ELECTRONICS
INSTITUTE

Demokrati i Neurale Net

Stig Porsgaard

31. December 1994

Danmarks Tekniske Universitet
DK-2800 Lyngby
Danmark

Abstrakt

Der er i denne rapport undersøgt ensembler af neurale net. Specielt er der simuleret hardware implementerede analoge neurale net. Vægtene i disse hardware net, modelleres med en spændingsdifferens på to kapacitorer. I hardware vil kapacitorerne aflade, og derved opstår fænomenet vægtdrift. Der er i denne rapport undersøgt hvorledes vægtdrift påvirker et ensemble af neurale nets præstationsevne, samt om ensemblet kan forvente en forlænget levetid, når de enkelte net optrænes med ensemblets consensus som mål. Der er brugt teacher net til generering af data. Dette teachernet afbilder vektorer af reelle tal over i to klasser. Med et teachernet fås en data-kilde, der kan generere data med velkendt kompleksitet, og i rigelig mængde.

Forord

Denne rapport tjener som dokumentation for et eksamensprojekt i forbindelse med civilingeniørstudiet ved Danmarks Tekniske Universitet. Projektet blev udført på Elektronisk Institut med lektor Lars Kai Hansen som vejleder og John A. Lansner og Torsten Lehmann som medvejledere. Projektet blev påbegyndt i juli 1994 og afsluttet i december 1994.

Jeg vil gerne rette en stor tak til mine vejledere, der altid var parat til en diskussion om dette og hint.

Lyngby d. 31.12.94

Stig Anders Porsgaard
C868571

Indholdsfortegnelse

Figurfortegnelse

Tabelfortegnelse

Kapitel 1

Introduktion

Neurale netværk er et videnskabeligt område, der i de seneste år er blevet studeret med stor intensitet. Oprindelig var man interesseret i at modellere hjernen. Man ønskede at undersøge hvad det var for mekanismer der gør at hjernen er så overlegen f.eks til visuel genkendelse, at selv simple organismer kan udføre denne mønstergenkendelse bedre end de største og hurtigste super-computere. Hjernen har i det hele taget mange egenskaber der er ønskelige, i kunstig intelligens. Blandt disse egenskaber kan nævnes [HKP91, s. 1]:

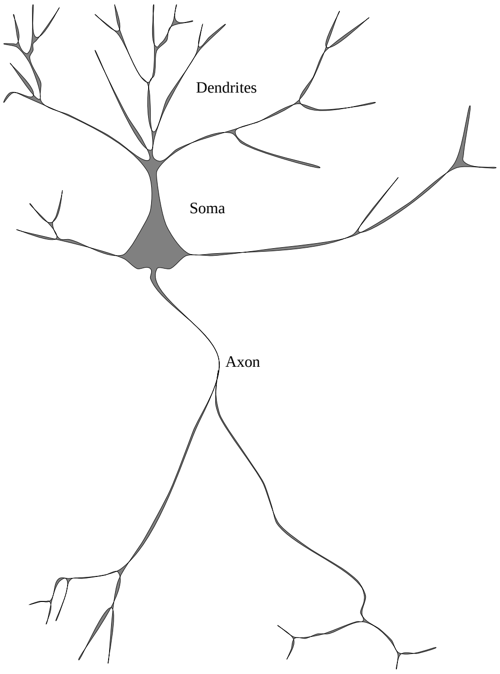
- Den er robust og fejltolerant. I hjernen dør nerveceller hver dag, uden at dette påvirker hjernens kapacitet synderligt.
- Den er fleksibel. Den tilpasser sig til sine omgivelser, ved at lære.
- Den kan behandle data, der er støjfyldte, og inkonsistente.
- Den arbejder på en yderst parallel måde.
- Den er lille, kompakt, og forbruger meget lidt energi.

Den "neurale metode" introducerer en anden måde at udføre databehandling på, end den traditionelle, baseret på von Neumans ideer. Neurale netværk er, med deres høje grad af parallelitet, og deres fejl- og støjtolerance indlysende emner til implementation i hardware.

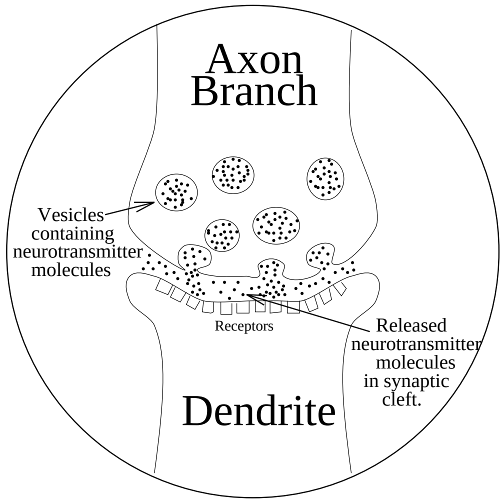
1.1 Den Biologiske Neuron

Inspirationen til, og modellen for de kunstige neurale netværk er hentet fra biologien. Det menneskelige nervesystem består af ca. 10^{10} nerveceller, kaldet neuroner.

En typisk neuron (se figur 1.1), består af cellekroppen, kaldet *soma*, der ligesom alle andre celler i kroppen indeholder de nødvendige celleorganeller for at kunne fungere. Desuden har neuronen "grene" der strækker sig ud fra soma.



FIGUR 1.1: Den biologiske neuron.



FIGUR 1.2: Den synaptiske kontakt, mellem én neurons axon (øverst), og en anden neurons dendrit (nederst).

Der er to typer grene, *dendritter* der leder elektriske signaler til soma, og *axoner* der leder elektriske signaler fra soma. Kontakten mellem to neuroner sker via en synapse. Synapsen transporterer det elektriske signal, fra én nervecelles axon, til en anden nervecelles dendrit. Selve transporten sker primært kemisk, ved hjælp af såkaldte transmitterstoffer (figur 1.1).

Når neuronen modtager signaler fra andre neuroner, skalerer den signalet, fordi nogle dendritter er bedre til at lede det elektriske signal, end andre dendritter. I selve soma sker der en summering af de elektriske input. Hvis inputtet overstiger en tærskelværdi, sender neuronen et elektrisk signal ud langs sin axon. Man siger, at den "fyrer". Der findes to slags synapser, den ene slags stimulerer neuronen til at fyre, den anden søger at forhindre at neuronen fyrer. De to slags synapser kaldes hhv exitatoriske, og inhibitoriske synapser. En exitatorisk synapse kan aldrig "konvertere", og blive inhibitorisk, og omvendt. En anden vigtig egenskab i biologiske neurale net, er at der aldrig er en synaptisk kontakt mellem egne dendritter og axoner. Der er ingen "self-coupling".

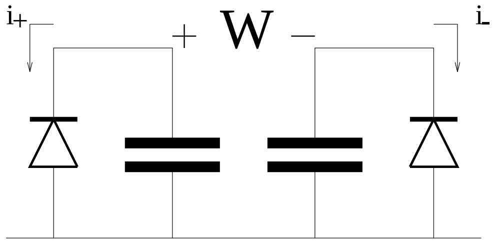
1.2 Hardware neurale netværk

Implementering af kunstige neurale net, forkortet til ANN[1], i analog hardware byder på en række interessante muligheder. Bl.a. muligheden for en meget høj

[1]ANN er den engelske forkortelse for "Artificial Neural Network", som på dansk betyder "Kunstige Neurale Netværk".

grad af parallelitet. Neurale net, er særdeles beregningstunge, og ved anvendelser hvor der kræves store net kan det være helt urealistisk at implementere nettet i software. Man kan vælge at implementere sit ANN ved hjælp af analoge VLSI[2] kredse. Denne implementationsform har en række positive egenskaber som f.eks [Leh93]:

- Lavt strømforbrug.
- Tæt pakning af synapse enheder.
- Mulighed for direkte interfacing til den virkelige verden.



FIGUR 1.3: Typisk implementation af en analog vægt, som en differens mellem spændingen på to kapacitorer. Bemærk de to lækstrømme i_- og i_+, der bevirker at vægten "driver".

Ønsker man at implementere et ANN ved hjælp af analoge VLSI kredse, løber man imidlertid ind i et problem omkring vægtene. Vægtene i et analogt ANN, kan være implementeret som en differensspænding mellem to kapacitorer, som skitseret på figur 1.3. I dette kredsløb optræder der to forstyrrende strømme, som langsomt vil aflade de to kapacitorer. Hvis disse to strømme ikke er lige store, vil den ene kapacitor aflade en smule hurtigere end den anden. Derved opstår fænomenet "vægtdrift".

[2]VLSI er den engelske forkortelse for "Very Large Scale Integration".

Kapitel 2

Formål

Formålet med dette projekt er at undersøge hardware implementerede kunstige neurale net. Specielt ønskes levetiden i ensembler af hardware implementerede kunstige neurale net undersøgt.

2.1 Problemformulering

Det ønskes belyst om ensembler af hardware implementerede analoge neurale net, kan forlænge levetiden ved hjælp af optræning på ensemblets consensus beslutning.

I hardware analoge neurale net, hvor vægtene er implementerede som differensspændinger på kapacitorer, vil præstationsevnen på det enkelte net degradere med tiden som følge af fænomenet vægtdrift. I hardware er der desuden problemer med ulineariter, støj og offset.

Der skal opstilles en model af hardware neurale net. Denne model skal bruges til at teste om man kan forlænge levetiden af et ensemble af rigtige hardware neurale net ved at bruge ensemblets consensus-beslutning som "target" for vægtopdateringsalgoritmen.

Det er meningen at denne software skal kunne kobles sammen med et rigtigt hardware ANN, for at teste at modellen udviser en rigtig opførsel.

2.2 Problemanalyse

Det er ikke helt let at konstruere et ANN i analog hardware. Ved at opstille en model i software, kan man undersøge net under forskellige vilkår. Man er ikke begrænset til en bestemt arkitektur, eller en bestemt vægtopdateringsalgoritme, og det er muligt at undersøge variable der kan være vanskelige, eller helt umulige, at måle på et rigtigt hardware net. Man har fuld kontrol over ulineariteter, støj og offset værdier. Når man kører et forsøg, hvor man kun ændrer en eller ganske få parametre, kan man være sikker på at forskellige resultater skyldes netop denne ændring. Specielt er det muligt at simulere ensembler af vilkårlig størrelse. Man kan simpelt hen tilføje et ekstra net, hvis man har brug for det.

2.2.1 Portabillitet

Det var oprindelig meningen at de simulerede net skulle kobles sammen med et "rigtigt" hardware ANN, og målinger skulle baseres på dette hybrid-system. Det var derfor krævet at simuleringerne kunne udføres på en PC, og at simuleringsprogrammet måtte kunne stå alene. Derfor blev programmeringssproget **C++** valgt. Dette programmeringssprog har en række fordele:

- **Maskin uafhængig**
  C++ er ligesom **C** et standard programmeringssprog, og der er udviklet compilere til stort set alle typer maskiner. Specielt er det valgt at bruge GNU **C++**[1]

- **Objekt orienteret**
  Da ideen med simuleringsprogrammet er at køre flere uafhængige ANN, er det oplagt at bruge et objekt-orienteret programmeringssprog, da et ANN kan betragtes som et objekt.

- **Type-check**
  C++ udfører type-check på funktions-kald. Dette betyder 2 ting.

  1. Compileren opdager hvis man kalder en funktion med forkerte parametre, eller en forkert rækkefølge af parametre. Derved undgår man tit "underlige" fejl, som let opstår i standard **C**.
  2. Compileren kan skelne mellem funktioner, på deres parameter liste. Dermed kan man bruge samme funktionsnavn til funktioner der har samme opgave, men med forskellige objekter. Et eksempel kunne være "`int load(obj_type *obj);`". Denne funktion skal bare defineres for de forskellige objekt typer man ønsker at kunne "load'e", så kan compileren finde ud af hvilken funktion man ønsker at kalde udfra objekt typen.

Desværre viste det sig at det rigtige hardware-net ikke virkede pålideligt nok, så målingerne er baseret på de simulerede net alene.

2.3 Problemafgrænsning

- Der skal opstilles en model for et hardware analogt ANN.
- Denne model skal implementeres i et program, så ensembler af hardware net kan undersøges.

[1]GNU er en international organisation, der udvikler software, som gratis kan bruges af alle

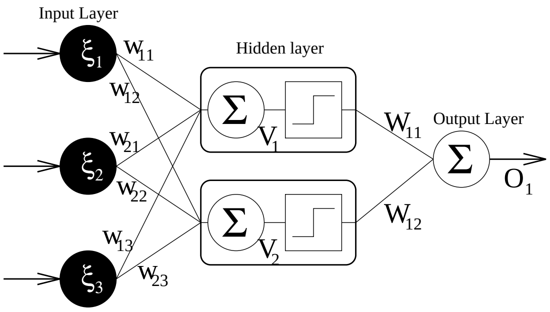
- Der skal køres forsøg der skal belyse om ensembler af hardware net, kan forvente en levetidsforlængelse ved brug af ensemblets consensus beslutning som mål for de enkelte net.

Kapitel 3

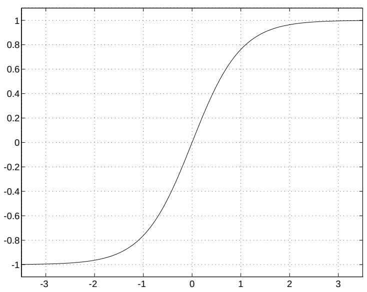
Teori

3.1 Perceptronen

Et *perceptron-netværk* er karakteriseret ved at have et input lag, et antal skjulte lag, og et output lag. Et 2-lags netværk har således et inputlag, et skjult lag, og et outputlag, idet inputlaget ikke tælles med. Enhederne i de enkelte lag kaldes



FIGUR 3.1: Eksempel på et 3:2:1 perceptron-netværk.



FIGUR 3.2: $tanh$ er en ofte anvendt aktiverings-funktion i perceptron-netværk.

neuroner. Hvert lag i netværket, er forbundet med det næste lag, med et sæt vægte. Disse vægte skalerer signalet fra neuronen i lag "i" til neuronen i lag "$i+1$". I denne rapport arbejdes der kun med 2 lags netværk. Et eksempel på et sådant net ses på figur 3.1. Input enhederne betegnes ξ_k, de skjulte enheder betegnes V_j og outputenhederne betegnes O_i. Vægtene fra inputenhederne til de skjulte enheder betegnes w_{jk} og indekseres altså fra skjulte til input. Vægtene mellem de skjulte og outputenhederne betegnes W_{ij}.[1]

[1]For fuldstændig nomenklatur-liste se evt. Appendix A.

