svaret bliver noget der kun kan tolkes som et entydigt c.

Dels opstår der et problem ved usikre klassifikationer hvor netværkets beregninger resulterer i at der er sandsynlighed for mere end ét bogstav. Med flere neuroner kan usikkerheden aflæses i form af at flere neuroner har en høj værdi, men med én neuron risikerer man at netværket eftersom det er en kontinuert funktion[1], returnerer en værdi der ligger mellem de to intervaller i et helt tredje interval, se figur 4.2 på foregående side – i så fald ville man ikke engang være klar over at der var tvivl om resultatet. Desuden påfører man netværket en kunstig sammenhæng mellem bogstaverne i alfabetet idet to bogstaver hvis intervaller ligger ved siden af hinanden, konceptuelt kommer til at ligge tættere på hinanden end to hvor der er større afstand mellem intervalgrænserne.

4.1.5 Fra resulterende vektor til bogstav

Den vektor som netværket returnerer kan slutteligt trivielt fortolkes ved at lede efter indgangene med de højeste værdier. På den måde fås et antal mulige bogstaver – i bedste fald kun ét, nemlig det som brugeren havde tænkt sig at skrive.

En implementering på en håndholdt computer ville blot skrive tegnet på skærmen hvis kun én indgang har en høj værdi, og ellers enten bede brugeren om at vælge mellem de sandsynlige bogstaver hvis der er flere indgange med høje værdier, eller også bede vedkommende om at prøve igen.

4.2 Forbehandling

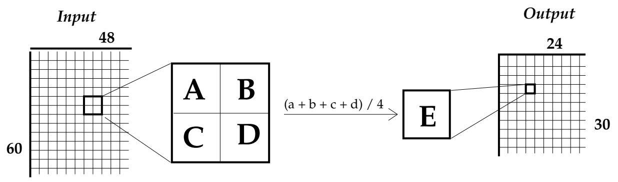
Det billede af et bogstav som brugeren tegner, kan umiddelbart omsættes til noget der kan sendes ind i det neurale netværk, som beskrevet i afsnit 4.1. Imidlertid kan der være gode grunde til at foretage en forbehandling af dataene:

1. En nedskalering eller anden transformation af dataene der reducerer deres dimensionalitet fra n til m, sparer $n - m$ neuroner i det første lag i netværket og skærer dermed også ned på antallet af vægte hvilket kan hjælpe til at undgå problemer med overtræning og gøre optræningen hurtigere. Til gengæld er en skalering tabsbetinget, så spørgsmålet er om det går ud over netværkets genkendelsesprocent.

2. En transformering af dataene kan gøre dem uafhængige af f.eks. placering af bogstavet i billedet eller bogstavets rotation eller hældning og således give en bedre genkendelsesevne.

Metoden til at forbehandle billederne er afgørende for forbehandlingens succes – en forkert ændring af billederne kan få alvorlige konsekvenser for netværkets genkendelsesevne. Derfor er der brug for grundige overvejelser inden anvendelse af en bestemt forbehandling.

[1]Hvis aktiveringsfunktionerne er kontinuerte, består netværket udelukkende af additioner, multiplikationer og kontinuerte transformationer og er derfor som funktion betragtet også kontinuert.



Figur 4.3: Startbilledet opdeles i 2x2 kvadrater. Hvert kvadrat udgør et punkt i det resulterende billedet som beregnes som gennemsnittet af de fire punkter i kvadrater.

4.2.1 Skalering af billedet

En oplagt metode til at begrænse dimensionen af dataene er ved at gøre billedet mindre og derved begrænse antallet af elementer i den vektor der skal til at beskrive billedet. Træningssættet består af billeder med størrelsen $48 \cdot 60 = 2880$ punkter. Hvis billedet skaleres ned til $24 \cdot 30 = 720$ punkter, kan antallet af neuroner i det første lag skæres ned til en fjerdedel.

En algoritme til at foretage en 1:2 skalering på hver led kan implementeres simpelt ved at opdele det oprindelige billede i 2x2 punkters kvadrater (se figur 4.2.1), beregne gennemsnittet af punkterne ved hjælp af

$$E = \frac{A + B + C + D}{4}$$

og tilføje E til det nye billede i stedet for 2x2 kvadratet. Bemærk at dette forudsætter sidelængder som 2 går op i, hvilket sidelængderne i vores træningssæt gør.

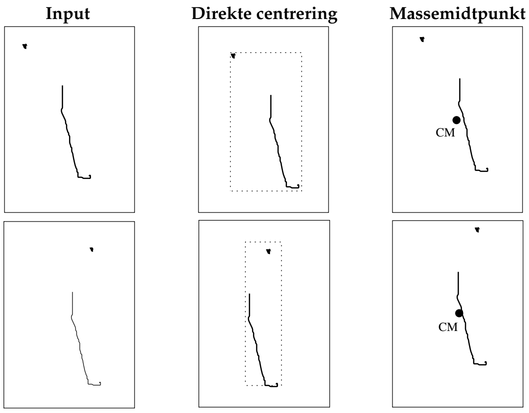
En sådan nedskalering af billedet er tabsbetinget, så der er en grænse for hvor meget det er muligt at skalere før det går ud over netværkets evne til at genkende bogstaverne. Umiddelbart skulle man tro at enhver tabsbetinget skalering af billedet vil påvirke genkendelsesevnen negativt, men som nævnt kan det begrænse antallet af vægte hvilket kan reducere problemer med overtræning og give en bedre generaliseringsevne. I appendiks B.2 på side 95 fremgår nogle skalerede bogstaver fra datasættet.

4.2.2 Centrering af bogstavet

To umiddelbart ens bogstaver kan opfattes meget forskelligt af netværket hvis de er placeret i hver sin side af billedet. Derfor er det oplagt at benytte en centrering for at ensarte billedet inden genkendelsen.

En simpel metode til at foretage centrering virker ved at man finder bogstavets yderste punkter og centrerer efter disse. Dette giver den efterspurgte ensartning ved meget ens bogstaver som næsten kun adskiller sig fra hinanden ved deres placering inden centreringen. Ved bogstaver der indeholder små ændringer ud over selve placeringen, er metoden dog ikke særlig effektiv til at ensarte billederne. F.eks. kan det få alvorlige konsekvenser hvis prikken over bogstavet „i" er placeret til højre på det ene billede og til venstre på det andet – her vil en centrering ikke medføre en ensartning, selvom kun en lille del af billedet er anderledes.

For at løse dette problem kan bogstavets „massemidtpunkt" tages til hjælp fordi dette i mindre grad end de yderste punkter er følsomt over for små ændrin-



Figur 4.4: I de to ovenstående eksempler forsøges centrering efter massemidtpunkt og midten af bogstavet. Det bemærkes at når den lille prik flyttes, ensartes billederne bedst ved brug af massemidtpunktet.

ger. Man kan i centreringen flytte bogstavet således at massemidtpunktet kommer til at ligge i midten af billedet og dermed opnå den ønskede ensartning. De to metoder er illustreret på figur 4.4.

Som det ses på figuren giver centreringen efter massemidtpunktet det bedste resultat da det netop er ønsket at ensarte de fleste punkter i to næsten ens bogstaver. Man kan dog i særlige situationer risikere at miste data ved brug af centreringen efter massemidtpunktet, f.eks. hvis der findes sorte punkter helt ud til billedets kant. Dette mener vi dog er et mindre problem da det næsten ikke opstår i praksis, hvorfor vi vælger at centrere efter massemidtpunket.

En algoritme til at foretage centreringen kan implementeres ved at betragte de enkelte værdier i punkterne som masser og herefter benytte samme formel som [Serway og Beichner, 2000, s. 269-273] til at finde massemidtpunktet (cm) i et fysisk objekt. Hvis $p(x, y)$ angiver værdien i punktet (x, y) på billedet kan formlen skrives som

$$x_{\mathrm{cm}} = \frac{1}{\sum_y \sum_x p(x,y)} \cdot \sum_x \sum_y x \cdot p(x,y)$$

$$y_{\mathrm{cm}} = \frac{1}{\sum_y \sum_x p(x,y)} \cdot \sum_y \sum_x y \cdot p(x,y)$$

hvor man summerer over alle punkter.

Det stiller dog som krav at værdierne opfører sig som en masse i fysikkens verden ved at massen aldrig kan blive mindre end 0. Det opfylder vores træningssæt ikke da gråtonerne beskrives i intervallet $[-1, 1]$, men dette kan løses ved at sætte $p(x, y) = p(x, y) + 1$ ved algoritmens start.

Når billedets massemidtpunkt er fundet, kan dette sammenholdes med billedets midtpunkt $(\bar{x}, \bar{y})$ som er defineret ved

$$\bar{x} = (b + 1)/2$$

$$\bar{y} = (h + 1)/2$$

hvor b er billedets bredde, og h er billedets højde.

Vi kan således beregne hvor mange punkter billedet skal flyttes ved

$$\Delta x = \bar{x} - x_{\mathrm{cm}}$$

$$\Delta y = \bar{y} - y_{\mathrm{cm}}.$$

Disse værdier skal naturligvis afrundes for at kunne bruges på de diskrete punkter i billedet.

For at illustrere effekten af en centering efter massemidtpunktet er der på bilag B.1 på side 94 vist nogle bogstaver der er centreret vha. vores forbehandlingsprogram.

4.2.3 Praktiske overvejelser

Da der i begge ovenstående tilfælde er tale om en forbehandling der skal implementeres på den håndholdte computer, er det værd at gøre sig nogle overvejelser om det vil få hastighedsmæssige konsekvenser at benytte forbehandlingen.

Det umiddelbare input består af 2880 punkter der hver kan beskrives med 1 bit. På trods af den mindre billedstørrelse forøger en skalering dette antal idet der er behov for flere bit til at beskrive decimaltal. Selv med fire byte pr. punkt, bliver dette dog ikke mere end højst 2880 byte ved en skalering i forholdet 1:2 på hver led. Det vil sige at lagerforbruget ikke er noget problem.

Beregningsmæssigt er centrering det mest krævende. Beregningen af massemidtpunktet kræver ét gennemløb pr. billede med to gangeoperationer for hvert punkt (én for $x \cdot p(x, y)$ og én for $y \cdot p(x, y)$). Dette giver dog ikke mere end omtrent $2 \cdot 2880 = 5760$ operationer, hvorfor forbehandlingen næppe vil udgøre noget praktisk hastighedsproblem.

4.3 Netværksarkitektur

4.3.1 Perceptronen

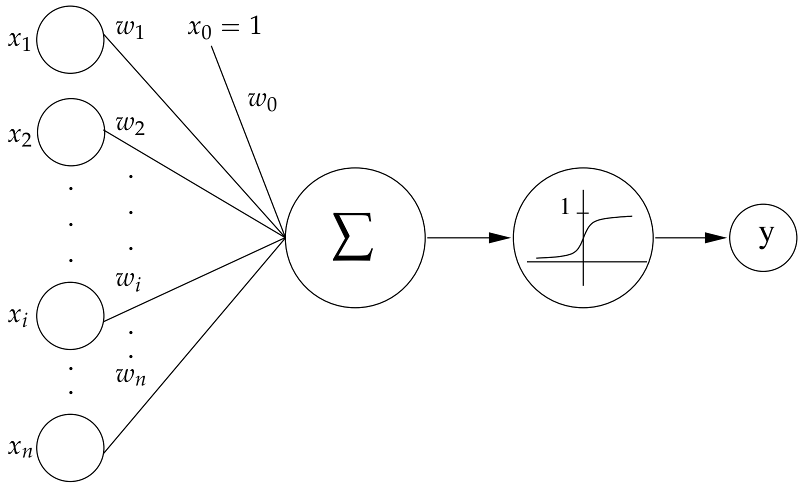
Som navnet antyder, sendes data i et fremadførende netværk kun fremad i netværket i den forstand at der ikke går forbindelser til det samme lag eller bagud. Den simpleste type fremadførende netværk er en perceptron som består af en enkelt neuron i det sidste lag, se figur 4.5 på den følgende side. Ud fra denne model kan sammensættes større netværk til at løse mere komplekse opgaver.

Bemærk at den klassiske perceptronmodel bruger fortegnsfunktionen (beskrevet i afsnit 2.2) som aktiveringsfunktion, mens figur 4.5 viser den logistiske funktion – denne bruges i dag i stedet for fortegnsfunktionen da den har nogle bedre egenskaber (se afsnit 4.3.3 på side 24).

Perceptronen består af en række input $x_1, x_2, \ldots, x_n$ der multipliceres med en række vægte $w_1, w_2, \ldots, w_n$ og derefter summeres

$$a = \sum_{i=1}^{n} w_i x_i$$

og en aktiveringsfunktion g virkende på den vægtede sum a, hvilket giver aktiveringsværdien $y = g(a)$ af neuronen.



Figur 4.5: Illustration af en perceptrons opbygning

Den vægtede sum repræsenterer en hyperplan som opdeler det n-dimensionale rum i to, men da denne opdeling ikke nødvendigvis er optimal når planen går gennem origo, definerer man ofte den vægtede sum med en konstant i form af en ekstra vægt lagt til

$$a = \sum_{i=1}^{n} w_i x_i + w_0.$$

Ved at indskyde en ekstra inputneuron x_0 (en „bias“) med den faste værdi 1 som illustreret på figur 4.5, kan dette omskrives til

$$a = \sum_{i=1}^{n} w_i x_i + w_0 x_0 = \sum_{i=0}^{n} w_i x_i$$

så den ekstra vægt ikke kræver særbehandling i det følgende.

4.3.2 Flerlagsmodeller

Ud fra den simple perceptronmodel kan modeller med flere lag udvikles, se f.eks. figur 4.6. Den vægtede sum for den j'te neuron bliver da

$$a_j = \sum_{i=0}^{n} w_{ji} x_i. \tag{4.1}$$

Ideen med flerlags neurale netværk er at de kan tilpasses til at approksimere enhver funktion [Bishop, 1995] og dermed også den ukendte funktion som kan klassificere bogstaver ud fra billeder. Denne approksimationsegenskab kan formuleres som at enhver funktion kan approksimeres med en vilkårlig lille fejl ved hjælp af et netværk med ét skjult lag hvis blot der er tilstrækkeligt mange neuroner i laget (og de benytter f.eks. den logistiske funktion som aktiveringsfunktion, se afsnit 4.3.3). Beviset for dette er et konstruktionsbevis (om end ikke praktisk konstruktivt) [Bishop, 1995], og for tilfældet med booleske funktioner er det meget simpelt.

Hvis man betragter et netværk med binære input så $x_i = 1$ eller 0, og outputtet enten er 1 eller 0 idet fortegnsfunktionen f benyttes som aktiveringsfunktion, kan netværket opfattes som at beregne booleske funktioner. For d input er antallet af mulige binære kombinationer der kan beskrives 2^d. For hver kombination med målværdien 1 tilføjes en neuron i det skjulte lag som „fyrer" på lige netop den udvalgte kombination ved at vægten i en forbindelse sættes til 1 hvis værdien fra neuronen i det foregående lag er 1, og -1 hvis værdien er et 0. Biasværdien for neuronen sættes så til $1 - b$ hvor b er antallet af værdier forskellig fra 0 i kombinationen. Endelig forbindes neuronerne i det skjulte lag med vægtværdien 1 til outputneuronen hvis bias sættes til $\frac{1}{2}$.

Når den rigtige kombination bliver præsenteret for netværket, vil den vægtede sum (uden biasværdien) derfor give b så neuronen giver værdien $f(b + 1 - b) = 1$. Alle andre kombinationer (inkl. dem med målværdien 0) giver højst $b - 2$ hvilket giver værdien $f(b - 2 + 1 - b) = 0$. Værdien ved outputneuronen bliver dermed $f(1 - \frac{1}{2}) = 1$ hvis en skjult neuron fyrer, og $f(0 - \frac{1}{2}) = 0$ hvis det ikke tilfældet. Dermed kan alle booleske funktioner udføres af et neuralt netværk hvis der er tilstrækkeligt mange neuroner i det mellemste lag.

Flerlagsmodeller giver dog anledning til nogle arkitekturmæssige vanskeligheder idet bestemmelsen af antallet af lag og antallet af neuroner i lagene ikke kan foretages analytisk, men må afprøves. Det generelle bevis for netværkenes universelle approksimationsevne indeholder ikke en praktisk opskrift.

Målet med designet af netværksarkitekturen er at opnå så få vægte som muligt samtidig med at netværket stadig har tilstrækkeligt med frihedsgrader til at kunne tilpasse sig den ønskede funktion. Hver vægt repræsenterer en frihedsgrad, og problemet med at have for mange af disse er dels at flere vægte betyder længere trænings- og processeringstid, dels at flere vægte giver større risiko for et fænomen der kaldes *overtræning* der går ud over netværkets generaliseringsevne (se afsnit 4.4.5).

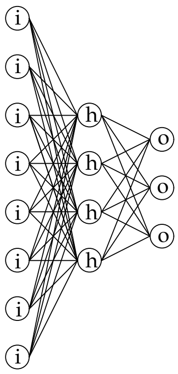
Som udgangspunkt viser sætningen om approksimationsevnen at der ikke er behov for mere end ét skjult lag, men ekstra lag kan muligvis gøre det muligt at skære ned på det samlede antal neuroner og forbindelser.

Den simpleste forbindelsesart er et fuldt forbundet netværk. Ud fra de inputdata som er indsamlet, skal netværket have 2880 inputneuroner. Outputlaget indeholder 29 neuroner, én neuron for hver outputværdi. Derudover er der et mellemlag som kan indeholde en variabel mængde af neuroner f.eks. 100. Hvis antallet af neuroner i inputlaget betegnes med i, antallet i det skjulte lag med h og antallet i outputlaget med o, bliver det samlede antal vægte n

$$n = i \cdot h + h \cdot o = (i + o) \cdot h \tag{4.2}$$

da der er i forbindelser bagud for hver neuron i det skjulte lag og tilsvarende h forbindelser for hver neuron i det sidste lag, se figur 4.6 på den følgende side.

Ved hjælp af (4.2) ses at med 100 neuroner i mellemlaget bliver antallet af vægte 290.900. Med 200 neuroner i mellemlaget bliver antallet af vægte fordoblet. En fordobling af vægte kan give en bedre tilpasningsevne da det giver en større mængde frihedsgrader til at tilpasse til funktionen. Men samtidig er der større risiko for overtræning som modsat giver en dårligere generaliseringsevne.



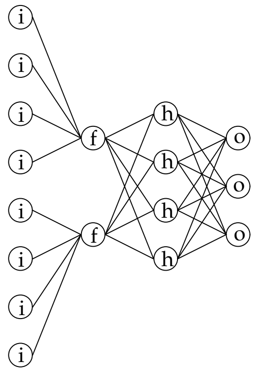
Figur 4.6: Et fuldt forbundet netværk med 44 vægte.

Det ses af (4.2) at en reduktion af antallet af vægte lettest foretages ved at skære ned på antallet af mellemlagsneuroner eller antallet af neuroner i det første lag, mens det omvendt næsten ikke giver noget at ændre antallet af neuroner i det sidste lag da i er meget større end o.

En anden mulighed er at lade være med at forbinde neuronerne fuldt ud. Man kan f.eks. indsætte et skaleringslag lige efter inputlaget, se figur 4.7, som ikke er fuldt forbundet, men hvor hver neuron i inputlaget er forbundet til f.eks. fire neuroner i næste lag. På denne måde opnås næsten samme effekt som hvis dataene først gennemgår en forbehandling i form af en ekstern skalering, bortset fra at netværket har mulighed for at justere på vægtene i den interne skalering. Antallet af vægte bliver da

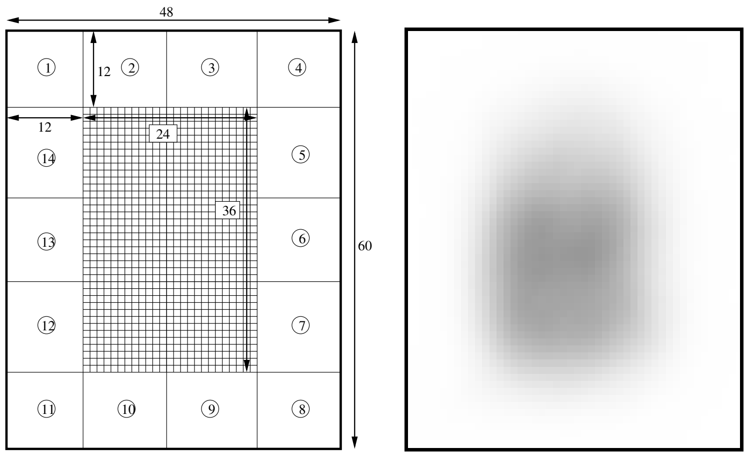
$$n = i + i/4 \cdot h + h \cdot o = i + (i/4 + o) \cdot h. \tag{4.3}$$

Antallet af vægte i et netværk med 100 skjulte neuroner bliver så 77.780 i stedet for 290.900, en betragtelig reduktion i antallet af vægte.



Figur 4.7: Et netværk med et nedskaleringslag – antallet af vægte er 28.

En anden form for intern skalering, *regionsskalering*, kan foretages ved at dele billedet op i flere regioner der skaleres uafhængigt af hinanden. Motivationen for dette er at der sjældent skrives helt ud til kanten af billedet – man kan derfor arbejde med en ramme og en midte som vist på figur 4.8 på den følgende side.



Figur 4.8: Til venstre et eksempel på regionsskalering hvor kanten består af et område med 14 kvadrater på 12×12 punkter, og det inderste område består af 212 kvadrater på 2×2 punkter. Til højre fremgår alle bogstaverne fra datasættet lagt oven i hinanden (efter en centrering som beskrevet i afsnit 4.2.2) og normaliseret – det ses at hovedparten af punkterne ligger i midten af billedet.

Selve skaleringen foretages på samme måde ved at undlade at forbinde neuronerne fuldt som ved den forrige interne skalering.

Det vil sige at $12 \cdot 12 = 144$ neuroner i rammen forbindes til én neuron, og tilsvarende $2 \cdot 2 = 4$ i midten. På denne måde bliver antallet af vægte

$$n = i + \frac{12 \cdot 12 \cdot 14}{144} \cdot h + \frac{24 \cdot 36}{4} \cdot h + h \cdot o \tag{4.4}$$

og reduceres fra 77.780 med almindelig 1:2 intern skalering til 28.780 ved brug af regionsskalering. Altså en betydelig reducering uden at det burde gå ud over det væsentlige indhold i billedet.

Selv om det er simpelt at konstatere at det giver færre vægte ved at justere på antallet af lag og neuroner og på den måde de er forbundet, så er det ikke muligt præcist at forudse hvilken effekt det får på netværkets genkendelsesprocent. Dette kan kun ske gennem eksperimenter som foretages i kapitel 7.

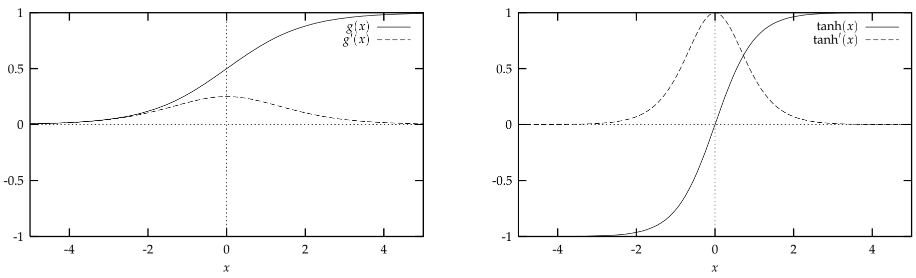
4.3.3 Aktiveringsfunktioner

Valget af aktiveringsfunktion dikteres naturligvis af et ønske om at opnå en transformation med bestemte egenskaber af den vægtede sum i de enkelte neuroner.

Den simpleste aktiveringsfunktion er identitetsfunktionen

$$g(x) = x,$$

svarende til at værdien ikke transformeres. Problemet med denne er at den er lineær hvilket begrænser hele netværket til at være en lineær afbildning idet det udelukkende består af summationer og multiplikation med konstanter (vægtene). At sætte flere lag sammen hjælper ikke på det problem for når hvert lag udgør en lineær afbildning, udgør sammensætningen af to lag også en lineær afbildning –



Figur 4.9: Plot af aktiveringsfunktionerne og deres afledede, til venstre sigmoid og til højre tanh

dette følger generelt af

$$g(f(c_1x_1 + c_2x_2)) = g(c_1f(x_1) + c_2f(x_2)) = c_1g(f(x_1)) + c_2g(f(x_2))$$

hvor f og g er lineære transformationer, hvoraf det ses at $g(f(x))$ også opfylder linearitetskravet [Jensen et al., 2000]. Aktiveringsfunktionens hovedformål er altså at bringe ikke-liniaritet ind i netværket.

En anden vigtig egenskab aktiveringsfunktionen skal besidde, er at den skal være differentiabel for at muliggøre træning af netværket ved hjælp af tilbageførselsalgoritmen som benytter sig af partielle afledede (algoritmen gennemgås i afsnit 4.4.2). Af samme grund har aktiveringsfunktionens afledede også en vis betydning.

En af de populære funktioner er den logistiske funktion, eller sigmoidfunktionen,

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

som tilhører en gruppe funktioner der „presser“ værdien sammen ved at transformere x fra intervallet $]-\infty;\infty[$ til et endeligt interval i $\mathbb{R}$, her $]0;1[$ (se figur 4.9). Sigmoidfunktionen er monoton og differentiabel med den simple afledede

$$\begin{aligned} g'(x) &= \frac{-\frac{d}{dx}(1+e^{-x})}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}} \frac{1+e^{-x}-1}{1+e^{-x}} \\ &= g(x)(1-g(x)) \end{aligned}$$

som kan udtrykkes ved funktionsværdien selv, hvilket sparer på antallet af numeriske operationer i en given implementering.

En anden populær funktion af samme type er hyperbolsk tangens

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

hvis værdimængde er $]-1;1[$. Funktionen er symmetrisk omkring 0 (se figur 4.9) hvilket kan have tendens til at give hurtigere konvergens under optræningen end f.eks. sigmoidfunktionen [Sarle, 1999], tilsyneladende af ikke helt forståede årsager. Desuden har tanh den egenskab at den blot er en lineær transformation af sigmoidfunktionen, hvor

$$\begin{aligned}\tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{1 + 1 - 1 - e^{-2x}}{1 + e^{-2x}} \\ &= \frac{2}{1 + e^{-2x}} - 1 = 2g(2x) - 1,\end{aligned}$$

der viser at den har samme egenskaber, f.eks. at den differentiabel.

Dens afledede er

$$\begin{aligned}\tanh'(x) &= \frac{d}{dx}\frac{\sinh(x)}{\cosh(x)} \\ &= \frac{\cosh(x)\frac{d}{dx}\sinh(x) - \sinh(x)\frac{d}{dx}\cosh(x)}{\cosh^2(x)} \\ &= \frac{\cosh(x)\cosh(x) - \sinh(x)\sinh(x)}{\cosh^2(x)} \\ &= 1 - \tanh^2(x)\end{aligned}$$

Begge funktioner har den egenskab at de er tilnærmelsesvist linieære i et interval på x-aksen omkring origo og derefter nærmer sig deres to vandrette asymptoter. Af denne grund kan det være en fordel ikke at sætte målværdierne helt til værdierne ved asymptoterne fordi vægtene så bliver drevet mod uendelig. Dette er i sig selv ikke så stort et problem da afrundingsfejl i beregningen af aktiveringsfunktionen hurtigt kan få værdierne til at konvergere – f.eks. er $\tanh(8) \approx 0{,}99999977493$ som med de seks decimalers nøjagtighed almindelige kommatal i C har, bliver til 1 (kommatal med dobbeltpræcision har 15 cifre, men $\tanh(18) \approx 0{,}99999999999999953610$). Problemet er snarere at de afledede af funktionerne ved disse x-værdier er meget tæt på nul hvilket får den ene af de træningsalgoritmer der gennemgås i afsnit 4.4, gradientnedstigning, til at tage små skridt så træningen er lang tid om at konvergere.

For tanh kunne man f.eks. vælge målværdierne til at være enten $-0{,}9$ eller $0{,}9$. På denne måde er målværdierne stadig tæt på asymptoterne og over det lineære stykke så netværket tvinges til enten at vælge enten den ene yderlighed eller også den anden – derved opnås en effekt som ved fortegnsfunktionen.

4.3.4 Startværdier for vægte

Tildeling af startværdier for vægtene kan principielt ske helt vilkårligt, bortset fra at de ikke må være ens for at undgå symmetri i netværket og ikke må være nul (da de partielle afledede så bliver nul, se afsnit 4.4.2). For at opnå hurtigst mulig konvergens under optræningen kan det dog være en fordel af begrænse den måde vægtene vælges på [Sarle, 2001].

Hvis vægtene vælges for små, tager gradientnedstigningen små skridt[2] – hvis vægtene er for store, driver de vægtede summer aktiveringsfunktionerne ud til asymptoterne hvor de afledede er meget små så gradientnedstigning som nævnt igen tager små skridt. Et kompromis mellem disse to yderligheder er derfor at

[2] Dette kan ses af udtryk (4.9) i afsnit 4.4.2.

vælge vægtene så de vægtede summer kommer til at ligge i intervallet $[-1;1]$ [Bishop, 1995] – omkring det sted hvor aktiveringsfunktionerne krummer mest. For at undgå symmetriproblemet kan det gøres efter en metode der genererer vilkårlige tal med en passende fordeling med variansen σ^2 og middelværdien 0.

Det viser sig så at σ skal skalere efter $\sigma \propto n^{-\frac{1}{2}}$ [Bishop, 1995], se evt. appendiks C på side 96.

4.4 Træningsmetoder

Træning af et neuralt netværk foregår ved at ændre på vægtene i forbindelserne og kan formaliseres ved at definere en *fejlfunktion* E over netværket som udtrykker hvor stor fejlen er for en datavektor $\mathbf{x}$ givet den ønskede værdi $\mathbf{t}$. Fejlen er naturligvis en funktion af vægtene, $\mathbf{w}$, hvorfor træningen kan anskues som en tilpasning af en funktion af flere variable. Det der søges efter, er den værdi $\mathbf{w}^*$ af vægtene som giver den mindste fejl over alle træningseksemplerne, hvilket svarer til at minimere $E(\mathbf{w})$.

Minimeringen af funktionen E kan ikke løses analytisk for et ikke-lineært netværk. I stedet kan iterative, numeriske algoritmer bruges, men disse garanterer dog kun at finde et lokalt minimum. Fælles for algoritmerne er at de benytter sig af de partielle afledede af E med hensyn til de forskellige vægte hvilket giver $\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n})$. Denne kan beregnes effektivt vha. tilbageførselsalgoritmen der, som nævnt i afsnit 2.2, blev populariseret af en artikel af Rummelhart, Hinton og Williams i 1986.

I Rummelhart et al.'s artikel benyttes gradientvektoren direkte til at opdatere vægtene ved *gradientnedstigning* [Madsen, 1993] som blot går ud på for hvert skridt at gå i retningen $-\nabla E$, men der findes andre algoritmer som reducerer fejlen hurtigere [Bishop, 1995], bl.a. *konjugeret gradient* som behandles senere i dette kapitel.

4.4.1 Fejlfunktioner

For alle de betragtede fejlfunktioner gælder at den samlede fejl som netværket forårsager over et træningssæt, er summen af fejlene ved for de enkelte træningseksempler $(\mathbf{x}_d,\mathbf{t}_d)$:

$$E = \sum_d E_d.$$

Fejlfunktionen for et træningseksempel kan defineres på mange måder, en ofte benyttet er *sum af kvadrater*

$$E_d(\mathbf{w}) = \frac{1}{2}\sum_k (t_{kd} - y_{kd}(\mathbf{x}_d, \mathbf{w}))^2, \tag{4.5}$$

hvor man summerer over kvadratet på hver komposant i vektoren $\mathbf{t}_d - \mathbf{y}_d(\mathbf{x}_d, \mathbf{w})$ for et givet træningseksempel d; $\mathbf{y}_d(\mathbf{x}_d, \mathbf{w})$ angiver outputtet fra netværket for inputtet $\mathbf{x}_d$ og vægtene $\mathbf{w}$.

4.4.2 Beregning af ∇E med tilbageførselsalgoritmen

I et flerlagsnetværk kan det sidste lag trænes som ved en model uden skjulte lag fordi det foregående skjulte lag kan betragtes som et inputlag. Dette gør det muligt at beregne fejlen for hver af neuronerne i det sidste lag direkte idet der er adgang til både outputværdi og ønsket værdi. Det er straks sværere at beregne fejlene i de skjulte lag fordi der i dem ikke er adgang til de ønskede værdier. Her kommer den algoritme der beskrives i det følgende [Bishop, 1995; Mitchell, 1997] ind, idet man ved hjælp af den kan propagere et udtryk for fejlen baglæns gennem netværket på en effektiv måde.

Det der ønskes beregnet til de forskellige træningsalgoritmer, er de partielle afledede af fejlfunktionen med hensyn til hver vægt i netværket:

$$\frac{\partial E}{\partial w_k}.$$

Bemærk at vi i det følgende nøjes med at betragte den afledede af fejlen for et enkelt træningseksempel d – da fejlen over hele træningssættet blot er summen af de fejlene for hver træningseksempel, kan den partielle afledede af E findes ved at summere de partielle afledede for hvert træningseksempel som en følge af reglerne for differentiation:

$$\frac{\partial E}{\partial w_k} = \sum_{d=1}^{m} \frac{\partial E_d}{\partial w_k}$$

hvor m er antallet af træningseksempler. Algoritmen har som forudsætning at aktiveringsfunktionerne er differentiable samt at netværket er fremadførende så de partielle afledede kan udledes ved hjælp af kædereglen.

Idet fejlfunktionen ved en neuron afhænger af både aktiveringsværdierne og de tilknyttede vægte i neuronens tilgående forbindelser, jvf. (4.1), kan kædereglen benyttes på den partielle afledede af E_d med hensyn til vægten w_{ji} mellem den i'te neuron i det foregående lag og neuronen j i det betragtede lag

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial a_j}\frac{\partial\left(\sum_{i=1}^{n} w_{ji}x_i\right)}{\partial w_{ji}} = \frac{\partial E_d}{\partial a_j}x_i$$

hvor n er antallet af neuroner i det foregående lag, x_i betegner den i'te neurons aktiveringsværdi, a_j betegner den j'te neurons vægtede sum, og andet skridt følger af (4.1).

Defineres nu

$$\delta_j = \frac{\partial E_d}{\partial a_j},$$

får man

$$\frac{\partial E_d}{\partial w_{ji}} = \delta_j x_i \tag{4.6}$$

hvoraf et generelt udtryk for den partielle afledede er fundet. Bestemmelsen af δ_j afhænger af hvordan netværket er opbygget – for et simpelt fremadførende netværk er der to tilfælde: enten sidder neuron j i et skjult lag eller også i det sidste lag.

Neuron i sidste lag

Den vægtede sum a_j påvirker fejlfunktionen gennem outputtet $o_j = g(a_j)$ hvor g betegner aktiveringsfunktionen, f.eks. den logistiske funktion eller hyperbolsk tangens, så anvendelse af kædereglen giver

$$\delta_j = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial a_j} = \frac{\partial E_d}{\partial o_j} g'(a_j) \tag{4.7}$$

Benyttes fejlfunktionen fra (4.5), ses at E_d er konstant for alle andre neuroner end j hvorved man får

$$\begin{aligned}
\frac{\partial E_d}{\partial o_j} &= \frac{1}{2}\frac{\partial (t_j - o_j)^2}{\partial o_j} \\
&= \frac{1}{2} \cdot 2(t_j - o_j)\frac{\partial (t_j - o_j)}{\partial o_j} \\
&= (t_j - o_j) \cdot (-1) \\
&= o_j - t_j
\end{aligned}$$

Indsættelse i (4.7) giver

$$\delta_j = g'(a_j) \cdot (o_j - t_j). \tag{4.8}$$

Dvs. at δ-værdien kan vha. af differensen mellem outputtet o_j og målværdien t_j.

Skjult neuron

Måden hvorpå den vægtede sum i en skjult neuron, a_j, kan påvirke resultatet af netværket og dermed ændre fejlfunktionen, er igennem de neuroner hvis indgang afhænger af a_j, dvs. de neuroner k som er direkte forbundet med neuronen j. Hvis disse neuroners vægtede summer betegnes med a_k, kan kædereglen benyttes til at bestemme fejlen:

$$\begin{aligned}
\delta_j = \frac{\partial E_d}{\partial a_j} &= \sum_k \frac{\partial E_d}{\partial a_k}\frac{\partial a_k}{\partial a_j} \\
&= \sum_k \delta_k \frac{\partial a_k}{\partial o_j}\frac{\partial o_j}{\partial a_j} \\
&= \sum_k \delta_k w_{kj} g'(a_j)
\end{aligned}$$

hvor $o_j = g(a_j)$ angiver aktiveringen af neuron j. For en skjult neuron gælder altså

$$\delta_j = g'(a_j) \cdot \sum_k \delta_k w_{kj} \tag{4.9}$$

Som det ses, kan δ_j udtrykkes på en simpel måde ved δ_k, og såfremt neuronen k ikke er en neuron i det sidste lag, kan proceduren gentages for det næste lag osv. til man når en neuron i det sidste lag hvor δ kan beregnes direkte.

I praksis starter man naturligvis bagerst i netværket med at propagere δ-værdierne tilbage gennem neuronerne, hvoraf navnet på algoritmen:

1. For hver neuron i det sidste lag bestem fejlen δ_k ved hjælp af (4.7).
2. For hver neuron i det foregående skjulte lag (lag $n-1$) beregn fejlen δ_j ved hjælp af (4.9). Bemærk at formlen benytter δ_k-værdierne fra lag n.
3. Gentag det foregående skridt med n én mindre indtil alle de skjulte lag er gennemgået.

Den partielle afledede i hver vægt kan derefter findes ved (4.6).

Tidskompleksitet

Tilbageførselsalgoritmen har vist sig i praksis at være meget anvendelig pga. tidskompleksiten af algoritmen [Bishop, 1995]. Tidskompleksiteten for en beregning af E_d som funktion af $\mathbf{w}$ er $O(n)$ hvor n er antallet af vægte, fordi den tunge del af beregningen ligger i fremadpropageringen af værdier hvor der er én multiplikation samt én addition pr. vægt.

Uden tilbageførselsalgoritmen kan bestemmelsen af de partielle afledede af E_d gøres numerisk for hver enkelt vægt ved hjælp af differentialkvotienten

$$\frac{E_d}{w_{ji}} \approx \frac{E_d(w_{ji}+\epsilon) - E_d(w_{ji})}{\epsilon}$$

hvor ϵ er en lille værdi. For at beregne $E_d(w_{ji}+\epsilon)$ kræves en ekstra fremadføring for hver vægt, dvs. tidskompleksiteten pr. vægt er af størrelsesordenen $O(n)$ hvilket giver $O(n^2)$ i alt.

Tilbageførselsalgoritmen derimod kan bruge delresultater i form af δ'erne beregnet undervejs til at bestemme de partielle afledede ved (4.6). Dette kræver en enkelt gangeoperation og en addition pr. vægt så tidskompleksiteten af algoritmen samlet bliver $O(2n)$ (en fremadføring og en tilbageføring) i stedet for $O(n^2)$. Da antallet af vægte som oftest er meget stort, f.eks. 1.000-500.000, giver tilbageførselsalgoritmen derfor en meget stor hastighedsforbedring. Desuden er der den fordel at beregningen ikke bygger på en approksimation.

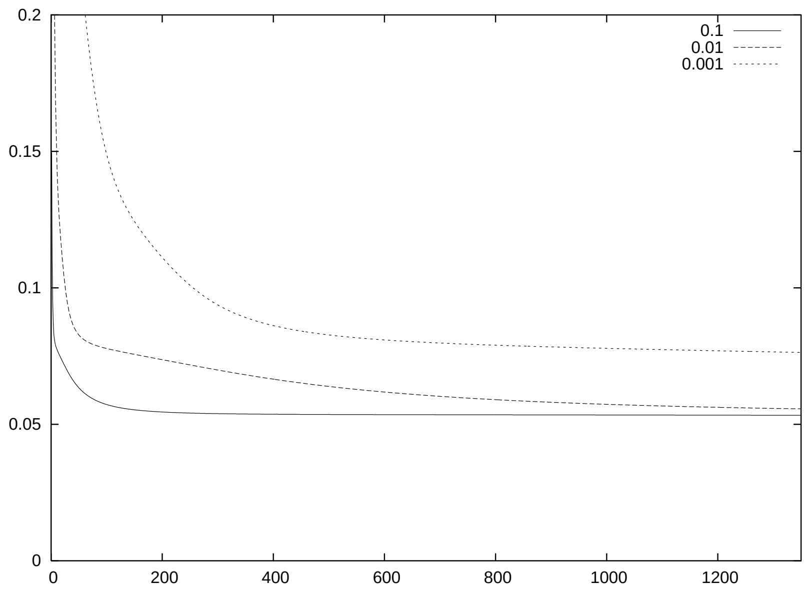
4.4.3 Gradientnedstigning

Differentieres fejlfunktionen med hensyn til samtlige vægte fås gradientvektoren som peger i den retning hvor fejlfunktionen stiger mest, så ved at gå i modsat retning af gradienten minimeres E. Hver vægt kan opdateres ved hjælp af udtrykket

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \tag{4.10}$$

hvor Δw_{ji} er den værdi der skal lægges til vægten w_{ji}, og træningskonstanten η fastsætter skridtstørrelsen.

Værdien af η kan dog være lidt svær at fastlægge. Holdes η konstant under hele optræningen, kan træning med en lille værdi af η være meget langsom om at konvergere mod et minimum som det ses på figur 4.10 på næste side.



Figur 4.10: Fejlen som funktion af antallet af iterationer med forskellige værdier af η for et netværk med ét skjult lag med to neuroner og én input- og én outputneuron; bemærk at en iteration svarer til én komplet gennemløbning af træningssættet. Netværket forsøger at approksimere 30 træningspunkter i intervallet $[0; 2\pi]$ genereret efter forskriften $0{,}9 \sin(x) + \epsilon$ hvor ϵ er normalfordelt støj med $\mu = 0$ og $\sigma = 0{,}07$. Når η sættes lavere, konvergerer netværket langsommere.

Hvis man derimod sætter η for højt, kan det få algoritmen til at springe over et minimum og aldrig konvergere, men kan til gengæld også ende i et bedre lokalt minimum. Et eksempel på dette fremgår af figur 4.11 på den følgende side. En løsning kunne være at lade η være en funktion af antallet af iterationer, men spørgsmålet er så hvad funktionen skulle være.

Et andet problem med η er at den samlede ændring i den vægtede sum for en neuron er afhængig af hvor mange indgående forbindelser der er til neuronen, fordi bidragene summeres:

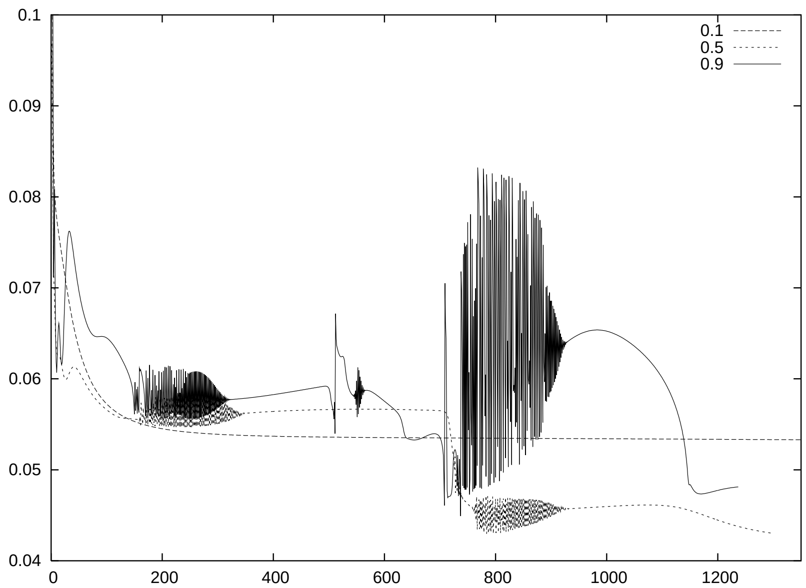
$$a_j = \sum_{i=0}^{n} w_{ji} x_i$$

Dette betyder at der er mulighed for at en neuron i det første skjulte lag med f.eks. 2880 forbindelser oplever en for stor ændring i aktiveringsværdien for en givet η, mens en neuron i det sidste lag med f.eks. 100 forbindelser samtidig opnår små ændringer for samme η. Dividerer man for hver neuron den benyttede værdi af η med antallet af forbindelser n til neuronen, undgås dette:

$$\bar{\eta} = \frac{\eta}{n}$$

Den ovenstående definition af gradientnedstigning opdaterer vægtene efter hvert træningseksempel og kaldes *stokastisk gradientnedstigning*. Proceduren er:

1. Fremadfør et træningseksempel.



Figur 4.11: Fejlen som funktion af antallet af iterationer på samme problem som på figur 4.10. Når η sættes højt, opfører algoritmen sig mere ustabilt, og netværket ender i forskellige minima.

2. Beregn δ-værdierne ved tilbageførsel.
3. Opdater vægtene efter (4.10) og (4.6).
4. Afslut hvis termineringskriteriet er opfyldt, ellers gå til første skridt igen med et nyt træningseksempel.

Termineringskriteriet kan vælges på flere måder. En mulighed er at stoppe når fejlen når under en bestemt tærskel – afsnit 4.4.5 gennemgår hvorfor dette dog kan være en dårlig idé.

En alternativ mulighed for gradientnedstigning er at finde de partielle afledede for hele træningssættet før vægtene opdateres. Algoritmen bliver så

1. Fremadfør et træningseksempel.
2. Beregn δ-værdierne ved tilbageførsel.
3. Beregn gradienten vha. (4.6) og læg den til den summerede gradient.
4. Gå til det første skridt medmindre hele træningssættet er løbet igennem.
5. Opdatér vægtene efter
$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}.$$
6. Afslut hvis termineringskriteriet er opfyldt, ellers gå til det første skridt idet der startes forfra på træningssættet.

Imidlertid kan den stokastiske gradientnedstigning have lettere ved at slippe ud af lokale minima da gradienten for hvert træningseksempel kan ændre sig en del. Desuden kan stokastisk nedstigning konvergere væsentligt hurtigere ved store træningssæt med megen redundans [LeCun et al., 1998]. Dette kan forstås intuitivt ved at betragte et datasæt som består af træningseksempler der er meget ens. Benytter man sig af en ikke-stokastisk gradientnedstigning vil gradienten kun tage højde for den redundante data én gang idet resultatet blot bliver en længere gradient som man så må kompensere for ved en lavere værdi af η for at undgå ustabilitet. Benytter man derimod den stokastiske variant så har vægtene allerede flyttet sig når det næsten samme eksempel optræder anden gang hvorved algoritmen også får gavn af træningseksemplet anden gang. Resultatet er hurtigere konvergens, hvorfor den stokastiske udgave benyttes i dette projekt.

Nedstigning med inerti

Tilbageførselsalgoritmen kombineret med gradientnedstigning giver nogle problemer: rummet for fejlfunktionen indeholder flere lokale minima og simpel gradientnedstigning kan være tilbøjelig til at sidde fast i et af disse [Mitchell, 1997]. Desuden er skridtstørrelsen for gradientnedstigningen afhængig af gradientvektoren for fejlfunktionen så på stejle områder af fejlfunktionens overflade vil gradientnedstigningen foregå hurtigt, mens den på lange flade stykker vil gå meget langsom.

En måde at komme uden om dette problem er ved at variere η som funktion af gradientvektoren til fejlfunktionen. En anden metode som kan afhjælpe begge ovenstående problemer, er at tilføje en vis afhængig af den foregående opdatering. Ved at lade en vægtopdatering $\Delta w_{ji}(n)$ ved skridt n afhænge af den foregående vægtopdatering $\Delta w_{ji}(n-1)$ opnås en inertieffekt:

$$\Delta w_{ji}(n) = -\eta \frac{\partial E}{\partial w_{ji}} + \mu \cdot \Delta w_{ji}(n-1) \tag{4.11}$$

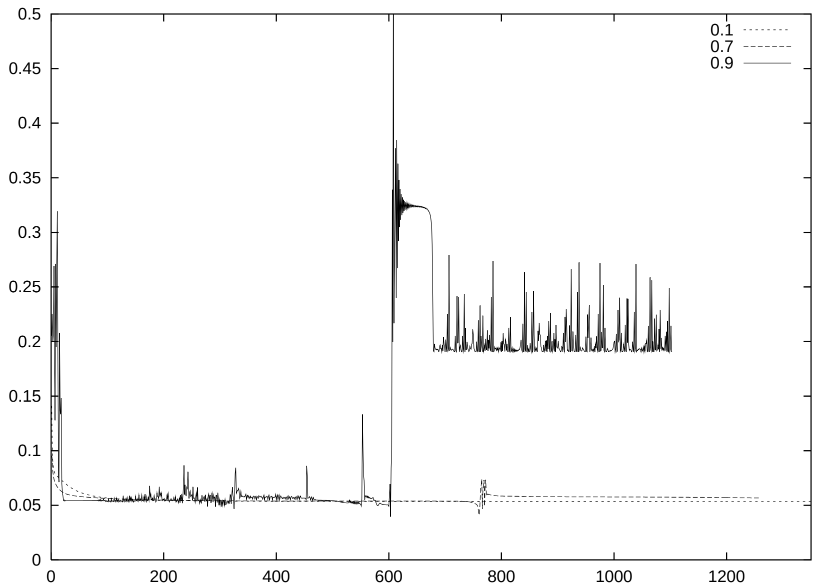
hvor inertikonstanten μ som bestemmer graden af inerti, skal opfylde $0 \leq \mu \leq 1$. Inertieffekten kan sammenlignes med en tung kugle på en bakket overflade – ved små minima vil kuglen have inerti nok til at trille forbi, og på flade strækninger accelererer kuglen så konvergensen foregår hurtigere.

Effekten af den tilføjede inerti skulle med andre ord gerne være hurtigere konvergens mod et lavere minimum. Hvis inertikonstanten sættes for højt, risikerer man dog at „skyde forbi" målet eller havne i uendelige oscillationer omkring minimum. Sættes den omvendt for lavt, er der ikke nogen fordel i forhold til almindelig gradientnedstigning. På figur 4.12 på næste side vises træningsfejlen som funktion af antallet af iterationer ved varierende inertikonstanter.

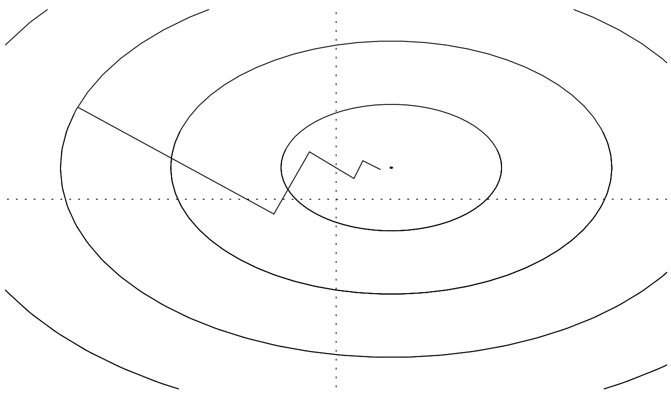
4.4.4 Konjugeret gradient

Nedstigning efter den retning som $\mathbf{g} = -\nabla E$ angiver, er ikke nødvendigvis direkte i retning af minimum. Figur 4.13 på den følgende side viser et eksempel på dette i $\mathbb{R}^2$.

Problemet er at gradienten for fejlfunktionen i et givet punkt er vinkelret med tangenten til niveaukurven i dette punkt, så hvis niveaukurven er elliptisk, bety-



Figur 4.12: Fejlen som funktion af antallet af iterationer med de angivne inertikonstanter. I nogle tilfælde „overskyder" træningen minimum og netværket havner i et andet minimum.



Figur 4.13: Niveaukurver for en fejlfunktion med indtegnede gradienter ved en gradient-nedstigning.

der det at den linje som gradienten ligger i ikke indeholder centrum af niveaukurverne. Altså er det ikke muligt direkte at gå til minimum. Hvis hvert skridt i gradientnedstigningen altid går til minimum på den linje som gradienten ligger i, kan det faktisk vises at to på hinanden følgende gradienter bliver ortogonale [Shewchuk, 1994] så algoritmen tvinges til at gå flere gange i samme retning og zigzagge hen til målet. Dette betyder at træningen konvergerer langsomt hvis netværket starter i et uheldigt punkt.

Det ville være smartere hvis man udvalgte et sæt af vektorer der udspændte vores søgerum og dernæst fandt længden af hver enkelt af disse vektorer. I $\mathbb{R}^n$ skal man finde et sæt af n lineært uafhængige vektorer – så ved man at de udspænder rummet.

Ved at definere vægtvektoren $\mathbf{w}_{i+1}$ for iterationspunktet $i+1$ som startpunktet $\mathbf{w}_0$ plus en linearkombination af n lineært uafhængige vektorer $\mathbf{d}_0, \ldots, \mathbf{d}_n$

$$\begin{aligned}
\mathbf{w}_{i+1} &= \mathbf{w}_0 + \sum_{j=0}^{i} \alpha_j \mathbf{d}_j \\
&= \mathbf{w}_0 + \alpha_0 \mathbf{d}_0 + \sum_{j=1}^{i} \alpha_j \mathbf{d}_j \\
&= \mathbf{w}_1 + \sum_{j=1}^{i} \alpha_j \mathbf{d}_j \\
&\vdots \\
&= \mathbf{w}_i + \alpha_i \mathbf{d}_i,
\end{aligned} \tag{4.12}$$

ses hvordan en algoritme kan udledes. Vægtvektoren $\mathbf{w}_i$ opdateres iterativt indtil man opnår et minimum ved $\mathbf{w}_i = \mathbf{w}^*$. For hvert skridt skal α_i og $\mathbf{d}_i$ bestemmes, og produktet mellem de to er så vægtopdateringen $\Delta\mathbf{w}$.