3.2 Indlæring

Formålet med et perceptron netværk, er at netværket skal lære en relation mellem et sæt af inputmønstre, og dertil hørende outputmønstre. Man opretter derfor en database med par af inputmønstre og outputmønstre. Et sådant mønsterpar betegnes $[\xi^\mu, \zeta^\mu]$. Når nettet præsenteres for et inputmønster beregner den et outputmønster, som betegnes O^μ. Man ønsker derfor:

$$O^\mu := \zeta^\mu$$

:= betyder at ligningen ikke holder, men er et ønske

Man ønsker desuden at nettet lærer at generalisere over mønstrene, således at nye mønstre som nettet ikke har set før, bliver korrekt genkendt [HS90].

3.2.1 Kostfunktionen

Som mål for hvor godt netværket har lært input–output relationen defineres en kostfunktion, f.eks den kvadratiske:

$$E_\omega = \frac{1}{2}\sum_{\mu i}(\zeta_i^\mu - O_i^\mu)^2 \tag{3.1}$$

Man søger at lære netværket relationen mellem input og output, ved succesivt at minimere E_ω's værdi.

Hvis netværket har for mange frie parametre (dvs. vægte) i forhold til databasens størrelse og relationens kompleksitet, sker der ofte det at netværket lærer databasen for nøjagtigt, og derved mister evnen til at generalisere [SHL, WHR90]. For at undgå dette er der udviklet metoder til at begrænse antallet af frie parametre i netværket. En af disse metoder går ud på at straffe numerisk store vægte, ved at indføre et vægthenfaldsled i kostfunktionen [KH]. Kostfunktion for et 2 lags netværk, med vægthenfaldsled ser således ud:

$$E_{\omega\alpha} = \frac{1}{2}\sum_{\mu i}(\zeta_i^\mu - O_i^\mu)^2 + \frac{1}{2}\alpha_1\sum_{jk}(w_{jk})^2 + \frac{1}{2}\alpha_2\sum_{ij}(W_{ij})^2 \tag{3.2}$$

Her er α_1,og α_2 skalaparametre der bestemmer hvor stor betydning vægthenfaldet tillægges. En typisk værdi kan f.eks være $\alpha_1 = \alpha_2 = 0.01$.

3.2.2 Andre kostfunktioner

Den kvadratiske kostfunktion (3.1) er ikke den eneste mulige. En hvilken som helst differentiabel funktion af ζ_i^μ og O_I^μ, der har minimum i $\zeta_i^\mu = O_i^\mu$ kan bruges. Vægtopdatering ved gradient reglen $\Delta w_{ik} \propto -\partial E/\partial w_{ik}$ giver derefter en forskrift for hvorledes vægtene i netværket skal opdateres.

Som et eksempel på en alternativ kostfunktion, kan nævnes den *entropiske* [HKP91]:

$$E = \sum_{i\mu} \left[\frac{1}{2}(1+\zeta_i^\mu) \log \frac{1+\zeta_i^\mu}{1+O_i^\mu} + \frac{1}{2}(1-\zeta_i^\mu) \log \frac{1-\zeta_i^\mu}{1-O_i^\mu} \right] \tag{3.3}$$

Denne funktion er ligesom 3.1 altid positiv, undtagen når $\zeta_i^\mu = O_i^\mu$ for alle $\{i, \mu\}$, hvor $E = 0$. Funktionen har en fordel, i forhold til den kvadratiske kostfunktion, at hvis nettet oplæres på værdierne $\{-1, +1\}$, som det ofte er tilfældet i klassificeringsopgaver, divergerer gradienten hvis output går i mætning på det forkerte output [HKP91].

3.2.3 Forward-propagation

Forward-propagation er betegnelsen for den proces, hvor signalet der påtrykkes inputlaget vandrer frem gennem lagene i nettet, indtil værdien i outputlaget er beregnet. For et 2 lags netværk er den matematiske formulering således:

$$V_j^\mu = g_h\left(\sum_k w_{jk}\xi_k^\mu\right) \tag{3.4}$$

$$O_i^\mu = g_o\left(\sum_j W_{ij}V_j\right) \tag{3.5}$$

Som aktiverings-funktion g_h bruges ofte en ulineær funktion f.eks tanh (se fig 3.2) For g_o anvendes ofte en rent lineær ($y = x$) afbildning. Derved bliver ligningerne (3.4) og (3.5) til

$$V_j^\mu = \tanh\left(\sum_k w_{jk}\xi_k^\mu\right) \tag{3.6}$$

$$O_i^\mu = \sum_j W_{ij}V_j \tag{3.7}$$

3.2.4 Error-backpropagation

Error-backpropagation, eller blot back-propagation, er en algoritme der giver en "opskrift" på hvordan man ændrer vægtene i et perceptronnetværk, således at kostfunktionen E_ω, succesivt gøres mindre. Algoritmen bygger på, at $(\partial E/\partial w) = 0$ hvor E_ω har globalt minimum.

Betragt et 2 lags netværk med tanh som aktiverings-funktion i det skjulte lag, og et lineært output lag. Forward propagation er da beskrevet ved ligningerne (3.6) og (3.7). Indsættes disse 2 ligninger i kostfunktionen (3.1) fås:

$$E_\omega = \frac{1}{2}\sum_{\mu i}\left(\zeta_i^\mu - \sum_j W_{ij}\tanh(\sum_k w_{jk}\xi_k^\mu)\right)^2 \tag{3.8}$$

Det ses tydeligt af denne formel, at E_ω er kontinuert differentiabel i alle vægte. Man kan derfor bruge en metode kaldet *gradient descent* til at opdatere vægte efter.

Gradient descent

Vægt-tilpasning efter *gradient descent* metoden går ud på at beregne gradienten $(\partial E_\omega / \partial w)$, for en given sammensætning af vægte, og så tage et lille skridt i den modsatte retning. Er skridtet lille nok, vil det føre til en reducering af E_ω.

Bevis:

Betragt kostfunktionen, ved en ændring af vægtene. Ændres vægtene med δw, fås kostfunktionsændringen δE. Ved rækkeudvikling til anden orden fås:

$$\delta E = \nabla E \delta w + \frac{1}{2}\delta w^T \mathbf{H} \delta w + \cdots \tag{3.9}$$

lad nu

$$\delta w = -\eta \nabla E$$

da fås

$$\delta E = -\eta (\nabla E)^2 + O(\eta^2) + \cdots \tag{3.10}$$

hvor $\mathbf{H}$ er hessian matricen. Man ser at hvis η er lille nok vil (3.10) domineres af første led, der klart er negativt. Man får altså en negativ kostfunktionsændring, og dermed en mindre kostfunktionsværdi. Opdateringsreglen, for vægtene i et 2-lags netværk, bliver da som følger:

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} \tag{3.11}$$

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} \tag{3.12}$$

Man kalder parameteren η for *learning rate*. Anvendes (3.11) og (3.12) på (3.8), bliver opdateringsreglerne (se f.eks [HKP91, s.116–117]) :

For vægtene mellem det skjulte lag og output laget:

$$\begin{aligned}\Delta W_{ij} = -\eta \frac{\partial E_\omega}{\partial W_{ij}} &= \eta \sum_\mu [\zeta_i^\mu - O_i^\mu] V_j^\mu \\ &= \eta \sum_\mu \delta_i^\mu V_j^\mu \end{aligned} \tag{3.13}$$

hvor

$$\delta_i^\mu = [\zeta_i^\mu - O_i^\mu] \tag{3.14}$$

For vægtene mellem inputlaget og det skjulte lag fås

$$
\begin{aligned}
\Delta w_{jk} = -\eta \frac{\partial E_\omega}{\partial w_{jk}} &= -\eta \sum_\mu \frac{\partial E}{\partial V_j^\mu} \frac{\partial V_j^\mu}{\partial w_{jk}} \\
&= \eta \sum_{\mu i} [\zeta_i^\mu - O_i^\mu] W_{ij} \tanh' \left(\sum_k w_{jk} \xi_k^\mu \right) \xi_k^\mu \\
&= \eta \sum_{\mu i} \delta_i^\mu W_{ij} (1 - V_j^2) \xi_k^\mu \\
&= \eta \sum_\mu \delta_j^\mu \xi_k^\mu \qquad (3.15)
\end{aligned}
$$

hvor

$$
\delta_j^\mu = (1 - V_j^2) \sum_i W_{ij} \delta_i^\mu \qquad (3.16)
$$

idet man benytter $\tanh'(x) = 1 - \tanh^2(x)$

Betragt nu kostfunktionen der inkluderer et vægthenfaldsled (3.2). Opskrives gradient metoden for denne formel fås let:

$$
\begin{aligned}
\Delta W_{ij} &= -\eta \left(\frac{\partial E_\omega}{\partial W_{ij}} + \alpha_2 W_{ij} \right) \qquad (3.17) \\
\Delta w_{jk} &= -\eta \left(\frac{\partial E_\omega}{\partial w_{jk}} + \alpha_1 w_{jk} \right) \qquad (3.18)
\end{aligned}
$$

Gradient metoden er ikke særlig effektiv, og der er derfor udviklet en del andre metoder der forbedrer gradient metoden.

Gradient descent med moment

En simpel forbedring af gradient metoden, der er særlig effektiv hvis man opdaterer vægtmatricerne efter hvert eksempel, er *moment metoden*. Denne metode går ud på at tilføje gradient descent metoden en form for midling af vægtopdateringsmatricerne [HKP91, s. 123].

$$
\Delta w(t+1) = -\eta \frac{\partial E}{\partial w} + \beta \Delta w(t) \qquad (3.19)
$$

I denne formel er momentet beskrevet ved parameteren β, der kan antage værdier i intervallet $[0;1]$. Hvis udregningen af $\partial E / \partial w$ i (3.19) giver næsten samme værdi, i en lang række udregninger, ses (3.19) at konvergere til

$$
\Delta w \approx -\frac{\eta}{1-\beta} \frac{\partial E}{\partial w} \qquad (3.20)
$$

der ses at have en effektiv learning rate på $\eta/(1-\beta)$. Effekten af at indføre et momentled, er at udglatte fluktuationer, og forstærke nedstigningen i den retning der over et længere tidsrum udstikkes af gradient metoden. Forstærkningen er

$1/(1-\beta)$. Man sætter typisk β til 0.9, der empirisk har vist sig som en god værdi.

Matrix inversions metoden

Matrix inversions metoden [Bar91] kan bruges til ANN, med lineært output. Betragtes et ANN med kun én output-neuron, fås udtrykket for kostfunktionen, med vægthenfaldsled (3.2), for fastholdt V_j til:

$$E_\omega = \frac{1}{2}\sum_\mu \left(\zeta^\mu - \sum_j W_j V_j\right)^2 + \frac{1}{2}\alpha_1 \sum_{jk}(w_{jk})^2 + \frac{1}{2}\alpha_2 \sum_j (W_j)^2 \tag{3.21}$$

For at finde minimum i (3.21) løses

$$\begin{aligned} \frac{\partial E_\omega}{\partial W_k} &= 0 & (3.22) \\ &= \sum_\mu \left(\left[\zeta^\mu - \sum_{n=0}^{N} W_n V_n^\mu\right]\right) V_k^\mu - \alpha_2 W_k & (3.23) \end{aligned}$$

Bytter man rundt på summationsordenen fås:

$$\sum_\mu \zeta^\mu V_k^\mu = \sum_{n=0}^{N} W_n \left[\sum_\mu V_n^\mu V_k^\mu\right] + \alpha_2 W_k \tag{3.24}$$

Denne formel ses at være ækvivalent med $(N+1)$ sammenhørende ligninger i $(N+1)$ ubekendte :

$$\sum_\mu \zeta^\mu V_k^\mu = W_0 \left[\sum_\mu V_0^\mu V_k^\mu\right] + \cdots W_k \left[\sum_\mu V_k^\mu V_k^\mu + \alpha_2\right] + \cdots W_N \left[\sum_\mu V_N^\mu V_k^\mu\right] \tag{3.25}$$

Skrives denne formel i matrix notation fås den simple sammenhæng

$$[\mathbf{A}+\alpha_2\mathbf{I}]\mathbf{C} = \mathbf{Y} \tag{3.26}$$

hvor

$$\begin{aligned} A_{nk} &= \sum_\mu V_n^\mu V_k^\mu = A_{kn} & (3.27) \\ \mathbf{I} &= \textit{enheds matricen} & \\ Y_k &= \sum_\mu \zeta^\mu V_k^\mu & (3.28) \end{aligned}$$

hvor $\mathbf{C}$ er en vektor med elementer $\{W_0, W_1, \ldots, W_N\}$. Løses systemet med hensyn til $\mathbf{C}$ fås:

$$\begin{aligned} \mathbf{C} &= [\mathbf{A} + \alpha_2\mathbf{I}]^{-1}\mathbf{Y} & (3.29) \\ |\mathbf{A} + \alpha_2\mathbf{I}| &\neq 0 & (3.30) \end{aligned}$$

Dette system af ligninger kan altså løses, f.eks ved gauss-eliminering, så længe determinanten af $[\mathbf{A}+\alpha_2\mathbf{I}]$ ikke nærmer sig 0, svarende til et singulært ligningssystem.

Det indses let at ovenstående kan generaliseres til flere output-neuroner, der så hver især løses med matrix-inversionsmetoden.

Anden ordens metoder

En af disse metoder går ud på at betragte hvad der sker hvis man ændrer en lille smule på vægtene når man er “tæt” på et minimum for E_ω. Hvis ændringen af vægten ω_i sættes til $\delta\omega_i$, og den dertil svarende ændring af E_ω til δE_ω, fås ved rækkeudvikling af (3.8) til anden orden [GHAKW, s.4].

$$\delta E_{total} \simeq \sum_{i=1}^{N} \frac{\partial E_\omega}{\partial w_i}\delta w_i + \frac{1}{2}\sum_{i=1}^{N}\frac{\partial^2 E_\omega}{\partial w_i^2}\delta w_i^2 + \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1,i\neq k}^{N}\frac{\partial^2 E_\omega}{\partial w_i \partial w_k}\delta w_i \delta w_k \quad (3.31)$$

Antages det at $\delta E_{total} \simeq 0$, og antages det ydermere at man kan se bort fra det sidste led i (3.31), finder man opdateringsreglen til:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{jk}} \bigg/ \frac{\partial^2 E}{\partial w_{jk}^2} \quad (3.32)$$

Den tilsvarende formel, der inkluderer et vægthenfaldsled bliver:

$$\Delta w_{jk} = -\eta \left(\frac{\partial E_\omega}{\partial w_{jk}} + \alpha_1 w_{jk}\right) \bigg/ \left(\frac{\partial^2 E_\omega}{\partial w_{jk}^2} + \alpha_1\right) \quad (3.33)$$

3.3 Tilfældige tal

Når der arbejdes med ANN, kommer man i mange tilfælde ud for at skulle bruge et “tilfældigt” tal. F.eks når man skal initialisere en vægt, eller som i tilfældet med *teacher-net* (se afsnit 3.6), der bruger vektorer af normalfordelte tal som input.

3.3.1 Definition

Hvad er et “tilfældigt” tal? Skal man være stringent så kan man ikke generere tilfældige tal på en computer, idet et program med samme udgangspunkt altid vil producere samme output. Man kan definere tilfældige tal, som tal målt på en proces med tilfældigt output. F.eks måling med et Geiger-Müller rør, på et radioaktivt stof. Denne definition er dog ikke så praktisk i simuleringsøjemed.

Tilfældige tal genereret af et computerprogram skal være ukorrelerede med det program der skal bruge de tilfældige tal. Det vil sige at to forskellige tilfældigtals generatorer, skal give samme statistiske resultat på bruger-programmet.

3.3.2 Begrænsninger

Ethvert **C**-bibliotek indeholder en rutine `int rand();` der kan generere tilfældige tal i området 0...`RAND_MAX`, hvor `RAND_MAX` som regel er det største positive tal, der kan repræsenteres med datatypen `int`. I mange tilfælde kan denne rutine opfylde de krav man stiller til en generator af tilfældige tal, men i forbindelse med ANN, kan følgende begrænsninger nævnes:

1. **Lav opløselighed.** ANSI C standarden foreskriver at `RAND_MAX`, skal være mindst 32767, og et så lavt `RAND_MAX` kan let blive et problem. Skal man f.eks bruge 10^6 tilfældige tal til en simulering, får man evalueret de samme 32767 tilfældige tal ca. 30 gange hver.

2. **Sekvenskorrelation.** De fleste rutiner til generering af tilfældige tal er af typen *lineære sekventielle generatorer* (se afsnit 3.3.3). Hvis man fra en sådan generator tager vektorer af k på hinanden følgende tal, vil disse vektorer ikke udfylde det k-dimensionale rum, men vil ligge på $k-1$ planer i dette rum. Der er højest $m^{\frac{1}{k}}$ af disse planer, og hvis ikke parametrene til rutinen er valgt omhyggeligt kan der være færre [PFTV90, s. 206].

3. **Reproducérbarhed.** Simuleringsprogrammerne i dette projekt er skrevet i standard kode, således at programmerne kan compileres og udføres på en vilkårlig 32-bit computer. Der er ingen garanti for at `int rand();` giver samme sekvens af tal, på forskellige maskiner.

4. **Fordeling.** `int rand();` leverer uniformt fordelte tilfældige tal. I mange simuleringer skal man bruge tilfældige tal, fra en anden type fordeling. F.eks. simuleres støj bedst med tilfældige tal trukket fra en normal fordeling.

3.3.3 Sekvens generatorer

De allerfleste rutiner til generering af tilfældige tal, er af typen *lineære sekventielle generatorer*, der genererer en sekvens af heltal $I_1, I_2, I_3, \ldots$ hvor hvert tal ligger mellem 0 og $m-1$ (m f.eks `RAND_MAX`). Sekvensen dannes af den rekursive formel:

$$I_{j+1} = aI_j + c \pmod{m} \tag{3.34}$$

hvor a, c, m er positive heltal. Sekvensen har en endelig længde inden den gentager sig selv. Hvis konstanterne a, c, m er valgt med omhu, er længden maksimal, og alle tal mellem 0 og $m-1$ optræder da nøjagtig en gang inden sekvensen gentager sig selv. Denne generator danner udgangspunkt for en tilfældigtals-generator, der kan generere uniformt fordelte tilfældige tal.

For at undgå den sekventielle korrelation nævnt i afsnit 3.3.2 punkt 2, benytter man følgende fremgangsmåde:
Først skal rutinen initialiseres

- Der skal bruges 2 tilfældigtalsgeneratorer.
- En vektor fyldes med tal fra den ene generator.

Derefter kan rutinen kaldes:

- Med et tilfældigt tal fra den anden generator, findes et element i vektoren. Dette tal er det søgte tilfældige tal. Elementet fyldes med et nyt tilfældigt tal fra den første generator.

Denne rutine er fri for sekventiel korrelation.

3.3.4 Koden

Koden der implementerer en generator af uniformt fordelte tilfældige tal, er indført herunder. Der bruges 3 individuelle sekvensgeneratorer. To generatorer til selve det tilfældige tal, og en generator til at undgå sekventiel korrelation. Konstanterne til de 3 generatorer `{IA?, IC?, IM?}`, hvor ? erstattes med et tal $\{1,2,3\}$ findes som en linie i tabel 3.2 på side 23. Denne rutine er portabel, og giver samme sekvens af tilfældige tal, for et givet I_0, på en vilkårlig 32-bit maskine. I_0 er i denne rutine kaldet `ran_seed`.

```
REAL RandomGenerator :: rand(void)
{
  REAL   temp;
  int j;

  if ( !ran_init ){
    ix1  = (ran_seed-IC1)%M1; /*Seed the first routine*/
    if (ix1<0) ix1=-ix1;
    ix1  = (IA1*ix1+IC1)%M1;
    ix2  = ix1%M2;          /*And use it to seed the second*/
    ix1  = (IA1*ix1+IC1)%M1;
    ix3  = ix1%M3;          /*And third routines*/
    j=0;
    for (j=0; j<98; j++) {  /*Fill the table with sequential uniform*/
      ix1=(IA1*ix1+IC1)%M1;/*deviates generated by the two routines*/
      ix2=(IA2*ix2+IC2)%M2;
      r[j]=(ix1+ix2*RM2)*RM1;/*Low- and High-order pieces combined here*/
    }
    ran_init=TRUE;
  }
  ix3=(IA3*ix3+IC3)%M3;
  j=((98*ix3)/M3);/*Use the third sequence to get an integer in [0..97]*/
  if ( (j>97) || (j<0) )nrerror("rand(). This cannot happen\n");
  temp=r[j]; /*Return that table entry*/

  ix1=(IA1*ix1+IC1)%M1; /*Except when initializing, this is where we */
  ix2=(IA2*ix2+IC2)%M2; /*start. Generate next number in each sequence*/
  r[j]=(ix1+ix2*RM2)*RM1; /*and refill it*/
  return temp;
}
```

Tabel 3.1: Kode til portabel uniform tilfældigtals generator.

overflow at	im	ia	ic
2^{20}	6075	106	1283
2^{21}	7875	211	1663
2^{22}	7875	421	1663
2^{23}	11979	430	2531
	6655	936	1399
	6075	1366	1283
2^{24}	53125	171	11213
	11979	859	2531
	29282	419	6173
	14406	967	3041
2^{25}	134456	141	28411
	31104	625	6571
	14000	1541	2954
	12960	1741	2731
	21870	1291	4621
	139968	205	29573
2^{26}	81000	421	17117
	29282	1255	6173
	134456	281	28411
2^{27}	86436	1093	18257

overflow at	im	ia	ic
	259200	421	54773
	116640	1021	24631
	121500	1021	25673
2^{28}	117128	1277	24749
	312500	741	66037
	121500	2041	25673
2^{29}	120050	2311	25367
	214326	1807	45289
	244944	1597	51749
	233280	1861	49297
	175000	2661	36979
	121500	4081	25673
	145800	3661	30809
2^{30}	139968	3877	29573
	214326	3613	45289
	714025	1366	150889
2^{31}	134456	8121	28411
	243000	4561	51349
	259200	7141	54773
2^{32}	233280	9301	49297
	714025	4096	150889
2^{33}	1771875	2416	374441
2^{34}	510300	17221	107839
2^{35}	312500	36261	66037

Tabel 3.2: Tabel over konstanter, der bruges til at initialisere konstanterne i `REAL RandomGenerator :: rand(void)`.

3.3.5 Transformering

I dette projekt bruges der tilfældige tal trukket fra en normalfordeling. De fleste statistiske fordelinger, kan fremstilles ved transformering af uniformt fordelte stokastiske variable [PFTV90, 214–217]. En uniformt fordelt stokastisk variabel X, med tilhørende frekvensfordeling :

$$p(x)dx = \begin{cases} dx & 0 < x < 1 \\ 0 & \text{ellers} \end{cases} \tag{3.35}$$

hvor

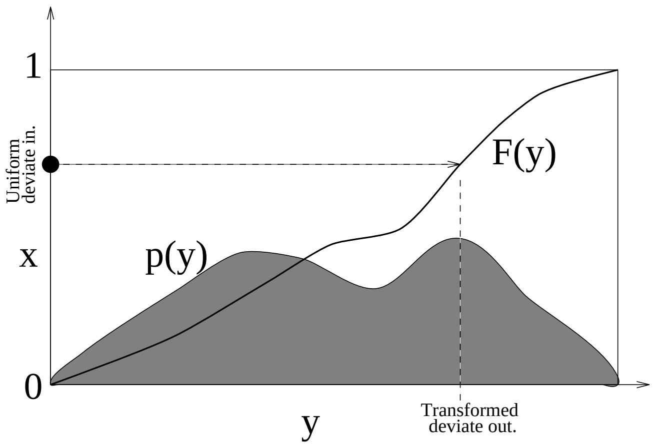
$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{3.36}$$

kan transformeres til en anden stokastisk variabel Y, ved at påtrykke en funktion $Y = y(X)$. Fordelingen for Y adlyder :

$$|p(y)dy| = |p(x)dx| \qquad \Updownarrow \tag{3.37}$$

$$p(y) = p(x)\left|\frac{dx}{dy}\right| \tag{3.38}$$

Ønsker man et Y med fordeling $p(y) = f(y)$, hvor $f(y) \geq 0$ for alle y, og $\int_{-\infty}^{\infty} f(y)dy = 1$, opnås dette ved at sammenholde formel (3.35) og (3.38). Man



FIGUR 3.3: Transformation af en uniformt fordelt stokastisk variabel, til en stokastisk variabel med en anden frekvensfordeling. På figuren er angivet $F(y) = \int_0^y p(y)dy$, som skal være kendt, og det skal være muligt at finde den omvendte funktion $F^{-1}(x)$

indser da at man skal løse differential ligningen

$$\begin{aligned} \frac{dx}{dy} &= f(y) \qquad \Updownarrow \\ x &= \int f(y)dy \\ &= F(y) \end{aligned} \tag{3.39}$$

hvor $F(y)$ er stamfunktionen til $f(y)$. Transformationen der bringer X over i Y, er nu givet ved:

$$y(x) = F^{-1}(x) \tag{3.40}$$

Om $F^{-1}(x)$ er mulig at finde, afhænger selvfølgelig af f. Transformationen kan fortolkes geometrisk. Eftersom $F(y)$ er arealet under frekvensfunktionen for Y, kan transformationen ses som: Lad x være et tilfældigt tal, trukket fra en uniform fordeling, find det y for hvilket arealet under kurven er x, og løsningen er dét y. Se figur 3.3.

Transformationsmetoden kan generaliseres til flere dimensioner. Hvis $\{X_1, X_2, \ldots\}$ er stokastiske variable, med simultan frekvensfordeling $p(x_1, x_2, \ldots)dx_1dx_2\ldots$, og $\{Y_1, Y_2, \ldots\}$ er stokastiske variable, der er fremkommet som funktioner af "x-erne", dvs $Y_1 = f_{y_1}(x_1), Y_2 = f_{y_2}(x_2)\ldots$ er den simultane frekvensfunktion:

$$p(y_1, y_2, \ldots) = p(x_1, x_2, \ldots) \left| \frac{\partial(x_1, x_2, \ldots)}{\partial(y_1, y_2, \ldots)} \right| dy_1 dy_2 \ldots \tag{3.41}$$

hvor $|\partial()/\partial()|$ er Jacobi-determinanten.

En meget vigtig anvendelse af transformationsmetoden, og brug af (3.41), er generering af normalfordelte tilfældige tal, ved hjælp af *Box Muller*'s metode. Frekvensfunktionen for en normalfordelt stokastisk variabel, med forventningsværdi

$\mu = 0$, og varians $\sigma^2 = 1$ er:

$$p(y)dy = \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} dy \tag{3.42}$$

Transformationen følger af:

$$\begin{aligned} y_1 &= \sqrt{-2\ln x_1}\cos 2\pi x_2 \\ y_2 &= \sqrt{-2\ln x_1}\sin 2\pi x_2 \end{aligned} \tag{3.43}$$

Som løses for x_1 og x_2, der giver

$$\begin{aligned} x_1 &= \exp\left[-\tfrac{1}{2}(y_1^2 + y_2^2)\right] \\ x_2 &= \tfrac{1}{2\pi}\arctan\tfrac{y_2}{y_1} \end{aligned} \tag{3.44}$$

Jacobi-determinanten udregnes derefter til [PFTV90, 216]:

$$\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = -\left[\frac{1}{\sqrt{2\pi}} e^{\frac{-y_1^2}{2}}\right]\left[\frac{1}{\sqrt{2\pi}} e^{\frac{-y_2^2}{2}}\right] \tag{3.45}$$

Eftersom resultatet af (3.45) er et produkt af to led, et led der alene afhænger af y_1, og et led der alene afhænger af y_2, ses det at både y_1, og y_2 er normalfordelte.

Ved udregningen af (3.43) kan man undgå udregningen af de to trigonometriske funktioner: I stedet for at trække x_1, x_2 fra en uniform fordeling på et kvadrat: $[0;1] \times [0;1]$, trækker man fra en uniform fordeling på enhedsskiven. Lad (v_1, v_2) være koordinaterne på dette punkt. Da fordelingen på skiven er uniform, er radius $R = v_1^2 + v_2^2$ også uniform fordelt. Radius R kan da bruges som x_1, og vinklen som vektoren (v_1, v_2) danner med v_1-aksen, kan bruges som vinklen $2\pi x_2$. Herved kan de to vinkler i (3.43) udregnes som $v_1/\sqrt{R}$, og $v_2/\sqrt{R}$. Indsættes i formel (3.43) fås:

$$\begin{aligned} y_1 &= \sqrt{-2\ln R}\,(v_1/\sqrt{R}) \\ &= v_1\sqrt{\frac{-2\ln R}{R}} \end{aligned} \tag{3.46}$$

$$\begin{aligned} y_2 &= \sqrt{-2\ln R}\,(v_2/\sqrt{R}) \\ &= v_2\sqrt{\frac{-2\ln R}{R}} \end{aligned} \tag{3.47}$$

Disse formler danner udgangspunkt for de rutiner der genererer tilfældige tal, trukket fra en normal fordeling.

3.4 Målinger på Neurale net

Som nævnt i afsnit 3.2, er formålet med optræningen af ANN, at det enkelte net "lærer" relationen mellem de input- og outputmønstre, man har i sin træningsdatabase. Man ønsker yderligere at nettene skal ekstrahere den relevante information i databasen, så de også bliver i stand til at klassificere andre mønstre [HS90]. Man ønsker altså, at nettene bliver i stand til at generalisere over træningsdatabasen. En teknik til at måle denne "evne" til at generalisere, kaldes *krydsvalidering*[2] [HS90, PC92].