Dette vil altså sige at man ved at reducere fejlen en vektor af gangen kan komme til et minimum i n iterationer. Spørgsmålet er hvilke vektorer man skal vælge, og hvordan de skal skaleres.

Omskrivning til matrixligning

Første skridt er at forsimple problemet gennem en approksimation til en kvadratisk fejlfunktion fra $\mathbf{w}'$ [Bishop, 1995]

$$E(\mathbf{w}) \approx E(\mathbf{w}') + \mathbf{w}^T\mathbf{b} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w}. \tag{4.13}$$

hvor $E(\mathbf{w}')$ findes ved at føre træningseksemplerne gennem netværket med vægtene $\mathbf{w}'$ og $\mathbf{b} = \nabla E(\mathbf{w}')$ kan findes med tilbageførselsalgoritmen. $\mathbf{H}$ er en kvadratisk matrix af andenordensafledte, den såkaldte Hessian-matrix:

$$\mathbf{H} = \begin{bmatrix}
\frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\
\frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 E}{\partial w_n \partial w_1} & \frac{\partial^2 E}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_n^2}
\end{bmatrix}$$

Her betegner w_j komposant j i vægtvektoren $\mathbf{w}'$. Bemærk at $\mathbf{b}$ og $\mathbf{H}$ i praksis evalueres i hver iteration for at approksimationen vil være rimelig, men under udledningerne af konjugeret gradient betragtes disse to som værende konstante.

Afledes (4.13) ud fra denne betragtning, fås [Bishop, 1995]

$$\nabla E(\mathbf{w}) \approx \mathbf{b} + \mathbf{H}\mathbf{w} \tag{4.14}$$

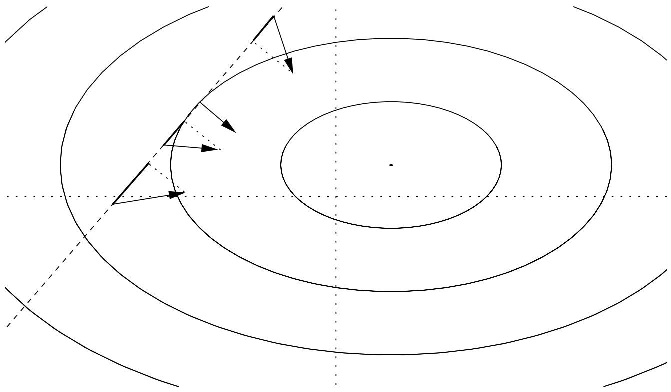
Hvis man nu sætter

$$\mathbf{b} + \mathbf{H}\mathbf{w} = 0 \quad \Leftrightarrow \quad \mathbf{H}\mathbf{w} = -\mathbf{b} \tag{4.15}$$

og løser ligningen mht. $\mathbf{w}$ så vil den opnåede $\mathbf{w}^*$ være minimum for den approksimerede fejlfunktionen, givet at $\mathbf{H}$ opretholder specielle egenskaber [Bishop, 1995]. Dette minimum vil altså være den bedste konfiguration af vægtene i netværket givet (4.13) som fejlfunktion og altså ikke nødvendigvis den oprindelige E den approksimerer.

Konjugerede retninger

Da $\mathbf{H}$ er en $n \times n$ matrix hvor n er antallet af vægte, vil løsning vha. rækkereduktionsalgoritmen normalt være for tidskrævende da denne kræver $O(\frac{2}{3}n^3)$ operationer [Lay, 2000]. I stedet forsøger vi at løse ligning (4.15) iterativt som før foreslået.



Figur 4.14: Den forrige retning ligger i en plan der skærer fejlfunktionen i en parabel (markeret med en stiplet linje), og hældningen af parablen i et punkt er den komposant af gradienten der ligger i tværsnittets plan – dvs. projektionen af gradienten på planen. Det betyder at minimum for parablen kan findes hvor gradienten står vinkelret på tværsnitsplanen da længden af projektionen i det punkt er nul.

Ved gradientnedstigning kunne man for hvert skridt gå til minimum i den retning som gradienten angiver, hvilket svarer til at gradienten (med modsat fortegn) $\mathbf{g}_i$ ved iteration i skal være ortogonal med den følgende gradient så $\mathbf{g}_i^T \mathbf{g}_{i+1} = 0$, se figur 4.14. Hvis niveaukurverne er ikke cirkulære, som de ikke er i det generelle tilfælde, zigzagger algoritmen dog hvilket vi som nævnt ønsker at undgå.

Hvis søgeretningerne i stedet er $\mathbf{H}$-*ortogonale* med hinanden, eller *konjugerede*, kan man være sikker på at disse udspænder rummet da sådanne søgeretninger er lineært uafhængige hvis $\mathbf{H}$ opretholder de egenskaber som skulle til for at der fandtes ét minimum (et bevis for dette fremgår i bilag A.1 på side 91). Hvad vigtigere er, er at det kan vises at man kan nå til minimum i n skridt hvor n er antallet af vægte [Shewchuk, 1994]. For to vektorer $\mathbf{d}_{i+1}$ og $\mathbf{d}_i$ er konjugerethed defineret som

$$\mathbf{d}_i^T \mathbf{H} \mathbf{d}_{i+1} = 0,$$

for $\mathbf{d}_i \neq \mathbf{d}_{i+1}$ og kan fortolkes som at $\mathbf{d}_{i+1}$ er ortogonal med $\mathbf{d}_i$ efter transformation med $\mathbf{H}$. En algoritme kan så bygges op om at vælge en startvektor $\mathbf{d}_0$ (den sættes som regel til $-\nabla E(\mathbf{w}_0)$), hvorefter de næste vektorer $\mathbf{d}_1, \ldots, \mathbf{d}_n$ skal vælges så de alle er konjugerede med hinanden.

Bestemmelse af α_i

Hvis $\mathbf{w}^*$ er en stedvektor til minimum af fejlfunktionen, dvs. en vektor fra origo til minimum, kan man definere en fejlvektor fra minimum til et punkt $\mathbf{w}_i$ som $\mathbf{e}_i = \mathbf{w}_i - \mathbf{w}^*$. Dette kan tolkes som fejlen man ønsker at få reduceret til $\mathbf{0}$ således at $\mathbf{w}_i = \mathbf{w}^*$. Vektoren indeholder en linearkombination af $n - i$ konjugerede vektorer [Bishop, 1995]:

$$\begin{aligned} \mathbf{e}_i &= \mathbf{w}_i - \mathbf{w}^* \\ &= \sum_{j=i}^{n} \alpha_j \mathbf{d}_j. \end{aligned} \tag{4.16}$$

Denne linearkombination består af alle de retninger algoritmen mangler at gå i – f.eks. er $\mathbf{e}_0 = \sum_{j=0}^{n} \alpha_j \mathbf{d}_j$. For at metoden skal virke, skulle $\mathbf{d}_i$ være konjugeret med alle andre vektorer i sættet. Hvis man opretholder at $\mathbf{d}_i$ er konjugeret med $\mathbf{d}_{i+1}, \ldots, \mathbf{d}_n$, så vil dette betyde at alle vektorerne er indbyrdes konjugerede. Da $\mathbf{e}_{i+1}$ indeholder alle disse vektorer vil det sige at der skal opretholdes

$$\mathbf{d}_i^T \mathbf{H} \mathbf{e}_{i+1} = 0.$$

Da

$$\begin{aligned} \mathbf{e}_{i+1} &= \mathbf{w}_{i+1} - \mathbf{w}^* \\ &= \mathbf{w}_i + \alpha_i \mathbf{d}_i - \mathbf{w}^* \\ &= \mathbf{e}_i + \alpha_i \mathbf{d}_i \end{aligned}$$

kan dette skrives om til

$$\mathbf{d}_i^T \mathbf{H} (\mathbf{e}_i + \alpha_i \mathbf{d}_i) = 0$$

og dermed bliver

$$\alpha_i = -\frac{\mathbf{d}_i^T \mathbf{H} \mathbf{e}_i}{\mathbf{d}_i^T \mathbf{H} \mathbf{d}_i}.$$

Ved at multiplicere (4.16) med $\mathbf{d}_i^T \mathbf{H}$ og benytte (4.15) får man

$$\begin{aligned} \mathbf{d}_i^T \mathbf{H} \mathbf{e}_i &= \mathbf{d}_i^T \mathbf{H} (\mathbf{w}_i - \mathbf{w}^*) \\ &= \mathbf{d}_i^T (\mathbf{H} \mathbf{w}_i + \mathbf{b}) \\ &= -\mathbf{d}_i^T \mathbf{g}_i \end{aligned}$$

da $\mathbf{g}_i = -\nabla E(\mathbf{w}_i) \approx -(\mathbf{H}\mathbf{w}_i + \mathbf{b})$. Dvs. at

$$\alpha_i = \frac{\mathbf{d}_i^T \mathbf{g}_i}{\mathbf{d}_i^T \mathbf{H} \mathbf{d}_i} \tag{4.17}$$

Bestemmelse af $\mathbf{d}_i$

Det vides at retningen $\mathbf{d}_{i+1}$ skal være konjugeret med alle de forrige retninger $\mathbf{d}_i, \ldots, \mathbf{d}_0$. Dette kan opretholdes ved at sætte [Bishop, 1995]

$$\mathbf{d}_{i+1} = \mathbf{g}_{i+1} + \beta_i \mathbf{d}_i \tag{4.18}$$

hvor β_i kan findes ved multiplikation med $\mathbf{H}\mathbf{d}_i$:

$$\begin{aligned} \mathbf{d}_{i+1}^T \mathbf{H} \mathbf{d}_i &= (\mathbf{g}_{i+1} + \beta_i \mathbf{d}_i)^T \mathbf{H} \mathbf{d}_i \\ 0 &= \mathbf{g}_{i+1} \mathbf{H} \mathbf{d}_i + \beta_i \mathbf{d}_i \mathbf{H} \mathbf{d}_i \\ \beta_i &= -\frac{\mathbf{g}_{i+1}^T \mathbf{H} \mathbf{d}_i}{\mathbf{d}_i^T \mathbf{H} \mathbf{d}_i}. \end{aligned} \tag{4.19}$$

Det ønskes at undgå at beregne $\mathbf{H}$, og ved hjælp af (4.14) og (4.12) kan dette opnåes

$$\begin{aligned} \mathbf{g}_i - \mathbf{g}_{i+1} &= -(\mathbf{b} + \mathbf{H}\mathbf{w}_i) + (\mathbf{b} + \mathbf{H}\mathbf{w}_{i+1}) \\ &= \mathbf{H}(\mathbf{w}_{i+1} - \mathbf{w}_i) \\ &= \alpha_i \mathbf{H} \mathbf{d}_i \end{aligned} \tag{4.20}$$

Dette kan nu substitueres ind i (4.19) hvorved α_i går ud, og man får

$$\beta_i = -\frac{\mathbf{g}_i^T (\mathbf{g}_i - \mathbf{g}_{i+1})}{\mathbf{d}_i^T (\mathbf{g}_i - \mathbf{g}_{i+1})}.$$

Det vises af Bishop [1995] at dette yderligere kan skrives om til

$$\beta_i = \frac{\mathbf{g}_{i+1}^T \mathbf{g}_{i+1}}{\mathbf{g}_i^T \mathbf{g}_i} \tag{4.21}$$

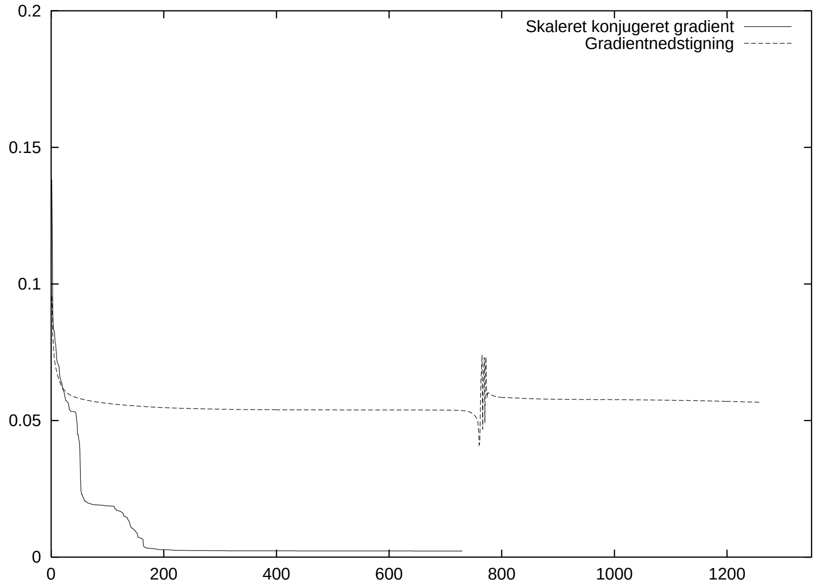
som er den form der benyttes i praksis i dette projekt.

Algoritmen i praksis

Det foregående afsnit viste at $\mathbf{d}_i$ kan findes uden brug af $\mathbf{H}$ hvilket er helt essentielt da $\mathbf{H}$ har $n^2/2$ unikke indgange som skal beregnes, hvor n er antallet af vægte – med f.eks. 50.000 vægte bliver det 1.250.000.000 indgange. Det er imidlertid ikke umiddelbart muligt at omskrive udtrykket med α_i så $\mathbf{H}$ undgås, hvorfor det er nødvendigt at bruge andre teknikker til at bestemme α_i.

En af disse er *skaleret konjugeret gradient* som er beskrevet i appendiks A.2 på side 92. Grundtanken i denne er at approksimere matrixproduktet $\mathbf{H}\mathbf{d}_i$ direkte, men dette medfører imidlertid visse vanskeligheder der gør algoritmen noget kompliceret. Af samme grund fremgår oversigten over hvordan den endelige algoritme kommer til at se ud også af appendikset i stedet for her.

Generelt for alle implementeringer af konjugeret gradient, om de bruger **H** eller skaleret konjugeret gradient eller noget andet, er dog det problem at den kvadratiske fejlfunktion kun er en approksimation til den virkelige fejlfunktion og denne approksimation bliver man nødt til at evaluere i hvert iterationspunkt for at den er nøjagtigt. Ovenstående udledninger baserer sig på at **b** og **H** var konstante, og af denne grund kan man ikke være sikker på at den indbyrdes konjugerethed opretholdes og man er således ikke sikker på at netværket har konvergeret efter n iterationer. Man bliver altså nødt til lade algoritmen køre flere end n iterationer, og dette gøres blot ved at genstarte algoritmen efter hver n iterationer. Algoritmen har dog vist sig at opretholde sin effektivitet på trods af disse betingelser.



Figur 4.15: Træningsfejlen som funktion af antallet af iterationer ved brug af skaleret konjugeret gradient sammenlignet med gradientnedstigning med inerti med de bedste værdier af konstanterne fra figur 4.12. Funktionen som aproksimeres beskrives i 4.4.5 på den følgende side

På figur 4.15 ses en iterationsmæssig sammenligning af gradientnedstigning og skaleret konjugeret gradient. Skaleret konjugeret gradient konvergerer her meget hurtigere end gradientnedstigning, men er samtidig mere beregningsintensiv pr. iteration. Dette skyldes at der skal beregnes to fejlværdier samt to gradienter pr. iteration.

Gradienterne beregnes i øvrigt over hele træningssættet som ved den ikke-stokastiske gradientnedstigning da det hverken giver mening at træne netværket helt op på de enkelte træningseksempler hver for sig, eller træne med skiftende eksempler for hvert skridt idet grundlaget for algoritmen så bryder sammen med de skiftende gradienter.

4.4.5 Generalisering, overtræning og termineringskriteriet

En af de stærke sider ved at bruge neurale netværk som et værktøj, er at netværket kan trænes op til at generalisere ud over de præsenterede træningseksempler. Der er imidlertid to fænomener der kan spolere generaliseringsevnen – man kan opnå en intuitiv fornemmelse for hvad disse gør ved netværk ved at betragte regressionsproblemer hvor netværket skal approksimere kontinuerte funktioner. Som eksempel benyttes funktionen $h(x) = 0{,}9 \cdot \sin(x) + \epsilon$ fra figur 4.10 hvor ϵ repræsenterer normalfordelt støj med $\mu = 0$ og $\sigma = 0{,}07$ og et netværk med én inputneuron, én outputneuron og et varierende antal neuroner i det skjulte lag, alle med tanh som aktiveringsfunktion, optrænet med gradientnedstigning. Konstanten 0,9 er valgt fordi den vægtede sum til aktiveringsfunktionen i outputneuronen skal gå mod uendelig for at resultatet bliver 1 så netværket får svært ved at klare approksimationen.

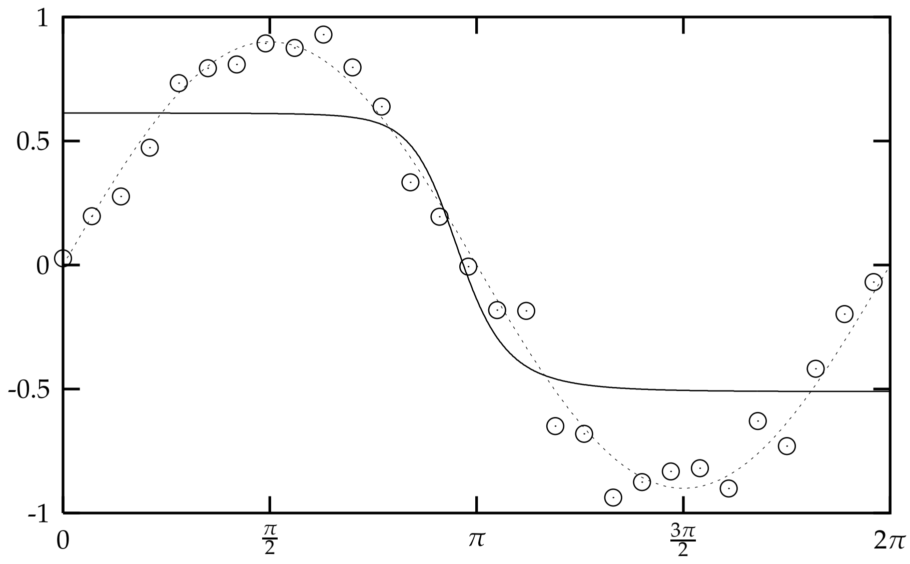
Når et neuralt netværk skal optrænes til at kunne generalisere ud fra de 30 datapunkter der genereres med h, er det meningen at netværket skal kunne ignorere den tillagte støj og i stedet approksimere $\hat{h}(x) = 0{,}9 \cdot \sin(x)$. Der kan nu gå to ting galt: enten er der for få neuronerne og vægte så netværket ikke har tilstrækkeligt med *frihedsgrader* til overhovedet at være i stand til at repræsentere $\hat{h}$, eller også er der for mange frihedsgrader så netværket approksimerer støjen i dataene i stedet for $\hat{h}$.

Det sidste yderpunkt, benævnt overtræning, er karakteriseret ved at netværket kan give funktionsværdier der er helt i overenstemmelse med træningseksemplerne hvor de er placeret, men til gengæld ikke er i stand til at „udjævne" den approksimerede funktion mellem datapunkterne så den svarer til $\hat{h}$. Modsat repræsenterer det første yderpunkt den situation at netværket udjævner punkterne for meget så den approksimerede funktion er for jævn til at være i overenstemmelse med $\hat{h}$ [Bishop, 1995].

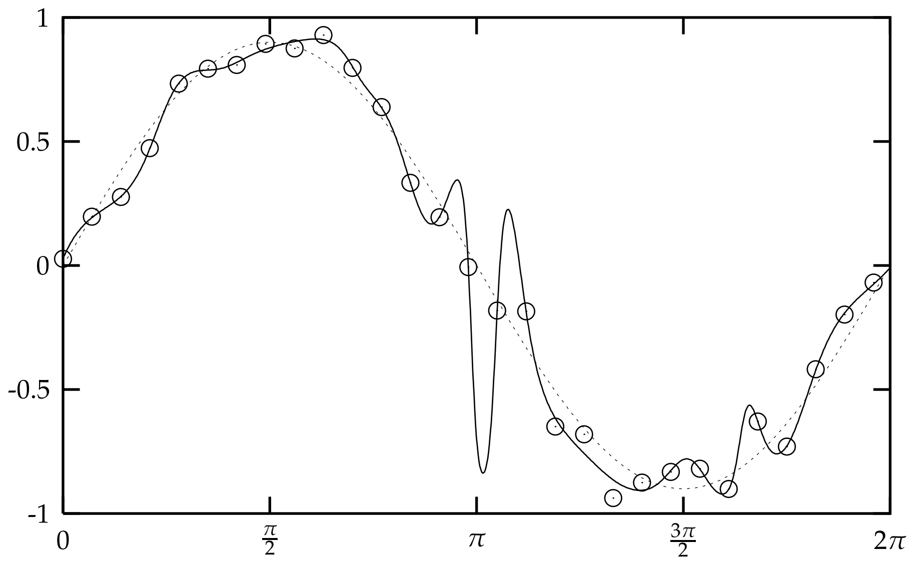
Et eksempel på for få frihedsgrader kan ses på figur 4.16 på den følgende side. Da netværket kun består af to vægte i hvert lag, en fra det forrige lag og en fra biasneuronen, kan netværket kun skalere inputtet og læggge en konstant til. Resultatet er ikke overraskende at netværkets resultat ligner aktiveringsfunktionen $\tanh(x)$, blot skaleret med et negativt tal således at den er blevet spejlet om x-aksen.

For at forbedre approksimationen er det nødvendigt at give netværket flere frihedsgrader i form af flere neuroner og dermed vægte i det skjulte lag. Figur 4.17 på næste side viser hvad der sker hvis der tilføjes for mange frihedsgrader, i dette tilfælde 10 skjulte neuroner. Det ses at netværkets resultat går meget præcist gennem næsten alle datapunkter, men at dette er en dårlig approksimation af den underliggende funktion $\hat{h}$. Netværket har tilstrækkeligt med frihedsgrader til at „huske" hvert datapunkt i stedet for at udtrække den ønskede information og bliver dermed meget følsomt over for støj.

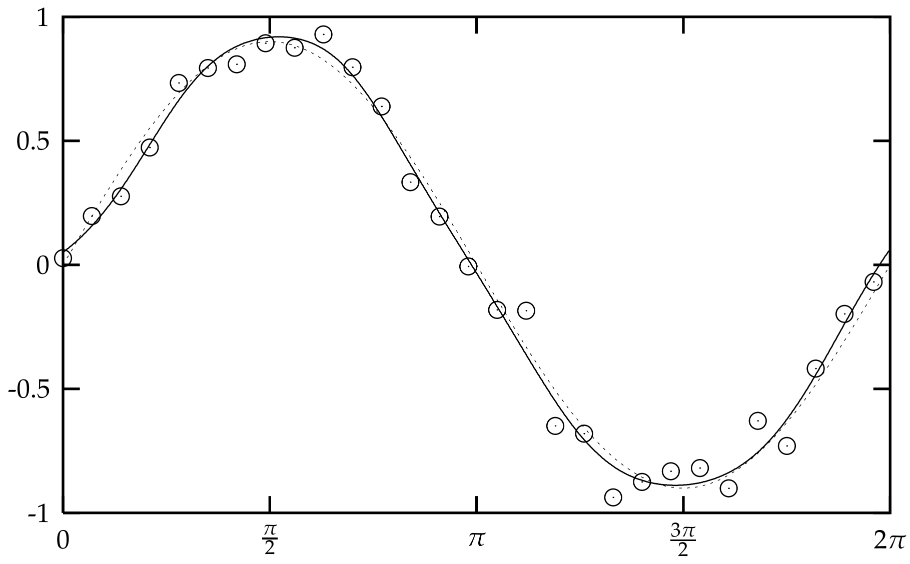
For at opnå den bedste approksimation skal netværket altså have for få frihedsgrader til at det kan gå gennem alle punkterne, men samtidig nok til at dets struktur er i stand til at repræsentere den underliggende funktion. At finde denne balance er netop det vanskelig ved neurale netværk da det generelt ikke kan gøres analytisk, men må afgøres eksperimentelt. Vi fandt frem til at to neuroner i det



Figur 4.16: Et netværk med én skjult neuron forsøger at approksimere $0{,}9 \cdot \sin(x)$. *De indtegnede punkter er træningseksemplerne, og* $0{,}9 \cdot \sin(x)$ *er desuden indtegnet.*



Figur 4.17: Et netværk med 10 skjulte neuroner forsøger at approksimere $\hat{h}$.



Figur 4.18: Et netværk med to skjulte neuroner er optimalt til at approksimere $\hat{h}$ i dette tilfælde. Bemærk hvor nøje netværket følger den unnderliggende funktion.

skjulte lag var nok til approksimere $\hat{h}$, se figur 4.18.

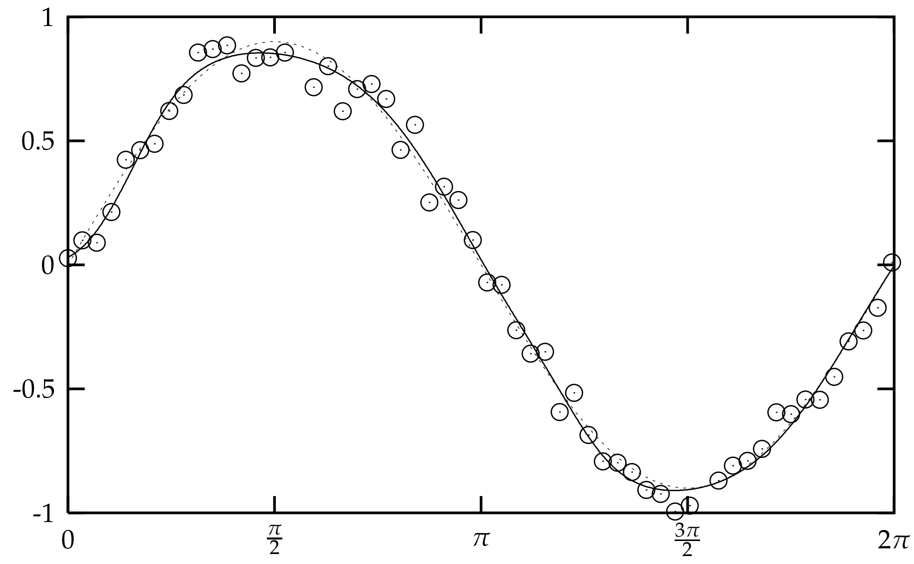
Dette betyder dog ikke at to skjulte neuroner er nok til at approksimere sinus i ethvert tilfælde. Hvis man f.eks. vil approksimere et større interval af sinus, eller hvis man har flere træningseksempler til rådighed, kan det optimale antal neuroner ændre sig. Faktisk kan man blot ved at forøge antallet af træningseksempler undgå problemet med overtræning. Dette kan ses på figur 4.19 på næste side.