3.4.1 Krydsvalidering

Ved krydsvalidering forstås at man måler fejlen på et sæt data, nettet ikke har trænet på. Man udfører altså krydsvalidering ved at dele sit datamateriale i 2 dele. En del til at træne sine ANN med, og en del til at krydsvalidere sine ANN med. Fejlen målt ved krydsvalidering, er et mål for generaliseringsevnen af de ANN, man har trænet op.

3.4.2 Lokale minima

Hvis man betragter opbygningen af ANN ser man at der er mange symmetriske konfigurationer. Ved en symmetrisk konfiguration forstås, at to net er symmetriske, hvis de for alle input giver samme output. En symmetri kan opstå på flere måder:

- **Permutation af skjulte neuroner.**
  Forestilles de skjulte neuroner som kugler, og vægtene som elastikker, kan man flytte rundt på den indbyrdes orden af kuglerne.

- **Fortegnsskift på vægte**
  I et to-lags ANN, med tanh() som aktiveringsfunktion, i det skjulte lag, kan man multiplicere de ind- og udgående vægte, fra en skjult neuron med -1.

- **Vægt multiplikation**
  Efter optræningen af netværket, vil det ofte være sådan at mange af de skjulte neuroner har så høje indgangsvægte at de er "mættede". Hermed menes at neuronerne for (næsten) alle input-mønstre i input-rummet, kun antager 2 værdier, svarende til at summen af vægtede input til neuronen antager et stor negativt, eller positivt tal.
  Heraf ses at en multiplicering af indgangsvægtene med et vilkårligt stort (> 1) tal, vil give (næsten) samme resultat på udgangen.

Ved optræningen af et ANN, sigter man på at ramme en konfiguration der svarer til det globale minimum for kostfunktionen. Der er til denne "optimale"

[2] På engelsk: "Crossvalidation"

konfiguration en hel række af symmetriske ANN, der også er "optimale". En blanding af disse konfigurationer svarer til et lokalt minimum i kostfunktionen, og optræningen af nettene, ender ofte i et sådant lokalt minimum [HS90].

3.5 Ensemble modellen

En almindelig fremgangsmåde i optræningen af ANN, er at lave en mængde net, intialisere dem forskelligt, og derefter optræne dem. Det net der efter optræningen har den laveste krydsvalideringsfejl, bliver udvalgt som kandidat til at løse det givne problem.

I [HS90] argumenteres der for at man beholder (en del af) nettene, og lader dem "stemme" om resultatet af et givet problem. En samling af net kaldes et *ensemble*, og deres fælles beslutning kaldes *consensus*. Ideen med ensembler, er at deres consensus-beslutning skulle give et bedre resultat end det bedste af nettene alene. Dette bygger bl.a. på at optræningen af ANN, er et *globalt optimerings-problem*, i et "problem-landskab" med mange *lokale minima*. Hvis optræningen af flere ANN, resulterer i at de enkelte net finder forskellige lokale minima, vil de enkelte net lave fejl på forskellige dele af input-rummet. Ved at bruge ensembler, skulle man derfor kunne "udjævne" de lokale minima.

Til analyse af ensembler, opstilles først en model for hvordan afstemningen skal udføres. Dernæst opstilles en model for ensemblers præstation, under antagelse af at ensemblets net tager fejl uafhængigt af hinanden. Endelig opstilles en model for ensemblers præstation, under hensyntagen til at nettene ikke tager fejl uafhængigt af hinanden.

3.5.1 Forudsætninger

Forudsætningen for at ensembler giver et bedre resultat end det bedste net alene, er:

- **Gode net**
  De enkelte net skal være i stand til klassificere mere end halvdelen af input-rummet korrekt. De skal have en krydsvalideringsfejl der er mindre end 0.5.

- **Uafhængige fejl**
  De enkelte net skal, til en vis grad, tage fejl på forskellige dele af input-rummet.

Den første forudsætning indses intuitivt, den anden siger at hvis nettene tager fejl på de samme input, så nytter en fælles beslutning alligevel ikke.

3.5.2 Afstemnings modeller

Når man benytter ensembler ved klassificeringsopgaver, er der to måder hvorpå nettene i et ensemble kan stemme sammen om deres consensus-beslutning.

- **Majoritet.**
  Ved majoritet, er consensus-beslutningen den klassifikation der har opnået halvdelen eller flere stemmer. Hvis der er flere end 2 mulige klassifikationer af et input, kan det ske at der ikke kan træffes en majoritets beslutning.

- **Pluralitet.**
  Ved pluralitet, er consensus-beslutningen den klassifikation der har opnået flest stemmer.

Det indses let at hvis der kun er to mulige klassifikationer, giver de to afstemningsformer samme resultat. Det ses også, at majoritet medfører pluralitet.

3.5.3 Uafhængige fejl

Antag at alle net i ensemblet kan klassificere et tilfældigt valgt input korrekt, med sandsynlighed $1 - p$ og at nettene indbyrdes er ukorrelerede med hensyn til fejl[3]. Sandsynligheden for at se præcis k fejl i et ensemble med N net er da:

$$\begin{pmatrix} N \\ k \end{pmatrix} p^k (1-p)^{N-k} \tag{3.48}$$

Antages ensemblet at bruge majoritets kriteriet for afstemningen, vil sandsynligheden for at ensemblets consensus tager fejl, være:

$$\sum_{k>N/2}^{N} \begin{pmatrix} N \\ k \end{pmatrix} p^k (1-p)^{N-k} \tag{3.49}$$

Under antagelse af at nettene laver fejl uafhængigt af hinanden, kan man vise at hvis fejlraten p opfylder $p < 0.5$, vil (3.49) være en monotont aftagende funktion med antallet af net N. Dette betyder, at hvis nettene tager fejl uafhængigt af hinanden, og hvis de hver især tager fejl for mindre end halvdelen af inputrummet, så kan man opnå en vilkårligt god consensus beslutning. Ensemblet skal blot være stort nok.

3.5.4 Input sværhedsgrad

Desværre opnår man ikke at nettene i et ensemble tager fejl uafhængigt af hinanden. Dette skyldes i overvejende grad at der er input der er sværere at lære

[3] p kaldes også for fejlraten. (På engelsk "error-rate").

end andre [HS90]. Der opstilles nu en model der tager højde for et inputmønsters "sværhedsgrad".

Lad ξ være et inputmønster, og lad $\theta(\xi)$ være defineret som den brøkdel af net i et uendeligt stort ensemble der ikke klassificerer ξ korrekt. Da kan man beregne sandsynligheden for at et ensemble af N net, med majoritetsafstemning, tager fejl på netop dette mønster:

$$\sum_{k>N/2}^{N} \binom{N}{k} \theta^k (1-\theta)^{N-k} \tag{3.50}$$

Man kan nu definere en frekvensfunktion μ af "sværheden"[4], af inputmønstrene i inputrummet. Med $\mu(\theta)$ menes, at der er mellem θ og $\theta + d\theta$ net der ikke kan klassificere brøkdelen $\mu(\theta)$ af inputtene i input-rummet. Med denne funktion kan man forudsige sandsynligheden for at et ensemble med N net tager fejl på et tilfældigt input til:

$$\int \sum_{k>N/2}^{N} \binom{N}{k} \theta^k (1-\theta)^{N-k} \mu(\theta) d\theta \tag{3.51}$$

Ved at måle på et ensemble med N net, kan man estimere en diskret tilnærmelse til fordelingen af μ. Dette gøres efter følgende forskrift:

1. Dan et stort antal (N_{input}) inputmønstre.
2. Dan en vektor $\boldsymbol{f}$ af tal, med længde $N+1$. Vektorens tal indekseres $0, \ldots, N$, og nulstilles.
3. Tag et inputmønster, og undersøg hvor mange af de N net der fejlklassificerer mønsteret. Dette antal kaldes N_{fejl}. Bemærk $N_{\text{fejl}} \in [0, \ldots, N]$.
4. Læg 1 til $\boldsymbol{f}$ med indeks N_{fejl}.
5. Fortsæt med trin 3 indtil alle de trin 1 dannede inputmønstre er undersøgt.
6. Normér $\boldsymbol{f}$ med antallet af inputmønstre (N_{input}).
7. Der er nu dannet et estimat af $\mu(\theta)$, og vektoren $\boldsymbol{f}$ indeholder de $N+1$ diskrete værdier, for $\theta = 0, 1/N, 2/N, \ldots, (N-1)/N, 1$.

Eksempler på sådanne målinger kan ses i kapitel I-A.

[4] På engelsk "difficulty distribution".

3.6 Teacher Net

Når man skal teste egenskaber ved ANN, og ensembler af ANN's, melder der sig ofte det problem, at man mangler data. Data opsamlet som målinger på en fysisk proces, som f.eks. solplet serien, lider ofte af en række uheldige egenskaber, der gør dem uegnede:

1. **Dimensionering.**
   Kompleksiteten af den underliggende proces er ofte ukendt. Dermed er det vanskeligt at dimensionere sit ANN, og man må ty til metoder der optimerer dimensionerne på sit ANN. Dette er ofte en vanskelig opgave i sig selv, og ikke nødvendigvis den opgave man ønsker at teste sit ANN for.

2. **Mangel på data.**
   Hvis den fysiske proces er tilstrækkelig kompleks kan det være yderst vanskeligt at skaffe data nok, til både optræning, og krydsvalidering.

3. **Målestøj.**
   På enhver måling af fysiske processer, er der indlejret en del støj. Støj, er ofte vanskelig at estimere, og varierer måske endda i styrke under forsøget.

4. **Hastighed.**
   Hvis man vil undersøge egenskaber ved ANN der ikke afhænger af topografien af nettet, vil man søge at bruge så små net som muligt, idet disse (selvfølgelig) kan evalueres hurtigere end store. Her melder sig da problemet med at skaffe data der er af en sådan kompleksitet at de lader sig modellere med et lille net.

Man ønsker sig altså en generator, der kan generere data med en veldefineret kompleksitet, uden støj, og i rigelig mængde. En sådan generator kan være et andet ANN, og man kalder dette net, for et *teacher net*.

3.6.1 Fremstilling

Der er i dette projekt brugt teacher net, der klassificerer et tilfældigt input i to klasser. Det er tilstræbt at teacher nettet klassificerer ligeligt i de to klasser, dvs. at den ene klasse ikke er overrepræsenteret, i outputtet fra teacher nettet, i forhold til den anden. Fremgangsmåden ved fremstillingen af teacher nettet er:

1. Bestem topografien, og dermed kompleksiteten af teacher nettet.

2. Tildel vægtene i teacher nettet, tilfældige værdier.

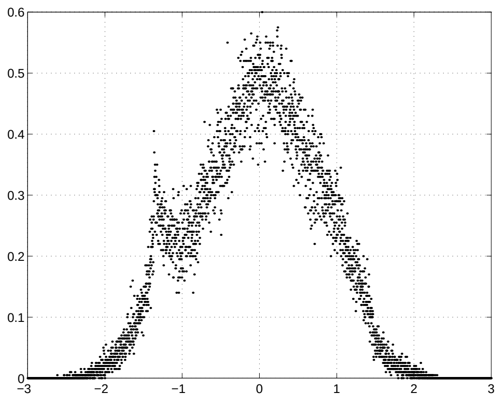
3. Bestem, over et stort antal inputs, hvor skævt teacher nettet klassificerer, idet outputtet klassificeres efter fortegn. Justér tærskelværdien i outputlaget i teacher nettet, således at nettet klassificerer ligeligt.

Man kan let bestemme tærskelværdien i outputlaget med følgende fremgangsmåde:

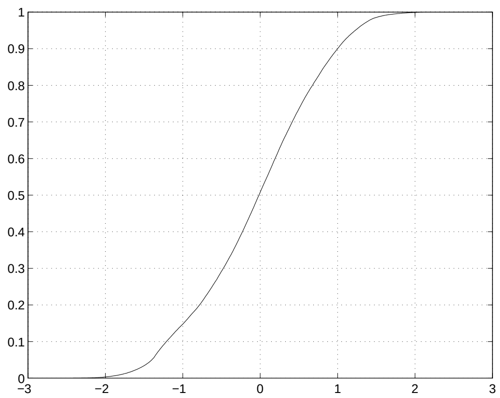
1. Sæt tærskelværdien i outputlaget til 0.
2. Kør et stort antal inputs igennem nettet. Undlad at klassificere outputtet.
3. Medianen i outputtet er den ønskede tærskelværdi, med modsat fortegn.

3.6.2 Test af fremstillingsmetode

Der er kørt en række forsøg der skal belyse, hvorledes den ovenstående fremgangsmåde virker. Et af disse forsøg viser hvordan outputtet fra et Teacher net er fordelt inden klassifikationen (se figur 3.4).



FIGUR 3.4: Frekvensfunktionsplot af outputtet fra et teachernet inden det "klippes". Der er kørt 10,000 tilfældige input eksempler igennem teachernettet, og frekvensværdien af output neuronen er målt. Teacherens er af typen 8:5:1, med uniformt fordelte vægte i intervallet $[-1.0; +1.0]$



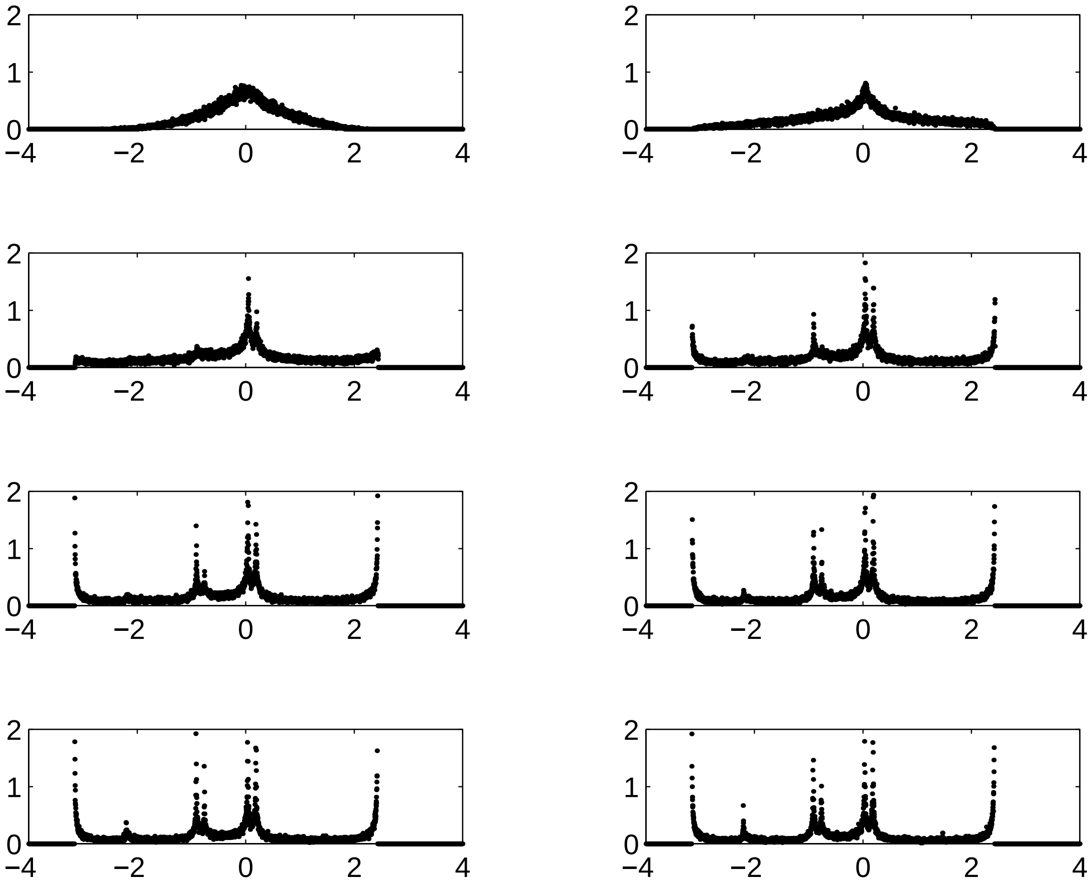
FIGUR 3.5: Fordelingsplot af outputtet fra et teachernet inden det "klippes". Tærskelværdien i outputlaget er justeret, således at der klassificeres ligeligt. Dette ses af at kurven løber igennem punktet $(0, 0.5)$.

Betragter man frekvensplottet på figur 3.4, ser man det typiske forløb af et frekvensplot for et teacher net. Der er en relativt stor vægt af output omkring output-neuron værdien 0.0, og derfra falder frekvensværdierne. Det er også typisk at der er visse output-værdier, der har en meget høj frekvens (toppene i plottet). Toppene skyldes at aktiveringsfunktionen i teachernettets skjulte lag er sat til tanh(). Når input til en sådan skjult neuron er tilstrækkeligt højt, går tanh() i mætning, og derved opstår der to toppe i frekvensplottet. Om man på udseendet af frekvensplottet kan afgøre om et teacher net er "let at lære", har det desværre ikke kunnet lade sig gøre at finde ud af.

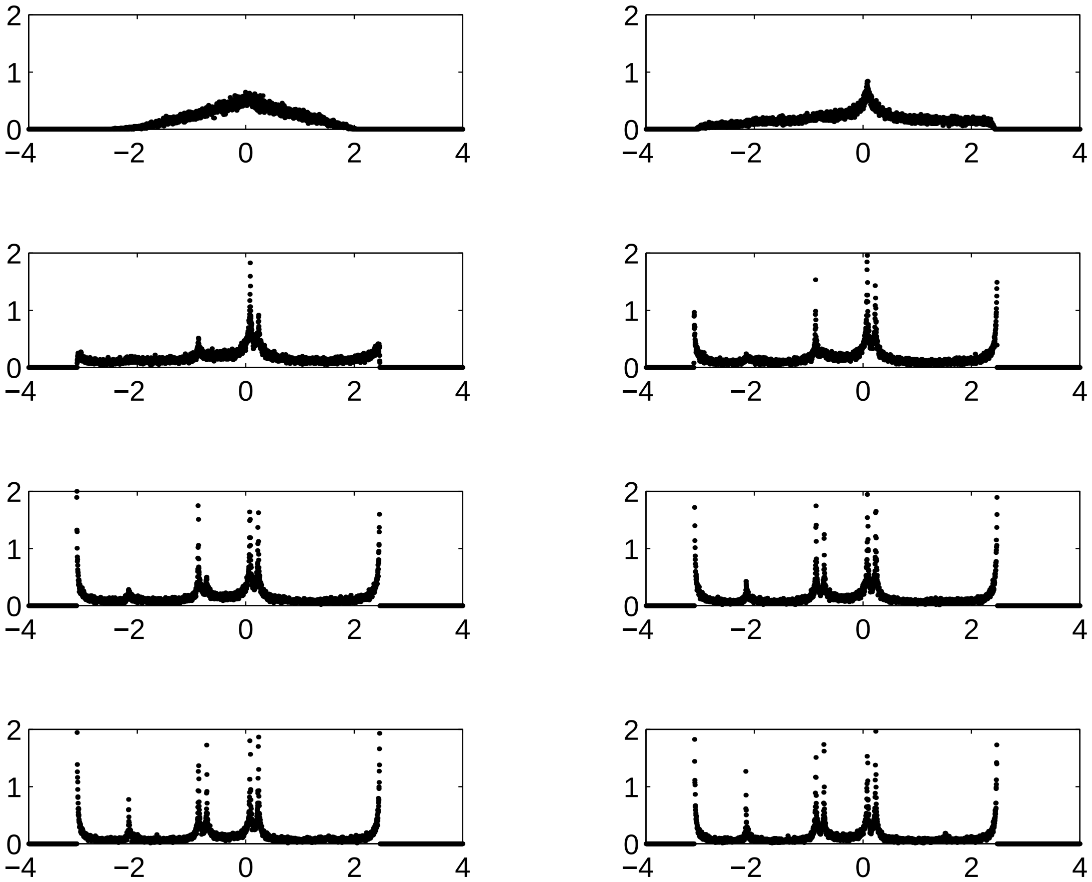
Man kan betragte samspillet mellem teachernettet og studenternettene som en “lærer-elev” situation. Eleven stiller læreren en række spørgsmål, som læreren svarer “ja” eller “nej” til. Eleven sammenligner derefter lærerens svar med egne svar. Udfra denne sammenligning kan eleven rette lidt på sin “forståelse”. Efter at have rettet på sin forståelse, har eleven nu et nyt bud på svarene, disse sammenlignes så med de rigtige fra læreren, og eleven kan så igen rette lidt på sin forståelse, og så fremdeles indtil man stopper eleven. Denne leg med spørgsmål og “ja-nej svar”, har de fleste prøvet. Det er velkendt fra denne leg at nogen spørgsmål er bedre at stille end andre, og hvis man derfor trækker analogien en smule, kan man stille sig det spørgsmål, om der er én type input til teachernettet der afslører mere end andre typer. I dette projekt kan man vælge input trukket fra en normal fordeling, eller fra en uniform fordeling. Spørgsmålet er altså om input trukket fra en normalfordeling, er “bedre” end input trukket fra en uniform fordeling. Til at belyse dette spørgsmål er der kørt et forsøg, og resultatet kan ses på figur 3.6 side 33 og 3.7 side 34.

I forsøget er teachernettet, der er af typen $5:3:1$, fasthold. Nettet er initialiseret med tilfældige uniformt fordelte tal i intervallet $[-2,+2]$, derefter er det testet med input genereret fra en normalfordeling (figur 3.6), og med input genereret fra en uniform fordeling (figur 3.7). Der er til hver del-forsøg testet på $100,000$ input. De to sæt af plot, udviser overordentlig stor overensstemmelse. Helt typisk er det at teachernettet har massen af output koncentreret omkring 0.0, for input der har lave værdier, og at frekvensplottet ikke væsentligt skifter udseende, når blot inputværdierne er over en vis størrelse. Dette skyldes, at teacheren har tanh() som aktiveringsfunktion på de skjulte neuroner. Betragter man teacheren som en funktion der afbilder inputrummet i to klasser, indser man at inputrummet må bestå af regioner der afbildes i klassifikationen +1, ved siden af regioner der afbildes i klassifikationen -1. For studenter nettene gælder det altså om at bestemme hvor disse grænser går, og det må derfor være fornuftigt valg at stille flest spørgsmål tæt på grænserne. Netop pga. at teacheren har tanh() som aktiveringsfunktion på de skjulte neuroner, må man forvente at grænserne mellem regioner i inputrummet der klassificeres forskelligt, må ligge tættest for lave værdier af input. Et fornuftigt valg af inputtype, må efter denne argumentation, være af normalfordelingstypen, med forventningsværdi omkring 0.0, og en spredning σ der lige akkurat bringer teacheren i mætning. Betragtes figur 3.6 ses dette at ske omkring spredningen $1 \leq \sigma \leq 1\frac{1}{4}$.

Stort set alle forsøg med teachernet i dette projekt er udført med normalfordelt input, med forventningsværdi $\mu = 0$ og spredning $\sigma = 1$.



FIGUR 3.6: Teacher nettet er præsenteret for 100,000 tilfældige input. De 8 plot viser frekvensfordelingen af output neuronen. Inputtet er normalfordelt med forventningsværdi $\mu = 0$. Hvert plot viser resultatet af forskellige værdier af spredningen σ. Nummeres plottene fra venstre mod højre, og oppefra og ned, er spredningen hhv $\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 1\frac{1}{4}, 1\frac{1}{2}, 1\frac{3}{4}, 2\}$



FIGUR 3.7: Teacher nettet er præsenteret for 100,000 tilfældige input. De 8 plot viser frekvensfordelingen af output neuronen. Inputtet er uniformt fordelt med symmetriske ($\pm$) grænser. Hvert plot viser resultatet af forskellige værdier af grænsen. Nummeres plottene fra venstre mod højre, og oppefra og ned, er grænserne hhv $\{\frac{1}{2}, 1, 1\frac{1}{2}, 2, 2\frac{1}{2}, 3, 3\frac{1}{2}, 4\}$

Kapitel 4

Hardware modellen

I dette kapitel opstilles den model for hardware implementerede ANN, der ligger til grund for simuleringerne i afsnit 5.3.

4.1 Modellen

Det er valgt at modellere et hardware ANN implementeret i VLSI med CMOS teknologi (se [LL93]). En implementering af et sådant hardware ANN kan findes i [Leh94].

Netværket bygges op af to typer elementer, synapser og neuroner, som vist på figur 4.1. Denne opdeling gør det let at tilpasse størrelsen af netværket til den specifikke opgave.

Formuleres *forward-propagation* i vektor-matrix notation fås:

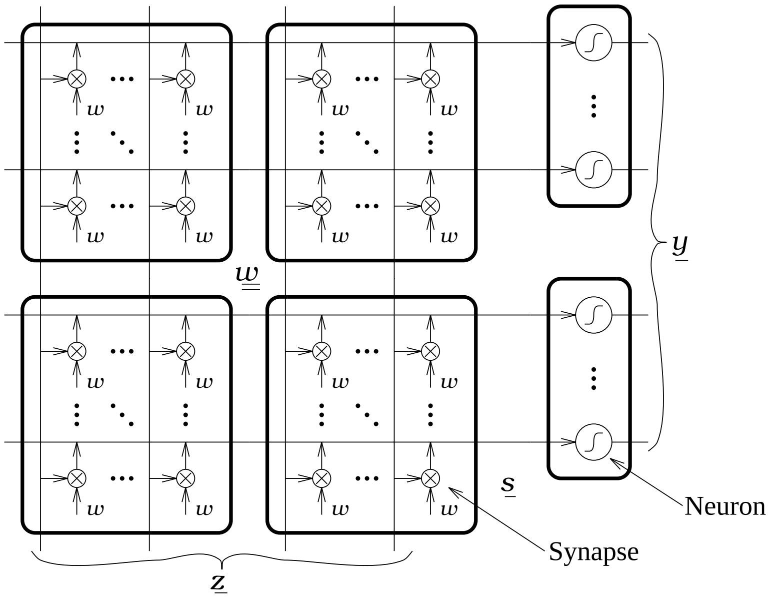
$$\mathbf{V}^l = \mathbf{g}(\mathbf{w}^l \mathbf{V}^{l-1}) \tag{4.1}$$

hvor $\mathbf{V}$ er en vektor af neuron output, med $\mathbf{V}^0 = \boldsymbol{\xi}$ som input til netværket. $\mathbf{g}$ er en vektor af sigmoide aktiveringsfunktioner (f.eks tanh()), og $\mathbf{w}^l$ er vægtmatricen mellem de to lag neuroner.

Af denne formulering opbygges et ANN med de to typer elementer: Synapser, der foretager multiplikationen $\mathbf{w}^l \mathbf{V}^{l-1}$, og neuroner der påtrykker synapseproduktet en aktiveringsfuntion $\mathbf{g}(\cdot)$. Derved opnås at netværket kan implementeres med kendte komponenter. Synapsen implementeres som en matrix-vektor-multiplikator med spænding-input og strøm-output. Neuronen implementeres med aktiveringsfunktionen tanh() og med strøm-input og spænding-output [LL93, Leh94].

Denne implementering har den fordel at output fra neuroner (i form af spænding) let distribueres til en række synapser, hvis disse har en meget høj indgangsimpedans. Summen af synapse-multiplikationen dannes ved at forbinde udgangene (strøm) fra de enkelte multiplikatorer til indgangen af den efterfølgende neuron.

Pga. af process-variationer i fabrikationen af VLSI-chips kan man ikke producere nøjagtig ens komponenter, og modellen af hardware ANN, må tage højde



FIGUR 4.1: Model af hardware implementeret ANN. Input til nettet er betegnet med $\underline{z}$. Synapse-produktet betegnet med $\underline{s}$. Outputtet $\underline{y}$. Vægtmatricen er betegnet $\underline{\underline{w}}$. Opbygningen af ANN i elementer af synapser, og elementer af neuroner, gør det let at skalere størrelsen af nettet. Man kan derfor let skrædersy nettet til den specifikke opgave.

for disse variationer. I beskrivelsen af modellen for et hardware ANN, indsættes derfor konstanter, der beskriver den enkelte komponents karakteristika. Disse konstanter varierer fra komponent til komponent, og denne variation beskrives bedst med en normalfordeling [LL]. For den enkelte komponent er konstanten dog fast, og i simuleringen vælges denne type konstanter som et tilfældigt tal, fra en normalfordeling, med passende parametre.

I disse konstanter indbygges implicit konverteringen mellem enheder.

Den fremkomne model er et resultat af eget arbejde, samtaler, og diskussion med de to medvejledere John Lansner, og Torsten Lehmann. Hvor modellen overvejende er baseret på samtale, er dette markeret med henvisningen til [LL]

4.1.1 Hardware neuronen

Hvis den j'te hardwareneuron i lag l betegnes V_j^l, er formlen for beregning af neuronværdien følgende:

$$V_j^l = k_1 \tanh\left(k_2 \sum\right) + V_{j,\text{offset}}^l \tag{4.2}$$

hvor $\sum$ er den vægtede sum af input til neuronen. Denne sum beregnes af synapse-delen i et hardware ANN.

I formelen (4.2) indgår der to neuronspecifikke konstanter k_1 og k_2. Disse konstanter er faste for den enkelte neuron, og modelleres med en normalfordeling, med parametre $(\mu_{k_1}, \sigma_{k_1})$ og $(\mu_{k_2}, \sigma_{k_2})$.