Ved en forøgelse af antallet af træningseksempler forøger man antallet af punkter som netværket forsøger at ramme med approksimationen. Når datasættets størrelse er tilstrækkeligt stort, har netværket ikke længere nok frihedsgrader nok til at ramme alle punkter som på figur 4.17 på forrige side og tvinges i stedet til vælge en middelvej som ligger tættere på $\hat{h}$.

Desværre er størrelsen af datasættet som regel ikke let at forøge selvom det kan være muligt at gøre det større kunstigt ved at duplikere nogle af eksemplerne og tilføje støj. En anden måde at modvirke overtræning hænger sammen med en antagelse om at netværket først justerer vægtene så det approksimerer den underliggende funktion og først derefter begynder at tilpasse sig støjen, deraf i øvrigt navnet *overtræning*.

Termineringskriteriet og validering

Når man træner et neuralt netværk er den intuitive tilgang at man gerne vil opnå så lille en fejl som muligt, og et muligt termineringskriterium er derfor at fortsætte træningen til fejlen enten er under en bestemt tærskel eller til den ikke ændrer sig væsentligt mere. Hvis man tester netværket efter hver iteration med en friholdt del af dataene der ikke trænes på, viser det sig imidlertid at fejlen på denne del efter et tidsrum begynder at stige, se figur 4.20 på side 44. Det vil sige at netværket



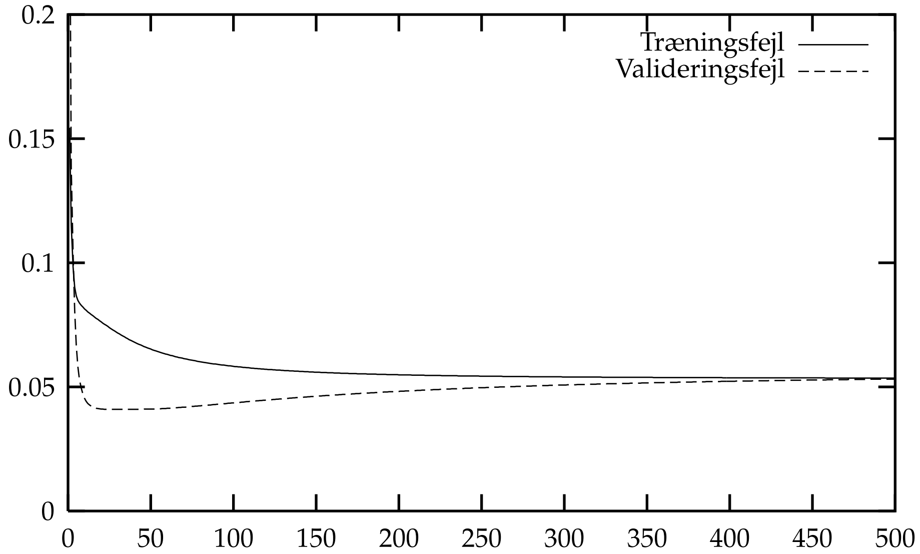
Figur 4.19: Et netværk med 10 neuroner i det skjulte lag bliver ikke længere overtrænet når antallet af træningseksempler forøges til 60.

bliver dårligere til at generalisere og i stedet begynder at tilpasse sig de specifikke træningseksempler.

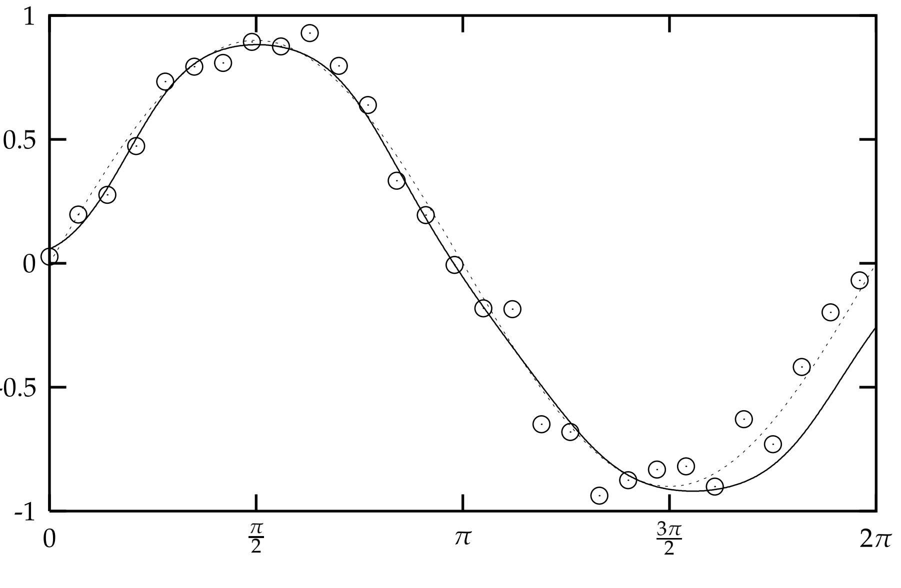
En løsning er at dele træningssættet op i to: et træningssæt som trænes på, og et valideringssæt som blot bruges til at holde øje med generaliseringsevnen. Når fejlen over valideringssættet har været stigende i et stykke tid, stoppes træningen så og man vælger den iteration hvor fejlen var lavest. Man skal dog være forsigtig med ikke at stoppe for hurtigt da valideringsfejlen sagtens kan stige midlertidigt over nogle iterationer.

Afslutning af træningen ud fra et valideringssæt betyder heller ikke at problemet med overtræning er helt undgået da man ikke direkte kan styre træningen af et netværk; man kan kun lade det fortsætte eller stoppe. Antagelsen om at netværket først approksimerer den underliggende funktion i dataene og først bagefter tilpasser sig støjen, holder naturligvis ikke helt i virkeligheden, og hvis der er for få data, kan man f.eks. komme ud for at træningen må stoppe så tidligt at netværket nærmest bliver undertrænet. Figur 4.21 på den følgende side viser samme eksempel som figur 4.17 bortset fra at 10% af dataene er blevet brugt til validering i stedet for træning. Det ses at netværket nu bedre approksimerer $\hat{h}$, men samtidig ikke er blevet ordentligt trænet i den sidste del af definitionsmængden.

Desuden betyder opdelingen af dataene at der er færre data til træning da man absolut ikke må træne på valideringsdataene hvis man skal have en vurdering af generaliseringsfejlen der ikke er for optimistisk. Og der opstår et nyt problem med hvor stor en andel af datene der skal reserveres til validering, samt hvilken del.



Figur 4.20: Fejlen over et træningssæt på 27 eksempler og det friholdte valideringssæt på 3 eksempler som funktion af antallet af iterationer. Valideringsfejlen begynder på et tidspunkt at stige hvorimod træningsfejlen aldrig stiger.



Figur 4.21: Netværk med 10 skjulte neuroner, men hvor 3 af de 30 træningseksempler er brugt til validering. Den sidste del af defitionsmængden nåede ikke at blive optrænet tilstrækkeligt.

Estimering af generaliseringsevne og krydstest

Spørgsmålet er om den opnåede valideringsfejl kan genbruges som et estimat af generaliseringsfejlen. For at kunne vurdere brugbarheden af et netværk i praksis er det nødvendig med et estimat at hvor godt det er til at generalisere, dvs. hvor godt det fungerer ved ukendte data. Desværre kan valideringssættet ikke bruges til denne bedømmelse da netværket er stoppet ud fra hvornår fejlen er lavest over dette så man opnår et for optimistisk estimat – i stedet må man helt friholde en del af dataene, testsættet, som ikke må indgå i træningen på nogen måde. Den resterende del, træningssættet, kan så videre inddeles i validerings- og træningssæt efter behov.

I stedet for at reservere en fast del af den samlede datamængde kan man dog udnytte dataene mest muligt til et mere præcist estimat ved at benytte sig af *krydstest* [Singh og Amin, 1999]: datasættet opdeles i k lige store dele, og et netværket trænes herefter på f.eks. de første $k-1$ dele, hvorefter det testes med den udeladte del. Et nyt netværk trænes derefter op med $k-1$ dele hvor en anden del udelades, testes med denne del, osv., og til sidst kan det endelige estimat fastsættes som gennemsnittet af testresultaterne.

4.5 Maksimering af rigtige klassifikationer

Man kan opbygge et simpelt mønstergenkendelsessystem ud fra et simpelt statistisk synspunkt, og principperne fra et sådant system vil senere vise sig at kunne bruges til at udlede summen af kvadrater [Bishop, 1995].

Man kan anskue problemet med at klassificere, og dermed genkende, bogstaver som et spørgsmål om at maksimere sandsynligheden for at klassifikationen er rigtig. Hvis man bliver stillet til opgave at klassificere bogstaverne 'a' og 'b', og begge optræder med 50% sandsynlighed i opgaven, kan man opnå en genkendelsesprocent på 0,5 blot ved altid at vælge det samme bogstav.

Hvis til gengæld 'a' optræder med en frekvens på 0,75 og 'b' dermed med en frekvens på $1 - 0{,}75 = 0{,}25$, vil det være mere fordelagtigt hvis man vælger at klassificere et eksempel som 'a' da man derved opnår en genkendelsesfrekvens på 0,75. Et klassifikationssystem der altid gætter på det samme bogstav, er naturligvis ikke meget værd, men når man tager yderligere oplysninger om forskellige visuelle egenskaber ved eksemplerne med i beregningen, kan ideerne fra en sådan sandsynlighedsanskuelse genbruges.

Beregningerne kan foretages ved hjælp af *Bayes' sætning*. Begynd med at definere:

- $P(C_k)$ som sandsynligheden for et eksempel tilhører klassen C_k og dermed sandsynligheden for at bogstavet er bogstav k.
- $P(X_l)$ som sandsynligheden for at observere egenskab X_l for hele datasættet.
- $P(C_k, X_l)$ som sandsynligheden for at et træningseksempel har egenskab X_l og tilhører klassen C_k.

- $P(C_k|X_l)$ som sandsynligheden for at træningseksemplet tilhører klasse C_k givet at vi har observeret egenskaben X_l.

Bayes' sætning er da

$$P(C_k|X_l) = \frac{P(C_k, X_l)}{P(X_l)} = \frac{P(X_l|C_k)P(C_k)}{P(X_l)}, \tag{4.22}$$

hvor

$$P(X_l) = \sum_k P(X_l|C_k)P(C_k),$$

og kan bruges til at finde sandsynligheden for at bogstavet tilhører en bestemt klasse når egenskaben X_l er observeret, idet sandsynlighederne på højre side af lighedstegnet i (4.22) alle kan bestemmes ud fra observationer der er foretaget på forhånd. En egenskab kunne f.eks. være om et bestemt punkt var sort. For at vise hvordan det kan bruges, udbygger vi eksemplet fra før.

Bogstavet kan som før enten være et 'a' eller et 'b' hvor frekvensen for 'a' er 0,25. For træningseksempler på 'a' er frekvensen 0,80 for at observere egenskaben X_1 og 0,20 for egenskaben X_2. Tilsvarende er på træningseksempler af 'b' frekvensen 0,40 for at observere egenskaben X_1 og 0,60 for at observere egenskaben X_2.

Hvis man for et testeksempel observerer egenskaben X_1, kan man beregne sandsynlighederne for at eksemplet tilhører én af klasserne:

$$\begin{aligned} P(C_a|X_1) &= \frac{P(X_1|C_a)P(C_a)}{P(X_1)} = \frac{0{,}80 \cdot 0{,}25}{0{,}50} = 0{,}4 \\ P(C_b|X_1) &= \frac{P(X_1|C_b)P(C_b)}{P(X_1)} = \frac{0{,}40 \cdot 0{,}75}{0{,}50} = 0{,}6. \end{aligned}$$

For at opnå flest korrekte klassifikationer skal man altså klassificere bogstaver hvor egenskaben X_1 er observeret som 'b' – dermed opnås en genkendelsesfrekvens på 0,60.

Hvis de egenskaber man ekstraherer fra inputtet ikke er diskrete, bliver man nødt til at benytte sandsynlighedsfordelingsfunktioner hvor sandsynligheden for at x ligger i intervallet $[a;b]$ er givet ved

$$P(x \in [a;b]) = \int_a^b p(x)\,dx,$$

hvor $p(x)$ er sandsynlighedsfordelingsfunktionen. Det kan vises [Bishop, 1995] at Bayes' sætning med sandsynlighedsfordelingsfunktioner bliver

$$P(C_k|x) = \frac{p(x|C_k)P(C_k)}{p(x)},$$

hvor

$$p(x) = \sum_k p(x|C_k)P(C_k).$$

Dette kan bruges til at vise at man kan udlede fejlfunktionen summen af kvadrater ved at forsøge at maksimere sandsynligheden for at hvert eksempel $\mathbf{x}_n$ i træningssættet resulterer i den ønskede værdi $\mathbf{t}_n$ for samme eksempel, svarende til at maksimere de fælles sandsynlighedsfordelinger $p(\mathbf{x}_n, \mathbf{t}_n)$ for hver værdi af n. Hvis man antager at træningseksemplerne er uafhængige, kan den fælles sandsynlighedsfordeling skrives som

$$p(\mathbf{x}_1, \mathbf{t}_1, \ldots, \mathbf{x}_n, \mathbf{t}_n) = \prod_n p(\mathbf{x}_n, \mathbf{t}_n) = \prod_n p(\mathbf{t}_n|\mathbf{x}_n)p(\mathbf{x}_n), \tag{4.23}$$

hvor multiplikationen løber over hele datasættet.

Den naturlige logaritme er en monotont voksende funktion, hvorfor maksimering af (4.23) vil være ækvivalent med at minimere det negative af logaritmen til (4.23). Dette giver fejlfunktionen E:

$$\begin{aligned} E &= -\ln\left(\prod_n p(\mathbf{t}_n|\mathbf{x}_n)p(\mathbf{x}_n)\right) \\ &= -\sum_n \ln\left(p(\mathbf{t}_n|\mathbf{x}_n)p(\mathbf{x}_n)\right) \\ &= -\sum_n \ln p(\mathbf{t}_n|\mathbf{x}_n) - \sum_n \ln p(\mathbf{x}_n). \end{aligned}$$

Hvis man yderligere antager at sandsynlighederne komposantvist er uafhængige, kan udtrykket skrives om til

$$E = -\sum_n \sum_k \ln p(t_{kn}|\mathbf{x}_n) - \sum_n \ln p(\mathbf{x}_n), \tag{4.24}$$

hvor k løber over komposanterne i $\mathbf{t}_n$.

Ved mange problemer vil der være forbundet en vis støj eller usikkerhed til målingerne af de værdier som netværket forsøger at approksimere – det vil sige at man kan skrive t op som genereret af en underliggende funktion h minus støjen:

$$t_{kn} = h_{kn}(\mathbf{x}) - \epsilon_{kn}. \tag{4.25}$$

Hvis denne støj antages at være normaltfordelt, kan dens fordeling udtrykkes ved hjælp af det generelle udtryk for sandsynlighedsfordelingsfunktion for normalfordelingen

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{4.26}$$

og bliver

$$p(\epsilon_{kn}) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{\epsilon_{kn}^2}{2\sigma^2}}$$

idet middelværdien μ antages at være 0.

Det ses af (4.25) at t_{kn} for en given værdi af $\mathbf{x}$ ikke er helt fastlagt, men i stedet har en sandsynlighedsfordeling $p(t_{kn}|\mathbf{x})$ som svarer til sandsynlighedsfordelingen $p(\epsilon_{kn})$ for støjen. Desuden ses at forskellen mellem $h_{kn}(\mathbf{x})$ og t_{kn} giver støjen – det vil sige at

$$p(t_{kn}|\mathbf{x}) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(h_{kn}(\mathbf{x})-t_{kn})^2}{2\sigma^2}}.$$

Den underliggende funktion h_{kn} approksimeres af netværket – dette kan repræsenteres med funktionen $y_{kn}(\mathbf{x}, \mathbf{w})$ hvor $\mathbf{w}$ er vægtene i netværket:

$$p(t_{kn}|\mathbf{x}) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(y_{kn}(\mathbf{x},\mathbf{w})-t_{kn})^2}{2\sigma^2}}. \tag{4.27}$$

Vi har nu fundet $p(t_{kn}|\mathbf{x})$ fra (4.24), og substitution giver mulighed for at omskrive fejlfunktionen

$$\begin{aligned} E &= -\sum_n \sum_k \ln\left(\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(y_{kn}(\mathbf{x},\mathbf{w})-t_{kn})^2}{2\sigma^2}}\right) - \sum_n \ln p(\mathbf{x}_n) \\ &= \sum_n \sum_k \left((y_{kn}(\mathbf{x},\mathbf{w}) - t_{kn})^2 - 2\sigma^2\right) - \sum_n \sum_k \ln \frac{1}{\sigma\sqrt{2\pi}} - \sum_n \ln p(\mathbf{x}_n) \\ &= \sum_n \sum_k (y_{kn}(\mathbf{x},\mathbf{w}) - t_{kn})^2 - 2NK\sigma^2 + NK\ln(\sigma\sqrt{2\pi}) - \sum_n \ln p(\mathbf{x}_n), \end{aligned}$$

hvor N er antallet af træningseksempler og K er antallet af komposanter i $\mathbf{t}$.

Hvis man nu dropper de led som ikke bliver ændret når man ændrer på vægtene i netværket $\mathbf{w}$ og ganger med $\frac{1}{2}$, bliver udtrykket til den genkendelige sum af kvadrater:

$$E = \frac{1}{2} \sum_n \sum_k (y_{kn}(\mathbf{x},\mathbf{w}) - t_{kn})^2 \tag{4.28}$$

Hvis antagelserne om normalfordelt støj og uafhængige data er opfyldt, har minimering af summen af kvadrater altså en direkte fortolkning som maksimering af sandsynlighederne for korrekt klassifikation.

Kapitel 5

Implementering

For at kunne afprøve teorien fra foregående kapitel i praksis implementerede vi et komplet netværk og fremstillede træningsdata til test af netværket. Desuden blev der behov for en række hjælpeprogrammer – disse er beskrevet til sidst i kapitlet.

5.1 Implementering af netværket

5.1.1 Krav til implementeringen

Ydelsesmæssigt er det mest centrale krav til netværket at opnå en høj genkendelsesprocent idet genkendelsesprocenten er den primære skala som netværkets præstation måles efter. Dette betyder at netværket skal være tilstrækkeligt skalérbart så der kan tilføjes ekstra lag og ekstra neuroner hvis det viser sig nødvendigt.

For at finde det bedste netværk skal der foretages en række test hvor der stilles på de forskellige variable dele. For at kunne gennemføre alle testene inden for en rimelig tid skal netværket være fleksibelt nok til at alle disse dele kan tilgås på en nem måde så der ikke bliver behov for et omdesign af strukturen med de efterfølgende nødvendige fejlsøgninger og lignende. Antallet af test betyder også at træningstiden kan blive et problem hvorfor det er væsentligt at nedbringe denne mest muligt.

En opsummering af kravene er altså:

- En datastruktur der muliggør store netværk.
- Modulær opbygning med tilstrækkelig fleksibilitet til at afprøve mange konfigurationer.
- Hurtig under træning.

5.1.2 Valg af programmeringssprog

Ud fra de opstillede krav overvejede vi nogle kategorier af programmeringssprog for at udvælge det bedste til opgaven: SML, Java, C++, en gruppe man kunne

Krav	SML	Java	C++	Scriptsprog	Assembler
Objektorienteret	-	+	+	+	-
Imperativt	-	+	+	+	+
Hastighed	-	-	+	-	+
Kendskab	alle	ingen	få	få	få

Tabel 5.1: Sammenligning af programmeringssprog.

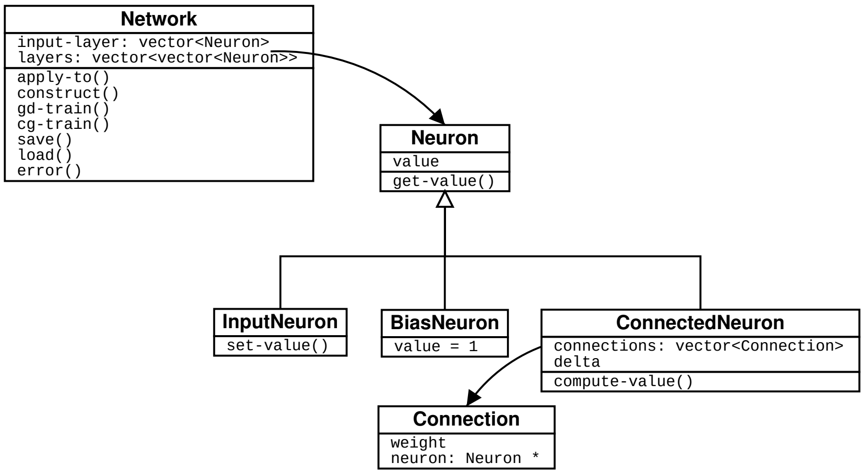
kalde scriptsprog som dækker over Perl, Python og lignende og endelig assemblerkode. En opsummering af overvejelserne fremgår af tabel 5.1

Da netværket i forbindelse med beregning af de partielle afledede af fejlfunktionen skal opbevare de beregnede δ-værdier for at opnå tidskompleksiteten $O(2n)$, er det ønskværdigt at det benyttede sprog er imperativt så det er nemt at tilgå datastrukturer. Af denne grund blev SML afvist eftersom den eneste anden fordel var at alle i gruppen havde kendskab til sproget. Scriptsprogene kunne evt. have den fordel at de var lettere at sætte sig ind i, men til gengæld lider de under et problem med hastigheden. For assembler er det lige modsat.

Tilbage er Java og C++ – af hensyn til hastigheden og kendskabet i gruppen valgte vi at implementere i C++.

5.1.3 Design af netværket

For at muliggøre mere komplekse netværksarkitekturer end blot fuldt forbundne er netværket blevet designet således at forbindelserne ligger gemt i den datastruktur der indeholder netværket. På denne måde er det muligt at ændre på designet blot ved at ændre den funktion, CONSTRUCT, der konstruerer netværket.



Figur 5.1: De vigtigste af de forskellige klasser og forbindelserne mellem dem – for hver klasse er angivet medlemsvariable og -funktioner, den hvide pil angiver nedarvning.

Da C++ understøtter objektorienteret programmering, har vi valgt at modellere netværket, neuronerne og forbindelserne med klasser. En skematisk fremstilling af forholdet mellem alle klasserne kan ses på figur 5.1 på forrige side. Selve kildekoden ligger på den medfølgende cd. Bemærk at navnene af pladshensyn i de følgende afsnit er forkortet lidt i forhold til deres faktiske navne i kildekoden.

Netværksklassen

Netværksklassen indeholder dels en række funktioner i sin grænseflade ud ad til (disse er erklæret „public"), se tabel 5.2, dels alle neuronerne i den private, indkapslede del. Brug af netværket foregår så ved at oprette et objekt af typen NETWORK, indlæse gemte vægte vha. LOAD og endelig benytte APPLY-TO til at sende en vektor gennem netværket og returnere resultatet ved det sidste lag – fortolkningen og omsættelsen af denne vektor til et tegn varetages af andre funktioner.

Navn	Parametre	Formål
APPLY-TO	vektor af input	Returnerer output fra netværket givet input.
CONSTRUCT	vektor med antal neuroner i lagene	Konstruerer netværket med de givne størrelser af lagene.
GD-TRAIN	styringsobjekt, træningssæt, η, μ	Træning med gradientnedstigning.
CG-TRAIN	styringsobjekt, træningssæt	Træning med konjugeret gradient-algoritmen.
SAVE	filnavn	Gemning af netværket.
LOAD	filnavn	Indlæsning og konstruktion fra gemt netværk.
ERROR	Testsæt	Returnerer E over testsættet.

Tabel 5.2: De forskellige funktioner i grænsefladen for netværksklassen

Træning af netværket foregår ved enten at konstruere netværket fra bunden vha. CONSTRUCT eller indlæse et tidligere netværk, hvorefter træningen kan sættes i gang med GD-TRAIN for gradientnedstigning med inerti (som dog kan slås fra ved at sætte $\mu = 0$) eller CG-TRAIN for træning med skaleret konjugeret gradient.

For at muliggøre forskellige stopkriterier og logføring af diverse ting undervejs så det færdige træningsprogram kan fortælle hvor langt det er nået, tager træningsfunktionerne mod et styringsobjekt som de kalder én gang for hver iteration. Styringsobjektet fortæller så træningsfunktionen om den skal fortsætte eller stoppe. Da styringsklassen blot indeholder en abstrakt funktion til at foretage denne afgørelse, kan nedarvede klasser overskrive den abstrakte funktion og dermed implementere forskellige termineringskriterier og logføringer.

I praksis viste det sig kun nødvendigt at implementere én nedarvet klasse – den tager bl.a. mod et filnavn til logføring og det maksimale antal iterationer, og benytter så valideringsfejlen til at stoppe et givet antal iterationer (f.eks. 100) efter at det hidtidige globale valideringsminimum er nået. Valideringsfejlen kan beregnes ved at kalde ERROR på netværket med valideringssættet. For hver iteration bliver netværket gemt – medmindre valideringsfejlen er steget – så det er muligt at afbryde træningsprogrammet uden at miste det hidtidige arbejde.

Neuronhierarkiet

Der viste sig at være brug for tre forskellige typer neuroner: inputneuroner, neuroner med konstant værdier („biasneuroner") og endelig forbundne neuroner. For at muliggøre forbindelser mellem vilkårlige typer neuroner benyttes et klassehierarki med en generel neurontype øverst – denne understøtter ikke andet end opbevaring af værdien for neuronen samt en funktion til at få fat i denne værdi. Herfra nedarves en inputneuron som indeholder en funktion til at indsætte en værdi i neuronen, en konstant neuron som blot sætter sin værdi til 1 når den konstrueres, samt en neuronklasse med forbindelser.

Sidstnævnte klasse indeholder dels en vektor af forbindelser – en forbindelse består blot af en vægt og en reference til en anden neuron – som peger bagud i netværket (dvs. f.eks. fra andet lag til første lag og aldrig omvendt), og dels en funktion, COMPUTE-VALUE, til at beregne og sætte den forbundne neurons værdi ved at løbe alle forbindelserne igennem, beregne den vægtede sum af de tilkoblede neuroners værdier og anvende aktiveringsfunktionen på resultatet. De forbundne neuroner indeholder desuden hver en δ-værdi til brug for beregning af de partielle afledede med tilbageføringsalgoritmen.

Neuronobjekterne for det første lag, inputlaget, opbevares i en særskilt vektor, mens resten af lagene opbevares i en samlet vektor der indeholder en vektor med neuroner for hvert lag.

5.1.4 Beskrivelse af netværksalgoritmer

De mest interessante af de algoritmer som benyttes i netværket, bliver gennemgået med pseudokode i det følgende.

Fremadføring af værdier

Fremadføring af værdier foregår ved at løbe den særskilte vektor med inputlaget igennem og indsætte en værdi fra inputvektoren i hver neuron, hvorefter resten af lagene kan løbes igennem fra den ene ende af – for hver neuron kaldes COMPUTE-VALUE som sætter neurons værdi, hvorved værdierne gradvist bliver beregnet efterhånden som lagene gennemgås. Til sidst indeholder den sidste vektor af neuroner resultatet af netværkets samlede beregning som derved let kan indsamles.