I formlen indgår også neuron offset værdien $V^l_{j,\text{offset}}$, der ligeledes er konstant, og specifik, for den enkelte neuron. Neuron offset værdien modelleres ved normalfordeling med parametre $(\mu_{V_{\text{off}}}, \sigma_{V_{\text{off}}})$

4.1.2 Hardware synapsen

Synapsen i et hardware ANN, foretager multiplikationen af spændingsdifferencen på indgangen og en lagret vægt. Denne vægt, implementeres som differensen af spændingerne på to kapacitorer (se figur 4.2). Denne implementering er simpel og fylder ikke ret meget i et VLSI design[Leh94]. Implementeringsformen har dog den ulempe, at vægten driver, pga. lækstrøm fra kapacitorerne.

Modellen for en hardware vægt er givet ved [LL]:

$$w = w_{\text{ideal}} + w_{\text{offset}} + w_{\text{noise}} + \Delta t \cdot w_{\text{droop}} \tag{4.3}$$

I modellen optræder følgende størrelser:

w_{ideal} Den værdi vægten ideelt skulle have. I hardware er der både en øvre og nedre grænse for den værdi en vægt kan antage. I simuleringen testes det om en vægt er nået uden for den tilladte værdi-område, og ifald dette er tilfældet, bliver vægten justeret så den ligger i det tilladte værdi-område.

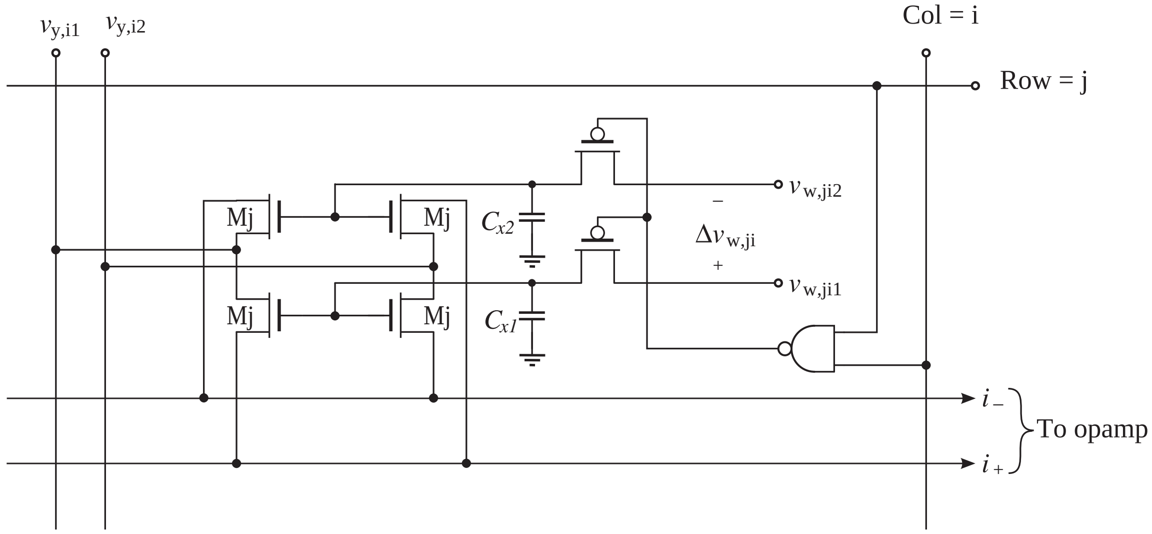
w_{offset} Offset værdi. Sættes en hardware vægt til værdien 0, vil synapsen alligevel bidrage med en lille værdi. Denne værdi opfattes som en offset værdi på vægten.

w_{noise} Bruges en vægt til to multiplikationer lige efter hinanden, med samme multiplikanter, vil der være en lille forskel på de to resultater. Denne forskel modelleres med "støj". Støjen på den enkelte vægt varierer fra gang til gang, og simuleres ved at lægge et tilfældigt tal trukket fra en normalfordeling, til w_{ideal}, inden man bruger den resulterende vægt.

w_{droop} På grund af repræsentationsformen af vægtene i hardware, som differensspændinger på to kapacitorer, opstår fænomenet vægtdrift. Driften er specifik, og konstant for den enkelte vægt, og proportional med tiden siden sidste opdatering. Vægtdriften simuleres ved at vælge et tilfældigt tal fra en normalfordeling (med parametre $(\mu_{\text{droop}}, \sigma_{\text{droop}})$), for hver vægt.

4.1.3 Optræning

Ved optræningen af et hardware netværk, er der typisk tale om ren gradient descent. Denne algoritme er som nævnt i afsnit 3.2.4 ikke særlig effektiv, og der



FIGUR 4.2: Model af hardware synapse. Synapsen foretager multiplikationen $(v_{w,ji1} - v_{w,ji2}) \cdot (v_{y,i1} - v_{y,i2})$, og adderer produktet, i form af en differensstrøm, til linierne i_- og i_+. Se [LL93].

er udviklet et lang række forbedringer til denne metode. De fleste af disse metoder er ikke særlig velegnede til implementering i hardware [Leh94], og derfor er det valgt, kun at implementere "vægthenfald", og moment[1]. I de net der simuleres, er aktiveringsfunktionen for både det skjulte, og output-laget sat til tanh().

Gradient descent til hardware ANN

Man skal ved vægtopdatering med gradient descent metoden bruge den afledte af aktiveringsfunktionen. I hardware er det ikke så let at beregne afledte, men da aktiveringsfunktionen er tanh(), kan man benytte

$$\frac{d}{dx} k_1 \tanh(k_2 x) = k_1 k_2 (1 - \tanh^2(k_2 x)) \tag{4.4}$$

Derved kan man beregne den afledte som en funktion af kvadratet på en neuronværdi. Dette leder frem til følgende formulering af gradient descent metoden for hardware ANN:

$$\Delta w^l_{jk} = \eta^l \delta^l_j V^{l-1}_k \tag{4.5}$$

hvor

$$\delta^l_j = k_1 k_2 (k_3 - (V^l_j)^2) \sum_i w^l_{ij} \delta^{l+1}_i \tag{4.6}$$

med

$$\delta^L_i = \zeta_i - O_i$$

[1]Disse ekstra metoder, kan selvfølgelig slås fra ved at sætte de tilhørende parametre til 0.

Hvor L er maksimalt lagindeks
Man bemærker at det er muligt at stille "skridtlængden" η specielt for hvert lag i et hardware ANN. Der introduceres herved en ekstra parameter, k_3, der implementeres som en værdi for hver neuron. Denne værdi er sædvanligvis højere end den ideelle værdi på 1.0. Dette skyldes at man vil være helt sikker på at man ikke får vendt fortegnet på gradienten, således at man i stedet for gradient-nedstigning, får gradient-opstigning.

I simuleringen sættes k_3 for hver vægt til et tilfældigt tal valgt fra en normalfordeling med parametre $(\mu_{k_3}, \sigma_{k_3})$.

Kapitel 5

Resultater

5.1 Solplet serien

Solpletter er mørke pletter på solen. Disse er ofte større end jorden, og blev observeret første gang omkring 1610, kort efter opfindelsen af teleskopet. Antallet af solpletter er registreret siden 1700. På figur 5.1 ses den normerede solplet serie[1], siden 1700, markeret med fuldt optrukket linie. Solpletter varierer i antal, og der er gennemsnitlig 11 år mellem maksima af solpletaktivitet. Den underliggende mekanisme der producerer disse solpletter er ikke kendt til fulde, men man ved at solpletaktiviteten hænger sammen med bl.a. den magnetiske aktivitet i solen. Solpletter optræder sædvanligvis i par, svarende til magnetiske dipoler. Solplet serien er karakteriseret ved at være et *ikke lineært dynamisk system* målt i den *virkelige verden*.

Solplet serien bruges som et standard test eksempel for neurale netværk, af TDNN[2] typen. Hvis $x(t)$ betegner solplet-aktiviteten målt i år t, foretager et solplet-TDNN afbildningen:

$$\hat{x}(t) = F_w([x(t-1), \ldots, x(t-N)]) \tag{5.1}$$

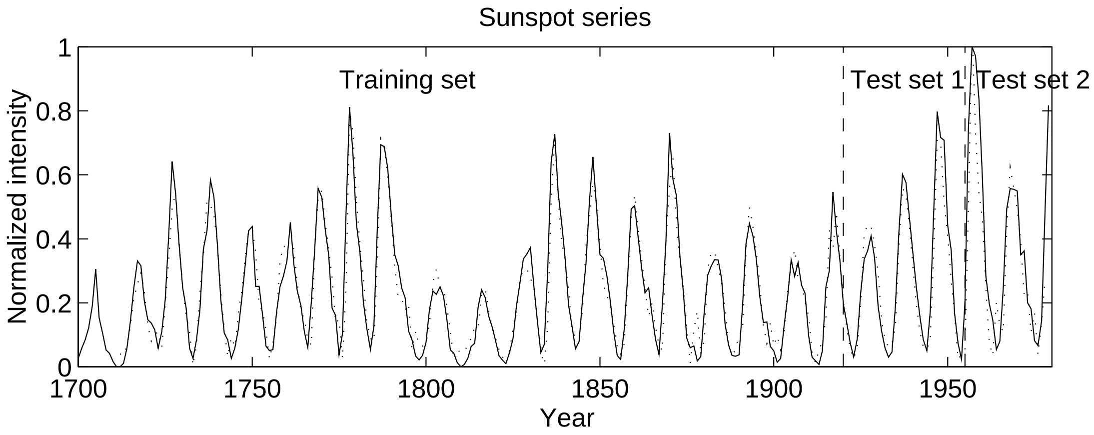
Hvor N er antallet af inputenheder, $\hat{x}(t)$ er nettets forudsigelse af solpletaktiviteten i år t og F_w er den ulineære funktion nettet implementerer.

Solplet serien er begrænset til 280 målinger (årene 1700–1979) og der opstår derfor nemt "overfitting" [WHR90]. Når man betragter de målte data, ses det at solplet aktiviteten skifter karakter omkring år 1950–1960. Perioden efter ca. 1955 har en betydelig større varians i målingerne end perioden før. Efter normering af hele serien fås $\sigma^2_{1700-1955} = 0.1855$ mod $\sigma^2_{1955-1979} = 0.2957$ en forskel på næsten 60%.

For at kunne sammenligne resultater med f.eks Weigend et. al. (se [WHR90] eller [SHL]), opdeles solplet–serien i tre perioder 1700–1920, 1921–1955 og 1956–1979 [WHR90]. Perioden 1700–1920 bruges som trænings-serie. Årrækken 1921–

[1] På engelsk: "Sunspot series".

[2] TDNN er engelsk for "Tapped Delay Neural Network"



FIGUR 5.1: Den normerede sunspot serie. Målinger på solpletaktiviteten er markeret med fuldt optrukket linie, den punkterede linie er et 12-8-1 ANN's forsøg på at forudsige aktiviteten. På figuren er der indtegnet lodrette stiplede linier, ved årene 1920, og 1955. Derved ses opdelingen af datasættet i et træningssæt, og to testsæt.

1955 kaldes "test serie 1" og er det primære krydsvalideringssæt [HS90]. Hermed menes at man med dette sæt data kan måle hvor godt nettet generaliserer. Når nettet begynder at overtræne vil det vise sig ved en dårligere forudsigelsesevne på netop dette sæt. Den sidste periode, 1956–1979 kaldes "test serie 2". Dette sæt kan også bruges som krydsvalideringssæt, men man skal huske at det ikke ligner resten af solpletserien så godt som "test serie 1".

Det benyttede netværk havde en 12:8:1 arkitektur, dvs. 12 input enheder, 8 skjulte neuroner, og 1 output neuron. Der er tærskelværdi[3] på neuronerne.

I det meste af den litteratur der behandler solpletserien, normerer man kostfunktionen, med et led, der er proportionalt med længden af den del af serien, kostfunktionsværdien udregnes for. Desuden normeres med variansen for hele solpletserien.

Således er de registrerede kostfunktionsværdier givet ved:

$$E_c = \frac{1}{\sigma^2 N} \sum_t (\hat{x}(t) - x(t))^2 \tag{5.2}$$

hvor σ^2 er variansen på hele solpletserien, og N er længden på det stykke af solpletserien kostfunktionen udregnes for.

5.1.1 Individuel træning

Der er skrevet et sæt *MatLab* rutiner, der kan optræne et netværk på "sunspot-data'ene".

Hjertet i *MatLab* rutinerne er disse tre dele:

[3] På engelsk: Threshold

1. **forward.m** : Signalet propageres frem igennem netværket. Se tabel 5.1.

2. **backprop.m** : Vægtopdatering udregnes. For at bruge anden ordens opdatering sættes **secOrder** = 1, svarende til logisk *true*. For at bruge almindelig gradient descent sættes **secOrder** = 0. Se tabel 5.2.

3. **cost.m** : Kostfunktionsværdien udregnes. Den værdi der udregnes og registreres, inkluderer ikke et vægthenfaldsled. Vægthenfaldsleddet optræder kun i opdateringsberegningerne, og har til hensigt at forhindre overtræning. Se tabel 5.3.

```
function [o,v] = forward(Inp, Whi, Woh)
% Forward propagate the Inp through the 2 layer network defined
%   by the weightmatrices Whi (Hidden <- Input)
%   and Woh (Output <- Hidden)
% Note that forward(..) implements a theshold on the output layer by
% setting v(1,:)=1
% the squashing function for the Hidden layer is   : y= tanh(x)
% the squashing function for the Output layer is   : y= x

% The functions return Output values in o
%   and                Hidden values in v

v = tanh(Whi*Inp);
v(1,:) = ones(1,size(v,2));
o = Woh*v;
```

Tabel 5.1: MatLab funktionen *forward.m*

5.1.2 Træningsresultater

Resultaterne bygger på optræningen af et 12:8:1 net, med tærskelværdier på det skjulte lag og outputlaget. Nettene er optrænet i 250 "runder". I hver "runde" har nettet set hele træningssættet én gang. En sådan runde kaldes en epoke[4].

Den første træningsserie

I den første træningsserie er nettene optrænet på solplet serien med ren *gradient descent* opdatering. I denne træningsserie er vægthenfaldsleddene $\alpha_1 = \alpha_2 = 0$, dvs. ingen vægthenfald. Der er trænet 25 net, med forskellig initiel vægtfordeling.

Betragter man figuren 5.2 ses det at nettene endnu ikke er trænet i bund. Man ser også at præstationsvariansen stiger fra træningssættet over test sæt 1, til test sæt 2.