I pseudokode ser algoritmen således ud:

```
løb gennem neuronerne i første lag:
    sæt værdien for hver neuron til værdien i inputvektoren
løb gennem lagene fra første skjulte lag:
    løb gennem neuronerne for hvert lag:
        kald COMPUTE-VALUE for hver neuron
```

Tilbageføring af δ-værdier

Tilbageføring af δ-værdier forløber under antagelse af at der først er foregået en fremadføring så alle neuronerne indeholder deres respektive aktiveringsværdier. δ-værdierne i det sidste lag kan beregnes direkte jvf. udtryk (4.8)

$$\delta_j = g'(a_j) \cdot (o_j - t_j),$$

mens summen i udtryk (4.9) for skjulte neuroner

$$\delta_j = g'(a_j) \cdot \sum_k \delta_k w_{kj}$$

ikke er mulig at beregne direkte da der i neuron j kun er adgang forbindelser til neuroner i det *foregående* lag og ikke til neuronerne som gennemløbes med indekset k da de befinder sig i *næste* lag.

Løsningen er at nulstille alle δ'er untagen det sidste lag og starte ved det næstsidste lag i netværket – for hver neuron i det sidste lag, k, kan hvert lille bidrag til summen i (4.9) for j beregnes. Ved at løbe hele laget igennem er hele summen beregnet og δ_j kan findes, idet aktiveringsfunktionerne er kendt. Proceduren gentages for alle resterende lag.

På pseudokodeform ser algoritmen således ud (y er aktiveringsværdien for neuronen, t_i er den i'te værdi i målvektoren, w er vægten i en forbindelse og $\hat{\delta}$ er δ-værdien for neuronen i den anden ende af forbindelsen):

```
løb gennem neuronerne i sidste lag:
    sæt δ for hver neuron til y − t_i
nulstil alle δ'er pånær i det sidste lag
løb gennem lagene bagfra pånær det første mellemlag:
    løb gennem neuronerne for hvert lag:
        sæt δ for hver neuron til g'(a) · δ
        løb gennem forbindelserne for hver neuron:
            sæt δ̂ for neuronen i hver forbindelse til δ̂ + δ · w
løb gennem neuronerne i det første mellemlag:
    sæt δ for hver neuron til g'(a) · δ
```

Træning med gradientnedstigning

Gradientnedstigning foretages ved at igangsætte en fremadføring af værdierne for et givet træningseksempel, efterfulgt af en tilbageførsel af δ'erne. Derefter kan alle de forbundne neuroner løbes igennem, og for hver forbindelse kan den partielle afledede for vægten bestemmes ved (4.6) som

$$\frac{\partial E_d}{\partial w_{ji}} = \delta_j x_i$$

så vægten kan opdateres som træningskonstanten η ganget med den partielle afledede plus en inertikonstant α gange den forrige vægtopdatering (som også gemmes i forbindelsen sammen med vægten), jvf. udtryk (4.11). η og α sendes med som parametre til træningsfunktionen idet den normaliserede værdi $\bar{\eta}$ som beskrevet i afsnit 4.4.3 beregnes og benyttes for hver neuron. Normaliseringen blev indført efter en række indledende test hvor nedstigningsalgoritmen opførte sig yderst ustabilt.

Ovenstående procedure gentages for alle træningseksemplerne, hvorefter styringsobjektet kaldes for at finde ud af om træningen skal stoppes eller fortsætte (og evt. også for at udskrive loginformationer). Hvis træningen fortsættes, gennemføres operationerne blot en gang til over hele træningssættet.

I pseudekode ser algoritmen ud på følgende måde (x betegner den tilknyttede neurons aktiveringsværdi i en forbindelse):

gentag
    løb gennem træningssættet:
        kald FREMADFØRING *på inputvektoren*
        kald TILBAGEFØRING *på målvektoren*
        løb gennem lagene:
            løb gennem neuronerne for hvert lag:
                løb gennem forbindelserne for hver neuron:
                    sæt Δw *til* $-\bar{\eta} \cdot \delta x + \alpha \Delta w$
                    sæt w *til* $w + \Delta w$
    kald styringsobjekt – afslut hvis påkrævet

Træning med skaleret konjugeret gradient

Træning med skaleret konjugeret gradient følger den matematiske formulering fra appendiks A.2. Ved en omsættelse til praksis viser det sig dog at nogle beregninger af gradienten af fejlfunktionen og selve fejlfunktionen kan springes over. Mere specifikt har man ved udregning af $-\nabla E(\mathbf{w}_i)$ i appendiks A.2 på side 92 allerede beregnet $E(\mathbf{w}_i)$ ved fremadføring. $E(\mathbf{w}_i + \alpha_i \mathbf{d}_i)$ benyttes to gange hvilket kun kræver en beregning. Til sidst kan $E(\mathbf{w}_i + \alpha_i \mathbf{d}_i)$ benyttes i det næste iteration af algoritmen. Dette betyder at tidskompleksiteten havner på $O(4n)$ i stedet for de $O(8n)$ som umiddelbart ser ud til at være tilfældet.

Beregningerne med gradienten involverer flere vektoroperationer, f.eks. $\alpha_i = \frac{\mathbf{d}_i^T \mathbf{g}_i}{\delta_i}$, så det er nødvendigt at udtrække den fra netværket. Dette kan gøres ved at løbe neuronerne igennem på samme måde som ved gradientnedstigningen, men blot indsamle de partielle afledede til en vektor i stedet for direkte at ændre på vægtene. Gradienterne ∇E_d beregnes så for hvert træningseksempel og lægges sammen til den færdige gradient $\nabla E = \sum \nabla E_d$ – gyldigheden af dette skyldes regnereglerne for differentiation som tillader differentiation af de enkelte led i en summering for sig.

Da alle vægtene ligger spredt rundt omkring i netværket og ikke kan udskiftes på en simpel måde på én gang, viste det sig lettest at foretage beregningen af f.eks. $\nabla E(\mathbf{w} + \sigma \mathbf{d})$ ved at indsætte vægtene i en kopi af netværket så vægtene i det rigtige netværk ikke blev forstyrret.

Implementeringen følger punktopstillingen i appendiks A.2 på side 92.

5.2 Indsamling af datasæt

Til testene skulle der produceres et datasæt. Vi besluttede os på forhånd til at indskrænke os til de små bogstaver i det danske alfabet, altså 29 forskellige bogstaver.

For at kunne få et realistisk estimat af netværkets genkendelse på en håndholdt computer, fremstillede vi selv datasættet på en håndholdt.

Til dette blev et tegneprogram på en håndholdt benyttet – ved at benytte et startbillede med et indtegnet net med i alt 40 felter på hver 60 x 48 punkter kan der indtegnes f.eks. 40 a'er hvorefter billedet gemmes og et nyt billede fremstilles med f.eks. 40 b'er. Med 6 gruppemedlemmer resulterede dette i et datasæt på $40 \cdot 29 \cdot 6 = 6960$ tegn. Desuden fremstillede et af gruppemedlemmer tre ekstra datasæt med hver 40 tegn af hvert bogstav. Valget af højde og bredde blev foretaget ud fra hvad der passede med en behagelig størrelse skrivefelt. Instrukserne til forsøgspersonerne gik ud på at skrive i almindeligt tempo uden overdreven akkuratesse som hvis det var en realistisk situation.

Billederne blev derefter overført til en pc, og de enkelte tegn udtrukket og omsæt til filer indeholdende -1 og 1.

5.3 Beskrivelse af hjælpeprogrammer

Det viste nødvendigt at fremstille et antal hjælpeprogrammer, bl.a. til forbehandlingen. De væsentligste af disse omtales i det følgende og kan findes på vedlagt cd.

5.3.1 Opdeling i validerings- og træningssæt

Programmet `splitsamples` blev fremstillet til at dele datasættet op i træningssæt og valideringssæt. Det tager mod en liste af filnavne og tilføjer en vis andel, f.eks. 10%, af hvert bogstav fra hver person til en liste over valideringssættet – resten af navnene blandes og udgør træningssættet. Hvilke bogstaver inden for en bestemt klasse der udvælges, sker tilfældigt, men altså på en sådan måde så det f.eks. altid er 10% af a'erne.

5.3.2 Centrering af bogstaverne

Programmet `center` har til formål at centrere bogstaverne. Det benytter ligningen beskrevet i afsnit 4.2.2 til at bestemme massemidtpunktet af bogstavet for derefter at centrere efter dette. Det benytter sig af et underprogram, `moveimage`, der bruges til at flytte billedet efter centrum er fundet. Programmet flytter billedet et antal punkter ved at bortkaste punkter der skæres fra, og tilføje blanke punkter til de to kanter hvor der bliver et tomrum.

5.3.3 Skalering af billederne

Skalering af billederne blev foretaget ved hjælp af programmet `scale`. Programmet tillader skaleringerne 1:2, 1:4, 1:6 og 1:12 og bruger den simple algoritme som er beskrevet i afsnit 4.2.1, altså middelværdier.

5.3.4 Sammenlægning af billeder

For bl.a. at kunne vurdere hvor meget bogstaverne fylder, skrev vi programmet `superimpose` som lægger de enkelte punkter i et antal billeder sammen og slutteligt dividerer med antallet af bogstaver. På den måde kan man få et indtryk af hvordan forskellige standardbogstaver ser ud, se figur B.1 på side 94.

5.3.5 Konvertering til billedformat

Til at lette visualiseringen af de forskellige tegn udformede vi et program `chartobmp` som omsætter en tegnfil til et .bmp-billede ved hjælp af grafikbiblioteket Allegro [1].

[1] http://www.talula.demon.co.uk/allegro/

Kapitel 6

Empirisk metode

Når det neurale netværk er blevet implementeret, er der behov for viden om hvordan det fungerer i samspil med data for at kunne tilpasse det så genkendelsesevnen bliver bedst mulig. Denne viden kan frembringes ved at eksperimentere med netværket, men disse eksperimenter skal ske gennem en metode der giver mulighed for at drage konklusioner på dem, samt at systematisere opstillingen af dem så de kan udføres effektivt for at gøre det praktisk muligt at gennemføre forsøgene.

Den *empiriske metode* giver en løsning på disse problemer idet den både indebærer måder at eftervise påstande på og en metode til at foretage eksperimenter. Først er der dog et spørgsmål om hvad viden er – uden en definition på dette er det ikke muligt at besvare andre interessante spørgsmål som f.eks. i hvor høj grad man kan stole på resultaterne af den empiriske metode.

6.1 Positivisme og empiri

En definition af hvad viden er, er ikke nødvendigvis helt ligetil. F.eks. kan man anlægge det synspunkt at det ikke er muligt at vide noget fordi man ikke kan stole helt på sine sanser. Den empiriske opfattelse af viden er lige modsat at det er noget som kun kan opnås gennem erfaringer og observationer [Kragh og Pedersen, 1991]. Hertil kommer verifikationsprincippet som en definition af hvilken slags viden det giver mening at beskæftige sig med. Ud fra dette princip giver et udsagn ikke mening medmindre det kan vises sandt kun ud fra sanseerfaringer. Derved afvises den metafysiske idé om en indre kraft der styrer omverdenen da den ikke er mulig at påvise med sanseerfaringer.

Denne filosofi om at viden kun kan opnås gennem sanseerfaringer kaldes også positivisme. Positivisme blev udformet af Auguste Comte (1798-1857) med udgangspunkt i hans lov om de tre erkendelsesstadier. Første stadie var det religiøse stadie hvor fænomener blev forklaret ved at det var guds vilje. Meget nyttigt til fænomener man ikke har en forklaring på, men ikke særligt god til at skabe ny viden. Andet stadie var det metafysiske stadie hvor fænomener blev forklaret ved at objektet havde en „indre kraft“. Tredje stadie er det positive eller reale stadie, under hvilket fænomenerne blev forklaret ud fra observationer og videnskabelige metoder.

Det tredje stadie er den sidste og endegyldige måde at samle viden på, og det danner grundlaget for hvordan viden bliver defineret i positivismen og empirismen. Ud fra dette kan videnskab defines som en systematisk frembringelse af viden og erkendelse [Dige, 2002], dvs. at videnskab bygger på at kunne tilegne sig viden og formidle den videre på en systematisk og gennemskuelig måde.

Man kan opdele udsagn efter om de er *a priori* eller *a posteriori*. At et udsagn er a priori betyder at det kan vises sandt eller falsk ud fra forudgående viden eller viden som vi har kendskab til uafhængigt af sanseerfaringen. Hvis udsagnet derimod kun kan vises sandt ved observationer siges det at være a posteriori. Desuden kan udsagn opdeles efter om de er *analytiske* eller *syntetiske*. Analytiske udsagn kan verificeres (vises sande eller falske) ved hjælp af en logisk, semantisk analyse, f.eks. er et udsagn som "solen ikke både skinner og ikke skinner"analytisk sandt idet det ikke er muligt at noget både kan have en egenskab og ikke have en egenskab på samme tid, uafhængigt af hvad dette noget er. Syntetiske udsagn er udsagn som ikke kan verificeres logisk ud fra deres semantik på samme måde – de kræver en stillingtagen til hvad de siger om virkeligheden.

	A priori	A posteriori
Analytisk	1	2
Syntetisk	3	4

Tabel 6.1: De mulige sammenhænge mellem forskellige typer udsagn

Ud fra disse to forskellige klassifikationer af udsagn kan man opstille fire forskellige klasser [Kragh og Pedersen, 1991], se tabel 6.1. Analytiske a priori udsagn er de udsagn som kan verificeres alene ud fra en logisk, semantisk analyse. Analytiske a posteriori udsagn kan på samme måde verificeres alene ved hjælp af logisk, semantisk analyse, men skulle samtidig afhænge af vores sanseerfaringer – dette giver ikke meget mening, og kategorien anses også normalt for at være tom.

De syntetiske a priori udsagn kræver en stillingtagen til hvad de siger om virkeligheden og sætter én i stand til at udtale sig om den, men kan alligevel verificeres uafhængigt af erfaringer. De syntetiske a posteriori udsagn sætter ligeledes én i stand til at opnå viden om virkeligheden, men kræver samtidig observationer af den for at kunne verificere om de er sande.

Spørgsmålet er om den tredje kategori er tom eller ej, og det er faktisk dette spørgsmål der adskiller den empiriske tankegang fra f.eks. tidligere tiders naturfilosofi. Ifølge det empiriske synspunkt er den tom fordi det ikke er muligt at udtale sig om virkeligheden uden at have foretaget observationer af den – f.eks. ville en religiøs verifikation som kunne foretages på forhånd ikke være tilstrækkeligt.

Tilbage er kun kategori 1 og 4. Kategori 1 anses også for at indeholde viden, selvom denne viden kan opnås alene ved hjælp af analyser af sproget – matematik hører f.eks. under denne klasse idet sandheden af matematiske sætninger kan afgøres ud fra de grundlæggende definitioner. Hvor det er muligt i f.eks. fysik, foretrækkes faktisk viden fra kategori 1 fordi denne viden er garanteret sand.

Den empiriske opfattelse af viden er altså at det består af dels analytisk a priori viden som kan opnås f.eks. gennem logik og matematik, men som ikke sætter én i stand til at udtale sig om verden, og dels syntetisk a posteriori viden der kun kan opnås gennem sanseerfaringer, om verden. Udsagn der ligger uden for disse to grupper, f.eks. udsagn det ikke er muligt at eftervise ved eksperimenter og som derfor må ligge i kategori 3, er meningsløse.

6.2 Empirismens baggrund

Det empiristiske syn på videnskaben begyndte tilbage i det 16. århundrede hvor Galileo Galilei (1564-1642) udførte kontrollerede iagttagelser af luftens tryk. Det var Galilei der for første gang systematisk brugte den empiriske metode til at beskrive sit arbejde [Petersen, 1998]. Formålet var at sikre at resultaterne var pålidelige (grunden til at de var det, analyseres i afsnit 6.5). På denne måde blev fysikken til en systematisk videnskab, og samtidig var grunden ved at blive lagt til en revolution af det videnskabelige syn. Fra de filosofiske og religiøst funderede måder at bedrive videnskab på blev eksperimenterne og den korrekte behandling af disse sat i centrum. På denne måde begyndte udviklingen at accelerere.

Grunden til denne accelerering skyldes at den empiriske metode var så meget mere effektiv til at skabe ny viden, og kan forklares med to ting: dels var den religiøse opfattelse konservativ idet den cirklede om de samme begreber så der ikke var motivation til egentlig ny viden (snarere tværtimod blev anslag til dette bekæmpet), dels var den filosofiske tilgang med a priori syntetisk videnskab ikke pålidelige, dvs. man kunne ikke være sikker på at resultaterne var i overstemmelse med virkeligheden. Og hvis resultaterne ikke er pålidelige, går udviklingen langsomt fordi hvert fremskridt så nærmest opstår ved en tilfældighed.

Ved at bruge eksperimenter som grundlag for deres forskning kunne videnskabsmænd som Blaise Pascal (1623-1662) og Isaac Newton (1642-1727) så efterhånden udvikle og cementere den empiriske metode. Senere blev metoden undersøgt af filosoffer som George Berkeley (1685-1753) og David Hume (1711-1776) og den positivistiske værdiopfattelse kom til.

6.3 Bevisførelse

Den empiriske metode indebærer forskellige teknikker til at kunne afgøre om en hypotese er rigtig eller ej, som skal følges for at det videnskabelige arbejde inden for et felt bliver anerkendt [Kragh og Pedersen, 1991]. Disse benytter sig i vid udstrækning af logisk argumentation.

6.3.1 Deduktive slutninger

To grundlæggende logiske slutningsskemaer er *modus ponens* og *modus tollens*. Begge gælder helt uafhængigt af indholdet af de udsagn der arbejdes med – hvis præmisserne er sande, vil udfaldet altid vise sig at være korrekt.

Modus tollens arbejder med negationer og kan dermed bruges til at *falsificere* med, dvs. bevise at noget er falsk – slutningsskemaet fremgår af tabel 6.2.

(1)	*Hvis* H, *så* K
(2)	*ikke*-K
(3)	*ikke*-H

Tabel 6.2: Slutningsskemaet modus tollens. H *og* K *er udsagn, og konklusionen er ikke*-H. *F.eks. kunne* H *være en hypotese og* K *være hypotesens forudsigelse af et eksperiment. Hvis eksperimentets udfald ikke er* K, *vil hypotesen* H *være falsk.*

Det ses at slutningen består af to præmisser. Første præmis (1) udledes af den teori der undersøges. Det er i praksis normalt et velbegrundet tankeeksperiment udtænkt af forskeren. Den anden præmis (2) er udfaldet af de eksperimenter forskeren udfører for at kunne bekræfte/afkræfte sin hypotese. Resultatet af disse to præmisser (3) sluttes logisk.

(1)	*Hvis* terminering ud fra validering undgår overtræning, *så* genkender store netværk trænet med validering flere bogstaver rigtigt end uden.
(2)	Genkendelsesprocenten bliver *ikke* bedre.
(3)	Terminering efter validering undgår *ikke* overtræning.

Tabel 6.3: Et eksempel på falsificering ved hjælp af modus tollens.

Et konkret eksempel på brugen af modus tollens fremgår af tabel 6.3 som ud fra eksperimentelle data i (2) giver en modstrid til den opstillede hypotese i (1); udfaldet (3) viser at hypotesen er falsk (bemærk at det er et tænkt eksempel).

Modus ponens arbejder ikke med negationer og kan bruges til at bevise udsagn. Slutningskemaet fremgår af tabel 6.4. Analogt til modus tollens er der to præmisser: (1) er at der ud fra et udsagn, f.eks. en teori, H kan udledes en konsekvens K, og (2) at teorien er sand. Så vil konsekvensen K også være sand.

(4)	*Hvis* H, *så* K
(5)	H
(6)	K

Tabel 6.4: Slutning ved hjælp af modus ponens – K *er et udsagn som følger af* H, *så hvis* H *er sand, er konklusionen at* K *er sand.*

Et konkret eksempel på brug af modus ponens fremgår af tabel 6.5.

De foregående eksempler har været af den simpleste art, men slutningsskemaerne kan også bruges i mere komplekse situationer. Hvis f.eks. der ud fra konsekvensen K af en hypotese H og hypotesen selv kan udledes en række forudsætninger F, og konsekvensen K ønskes eftervist ved hjælp af H, kan modus ponens bruges som det fremgår af tabel 6.6 på den følgende side. Hvis derimod sandhedsværdien af selve hypotesen H ønskes fastlagt, kan modus tollens anvendes til en evt. falsificering.

(1)	*Hvis* et neuralt netværk har en god generaliseringsevne, *så* kan det anvendes af forskellige mennesker.
(2)	Netværket har en god generaliseringsevne.
(3)	Det kan anvendes af forskellige mennesker.

Tabel 6.5: Et eksempel på brugen af modus ponens.

Modus ponens	Modus tollens
Hvis H *og* F, *så* K	*Hvis* H *og* F, *så* K
H *og* F	*ikke*-K
K	*ikke*-(H *og* F)

Tabel 6.6: Brug af modus ponens og modus tollens på mere komplekse problemer. Hvis forudsætningerne F *antages at være sande, kan modus ponens bruges når blot sandhedsværdien af* H *er bestemt, mens modus tollens omvendt kan bruges til at vise at* H *er falsk når* K *ikke er opfyldt.*

Modus ponens og modus tollens er særdeles vigtige inden for logikken og inden for den empiriske argumentation idet de muliggør systematiske konklusioner på sammenhængen mellem en teori og eksperimenterne. Slutningsskemaerne er udledt af nogle få grundlæggende aksiomer, så hvis man godtager dem, følger at slutningsskemaerne altid er sande. Hvis alle ens antagelser er korrekte, og man ikke har begået nogen fejl under udledningen af konsekvensen fra teorien, kan man hvis man benytter modus ponens og modus tollens, være sikker på at konklusionen er endegyldigt sand.

Denne korrekthed viser sig ved nøjere eftersyn imidlertid at være af udelukkende teoretisk karakter for mange problemer, hvorfor den logiske angrebsvinkel på empiriske problemer ikke er en komplet løsning.

6.3.2 Induktionsproblemet

Problemet ved at påvise en teori med eksperimenter er at der altid vil være en vis usikkerhed om resultatet af eksperimenterne, selv hvis man ser bort fra menneskelige fejl. Eksperimenterne vil altid kun med en vis sandsynlighed tale for eller imod en teori. Usikkerheden kan f.eks. dels komme fra måleusikkerhed eller lignende, men også fra hvad man kunne kalde statistisk usikkerhed som knytter sig til tilfældige begivenheder.

Et eksempel er kast med en terning. Kastes en terning 100 gange, og det usandsynlige sker at der slås en toer hver af de 100 gange, vil den umiddelbare konklusion på eksperimentet være at terningen igen vil ende på en toer den 101. gang. Men dette er en fejlagtig konklusion fordi terningen er homogen og derfor besidder sandsynligheden $\frac{1}{6}$ for at slå en toer og ikke $\frac{1}{1}$ som de foregående eksperimenter har vist.

Denne mulige situation vanskeliggør logisk deduktion fordi udsagn i logik enten er sande eller falske og ikke f.eks. 84% sande, og skaber dermed et stort problem, *induktionsproblemet*, i den empiriske bevisførelse [Kragh og Pedersen, 1991].

Det man stræber efter når man opstiller teorier, er slutninger fra det specielle til det generelle; induktive slutninger. Man ønsker at bevise udsagn på formen „for alle ... gælder ...“, hvilket f.eks. kan gøres ved at vise at forudsætningerne $F_1, F_2, \ldots$ for teorien er sande hvorved modus ponens giver at teorien også er sand. Imidlertid er man principielt nødt til at afprøve alle testscenarier eftersom hver mulighed udgør en forudsætning, og da der er uendeligt mange, er dette ikke muligt. En universel teori kan således ikke verificeres, kun falsificeres med modus tollens hvor der blot er behov for at én konsekvens af teorien viser sig at være forkert.

En mulig løsning på problemet er at indføre sandsynligheder i de logiske slutninger i stedet for kun at arbejde med værdierne sand og falsk. F.eks. skulle man så kunne konkludere at en teori var 99% sand hvis der var 99% sandsynlighed for at forudsætningerne var sande, se tabel 6.7. Det er imidlertid ikke lykkedes at opstille en sådan logik [Kragh og Pedersen, 1991], idet de hidtidige forsøg er endt med at have uønskede egenskaber, som f.eks. at sandsynligheden for alle udsagn på formen „for alle ... gælder ...“ er lig med nul, eller at observationer der tydeligvis ikke har med teorien at gøre, kan bidrage til at af- eller bekræfte den [Hüttel, 2001].

(7)	$P(\mathrm{H}\vert\mathrm{F}) = 0{,}99$
(8)	F
(9)	$P(\mathrm{H}) = 0{,}99$

Tabel 6.7: Et eksempel på en såkaldt probabilistisk slutning – hypotesen H *givet forudsætningen* F *har en sandsynlighed,* $P(\mathrm{H}|\mathrm{F})$. *Hvis* F *er sand, vil sandsynligheden for* H *være sandsynligheden 0,99.*

Induktionsproblemet har bl.a. som følge for dette projekt at det ikke er muligt at opstille en generel teori der beviser at det konstruerede netværk er det universelt bedste ud fra de gennemførte test. Det vil dog være muligt at benytte modus tollens til at opstille slutningsskemaer der afkræfter nogle teorier, og ligeledes vil det være muligt at bruge modus ponens til at bekræfte specifikke hypoteser i testene – kravet er blot at antallet af forudsætninger skal være overkommeligt at teste.

Da induktionsproblemet umuliggør logiske beviser for generelle hypoteser, men naturvidenskaben alligevel er fuld af den slags teorier – opstilling af sådanne teorier kan faktisk ses som selve *målet* – må der være andre muligheder for at påvise dem.

6.3.3 Slutning til bedste forklaring

Karl Popper (1902-1994) mente at muligheden for falsificering var tilstrækkelig idet udvælgelsen af en teori forløber efter en nærmest darwinistisk fremgangsmåde [Thornton, 2002].

På et givet tidspunkt vil der være en vis mængde teorier. Disse skal naturligvis være fri for selvmodsigelser og desuden være indrettet således at det er muligt at opstille påstande der kan afprøves eksperimentelt, ellers må de forkastes. Herefter udvælges en lovende teori efter en moderne videreudvikling af *Occams ragekniv* [Hüttel, 2001] efter den franciskanske munk William of Occam (1285-1349) som går ud på at den simpleste mulige teori er den bedste:

1. En hypotese der forklarer noget er bedre end en der blot kan beskrives af korrelationer. Det vil sige at selvom en matematisk kvantisering er vigtigt, så er det dog vigtigere at hypotesen indeholder en forklaring.

2. En generel hypotese er bedre end en specifik, hvilket til dels hænger sammen med næste princip – jo færre teorier der er nødvendige, jo mindre kompliceret.

3. En simpel hypotese er bedre end en kompliceret. Grunden til dette skal måske ses i den sammenhæng at en teori i forvejen er en forsimplet model af virkeligheden – jo mere simpelt den kan formuleres, jo mere vil den så bevæge sig i retning af det grundlæggende og dermed mest generelle.

4. En hypotese der hænger sammen og er integreret med eksisterende viden er bedre end en isoleret; troen på de andre teorier smitter nærmest af på den nye teori.

Når en teori er udvalgt, skal den udsættes for et antal eksperimenter. Hvis nogle af disse viser sig at være i modstrid med teorien, kan denne falsificeres ved hjælp af modus tollens, og må derfor forkastes. Ellers vil teorien gå styrket ud af prøvelsen [Kragh og Pedersen, 1991] idet den i endnu højere grad end før har vist sig at være *konsistent*.

På denne måde er det muligt at afvise en række teorier og løbende slutte sig til bedste forklaring som blot er den teori der hidtil har klaret sig bedst. Man arbejder ikke på at bevise teorierne, men på at vise at de er konsistente – herved forsvinder induktionsproblemet. Da værktøjet er falsifikation, er teorier der ikke kan falsificeres, ikke brugbare og er ifølge Popper også blot meninger [Thornton, 2002]. I centrum for metoden står altså stadig de empiriske forsøg.

6.4 Empiriske forsøg

Ifølge den empiriske metode skal de iagttagelser der erhverves ved eksperimenter beskrives og ordnes. To ting er kritiske ved alle empiriske forsøg idet de kan påvirke resultaterne af eksperimenterne: tilrettelæggelsen af forsøgene før selve processen er gået i gang, og den efterfølgende fortolkning af de opnåede resultater. Ved den empiriske arbejdsgang må altså forstås en sammenhæng mellem opstilling af hypoteser, korrekt afvikling af testene samt fortolkning af resultaterne [Kragh og Pedersen, 1991, s. 53-54].

Når der udføres empiriske forsøg for at påvise eller afvise hypoteser, er det sjældent at problemområdet er totalt isoleret. Det er derfor nødvendigt at kunne afgrænse det område man foretager eksperimenterne inden for, så de bliver teknisk og tidsmæssigt mulige at gennemføre, og samtidig vurdere hvordan afgrænsningen har en indflydelse på resultatet.

En almindelig måde at foretage afgrænsninger på er ved at gøre sig nogle antagelser om hvordan de forskellige dele i eksperimentet hænger sig, f.eks. at antage at de er uafhængige. På grund af den mængde tid det tager at udføre optræning af et neuralt netværk og de begrænsede resourcer der er til rådighed, vil eksempelvis de test som bliver foretaget i forbindelse med dette projekt også blevet opdelt efter en række antagelser. Da videnskab er systematisk og gennemskuelig tilegnelse af viden må disse antagelser imidlertid overvejes og evt. dokumenteres så eventuelle forbehold det måtte medføre for resultatet, kan inddrages i konklusionen på eksperimentet.

Både eksperimentets udformning og fortolkningen af det bliver præget af den forsker der udfører eksperimentet. Dels har forskellige personer forskellige tilgangsvinkler og metoder, f.eks. på grund af baggrund, uddannelse, personlige idealer og lignende, dels vil forskeren blive påvirket at vedkommende er ved at afprøve en bestemt teori og dermed i nogen grad tilpasse resultatet efter sin overbevisning om hvad udfaldet vil blive – man kigger efter bestemt „tegn". Det er derfor vigtigt at observatøren er så objektiv som muligt, og af samme grund foretrækkes resultater som afhænger mindst muligt af kontakt med observatøren; de skal helst kunne kvantificeres til tal.

I forbindelse med dette projekt er dette et mindre problem da resultaterne af forsøgene alle kan opgøres i tal uafhængigt af observatøren. Desuden er programmerne deterministiske så alle kørsler kan efterprøves.

Fortolkningen af eksperimentet indebærer en analyse af dataene og har til formål at kunne konkludere om eksperimentet har eftervist eller modvist teorien. Fortolkningen er dermed i endnu højere grad end dataene forbundet med teorien hvilket kan resulterer i fejlagtige konklusioner hvis dataene og de opstillede hypoteser i virkeligheden ikke hænger sammen. Eksempelvis kan fortolkningen indebære en frasortering af visse dele af dataene hvis de skønnes at være forstyrret af en fejlkilde der virker uafhængigt af teorien.

Konklusionen repræsenterer det endelige resultat af eksperimentet, men for at undgå fejl og sikre gennemskueligheden skal både data, analyse og fortolkning dokumenteres sådan at en vilkårlig person teoretisk kan gentage processen og få samme resultater. På denne måde sikres også den tillid der er nødvendig at det er muligt at bruge hinandens resultater.

Til dokumentationen af resultaterne er der endeligt opbygget et stort system af blade, bøger og konferencer med en standardiseret formidlingsform, den videnskabelige artikel, der varetager delingen af viden. Desuden varetager disse publikationskanaler en del af det sociale pres for korrekt empirisk fremgangsmåde ved at stille krav til det de publicerer.

6.5 Pålidelighed og anvendelighed

Til den empiriske metode som et hele hører altså:

1. En værdiopfattelse der tillægger betydning til det der kan erfares gennem sanserne, måles ved kontrollerede forsøg og ikke mindst eftergøres.

2. Et sæt logiske regler der accepteres som sande og derved utvetydigt kan bruges til at ræsonnere sig frem til konklusioner på eksperimenterne.

3. En erkendelse af at disse regler dog ikke er nok til at kunne udtale sig om generelle sammenhænge, og en pragmatisk udvej i form af en foreløbig accept af at en teori er brugbar når der er blevet gjort et vist antal forsøg på at falsificere den uden held.

4. En række udvælgelseskriterier for hvordan teorier foretrækkes; f.eks. teorier der kan indpasses godt med andre teorier inden for det givne felt, og teorier der er simple i deres opbygning og generelle i stedet for meget specifikke.

5. Nogle normer for hvordan eksperimenterne skal tilrettelægges, udføres, fortolkes og dokumenteres med henblik på at sikre at de kan eftergøres og er så alment gyldige som mulige i stedet for f.eks. udelukkende at afhænge af en bestemt forskers tilgangsvinkel. Samt et distributionssystem der muliggør vidensdelingen.

Betydningen af den empiriske metode er steget eksplosivt op gennem historien i takt med den teknologiske udvikling fra Galileis første forsøg til situationen i dag hvor alle teknologiske landvindinger gøres ved hjælp af metoden idet den er en integreret del af den arbejdsmetode der undervises i på f.eks. universiteter. Uden den viden som den empiriske fremgangsmåde har skabt, ville konstruktion af de mange teknologier der anvendes i dag ikke være muligt, og det danske samfund ville måske stadig være et landbrugssamfund i stil med 1500-tallets Danmark.

Grunden til metodens succes på det naturvidenskabelige område kan måske forklares med pålideligheden af de enkelte resultater, dvs. at de er i overensstemmelse med virkeligheden. Afsnittet om induktionsproblemet viste at resultaterne ikke nødvendigvis er korrekte i det generelle tilfælde, men samtidig gør den positivistiske værdiopfattelse der lægger vægt på fænomerne skal kunne måles, at teorierne udformes så det er muligt at vise forskellige specialtilfælde af teorierne. Dermed opbygges en vis pålidelighed inden for de områder man efterprøver – og selvom de foreslåede teorier viser sig at være forkerte, vil de nye teorier typisk vise sig at være i meget god overenstemmelse med de gamle på de områder hvor de tidligere teorier viste sig at være konsistente med målingerne.

Et kendt eksempel er Newtons mekanik som kan bruges til præcist at forudsige f.eks. hvor lang tid det tager for en genstand at tilbagelægge en strækning under påvirkning af en anden genstand – det viste sig senere at resultaterne var forkerte ved meget store hastigheder, hvorfor teorien måtte erstattes med Einsteins specielle relativitetsteori. Men denne giver nøjagtigt samme resultat for små hastigheder.

Det ses at pålideligheden afhænger af to ting: pålideligheden af kun at foretage „stikprøver“ når en teori testes inden for et område, og pålideligheden af de enkelte forsøg. Sidstnævnte søger normerne for det eksperimentelle arbejde at øge idet de skal sikre at resultaterne er uafhængige af den person der udfærdiger eksperimentet – rent praktisk set forsøges desuden altid at minimere andre fejlkilder der kunne påvirke pålideligheden, f.eks. usikkerhed på måleinstrumenterne.

At stikprøvemetoden er pålidelig, er ikke umiddelbart indlysende, men ikke desto mindre som regel et faktum der er blevet erfaret – tilsyneladende opfører verden sig tilregneligt. Desuden, hvis eksperimenterne involverer så komplekse sammenhænge, f.eks. på det medicinske område, at der mistanke om at der ikke udføres tilstrækkeligt mange, er en del af objektivitetskravet at konklusionerne også skal være tilsvarende forsigtige.

På områder hvor det er muligt at udfærdige hypoteser der kan testes gennem kontrollerede eksperimenter, vil den empiriske metode – under antagelse af at eksperimenterne udføres pålideligt – derfor være pålidelig som videnskabelig metode, også selvom det ikke er muligt at opnå en garanti for at de generelle teorier er gyldige i hele det område de postuleres inden for.

6.5.1 Anvendelse i dette projekt

Ved at anvende den empiriske metode med vekselvirkning mellem teoridannelse og forsøg til at skabe viden bliver det muligt at forbedre det neurale netværk vi har konstrueret. Desuden kan metoden hjælpe med til at vurdere hvor godt det endelige netværk er. Der synes ikke at være andre muligheder for at opnå tilsvarende resultater i løbet af kort tid.

Det ville f.eks. ikke være muligt at forbedre netværket gennem samtaler med forskellige brugere der har fået hver sit netværk, dels fordi det ville medføre problemer med om udsagnene var pålidelige, dels fordi det ikke er praktisk muligt at skabe tilstrækkelig viden om netværkets virkemåde gennem den metode til at det er muligt at stille på de mulige parametre. Der er brug for en opdeling af problemet gennem hypoteser om hvordan netværket fungerer, for at det er muligt at opnå tilstrækkeligt detaljeret viden.

Den empiriske metode giver derimod en effektiv og pålidelig løsning. Til gengæld er det ikke sikkert den ville være nær så effektiv til at vurdere om det endelige netværk implementeret på en håndholdt ville være tilfredsstillende nok til brugerne, også selvom metoden kan give præcise tal for f.eks. genkendelsesevnen.

De belyste metoder til at undersøge hypoteser er følgende:

- Slutning vha. modus tollens, falsificering.
- Slutning vha. modus ponens, logisk slutning.
- Poppers metode, med slutning til bedste forklaring.

Den type hypoteser der ønskes afklaret i forbindelse med eksperimenterne i dette projekt, vil typisk være meget specifikke, hvorfor modus ponens kan komme i anvendelse. Samtidig er der dog behov for også at kunne forsvare mere generelle hypoteser, f.eks. for at kunne begrunde de afgrænsninger der er foretaget i testene. Hertil er princippet om at benytte den bedste forklaring velegnet. Modus tollens kan bruges så snart et eksperiment viser sig at få et uventet resultat.

Kapitel 7

Test

Det overordnede mål for testene er at efterprøve den teori der er blevet gennemgået i kapitel 4. En test kunne f.eks. være at vi søger afklaret om en centrering af bogstaverne er en god idé eller ej. Ud fra disse resultater sammensætter vi det netværk som ud fra vores test giver den højeste genkendelsesprocent.

Efter at have sammensat dette endelige netværk udfører vi en fuldstændig krydstest og en optræning med forskellige startværdier for vægtene for at bedømme netværkets endelige genkendelsesevne.

Med dette netværk søger vi afklaret om en eftertræning af netværket på bogstaver fra en isoleret testperson vil forbedre netværkets evne til at genkende flere af vedkommendes bogstaver.

Herefter redegør vi for om det netværk vi har valgt ud, er praktisk muligt at implementere på en håndholdt computer.

Til sidst søger vi afklaret om vores testafsnit har levet op til den empiriske teori der er beskrevet i kapitel 6, og hvilke problemer der kan være med den metode vi benytter.

7.1 Empiriske overvejelser

Vores beslutning om at benytte den empiriske metode til at efterprøve teorien medfører nogle krav til planlægning, udførelse, fortolkning og dokumentation af testene. Vi har derfor opstillet en række krav til vores arbejdsgang og dokumentation:

- Inden en test udføres skal der opstilles en velbegrundet hypotese. I hypotesen skal der indgå hvad testen skal vise og hvilken tankeproces der ligger bag.
- Alle udførte test skal dokumenteres på en sådan måde at det er muligt at efterprøve resultaterne.
- Efter udførelsen af en test skal alle væsentlige resultater fremgå, og gyldighedsområdet for hypotesen vurderes.

- I fortolkningen af en test skal argumentation være klar så der skelnes mellem argumenter der bruger logiske slutninger, og argumenter der går på slutning til bedste forklaring.

7.2 Praktiske begrænsninger

De mange test er meget beregnings- og hukommelsestunge. Da vi ikke havde adgang til en supercomputer, har vi benyttet nogle af basisuddannelsens 1 GHz-maskiner med 256 MB ram til test. For at koordinere de enkelte maskiners arbejde konstruerede vi et databasesystem som kørte fra basisuddannelsens centrale server. Maskinerne fik fra denne tildelt en opgave og efter hver iteration blev vægtene gemt og logfilen opdateret.

For at modvirke overtræning som omtalt i afsnit 4.4.5 benyttede vi et valideringssæt som termineringskriterium hvor netværket forsøgte sig med maksimalt 50 iterationer efter det hidtidige minimum for valideringsfejlen blev opnået. Desuden blev krydstest benyttet til at bestemme genkendelsesprocenten. Desværre tillod tiden ikke at vi udførte en fuldstændig krydstest, så i stedet blev krydstesten udført med to personer hvor henholdsvis Anders' og Laus træningseksempler blev udeladt fra optræning. Den endelige genkendelsesprocenten blev så estimeret ved at teste med x's træningseksempler på det netværk hvor x's træningseksempler blev udeladt, og til sidst beregne gennemsnittet af disse genkendelsesprocenter.

For at sikre at den fundne genkendelsesprocent ikke er et udtryk for hvilke parametre der passer bedst til et lokalt område i hyperrummet, burde der udføres flere test med forskellige startværdier af vægtene. Dette ville dog forøge antallet af test kraftigt og var således ikke praktisk muligt, undtagen for det afsluttende estimat. For at gøre testene sammenlignelige holdes startværdierne for vægtene ellers ens.

Da nogle af parametrene er kontinuerte, er en fuldstændig test af alle parametersammensætninger umulig. Derfor bliver vi nødt til at begrænse os til en endelig mængde parametre. En del af denne begrænsning kan foretages ved hver test teoretisk – f.eks. giver det ikke mening at sætte inertikonstanten til værdier uden for intervallet $[0;1[$ – og ud fra dette udvælges nogle rimelige værdier på baggrund af gruppens erfaringer eller forslag i litteraturen.

Selv med nogle få udvalgte værdier er det dog ikke overkommeligt at teste alle kombinationer, og ved de fleste af testene bliver vi derfor nødt til at antage at de forskellige parametre ikke er korrelerede så vi kan teste dem en af gangen med de andre parametre fastholdt.

7.3 Opstilling, udførelse og fortolkning af indledende test

Vi opstiller følgende hypoteser:

- Centrering forbedrer genkendelsesprocenten.

- Tanh som aktiveringsfunktion fremfor sigmoid giver hurtigere optræning med en lignende genkendelsesprocent.
- Valget af η og μ har stor betydning for et givet netværk.
- Konjugeret gradient konvergerer hurtigere end gradientnedstigning med en lignende genkendelsesprocent.
- Antallet af skjulte neuroner har stor betydning for et givet netværk.
- En høj ekstern skaleringsgrad forringer genkendelsesprocenten.
- Intern skalering forbedrer genkendelsesprocenten i forhold til ekstern skalering.
- En regionsskalering giver bedre genkendelsesprocent end andre skaleringsmetoder.
- Størrelsen af valideringssættet har indflydelse på genkendelsesevnen

De enkelte test er opbygget ved at hypotesen først opstilles, herefter testes og til sidst fortolkes resultaterne. Denne opbygning sker for at skabe et overblik over de enkelte resultater og konsekvenserne heraf.

7.3.1 Ordforklaring

Forkortelse	Forklaring
Centr.	Hvis „til“, er billederne blevet centreret efter massemidtpunkt inden billedet bruges som input.
Skal.	1:x angiver graden af den eksterne skalering der er foretaget som en ekstern forbehandling.
Neur.	Antal neuroner i det skjulte lag.
Akt.fkt.	Aktiveringsfunktionen i neuronerne, enten tanh eller sigmoid.
Træning	Træningsmetoden, enten konjugeret gradient eller gradientnedstigning.
Genk.	Procentdel genkendte bogstaver fra det træningssæt som netværket ikke er blevet optrænet på.
Gr. neds.	Træning med gradientnedstigning med inerti.
Konj. gr.	Træning med skaleret konjugeret gradient.
V.str.	Procentdelen af træningssættet der er anvendt som valideringssæt.
η	Træningskonstanten til gradientnedstigning.
μ	Inertikonstanten der trænes med.
Ant. iter.	Antallet af gennemløb træningsalgoritmen foretager over hele træningssættet inden terminering.
Seed	Den værdi genereratoren af tilfældige tal startes med.
Testperson	Den person hvis bogstaver isoleres under selve optræningen og efterfølgende bruges til at bedømme netværkets genkendelsesprocent.

7.3.2 Opstilling af netværk

Netværket opbygges som et fuldt forbundet fremadførende netværk bestående af et inputlag, et skjult lag og et outputlag. Antallet af neuroner i inputlaget kan i vores opstilling antage 4 størrelser: 2880 uden ekstern skalering, 720 med 1:2 ekstern skalering, 180 med 1:4 ekstern skalering eller 80 med 1:6 ekstern skalering. Skaleringsgraden er angivet i resultattabellerne. Antallet af skjulte neuroner varierer mellem 50-200 og er ligeledes angivet under hver hypotese. Antallet af output neuroner er altid 29. Ved intern og regionsskalering benyttes dog en anden arkitektur, som beskrevet i afsnit 4.3.2.

Hvis intet andet er angivet, benyttes de følgende værdier for de forskellige parametre:

Parameter	Værdi
Centr.	Til
Akt.fkt.	tanh
Træning	Gradientnedstigning
η	0,3
μ	0,1
V.str.	10%
Seed	1
Testperson	Gennemsnit mellem Lau og Anders

7.3.3 Hypotese: Centrering forbedrer genkendelsesprocenten

Vi ønsker efterprøvet om centrering forbedrer genkendelsesprocenten.

Baggrunden for at opstille hypotesen er at en centrering vil ensarte billederne, hvilket for netværket formodentlig vil virke som et større antal træningseksempler for de vægte der ligger omkring træningseksemplets centrum. Hvis der var tilstrækkeligt mange træningseksempler, burde centreringen være unødvendig, men dette er næppe tilfældet.

Da en forøgelse af træningssættet generelt vil modvirke overtræning som er et problem for alle de opstillede netværk fordi antallet af vægte er stort i forhold til antallet af træningseksempler, burde tendensen også være ens for alle netværk. Vi tester derfor kun centrering med et enkelt netværk.

Resultater

Centr.	Skal.	Neur.	Træning	Genk.
Fra	1:2	50	Konj. gr	54,7%
Til	1:2	50	Konj. gr	76,0%

Fortolkning

Netværkets genkendelsesprocent er altså forbedret ved brug af centrering i ovenstående testscenarie. Ud fra testen kan man ikke med sikkerhed konkludere at forbedringen i genkendelsesprocenten skyldes at centreringen er ækvivalent med en forøgelse af træningseksemplerne, men dette synes at være den bedste forklaring.

Ved at slutte til denne fås at centrering også vil forbedre genkendelsesprocenten på andre netværk under antagelse af at disse også bliver overtrænet.

7.3.4 Hypotese: Tanh som aktiveringsfunktion fremfor sigmoid giver hurtigere optræning med en lignende genkendelsesprocent

Det ønskes afgjort hvilken aktiveringsfunktion der giver den hurtigste optræning af netværket. Samtidig ønskes efterprøvet om valget af aktiveringsfunktion kun giver en mindre forskel i genkendelsesprocenten.

Motivationen for hypotesen kommer fra litteraturen, bl.a. [Bishop, 1995; Sarle, 1999], hvor det fremgår at brug af tanh som aktiveringsfunktion erfaringsmæssigt giver hurtigere konvergens end sigmoid. Hvis tanh viser sig at være hurtigere end sigmoid, bliver det muligt at teste flere netværk på den tid der er til rådighed, hvilket indirekte kan hjælpe med til at finde et bedre netværk.

Under testen blev målværdierne for sigmoid naturligvis ændret til 0,1 og 0,9 i stedet for $-0{,}9$ og 0,9 som benyttes ved tanh.

Resultater

Testperson	Ant. iter.	Centr.	Skal.	Neur.	Akt.fkt.	Træning	Genk.
Lau	237	Fra	1:2	50	tanh	Konj. gr.	63,3%
Lau	335	Fra	1:2	50	sigmoid	Konj. gr.	64,0%
Anders	220	Fra	1:2	50	tanh	Konj. gr.	46,1%
Anders	388	Fra	1:2	50	sigmoid	Konj. gr.	46,2%

Fortolkning

Det ses at tanh omtrent giver den samme genkendelsesprocent som sigmoid, men at sigmoid skal bruge henh. 41% og 76% flere iterationer. Altså kan vi konkludere at hypotesen har vist sig at være rigtig for disse specifikke tilfælde. At tanh i begge tilfælde giver en lidt lavere genkendelsesprocent, kan tilskrives tilfældigheder – det ville være nødvendigt med flere test for at afgøre dette, men da forskellen er så lille, har vi ikke anset dette for berettiget; resultatet er klart med hensyn til antallet af iterationer.

Da vi ikke kan falsificere hypotesen som i øvrigt understøttes af erfaringerne fra litteraturen, vælger vi derfor at konkludere dens foreløbige korrekthed og følger den ved at bruge tanh i det følgende for at kunne udføre flest mulige test.

7.3.5 Hypotese: Valget af η og μ har stor betydning for et givet netværk

Det ønskes fastslået om værdierne af træningskonstanten og inertikonstanten ved gradientnedstigning har nogen større betydning for antallet af iterationer og størrelsen af genkendelsesprocenten for det færdige netværk. Desuden ønskes en god værdi for konstanterne bestemt.

Ifølge den teoretiske opfattelse af hvordan træningen forløber bør de optimale værdier af η og μ være afhængige af hvordan fejlrummet ser ud, og da enhver ændring i netværkets parametre giver et nyt fejlrum, forventer vi ikke at resultatet fra denne test vil stemme nøjagtigt overens med resultatet fra andre test. Hvis parametrene ikke ændres for drastisk – f.eks. ved at skifte aktiveringsfunktion – bør de lokale minima i fejlrummet dog være af nogenlunde af samme størrelse og udstrækning, så parameterværdierne fundet i denne test stadig er fornuftige til andre test.

Meningsfulde værdier for inertikonstanten ligger i intervallet $0 \leq \mu < 1$. Hvis inertikonstanten er større end eller lig med 1, vil optræningen aldrig konvergere, og hvis den ligger under 0, vil man modarbejde optræningen og gå væk fra minimum. Ligeledes skal η være større end 0. Vi har yderligere erfaret at fornuftige værdier ligger omkring 0,1.

Resultater med η

Ant. iter.	Skal.	Neur.	Træning	η	μ	Genk.
2623	1:2	50	Gr. nedst.	0,1	0	79,3%
1338	1:2	50	Gr. nedst.	0,3	0	76,8%

Fortolkning

Det ses at værdien af η har stor betydning for både antallet af iterationer og genkendelsesprocenten og at 0,1 gav bedst genkendelsesprocent. Dette kan forklares med at jo mindre vægtopdateringer der foretages pr. iteration, jo dybere i det lokale minima kan algoritmen nå. Det ses desuden at algoritmen ved værdien 0,1 skulle bruge omkring dobbelt så mange iterationer for at nå dette minimum.

Ovenstående resultater antyder at en mindre træningskonstant altid vil give en højere genkendelsesprocent, men man kan forestille sig at optræning med små værdier af træningskonstanten vil have en tendens til at havne i selv små lokale minima. Dette er dog ikke en konklusion der kan uddrages af ovenstående resultater.

Resultater med μ

Ant. iter.	Skal.	Neur.	Træning	η	μ	Genk.
2008	1:2	50	Gr. nedst.	0,1	0,1	78,7%
645	1:2	50	Gr. nedst.	0,1	0,5	78,0%
380	1:2	50	Gr. nedst.	0,1	0,9	77,7%
606	1:2	50	Gr. nedst.	0,3	0,1	78,6%
275	1:2	50	Gr. nedst.	0,3	0,5	75,4%
53	1:2	50	Gr. nedst.	0,3	0,9	19,7%

Fortolkning

Det ses at træning med inerti har sænket antallet af iterationer væsentligt. For træningskonstanten 0,1 er genkendelsesevnen dog også blevet forværret, jvf. den forrige tabel. For $\eta = 0{,}3$ er situationen omvendt med inertikonstanten 0,1 – en forklaring på dette kunne være at inertien sætter algoritmen i stand til at passere gennem et lokalt minimum. Det kunne imidlertid også blot skyldes at algoritmen pga. inertien tilfældigvis tog en anden rute der havde et andet lokalt minimum som endepunkt.

For begge værdier af η ses ellers en faldende tendens i genkendelsesprocenten. Af denne grund vælges værdierne $\eta = 0{,}3$ og $\mu = 0{,}1$.

7.3.6 Hypotese: Konjugeret gradient konvergerer hurtigere end gradientnedstigning med en lignende genkendelsesprocent

Ønsket er at få bekræftet om skaleret konjugeret gradient konvergerer hurtigere tidsmæssigt set (og ikke iterationsmæssigt) end gradientnedstigning. Samtidig skal konjugeret gradient have en lignende genkendelsesprocent for at være anvendelig.

Motivationen for hypotesen kommer fra afsnit 4.4.4 og fra Bishop [1995] hvor det påstås at konjugeret gradient er mere effektiv til optræning. Hvis dette er tilfældet, vil brug af konjugeret gradient sætte os i stand til at udføre flere test.

For at kunne afgøre det tidsmæssige aspekt er det nødvendigt at teste hvor lang tid hver iteration tager for algoritmerne. Dette blev testet på en typisk testmaskine, jvf. afsnit 7.2, med et fastholdt antal iterationer.

Resultater

Ant. iter.	Skal.	Neur.	Træning	η	μ	Genk.
384	1:2	50	Konj. gr.	–	–	76,0%
1097	1:2	50	Gr. nedst.	0,3	0,1	78,6%

Ant. iter.	Skal.	Neur.	Træning	η	μ	Tid (s)
15	1:6	50	Konj. gr.	–	–	335
15	1:6	50	Gr. nedst.	0,3	0,5	101

Fortolkning

Ud fra den iterationsmæssige sammenligning mellem træningsmetoderne ser det ud som om konjugeret gradient giver den hurtigste optræning da denne kun bruger 384 iterationer i forhold til 1097 for gradientnedstigning. Men ved den efterfølgende test af tid pr. iteration viser det sig at konjugeret gradient tager 3,3 gange så lang tid pr. iteration som gradientnedstigning. Hvis man ganger dette med antallet af iterationer det tager at optræne med konjugeret gradient, får man $3{,}3 \cdot 384 \approx 1267$ så gradientnedstigning faktisk er hurtigst. Det vil sige at hypotesen er falsificeret.

Det ses desuden at gradientnedstigning giver bedre genkendelsesprocent end konjugeret gradient. Dette kan hænge sammen med at gradientnedstigningen er stokastisk så algoritmen har mulighed for at slippe ud af lokale minima som kun er minima for visse dele af træningssættet.

Det at gradientnedstigningen er stokastisk i modsætning til konjugeret gradient kan også forklare hvorfor gradientnedstigning konvergerer hurtigere – som nævnt i afsnit 4.4.3 er den stokastiske version at foretrække når der er meget redundans i dataene, hvilket er tilfældet med 40 forholdsvist ens bogstaver fra hver person. LeCun et al. [1998] angiver et lignende resultat for et tilsvarende tegngenkendelsesproblem.

Denne forklaring forekommer den bedste, og ved at slutte til den vil gradientnedstigning være hurtigere end konjugeret gradient for de andre test også da træningssættet er det samme. Derfor benyttes gradientnedstigning til de følgende test.

7.3.7 Hypotese: Antallet af skjulte neuroner har stor betydning for et givet netværk

Ønsket er at få efterprøvet hvor meget netværkets genkendelsesprocent afhænger af antallet af skjulte neuroner og samtidig finde en god værdi for dette antal.

Vi viste i afsnit 4.4.5 at et for stort antal skjulte neuroner kan medføre overtræning hvilket betyder en forringelse af genkendelsesprocenten, men at det samtidig var nødvendigt med en vis mængde skjulte neuroner for at det neurale netværk kunne tilpasse sig den ønskede funktion. Det optimale antal skjulte neuroner må ligge i dette interval, men da både øvre og nedre grænse afhænger af datasættet, kan dette interval ikke specificeres yderligere.

Af denne grund må vi finde den optimale værdi empirisk. Vi har som udgangspunkt valgt at teste og dokumentere netværk med 50, 100 og 200 skjulte neuroner. Disse værdier er valgt ud fra nogle indledende (ikke-dokumenterede) forsøg og ud fra hvad der er tidsmæssigt muligt at teste.

Resultater

Skal.	Neur.	Genk.
1:2	50	78,6%
1:2	100	82,4%
1:2	200	84,3%

Fortolkning

Det ses at genkendelsesprocenten ændrer sig væsentligt, så hypotesen er blevet bekræftet. At genkendelsesprocenten stiger med antallet af skjulte neuroner, tyder på at netværket ikke har haft tilstrækkeligt mange frihedsgrader til at kunne tilpasse sig datasættet. Hvis motivationen for testen – at det optimale antal af skjulte neuroner ligger i et interval – er rigtig, skulle den stigende tendens på et tidspunkt vende. Denne teori kan ikke afgøres ud fra testresultaterne da det hverken er muligt at påvise eller falsificere.

Den øvre grænse er muligvis ikke nået ved 200 neuroner, men da yderligere forøgelse i antallet bliver for tidskrævende ved de følgende test, fortsættes med dette antal.

7.3.8 Hypotese: En høj ekstern skaleringsgrad forringer genkendelsesprocenten

Ønsket er at efterprøve hvor meget ekstern skalering det er muligt at anvende uden at det forringer genkendelsesprocenten.

Motivationen for hypotesen er at en ekstern skalering kan reducere problemet med overtræning da antallet af vægte reduceres som beskrevet i afsnit 4.2.1, men samtidig er tabsbetinget.

Den nedre grænse for skalering af træningseksemplerne er når der ikke længere eksisterer nok information i et træningseksempel til at skelne de 29 forskellige bogstaver. Teoretisk set er dette $\lceil \log_2 29 \rceil = 5$ bit, men en mere brugbar grænse er når et menneske ikke længere kan skelne bogstaverne fra hinanden. Hvis menneskelige neurale netværk ikke kan genkende de skrevne tegn, er det næppe muligt for vores begrænsede kunstige netværk. Vi har fundet den menneskelige grænse til at være 1:6, se f.eks. appendiks B.2, hvilket reducerer træningseksemplerne fra deres originale størrelse på 2880 punkter til 80 punkter. Spørgsmålet er hvad den tilsvarende grænse er for netværket.

Resultater

Skal.	Neur.	Genk.
1:2	50	78,6%
1:4	50	76,1%
1:6	50	66,6%
1:2	100	82,4%
1:4	100	80,7%
1:6	100	73,5%
1:2	200	84,3%
1:4	200	83,3%
1:6	200	77,7%

Fortolkning

Det ses at genkendelsesprocenten falder når billederne skaleres mere end 1:2 for alle de forskellige antal af mellemlagsneuroner, og hypotesen er dermed bekræftet for disse tilfælde. Da andre parametre er blevet fastholdt, og tendensen er klar, slutter vi os derfor til at hypotesen er den bedste forklaring og fortsætter med skaleringsforholdet 1:2.

7.3.9 Hypotese: Intern skalering forbedrer genkendelsesprocenten i forhold til ekstern skalering

Ønsket er at efterprøve om der er forskel i genkendelsesprocenten alt efter om man bruger intern eller ekstern skalering.

Motivationen for hypotesen er at den interne skalering, som beskrevet i afsnit 4.2.1, giver mulighed for at vægte de enkelte punkter i skaleringen hvor ekstern skalering blot tager et gennemsnit. Denne mulighed kan måske forbedre genkendelsesprocenten.

Der skal således udføres samme test som i afsnit 7.3.8 blot hvor skaleringen internt og eksternt er målt mod hinanden.

Resultater

Skal.	Neur.	Ekstern genk.	Intern genk.
1:2	50	78,6%	77,3%
1:4	50	76,1%	78,8%
1:6	50	66,6%	74,6%
1:2	100	82,4%	81,7%
1:4	100	80,7%	81,1%
1:6	100	73,5%	75,8%
1:2	200	84,3%	81,9%
1:4	200	83,3%	82,2%
1:6	200	77,7%	78,7%

Fortolkning

Det ses at hypotesen er falsificeret da intern skalering kun i 3 af de 9 test giver en bedre genkendelsesprocent. Den dårligere genkendelsesprocent kan måske skyldes at de flere vægte giver større risiko for overtræning. Da intern skalering under alle omstændigheder giver langsommere træning fordi de mindre eksempelstørrelser ved ekstern skalering nedsætter processeringstiden, undlader vi derfor at bruge intern skalering i det følgende.

Ellers ses det at samme tendens som ved ekstern skalering gør sig gældende for intern skalering idet genkendelsesprocenten falder med større skalering selvom 1:4 skalering i to af tilfældene faktisk gav bedre genkendelsesprocent end 1:2.

7.3.10 Hypotese: En regionsskalering giver bedre genkendelsesprocent end andre skaleringsmetoder

Ønsket er at efterprøve om en skalering der er tilpasset den typiske informationsfordeling i et træningseksempel, forbedrer genkendelsesprocenten i forhold til de andre skaleringsmetoder.

Motivationen for denne hypotese er at regionsskaleringen som beskrevet i afsnit 4.3.2 kun har vægte hvor der skulle være brug for det, så problemet med overtræning mindskes. Da regionsskalering består af forskellige skaleringsgrader, kan disse ikke direkte sammenlignes med de andre to skaleringsmetoder, og vi har derfor udvalgt resultater fra dem efter sammenlignelige antal af vægte.

Resultater

Skal.	Antal vægte	Testperson	Neur.	Genkend.
Ekstern 1:4	10.450	Lau	50	89,3%
Intern 1:4	13.330	Lau	50	85,4%
Region	15.830	Lau	50	87,0%
Ekstern 1:4	20.900	Lau	100	88,6%
Intern 1:4	23.780	Lau	100	91,0%
Region	28.780	Lau	100	90,0%
Ekstern 1:4	41.800	Lau	200	90,2%
Intern 1:4	44.680	Lau	200	93,5%
Region	54.680	Lau	200	90,6%

Fortolkning

Det ses at hypotesen er falsificeret da regionsskalering i ingen af tilfældene gav den bedste genkendelsesprocent, men altid ligger mellem intern og ekstern skalering. Resultaterne kan tolkes som at noget af den omkringliggende information i nogle tilfælde alligevel har været væsentlig. Denne måde at foretage regionsskalering på er altså ikke den optimale måde at skalere på, og vi vælger derfor ikke at fortsætte med denne.

7.3.11 Hypotese: Størrelsen af valideringssættet har indflydelse på genkendelsesevnen

Ønsket er at afgøre om andelen af data der bruges til valideringssæt, har indflydelse på genkendelsesevnen.

Motivationen for hypotesen er at valideringssættet bruges til at estimere generaliseringsevnen undervejs med henblik på at afgøre hvornår træningen skal stoppe. Hvis valideringssættet ikke er omfattende nok, vil estimatet ikke være korrekt, og træningen kan derfor blive stoppet enten for tidligt eller for sent.

Men når størrelsen af valideringssættet forøges, formindskes størrelsen af træningssættet samtidig. Derved forøges overtræningsproblemet. Den optimale størrelse af valideringsættet er derfor en afvejning af de to fænomener.

Resultater

V.str.	Skal.	Neur.	Genk.
10%	1:2	50	78,6%
20%	1:2	50	79,2%
30%	1:2	50	77,7%

Fortolkning

Det ses at hypotesen er konsistent med de opnåede resultater idet genkendelsesevnen er skiftet med størrelsen af valideringssættet. Forskellene i værdierne er dog ikke større end at det kunne skyldes tilfældige udsving – men værdierne følger dog den forudsete fordeling med et toppunkt ved 20%.

7.4 Opstilling af de endelige test

Ud fra de ovenstående test kan vi sammensætte et netværk bestående af:

- Centrering efter massemidtpunkt.
- Aktiveringsfunktionen tanh.
- Gradient nedstigning som træningsalgoritme.
- 200 neuroner i det skjulte lag.
- Ekstern 1:2 skalering (720 inputneuroner).

Vores indledende test synes at vise at flere skjulte neuroner og lavere skalering giver højere genkendelsesprocent, men der er imidlertid et problem med at træningstiden for et større antal neuroner og en 1:1 skalering vil være uhensigtsmæssig stor. Uhensigtsmæssig fordi antallet af test på det endelige netværk i så fald skulle begrænses til ganske få. Endvidere vil vi senere kigge på om netærket kan eftertrænes til en enkelt bruger. Dette skal i såfald ske på en håndholdt computer hvor der således vil være krav om en relativ kort træningstid. Vi har derfor valgt ikke at teste større netværk end dem der er beskrevet i de indledende test.

Et netværk med de udvalgte parametre blev testet i afsnit 7.3.7 og fik en genkendelsesprocent på 84,3% baseret på en krydstest mellem Anders og Lau. I de endelige test undersøger vi om andre startværdier for vægtene har stor betydning for dette netværks genkendelsesevne og om en fuldstændig krydstest vil ændre på resultatet.

7.4.1 Resultater – forskellige startværdier

Seed	Testperson	Skal.	Neur.	Genk.
1	Lau	1:2	200	92,3%
2	Lau	1:2	200	90,0%
3	Lau	1:2	200	91,5%

Fortolkning

Det ses at forskellige seed-værdier giver forskellige genkendelsesprocenter. Dette kan forklares med at netværket ved optræning ender i forskellige lokale minima.

Resultaterne fra denne test viser hvordan man selv om man har fundet et godt netværk, aldrig kan være sikker på at finde det bedste; der kan altid ændres på udgangspunktet for vægtene og denne ændring vil højst sandsynligt føre til en ændring i genkendelsesprocenten. Det giver derfor ikke mening konkludere at seed-værdien 3 er bedst da denne sagtens kan give andre resultater ved test på andre personer. For sammenligning med forrige resultater er næste test foretaget med værdien 1.

7.4.2 Resultater – krydstest

Testperson	Skal.	Neur.	Genk.
Jasper	1:2	200	64,5%
Rasmus	1:2	200	90,5%
Martin	1:2	200	74,0%
Ole	1:2	200	91,6%
Lau	1:2	200	92,3%
Anders	1:2	200	76,2%
Gen.snt.			81,5%

Fortolkning

Det ses at når forskellige personer udelades, giver netværket vidt forskellige genkendelsesprocenter. Tre af testpersonerne giver en bemærkelsesværdig lav genkendelsesprocent.

Den bedste forklaring på dette synes at være at der i de tre personers datasæt er større afvigelser fra den almindelige skrivemåde. Måske afviger disse personers håndskrift markant fra normen eller måske skriver de ikke så ensartet som resten af personerne. De fem andre personers datasæt der henholdsvist er blevet trænet på, har for disse tre personer i hvert fald ikke været nok til at give en god genkendelsesprocent. Muligvis skyldes problemet overtræning, eller muligvis er der bare ikke tilstrækkeligt med data.

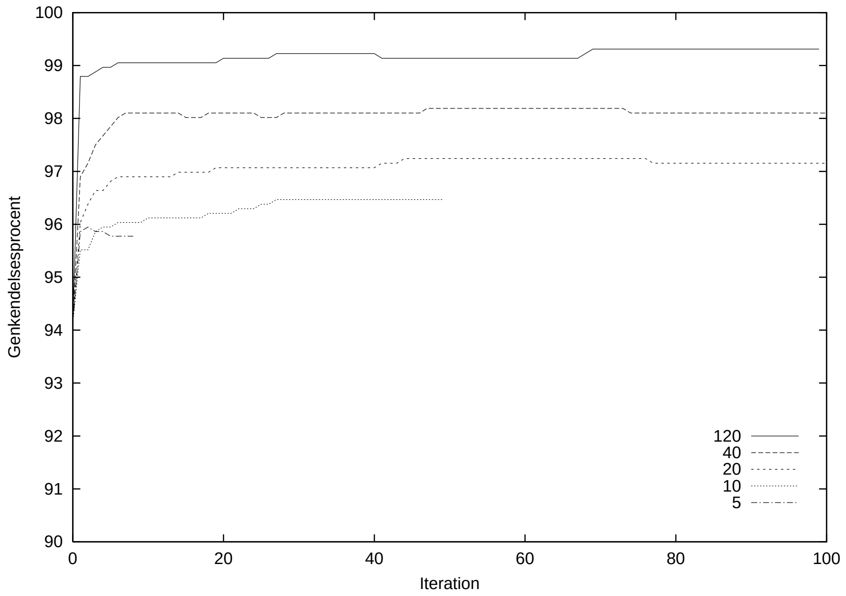
Den gennemsnitlige genkendelse blev under alle omstændigheder på 81,5%, hvilket betyder at der i gennemsnit vil opstå en fejl for hvert femte skrevne tegn – et uacceptabelt niveau for en bruger. En mulighed var at indsamle data fra flere personer – for en egentlig kommerciel implementering ville dette ikke være et problem – men da dette ikke har været praktisk muligt for os, forsøger vi i stedet at vende overtræningsproblemet til en fordel gennem en specialisering af netværket til den enkelte bruger.

7.5 Eftertræning

En sådan specialisering kunne gøres ved at optræne det neurale netværk fra bunden, men dette ville kræve meget tid og et stort datasæt fra brugeren. Især sidstnævnte er et problem. I stedet foreslår vi at eftertræne et færdigt netværk. Dette giver mulighed for at levere et generelt optrænet netværk til en bruger som så efterfølgende kan forbedre genkendelsesprocenten ved at lade det eftertræne. Om dette kan lade sig gøre ønskes efterprøvet i dette afsnit.

Det ville være fordelagtigt for en bruger at en eftertræning kunne forløbe hurtigt og med få indskrevne tegn. For at kunne få en idé om hvor stort et træningssæt der behøves, og hvor mange iterationer det kræver at forhøje genkendelsesprocenten, testes dette. Testen blev udført på det netværk der blev krydstestet i forrige afsnit. Dog var der kun data tilgængelige til at eftertræne på en enkelt person, nemlig Lau; af denne grund blev der eftertrænet på det netværk der ikke var blevet optrænet med træningseksempler fra Lau. Først blev 40 billeder af hvert bogstav isoleret og udelukkende brugt til at estimere genkendelsesprocenten med.

En test af forskellige størrelser datasæt fremgår af figur 7.1 hvor tallet angiver antallet af træningseksempler af hvert bogstav, og for hver af disse datasæt blev 20% brugt til at stoppe vha. validering.

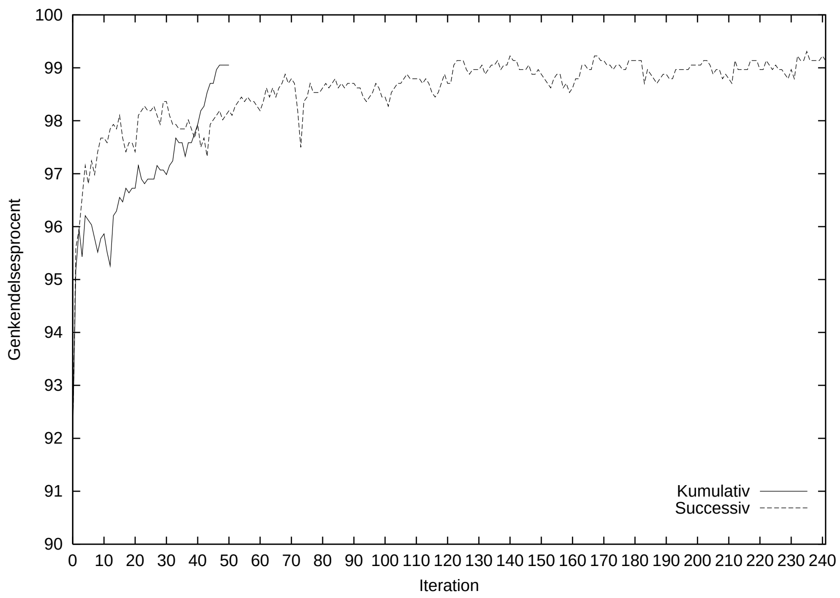


Figur 7.1: Genkendelsesprocent som funktion af antallet af iterationer med forskellige størrelser af træningssættet. Genkendelsesprocenten stiger kraftigt i løbet af de første 10 iterationer.

Det ses at jo større datasæt der bliver eftertrænet på, jo bedre bliver genkendelsesprocenten. Desuden ses det at størstedelen af forbedringen sker i løbet af de første 10 iterationer.

I situation på en håndholdt computer er der dog ikke adgang til alle dataene på en gang idet de skal indsamles undervejs. Der er så to muligheder for eftertræning: man kan lade netværket træne på alle de hidtil indsamlede data, eller man kan nøjes med at lade netværket træne på de senest indsamlede data. Hvilken af disse metoder der giver bedste genkendelsesprocent, er ikke umiddelbart klart så dette må testes. Disse test blev udført på det samme netværk som i testen ovenfor. Netværket bliver optrænet 10 iterationer hvorefter datasættet skiftes ud; af samme grund er der ikke benyttet validering.

Det ses at begge metoder til sidst opnår en genkendelsesprocent på over 99%. Iterationerne kan dog ikke bruges til at sammenligne træningstiden da datasættene ikke er af ens størrelse.



Figur 7.2: Netværket optrænet med løbende indsamling af træningseksempler. I kumulativ eftertræning er de første 10 iterationer med et datasæt på 5 af hvert bogstav, herefter 10, 20, 40 og 120 af hvert bogstav. I successiv eftertræning er de første 10 iterationer optrænet med 5 af hvert bogstav; herefter er disse udskiftet med 5 nye af hvert bogstav i 10 iterationer, osv.

Den mest praktiske metode at eftertræne på er den metode der kun bruger de 5 senest skrevne bogstaver da man med denne metode ikke behøver at opbevare større datasæt end disse $29 \cdot 5 = 145$ bogstaver.

7.6 Praktiske overvejelser

Spørgsmålet er om det netværk der er blevet konstrueret, opfylder de ydelsesmæssige krav der er blevet opstillet afsnit 3.3 om håndholdte:

- Genkendelsesprocessen skal være forholdsvis hurtig, dvs. hurtig nok til at kunne afvikles i realtid på den håndholdte.
- Netværket må ikke bruge for meget hukommelse, heller ikke midlertidigt.

For at undersøge om genkendelsesprocessen tager for lang tid skal det konkrete tidsforbrug ved genkendelsen af et bogstav undersøges. Inden bogstavet bliver sendt gennem netværket, skal det gennem en forbehandling som omfatter en centrering og en 1:2 skalering. Herefter sendes bogstavet gennem netværket hvorefter outputtet fortolkes.

Tidstesten blev udført på en bærbar computer med en Intel Celeron 600 MHz ved at indbygge en tidsmåler i de respektive implementeringer der måler tiden lige inden og efter funktionskaldene. Således bliver testen ikke afhængig af at filer skal læses og skrives eller at forbehandlingen og netværksgenkendelsen foregår i to forskellige processer. Desuden blev det maksimale hukommelsesforbrug for programmerne målt – dette tal er dog en øvre grænse da rapporteringen af hukommelsesforbruget ikke er helt pålideligt.

Resultaterne for tid er et gennemsnit af 100 test, og hukommelsen er målt som det maksimale forbrug af de 100 test.

Proces	Tid (ms)	Huk. (Mb)
Skalering	15	1,2
Centrering	1	1,1
Genkendelse	1	3,0
Samlet forbrug	17	3,0

Den samlede tid til at genkende et bogstav er således 17 ms hvilket er en del lavere end kravet på de 200 ms som nævnt i afsnit 3.3. Man skal dog være opmærksom på processorerne på håndholdte computere er en del langsommere end den processor der er testet på, hvorfor det må forventes at genkendelsesprocessen tager længere tid. Da det drejer sig om forskellige arkitekturer, er det ikke muligt at afgøre præcist hvor meget uden en direkte test på en håndholdt. Hvis man udelukkende kigger på frekvensforskellen, ville det dog tage ca. 3 gange så lang tid på f.eks. en iPAQ som har en 206 MHz StrongARM-processor, hvilket stadig med en god margin ligger under kravet.

Hukommelsesforbruget er heller ikke uden for f.eks. de 32 Mb ram som en iPAQ har – 10% af den samlede ram-mængde virker måske som meget i forhold til at genkendelsessystem skulle være aktivt det meste af tiden, men implementeringen er ikke blevet optimeret med hensyn til hukommelsesforbrug så det ville utvivlsomt være muligt at skære ned på forbruget; f.eks. kunne man gemme vægtene i variable med mindre præcision.

Med disse forbehold lever netværket op til de ydelsesmæssige krav vi har opstillet.

7.7 Evaluering af metoden

For at kunne konkludere på testene fra en anden synsvinkel evalueres den anvendte empiriske metode. Metoden, som blev belyst i kapitel 6, har dannet grundlag for den arbejdsgang der er blevet anvendt under testene. Den har været essentiel for at kunne bedømme om de teoretisk overvejelser som er foretaget i kapitel 4, har været rigtige, og det er derfor på sin plads at forholde sig kritisk til den.

De opstillede hypoteser har ved de indledende test hvor teorien blev afprøvet, været udgangspunktet for eksperimenterne, og der blev derfor gjort adskillelige overvejelser over hvordan strukturen af dem skulle være. Målet med dem var ikke at forudsige resultaterne af testene, men i stedet gennem deres formalisering at give en god struktur at arbejde efter. På denne måde har det været muligt at undersøge og behandle hvert teoretisk problem for sig og dermed sikre gennemsigtigheden i konklusionerne.

Det har dermed været muligt systematisk at indsamle tilstrækkelig viden til at konstruere et netværk med en genkendelsesprocent på 84,3% på to af testpersonerne i stedet for 54,7% som f.eks. det første resultat gav. Ydermere har det været muligt at nå helt op på 99% for en enkelt person ved at udnytte den erhvervede viden om eftertræning. Konklusionerne har altså i et vist omfang vist sig pålidelige.

De to hovedproblemer i forbindelse med anvendelsen af den empiriske metode har været at der for opdelingen af eksperimenterne ligger nogle antagelser til grund, samt at antallet af test er afgørende for hvor meget af en teori det er muligt at påvise, eller tilsvarende hvor mange forsøg det er muligt at gøre på at falsificere teorien. Vi er blevet tvunget ud i begge problemer på grund af den forholdsvis lange træningstid og begrænsede beregningskraft der har været til rådighed.

Med hensyn til antagelserne er problemet at de for en stor dels vedkommende bygger på at parametrene i netværket ikke er korrelerede hvilket giver mulighed for at teste dem uafhængigt af hinanden. Dette er næppe holdbart, og vores resultater er derfor ikke nødvendigvis pålidelige når flere parametre justeres på samme tid. Generelt kan nødvendigheden af at gøre sådanne antagelser vanskeliggøre den empiriske metode.

Med hensyn til antallet af test er problemstillingen dels at der er en vis usikkerhed på de opnåede resultater fordi forskellene mellem dem kan skyldes tilfældigheder der får dem til at havne i andre lokale minima; dels at de fleste af testene kun afprøver en meget begrænset del af de mulige værdier for parametrene. Det betyder at selv hvis det antages at forskellene mellem resultaterne aldrig skyldes tilfældigheder, så kan man kun være sikker på at konklusionerne er pålidelige i en lille del af hypotesernes område. Begge disse problemer er generelle for den empiriske metode: måleusikkerhed kan gøre resultaterne tvivlsomme, og en begrænset afprøvning af en teori gør at sikkerheden for konklusionen er tilsvarende begrænset.

Kapitel 8

Konklusion

Kunstige neurale netværk er opstået som forsimplede modeller af hjernen og kan bruges til at tilnærme funktioner som man ikke har kendskab nok til at opstille en forskrift for – når netværket indeholder flere skjulte lag, er approksimationsevnen universel.

En af de problemstillinger som neurale netværk er gode til, er tegngenkendelse. Dette er også et problem som er vigtigt i forbindelse med håndholdte computere da der ikke er plads til et tastatur. På grund af den begrænsede hardware er der dog nogle krav til hvor beregnings- og hukommelsestungt netværket må være. Genkendelsen af enkeltstående bogstaver foregår ved at brugeren skriver på et billede som bliver omdannet til en vektor. Vektoren føres gennem netværket, og resultatet herfra er igen en vektor som kan bruges til at afgøre hvilket bogstav brugeren skrev.

Afhængig af billedets størrelse kan man få et meget stort inputlag hvilket har indflydelse på udførselstiden og generaliseringsevnen. Dette kan løses ved at skalere billederne ned. En anden mulighed for forbehandling af billederne er at centrere bogstaverne; dette kan give bedre genkendelsesprocent.

For at et netværk kan approksimere en funktion skal det først trænes op på et sæt af eksempler på funktionsværdier. Træningen af netværket foregår ved at ændre på vægtene så en fejlfunktion E mindskes. Minimeringen af denne funktion foregår iterativt da problemet ikke kan løses analytisk, hvilket medfører at man kun kan være sikker på at nå et lokalt minimum. En ofte benyttet fejlfunktion er sum af kvadrater som kan udledes ud fra en maksimering af sandsynligheden for rigtige klassifikationer.

Flerlagsmodeller kan f.eks. trænes med gradientnedstigning som går i den modsatte retning af gradienten til E, eller konjugeret gradient der udnytter andenordensinformation om E til at undgå nogle af problemerne med gradientnedstigning. Fælles for de forskellige varianter er at de alle bruger de partielle afledede af fejlfunktionen E som kan findes ved tilbageførselsalgoritmen. Til gradientnedstigning kan tilføjes et ekstra led der forårsager en inertieffekt som kan øge hastigheden af konvergensen og afhjælpe problemer med lokale minima.

For at kunne opstille, udføre og fortolke test er den empiriske metode blevet undersøgt. Den empiriske metode bygger på en positivistisk opfattelse af et kun udsagn der kan efterprøves, er meningsfulde. Selvom induktionsproblemet gør at det ikke muligt endeligt at bevise generelle empiriske teorier, viser den empiriske metode sig dog alligevel at være pålidelig. Modus ponens, modus tollens og slutning til bedste forklaring har vist sig at være teknikker der er brugbare for dette projekts problemstillinger.

Efter et neuralt netværk er blevet implementeret i C++, er nogle specifikke test blevet opstillet som et kvalitativt udvalg da det ikke har været muligt at opstille et større antal test inden for den begrænsede tidsramme. Ud fra de indledende test har vi kunnet drage nogle konklusioner. Af de to aktiveringsfunktioner har *tanh* vist sig hurtigst. Konjugeret gradient har vist sig at være hurtigere iterationsmæssigt end gradientnedstigning, men til gengæld langsommere tidsmæssigt. Trænings- og inertikonstant har vist sig at have fordelagtige værdier ved 0,3 og 0,1. Centrering viste sig at have en positiv indflydelse på genkendelsesprocenten, mens en højere skalering end 1:2 viste sig at medføre en dårligere genkendelsesevne.

For at få et endeligt, pålideligt resultat har et netværk med de før fundne parametre gennemgået en fuldstændig krydstest. Resultatet var 81,7% gennemsnitlig genkendelse på ukendte tegn hvilket menes at være for lavt til en bruger. Dette kan skyldes at et træningssæt fra fem personer er for lidt, men kan også modvirkes gennem en eftertræning.

Eftertræning blev testet med et fuldt datasæt og med løbende indsamling af træningseksempler, og det viste sig at kunne forhøje genkendelsesprocenten til 99% på en person. En sådan genkendelsesprocent ser vi som værende tilfredsstillende, og vores mål er dermed opnået.

8.1 Perspektivering

8.1.1 Anvendelighed af neurale netværk

Til hvilke opgaver kan kunstige neurale netværk bruges? Et hurtigt svar på dette kunne være når et neurale netværk giver en mindre fejlprocent end andre løsninger, f.eks. mennesker. Da den menneskelige hjerne er et overordentligt kompliceret neuralt netværk, er det dog en utopisk tanke (endnu i hvert fald) at et kunstigt netværk skulle være bedre til meget komplekse opgaver som det at diagnosticere patienter på linje med en praktiserende læge eller bare det at føre almindelige samtaler – til gengæld kan et kunstigt netværk indbygges i en håndholdt computer i modsætning til et menneske. Fejlprocenten er mest interessant når der sammenlignes med andre kunstige modeller.

Neurale netværk har dog deres styrke i at de er et meget generelt løsningsværktøj som i teorien kan anvendes på alt hvad der kan opstilles et datasæt for, dvs. en mængde træningseksempler med tilhørende målværdier, uden man behøver at formulere en eksplicit teori for hvordan sammenhængen er mellem dataene og målværdierne. Det dækker over en stor mængde problemer som ellers kan være svært tilnærmelige, såsom[Sarle, 2001]:

- Ansigtsgenkendelse

- Musikkomposition
- Modellering af komplekse fysiske og kemiske processer
- Økonomisk modellering
- Selvstyrende fartøjer
- Styring af fabriksmaskineri
- Vejrforudsigelser
- Styring af robotter

Der er dog også nogle ulemper forbundet med neurale netværk. Man kan ikke ved at betragte netværket blive klogere på hvordan det når til sine konklusioner, og nødvendigheden af tilstrækkeligt mange træningseksempler er problematisk fordi det på nogle områder er svært eller direkte umuligt at skaffe nok data. De forskellige teknikker til at modvirke overtræning løser kun dette problem delvist.

Endeligt er der et problem med hensyn til konstruktionen af et neuralt netværk idet den teoretiske ramme ikke er tilstrækkelig omfattende til at en netværksmodel kan opstilles på forhånd uden eksperimenter – det er nødvendigt at inddrage empirien som værktøj. Neurale netværk kræver derfor at man sætter sig grundigt ind i teorien bag for at kunne anvende dem til at løse en given opgave hvilket begrænser den generelle anvendelighed. Den eksisterende løsning som beskrevet i afsnit 3.1 kræver derimod ikke nær så meget teoretisk indsigt.

8.1.2 Implementering på håndholdt

Hvis vores tegngenkendelsessystem skulle udgives som færdigt produkt på en håndholdt, skulle vi have konstrueret en brugergrænseflade så det blev let for brugeren at få genkendt et skrevet tegn. Vores forslag er beskrevet i afsnit 3.2. Denne skulle selvfølgelig sørge for at alt nødvendig for- og efterbehandling. Ydermere kunne man forestille sig at den blev udvidet til at kunne foretage følgende:

- Løbende eftertræning
- Fremstille valgmuligheder for brugeren
- Raffineret genkendelse ved hjælp af kontekst

8.1.3 Løbende eftertræning

Vi viste i afsnit 7.5 at eftertræning forøger genkendelsesprocenten. Desværre er en håndholdt computer ikke den bedste maskine at træne et netværk op på da optræningen er meget beregningskrævende. Hvorvidt det kan lade sig gøre at optræne netværket på en håndholdt afhænger af den enkelte håndholdte computer og må derfor afhænge af en test. Dog synes erfaringer mht. tid at antyde at dette ikke kan lade sig gøre på en håndholdt computer med dens begrænsede batteritid. I stedet kunne de tegn som brugeren indtegnede gemmes på den håndholdte, og når denne så senere kommer i kontakt med en stationær computer, kunne disse overføres og genoptræning foregå på denne.

Hvert af de tegn som vi bruger til optræning i dette projekt, fylder 2880 bit eller 360 byte, og på den håndholdte vi brugte til at indtegne disse var der en total lagerkapacitet på omkring $32 \cdot 10^6$ byte. Der er med andre ord plads til omkring 88.000 tegn eller et datasæt der er 10 gange så stort som det der er brugt til optræning i denne rapport. Det er derfor ikke noget problem at opbevare dataene.

Målværdien for hvert træningseksempel skal naturligvis findes. Dette kunne man gøre ved at holde øje med hvornår brugeren sletter det foregående tegn – hvis en bruger sletter det foregående tegn er det enten fordi der er skrevet forkert eller fordi tegnet blev genkendt forkert. For undgå nogen form for risiko fjernes tegnet helt fra datasættet, og optræning vil kun blive foretaget på de tegn som angiveligt er blevet genkendt korrekt. På denne måde kan man uden at forstyrre brugeren finde målværdierne for datasættet.

Fremstille valgmuligheder for brugeren

Netværket giver flere muligheder for forbedringer end blot tilpasselse til den enkelte bruger.

Hvis forskellen mellem de enkelte neuroners output er mindre end en tærskelværdi, dvs. hvis netværket har svært ved at genkende det skrevne tegn, kan man opstille en række valgmuligheder for brugeren der udnytter at vi har output fra 29 neuroner i stedet for blot 1. Hvis man eksempelvis finder de 3 neuroner der har de højeste outputværdier, så vil det korrekte bogstav sandsynligvis være at finde blandt disse 3, og man kan således give brugeren mulighed for at vælge et af de bogstaver der svarer til disse outputneuroner. På denne måde slipper brugeren for at slette bogstavet og skrive et nyt.

Hvis man kombinerer dette med genoptræning giver dette også mulighed for at optræne på de bogstaver som før kunne være genkendt forkert.

Raffineret genkendelse ved hjælp af kontekst

En yderligere metode til at forbedre genkendelsen er ved at tage konteksten til hjælp i form af de tegn der allerede er blevet genkendt. Hvis en bruger f.eks. har skrevet „mød“, er der stor sandsynlighed for at det næste bogstav er et „e“. Derved kan det blive lettere at afgøre tvivlstilfælde.

En mulighed ville være at sørge for at outputtet fra netværket direkte kan fortolkes som sandsynligheder efter principperne beskrevet i [Bishop, 1995]. Den samlede sandsynlighed for at det bogstav der er ved at blive genkendt, er et bestemt bogstav, kan så findes ved at gange sandsynligheden givet konteksten med sandsynligheden fra netværket.

Sandsynlighedstabeller for konteksterne kunne i en implementering opbygges løbende eller findes på forhånd til hvert enkelt sprog.

Generelt ville man ved større kontekst være i mindre tvivl om hvilket bogstav der skulle genkendes, men man bliver nødt til at begrænse størrelsen af konteksten da denne vokser eksponentielt med antallet af bogstaver. Ved ét bogstavs kontekst skal den håndholdte potentielt have en sandsynlighedstabel med $29^2 = 841$ indgange hvor man med 3 bogstavers kontekst skulle håndtere en tabel med $29^4 = 707.281$ indgange – i praksis kan dette antal dog begrænses betydeligt da visse kombinationer aldrig optræder.

Problemet med at bruge kontekstafhængig information til at forbedre gættet er at det foregår med den antagelse at konteksten faktisk har en stor betydning for hvordan det næste bogstav bliver. Ved almindelig tekstskrivning kan dette sagtens tænkes at være en god antagelse, men netop på en håndholdt er der måske større sandsynlighed for at brugeren benytter sig af forskellige forkortelser eller sammenskrivninger hvis teksterne kun er til notebrug. Desuden går det ikke at sætte sandsynligheden for usandsynlige bogstaver helt til nul da det så ikke vil være muligt at skrive disse bogstaver.

Forud for en eventuel implementering ville det altså være nødvendig at undersøge brugernes vaner. Men med et neuralt netværk er det muligt at få gavn af konteksten på en indlysende måde i modsætning til f.eks. den eksisterende løsning.

Litteratur

Shigeo Abe. *Neural Networks and Fuzzy Systems*. Kluwer Academic Publishers Group, 1995.

Peter Bickford. *Human Interface Online: Worth the Wait?* 1999. `http://developer.netscape.com/viewsource/bickford_wait.htm`.

Chrisopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

Søren Brunak og Benny Lautrup. *Neurale netværk - computere med intuition*. Munksgaard, 1988.

Steve Bush. *Brighthand - nothing but handhelds*. 2001. `http://www.brighthand.com/`.

Morten Dige. *Introduktion til videnskabsteori*. 2002. `http://www.teo.au.dk/html/rv-sem/dige/videnskabsteori.htm`.

Robert L. Harvey. *Neural Network Principles*. Prentice-Hall International, 1994.

Hans Hüttel. *Kursusnoter – Naturvidenskab, menneske og samfund*. 2001. `http://www.cs.auc.dk/~hans/Basis/NMS2001/DeEnkelteKursusgange/gang2-%noter.pdf`.

Helge Elbrønd Jensen, Steen Markvorsen, Paul Hjorth og Wolfhard Kliem. *Matematisk Analyse 1*. Institut for matematik, DTU, 2000.

Helge Kragh og Stig Andur Pedersen. *Naturvidenskabens teori*. Nyt Nordisk Forlag, 1991.

David C. Lay. *Linear Algebra and Its Applications*. Addison-Wesley, anden udgave, 2000.

Yann LeCun, Léon Bottou, Yoshua Bengio og Patrick Haffner. *Gradient-Based Learning Applied to Document Recognition*. IEEE, 1998.

Per Printz Madsen. *Neurale Netværk anvendt inden for proceskontrol*. Aalborg Universitet, 1993.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill Book CO, 1997.

Martin Møller. *Efficient Training of Feed-Forward Neural Networks*. Ph.d.-afhandling, Aarhus Universitet, 1997.

Niels K. Petersen. *Videnskabelighed og modelfrihed*. 1998. http://www.faklen.dk/dk/faklen/09/videnskabelighed.shtml.

Warren S. Sarle. *Ill-Conditioning in Neural Networks*. 1999. ftp://ftp.sas.com/pub/neural/illcond/illcond.html.

Warren S. Sarle. *Neural networks FAQ*, 2001. ftp://ftp.sas.com/pub/neural/FAQ.html.

Raymond A. Serway og Robert J. Beichner. *Physics: For Scientists and Engineers with Modern Physics, fifth edition*. Saunders College Publishing, 2000.

Jonathan Richard Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. 1994.

S. Singh og A. Amin. *Neural Network Recognition of Hand-printed Characters*. *Neural Computation and Applications*, (8):67–76, 1999.

Stephen Thornton. *Karl Popper*. 2002. http://plato.stanford.edu/entries/popper/.

Top500. *TOP500 List 11/2001*. 2001. http://www.top500.org/list/2001/11/.

Mark Willey. *Design and Implementation of a Stroke Interface Library*, 1997. http://www.etla.net/libstroke/libstroke.ps.

Bilag A

Konjugeret gradient i praksis

A.1 Bevis for at konjugerede vektorer er lineært uafhængige

Betragt $\mathbf{H}$ som værende positiv-definit, dvs. at $\mathbf{d}_i^T\mathbf{H}\mathbf{d}_i > 0$ for alle i og antag at

$$\mathbf{d}_i^T\mathbf{H}\mathbf{d}_j = 0 \tag{A.1}$$

for alle $i \neq j$.

Hvis sættet $\{\mathbf{d}_1,\mathbf{d}_2,\dots,\mathbf{d}_n\}$ skal være lineært afhængigt skal der være mindst en vektor som skal kunne skrives som en linearkombination af de andre, dvs. $\mathbf{d}_j = c_1\mathbf{d}_1 + c_2\mathbf{d}_2 + \cdots + c_{j-1}\mathbf{d}_{j-1} + c_{j+1}\mathbf{d}_{j+1} + \cdots + c_n\mathbf{d}_n$, hvor c_1, c_2, ..., c_{j-1}, c_{j+1}, ..., c_n ikke alle er nul. Hvis dette substitueres ind i (A.1) fåes

$$\mathbf{d}_i^T\mathbf{H}(c_1\mathbf{d}_1 + c_2\mathbf{d}_2 + \cdots + c_{j-1}\mathbf{d}_{j-1} + c_{j+1}\mathbf{d}_{j+1} + \cdots + c_n\mathbf{d}_n) = 0.$$

På grund af (A.1) bliver alle ledene på nær det hvor $\mathbf{d}_i$ indgår lig med nul. Derfor har vi nu

$$c_i\mathbf{d}_i^T\mathbf{H}\mathbf{d}_i = 0.$$

Dette kan kun give nul når $c_i = 0$ pga. at $\mathbf{H}$ er positiv-definit. Hvis man laver lignende omskrivning for $\mathbf{d}_1^T\mathbf{H}\mathbf{d}_j = 0$ op til $\mathbf{d}_n^T\mathbf{H}\mathbf{d}_j = 0$ får man

$$\begin{aligned}
c_1\mathbf{d}_1^T\mathbf{H}\mathbf{d}_1 &= 0\\
c_2\mathbf{d}_2^T\mathbf{H}\mathbf{d}_2 &= 0\\
&\vdots\\
c_{j-1}\mathbf{d}_{j-1}^T\mathbf{H}\mathbf{d}_{j-1} &= 0\\
c_{j+1}\mathbf{d}_{j+1}^T\mathbf{H}\mathbf{d}_{j+1} &= 0\\
&\vdots\\
c_n\mathbf{d}_n^T\mathbf{H}\mathbf{d}_n &= 0
\end{aligned}$$

og hvis disse alle skal give nul må $c_1 = c_2 = \cdots = c_{j-1} = c_{j+1} = \cdots = c_n = 0$. Dette giver altså en modstrid, og sættet $\{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_n\}$ er dermed lineært uafhængigt.

A.2 Skaleret konjugeret gradient

For at undgå at beregne $\mathbf{H}$ til bestemmelsen af α_i foreslår Møller [1997] at estimere beregningen af $\mathbf{d}_i^T\mathbf{H}\mathbf{d}_i$ som

$$\mathbf{d}_i^T\mathbf{H}\mathbf{d}_i \approx \delta_i = \mathbf{d}_i^T\left(\frac{\nabla E(\mathbf{w}_i + \sigma_i\mathbf{d}_i) - \nabla E(\mathbf{w}_i)}{\sigma_i} + \lambda_i\mathbf{d}_i\right) \tag{A.2}$$

for $0 < \sigma_i \ll 1$, hvor λ_i skal sikre at (4.15) kun har ét minimum. Hvis $\delta_i > 0$, er dette tilfældet [Møller, 1997], ellers må man forøge λ_i tilstrækkeligt. Dette kan ske ved at beregne en ny værdi [Møller, 1997]

$$\lambda_i = 2\left(\lambda_i - \frac{\delta_i}{\|\mathbf{d}_i\|^2}\right) \tag{A.3}$$

som bruges i beregningen af et nyt δ_i. Dette gentages indtil at $\delta_i > 0$.

Man kan dermed bestemme α_i som

$$\alpha_i = \frac{\mathbf{d}_i^T\mathbf{g}_i}{\delta_i}. \tag{A.4}$$

Imidlertid er (A.2) ikke altid er en god approksimation selv om $\delta_i > 0$ [Møller, 1997]. Derfor indføres

$$\Delta_i = \frac{E(\mathbf{w}_i) - E(\mathbf{w}_i + \alpha_i\mathbf{d}_i)}{E(\mathbf{w}_i) - \hat{E}(\mathbf{w}_i + \alpha_i\mathbf{d}_i)} \tag{A.5}$$

hvor $\hat{E}$ er den approksimerede, kvadratiske fejlfunktion, som et mål for hvor god approksimationen er. Hvis f.eks. $\Delta_i = 1$, er approksimationen nøjagtig. Det vises af Møller [1997] at (A.5) kan skrives om til

$$\Delta_i = \frac{2\delta_i(E(\mathbf{w}_i) - E(\mathbf{w}_i + \alpha_i\mathbf{d}_i))}{(\mathbf{d}_i^T\mathbf{g}_i)^2}. \tag{A.6}$$

hvor beregningen af $\hat{E}(\mathbf{w}_i + \alpha_i\mathbf{d}_i)$ ikke længere er nødvendig. Det ses ud fra (A.5) at hvis Δ_i er negativ så har nævneren ikke samme fortegn som tælleren. Dette er katastrofalt da vores approksimation ikke blot er unøjagtig, men en opdatering i retning $\mathbf{d}_i$ med den opnåede α_i faktisk vil forøge fejlen. Derfor må man starte forfra ved beregningen af δ_i og forsøge at approksimere med en større λ_i.

Man ønsker dog ikke at λ_i fra (A.2) skal være for stor da dette forstyrrer approksimationen af $\mathbf{H}\mathbf{d}_i$. Hvis $\Delta_i > 0{,}75$, formindsker man λ_i ved at sætte den nye $\lambda_i = \frac{1}{4}\lambda_i$, og hvis $\Delta_i < 0{,}25$ sætter man $\lambda_i = \lambda_i + \frac{\delta_i(1-\Delta_i)}{\|\mathbf{d}_i\|^2}$ for at få den fejl der beregnes på til at opføre sig kvadratisk.

En algoritme for den skalerede konjugerede gradient kan så opstilles idet den første retning sættes til $\mathbf{d}_0 = \mathbf{g}_0 = -\nabla E(\mathbf{w}_0)$, hvor $\nabla E(\mathbf{w}_i)$ kan beregnes vha. tilbageførselsalgoritmen ved at lægge de enkelte gradienter for hvert træningseksempel sammen som ved den ikke-stokastiske gradientnedstigning, og $E(\mathbf{w}_i)$ kan beregnes som i (4.5). Vægtene fra netværket til $\mathbf{w}_i$ kan blot indhentes ved at indsamle vægtene fra hver neuron for sig.

1. Konstruér en klon af netværket til beregning af $\nabla E(\mathbf{w}_i + \sigma_i\mathbf{d}_i)$.
2. Beregn $\delta_i = \mathbf{d}_i^T\left(\frac{\nabla E(\mathbf{w}_i+\sigma_i\mathbf{d}_i)-\nabla E(\mathbf{w}_i)}{\sigma_i} + \lambda_i\mathbf{d}_i\right)$.

3. Hvis $\delta_i < 0$:
   (a) Beregn $\lambda_i = 2\left(\lambda_i - \frac{\delta_i}{\|\mathbf{d}_i\|^2}\right)$.
   (b) Gå til beregningen af δ_i.
4. Beregn $\alpha_i = \frac{\mathbf{d}_i^T \mathbf{g}_i}{\delta_i}$.
5. Konstruér en klon af netværket til beregning af $E(\mathbf{w}_i + \alpha_i \mathbf{d}_i)$. Hvis fejlen kan reduceres, kan man benytte denne værdi som $E(\mathbf{w}_{i+1})$.
6. Beregn vurderingen $\Delta_i = \frac{2\delta_i(E(\mathbf{w}_i) - E(\mathbf{w}_i + \alpha_i \mathbf{d}_i))}{(\mathbf{d}_i^T \mathbf{g}_i)^2}$.
7. Hvis $\Delta_i > 0$, kan fejlen formindskes:
   (a) Opdatér vægtene i netværket som $\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i \mathbf{d}_i$.
   (b) Bestem $\beta_i = \frac{\mathbf{g}_{i+1}^T \mathbf{g}_{i+1}}{\mathbf{g}_i^T \mathbf{g}_i}$, og beregn derefter $\mathbf{d}_{i+1} = \mathbf{g}_{i+1} + \beta_i \mathbf{d}_i$.
   (c) Hvis $\Delta_i > 0{,}75$, så sæt $\lambda_{i+1} = \frac{1}{4}\lambda_i$ og brug denne i næste iteration.
8. Ellers hvis $\Delta_i < 0{,}25$, sæt $\lambda_{i+1} = \lambda_i + \frac{\delta_i(1-\Delta_i)}{||\mathbf{d}_i||^2}$ og brug denne i næste iteration.
9. Afslut hvis termineringskriteriet er opfyldt, ellers gå til det første skridt idet der startes forfra på træningssættet.

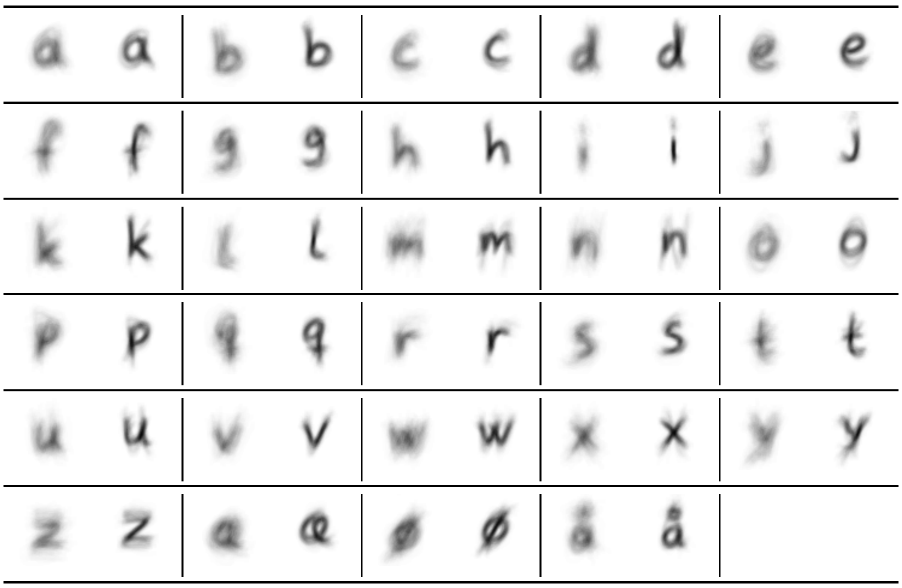
Da den kvadratiske fejlfunktion er en approksimation, kan det ske at netværket ikke har konvergeret efter n iterationer. I dette tilfælde kan algoritmen genstartes ved at sætte $\mathbf{d}_n = \mathbf{g}_n$ i stedet for at benytte (4.18) – dette svarer blot til at påbegynde træningen fra et nyt punkt. Termineringskriteriet kan vælges på samme måde som ved gradientnedstigning.

Bilag B

Datasættet

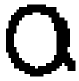
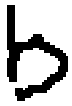
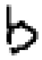
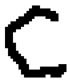
B.1 Centrering

Tabellen nedenfor indeholder alle bogstaver i vores datasæt. Hver figure er skabt ved at ligge alle 360 eksempler af det pågældende bogstav oven i hinanden vha. programmet `superimpose` beskrevet i 5.3 hvilket skaber et gennemsnitsbogstav. Figurene til venstre (de udviskede) er en sammensætning af ikke centrerede datafiler, hvor dem til højre er eksempler er der foretaget en centrering efter massemidtpunkt.



B.2 Skalering

For at danne et indtryk af hvordan netværket opfatter de skalerede bogstaver er der på tabellen nedfor vist et udsnit af datasættet i forskellige skaleringsgrader.

Ingen skalering	1:2	1:4	1:6
			
			
			

Bilag C

Startværdier for vægte

Det antages at inputværdierne har middelværdien 0, $\langle x_i \rangle = 0$, og variansen 1, $\langle x_i^2 \rangle = 1$. En neuron i det første skjulte lag har værdien a for den vægtede sum

$$a = \sum_{i=0}^{n} w_i x_i$$

hvor n er antallet af forbindelser til neuronen. Da vægtene på et utrænet netværk er uafhængige af inputværdierne, kan variansen af a findes ved [Bishop, 1995]

$$\langle a^2 \rangle = \left\langle \left(\sum_{i=0}^{n} w_i x_i \right) \left(\sum_{j=0}^{n} w_j x_j \right) \right\rangle = \sum_{i=0}^{n} \langle w_i^2 \rangle \langle x_i^2 \rangle = \sigma^2 n, \tag{C.1}$$

hvor sidste skridt følger af at $\langle x_i^2 \rangle = 1$, mens næstsidste fås fra at $\langle w_i w_j \rangle = \sigma^2$ hvis $i = j$, og $\langle w_i w_j \rangle = \langle w_i \rangle \langle w_j \rangle = 0$ hvis $i \neq j$ da de enkelte vægte ikke er korrelerede, og middelværdien for fordeling af vægtene sættes til 0.

Hvis variansen af de vægtede summer sættes til 1, er $\sigma = n^{-\frac{1}{2}}$ – hvis en anden værdi end $\langle a^2 \rangle = 1$ ønskes, skal σ ændres som $\sigma \propto n^{-\frac{1}{2}}$.