[4]På engelsk: "epoch".

```
v2 = 1 - v.^2;
dWhi = (v2 .* (Woh' * (z0-y0))) * x0t;
if secOrder,
   ddWhi = ((Woh.^2)'*onesNi) .* (v2.^2 * x2t);
   ddWhi(1,:)=onesWhi;
   delta = (dWhi + alfa1*Whi)./(ddWhi+alfa1);
else
   delta = dWhi + alfa1*Whi;
end,
eta = 2*eta; % allow for dynamic eta.
for j=1:patience,
   Wix = Whi -  eta * delta;
   [z0 v]=forward(x0,Wix,Woh);
   Etra = p0 * cost(z0,y0);
   if Etra<e0,
      Whi = Wix;
      ETA=j;
      A = v*v' +eyealf2;
      a = v*y0';
      Woh = (pinv(A)*a)';
      break;
   else
      eta = eta/2;
   end,
end,

[z0 v]=forward(x0,Whi,Woh);
e0 = p0*cost(z0,y0);
e1 = p1*cost(forward(x1,Whi,Woh),y1);
e2 = p2*cost(forward(x2,Whi,Woh),y2);
```

Tabel 5.2: MatLab script *backprop.m*

Den anden træningsserie

I den næste serie (se figur 5.3), er der først kørt 10 epoker med *gradient descent*, hvorefter der skiftes til anden ordens opdatering. Vægthenfaldsleddet er her $\alpha_1 = \alpha_2 = 0$.

På grafen for *test sæt 1* ses det at den normerede fejl for alle datasæt falder kraftigt i de første epoker, hvorefter den stiger svagt, for de to testsæt. Dette fænomen kaldes overtræning[5]. Overtræningen skyldes at nettet begynder at lære støjen i træningssættet, og bliver derfor dårligere til at forudsige solpletaktiviteten i krydsvalideringssættet.

Den tredie træningsserie

Den tredie serie (se figur 5.4), er kørt på samme måde som den anden serie, dog med vægthenfald. Vægthenfaldet er sat til $\alpha_1 = \alpha_2 = 0.1$.

Ved dette niveau af vægthenfald undgår man tydeligvis overtræning. Gennemsnitskurven for *test sæt 1* følger pænt kurven for træningsfejlen. Man ser dog også at der er betydelig større varians på *test sæt 1* end på fejlen på træningssættet.

[5] På engelsk "overfitting".

```
function e0 = cost(o,zeta)
%
% calculate cost output relative to desired
% output zeta. Using sum of squared deviations.

e1 = o - zeta;
e0 = e1*e1';
```

Tabel 5.3: MatLab funktionen *cost.m*

Grafen viser også at anden ordens opdatering, langt hurtigere opnår en rimelig præstation, end almindelig *gradient descent*.

Optimalt vægthenfald

De foregående kørsler viser at anden ordens opdatering med vægthenfald opnår de bedste resultater. Dette leder til spørgsmålet om hvad der er den optimale værdi for vægthenfalds-parametrene α_1 og α_2. Der er derfor opsat en forsøgsrække, der skal afsløre dette forhold.

Vægthenfaldsleddene, varieres uafhængigt, i et værdiområde der med sikkerhed indeholder det optimale. Desuden køres der et antal runder, for at sikre et minimum af statistisk sikkerhed.

Vægthenfaldsleddene variereres over 4 dekader, fra 1.00 til 0.0001, med 4 målepunkter i hver dekade. I første dekade er der således målt i α-punkterne $\{1.00, 0.75, 0.50, 0.25\}$. I anden dekade er der målt i $\{0.100, 0.075, 0.050, 0.025\}$, og så fremdeles. For hvert sæt af (α_1, α_2) koordinater er der kørt 250 epoker, og derefter er den normerede fejl på de tre datasæt blevet registreret. Ialt blev der kørt 6 runder over alle α_1/α_2 punkter. For at præsentere det indsamlede datamateriale på en overskuelig form, er det blevet behandlet som følger

- Der er opsamlet data over 6 runder, og det viste resultat er en gennemsnitsværdi.
- Resultatet er vist i en 3D graf, hvor de uafhængige variable er α_1 og α_2, og den afhængige er den normaliserede fejl.
- For yderligere at "blødgøre" 3D grafen, er der blevet interpoleret. Dette gøres ved at værdien i et punkt sættes til middelværdien af punktet, og de omkringliggende 8 punkter. Randen af kurven interpoleres, ved at dublere randen.
- Når grafen "plottes", dannes der "felter", hvor hjørnerne i et sådant felt, er punkter i grafen. Hvert felt farves med "flydende" farver. Således at hvis et felt ligger skråt, vil den øverste del være farvet med en anden farve end den nederste del. Dette gør at fornemmelsen af felter i grafen forsvinder.

- Under 3D grafen plottes et konturplot, således at man bedre kan identificere "toppe og dale".

Resultatet heraf ses på de tre figurer 5.5, 5.6 og 5.7.

Af figur 5.5 ser man af konturlinierne at for små værdier af vægthenfald, er præstationsevnen på træningssættet mest afhængig af størrelsen af α_2. Denne parameter styrer vægthenfaldet i laget mellem skjulte neuroner og output. Man ser også, som ventet, at mindre vægthenfald betyder bedre præstation for træningsdatasættet.

Af figur 5.6 ses det at for moderate værdier af vægthenfald, er præstationen mest følsom for ændringer i α_2 parameteren (Ses af at kontur-linierne overvejende ligger parallelt med α_1-aksen). Man ser også at der er et område i midten af grafen, hvor præstationen på dette testsæt er betydeligt bedre end udenfor. Minimum for fladen er bestem til $(\alpha_1, \alpha_2) = (0.0075, 0.01)$. I figuren er minimum tydeliggjort med en lodret stiplet linie igennem punktet, og et kryds i $\alpha_1 - \alpha_2$-planet. Kostfunktionsværdierne for de tre datasæt er i dette punkt bestemt til:

$$\begin{aligned} E_{c,\text{træningssæt}} &= 0.098 \pm 0.004 \\ E_{c,\text{test sæt 1}} &= 0.104 \pm 0.010 \\ E_{c,\text{test sæt 2}} &= 0.381 \pm 0.053 \end{aligned} \tag{5.3}$$

Af figur 5.7 ser man at præstationsevnen for test sæt 2 har bedst af høje vægthenfaldsled.

I [SHL] rapporteres om beskårede[6] TDNN der opnår en præstation:

$$\begin{aligned} E^*_{c,\text{træningssæt}} &= 0.090 \pm 0.001 \\ E^*_{c,\text{test sæt 1}} &= 0.082 \pm 0.007 \\ E^*_{c,\text{test sæt 2}} &= 0.35 \pm 0.05 \end{aligned} \tag{5.4}$$

I [WHR90] rapporteres resultater i samme størrelsesorden:

$$\begin{aligned} E^\star_{c,\text{træningssæt}} &= 0.082 \\ E^\star_{c,\text{test sæt 1}} &= 0.086 \\ E^\star_{c,\text{test sæt 2}} &= 0.35 \end{aligned} \tag{5.5}$$

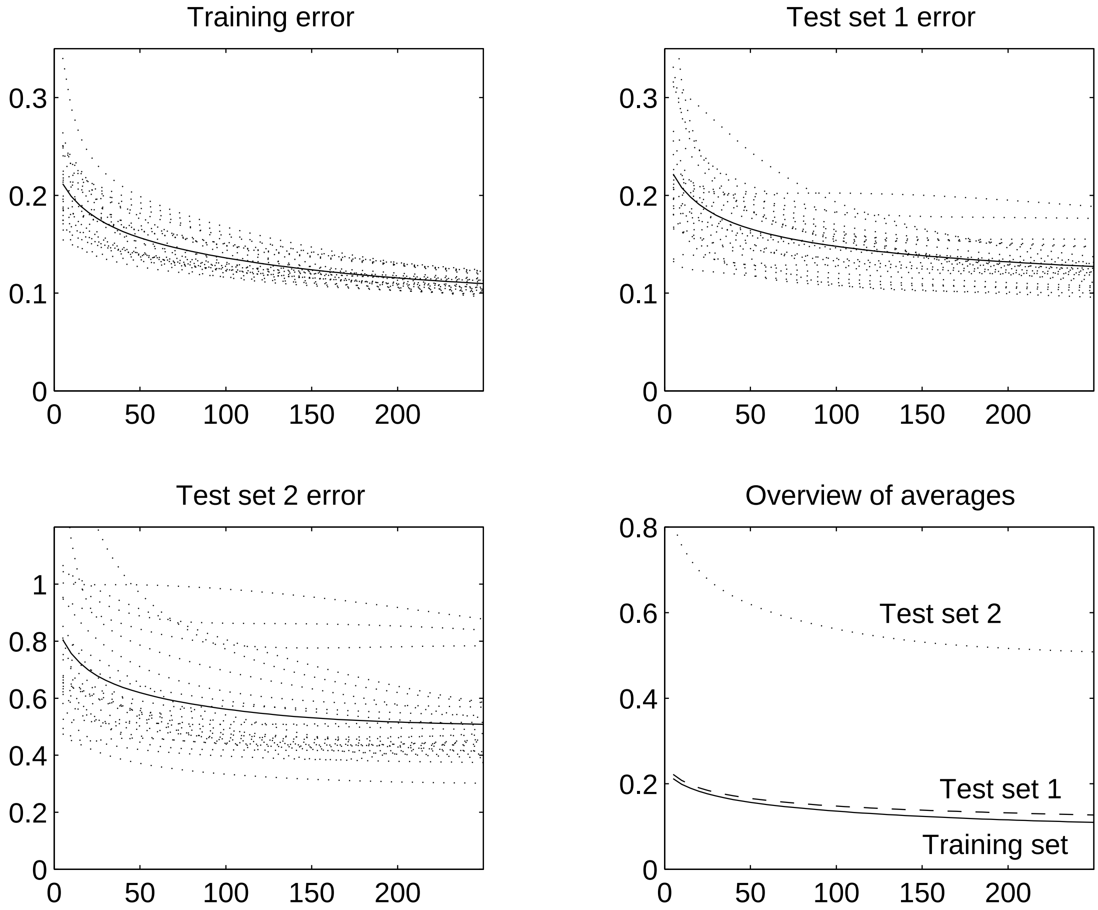
At der ikke er opnået helt så gode resultater, kan skyldes at der kun er trænet i 250 epoker. Hvis der havde været trænet over et større antal epoker kunne kostfunktionsværdien nok være bragt lidt længere ned. Resultaterne er dog absolut af samme størrelsesorden.

[6] På engelsk: "Prune".

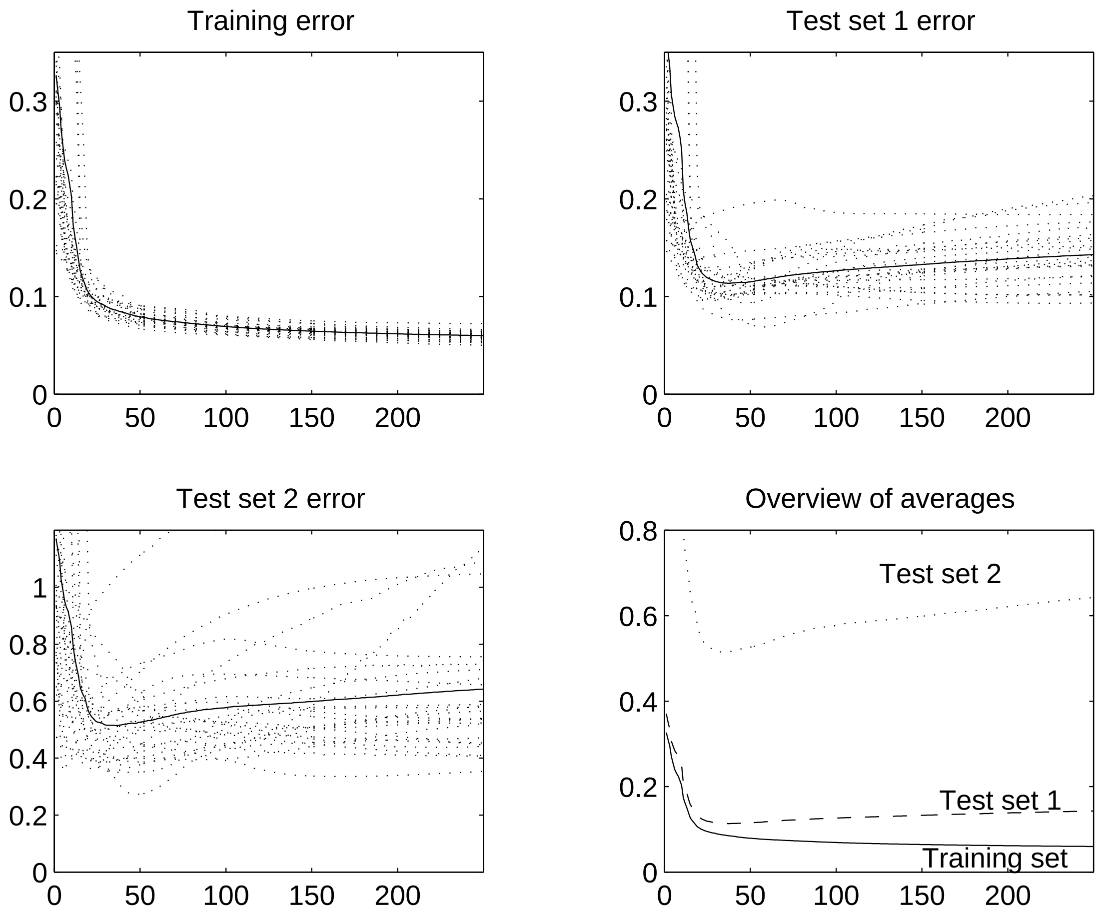
5.1.3 Delkonklusion

De foregående kørsler viser at anden ordens opdatering med vægthenfald giver de bedste resultater. Der er kørt en testserie der skulle belyse den optimale værdi for vægthenfaldet. Denne værdi er, på grundlag af grafen på figur 5.6, bestemt til $(\alpha_1, \alpha_2) = (0.0075, 0.01)$. Kostfunktionsværdierne, for de tre datasæt, er for dette vægthenfald og 250 epokers optræning bestemt over 6 runder til:

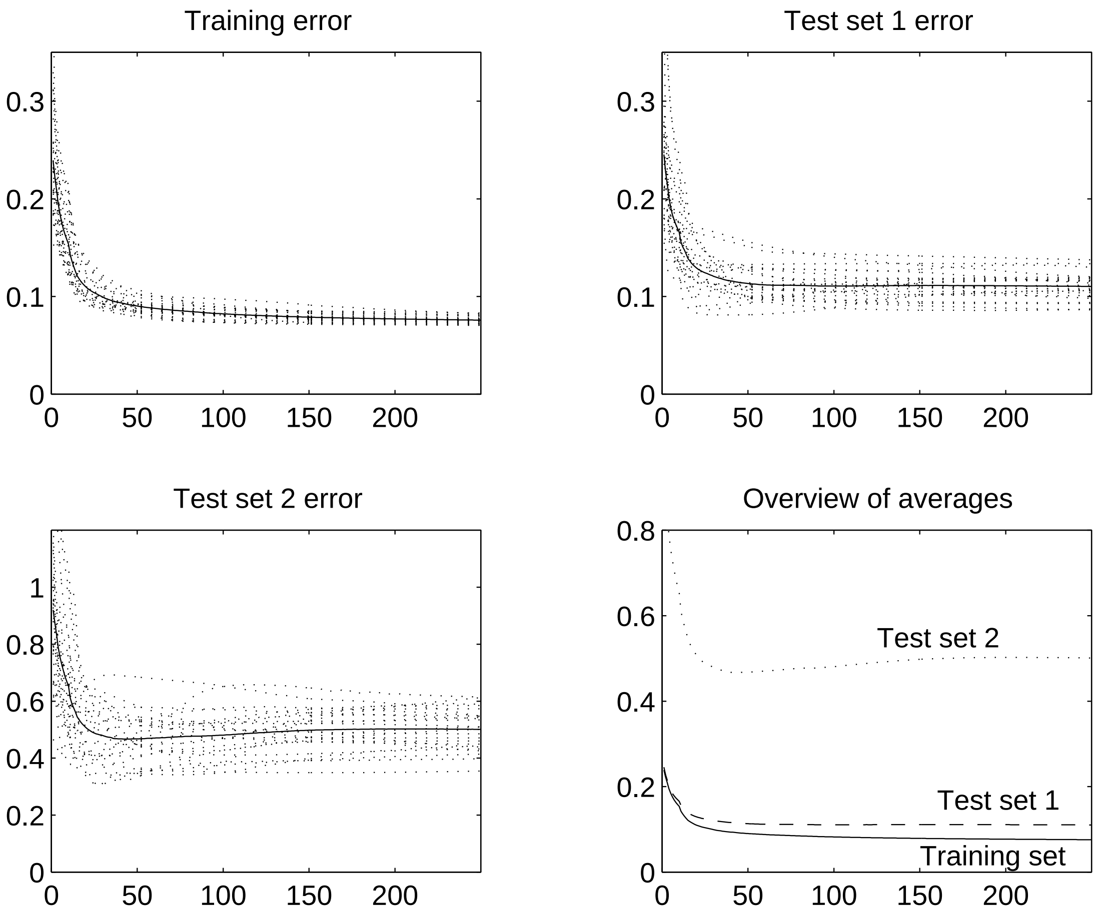
$$\begin{aligned} E_{c,\text{træningssæt}} &= 0.098 \pm 0.004 \\ E_{c,\text{test sæt 1}} &= 0.104 \pm 0.010 \\ E_{c,\text{test sæt 2}} &= 0.381 \pm 0.053 \end{aligned} \tag{5.6}$$



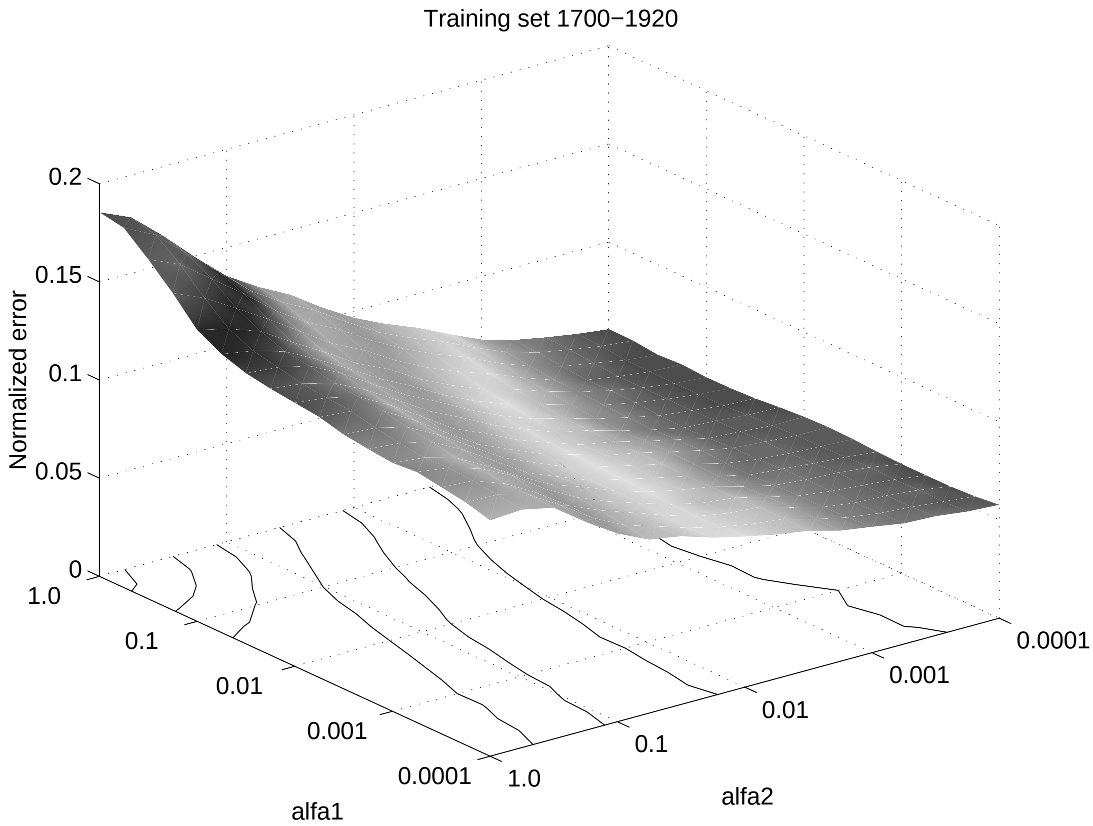
FIGUR 5.2: Sunspot trænet med ren *gradient descent*. Vægthenfaldsleddene $\alpha_1 = \alpha_2 = 0$, der er altså ikke vægthenfald. Der er trænet 25 net, med forskellig initiel vægtfordeling. De enkelte nets præstationer er afbildet med punkter (dots), og gennemsnittet er fuldt optrukket.Der er afbildet den normerede fejl som funktion af hvor mange gange nettet har set træningsættet.



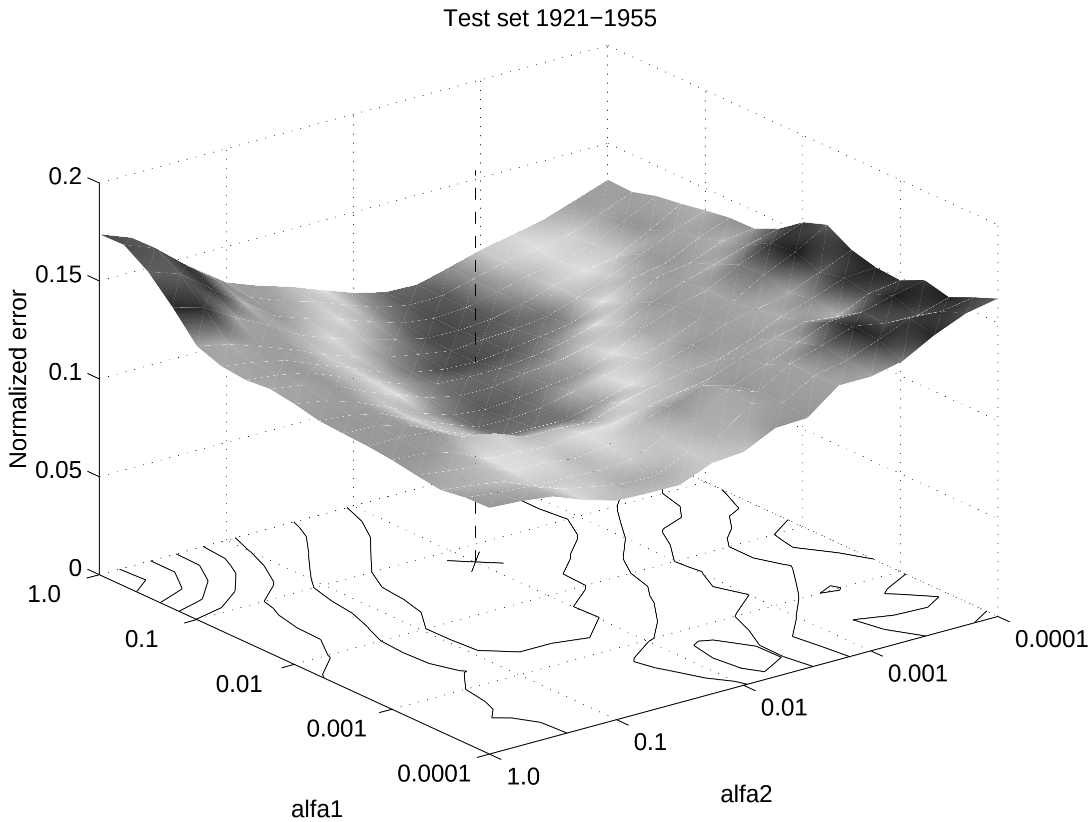
FIGUR 5.3: Sunspot trænet med anden ordens opdatering uden vægthenfald. Der er trænet 25 net, med forskellig initiel vægtfordeling. De enkelte nets præstationer er afbildet med punkter (dots), og gennemsnittet er fuldt optrukket. Der er afbildet den normerede fejl som funktion af hvor mange gange nettet har set træningsættet.(Epoker)



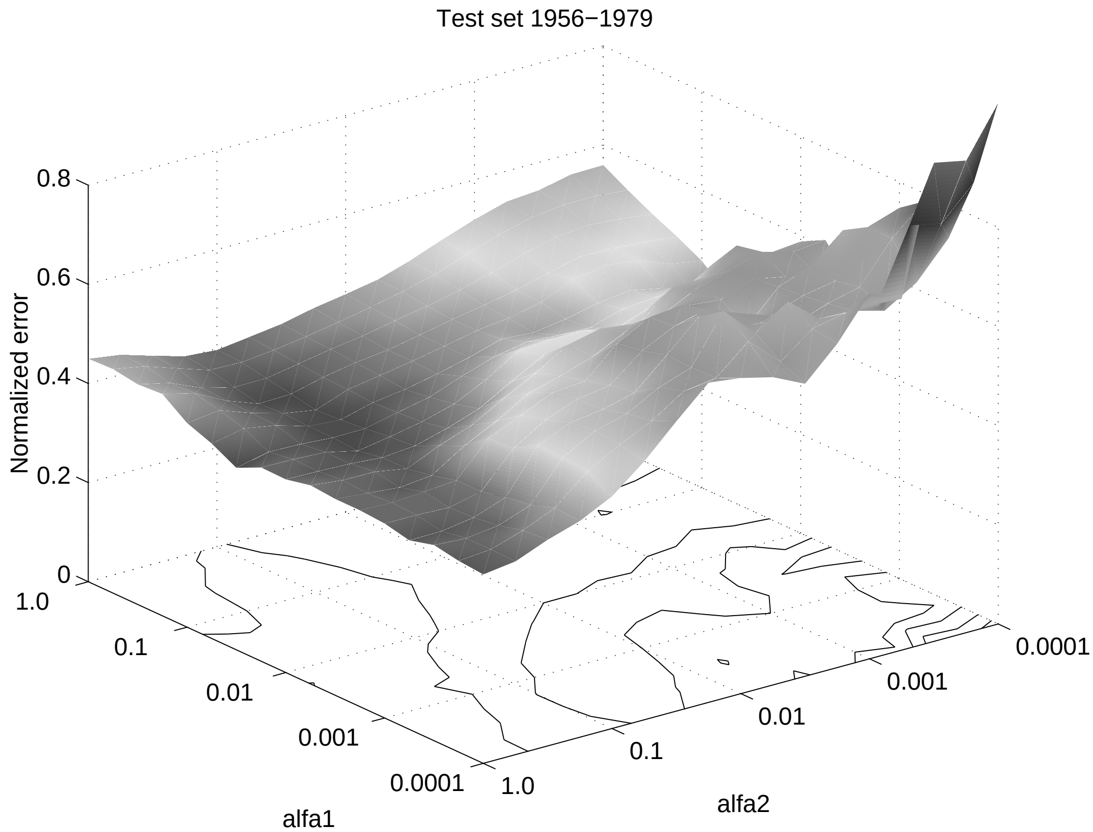
FIGUR 5.4: Sunspot trænet med anden ordens opdatering med vægthenfald. Vægthenfaldsleddene $\alpha_1 = \alpha_2 = 0.1$. Der er trænet 25 net, med forskellig initiel vægtfordeling. De enkelte nets præstationer er afbildet med punkter (dots), og gennemsnittet er fuldt optrukket. Der er afbildet den normerede fejl som funktion af hvor mange gange nettet har set træningsættet.(Epoker)



FIGUR 5.5: Optimalt vægthenfald, træningsserien. Vægthenfaldsleddene varieres over 4 dekader. Læg specielt mærke til konturlinierne, der viser at nettets præstation, med hensyn til træningsdatasættet, påvirkes væsentlig kraftigere af α_2 parameteren, end af α_1 parameteren. α_2 styrer vægthenfaldet på vægtene mellem det skjulte lag og outputlaget.



FIGUR 5.6: Optimalt vægthenfald, test sæt 1. På figuren er der tilføjet en linie der går lodret igennem minimum for fladen. I $\alpha_1 - \alpha_2$ planet er der indtegnet et kryds der viser koordinatet på dette minimum. Den optimale værdi for $\{\alpha_1; \alpha_2\}$ er derved bestemt til $\{0.0075; 0.01\}$.



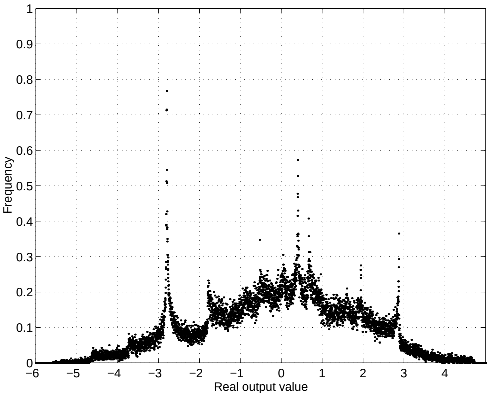
FIGUR 5.7: Optimalt vægthenfald, test sæt 2. Dette test sæt er atypisk for serien som et hele (se [SHL, afsnit V.]), idet variansen er tydeligt højere end i resten af solpletserien. Alligevel ses det af grafen, at der er mindre fejl på dette test sæt, for høje værdier af α_2. Sammenholdes graferne for de 2 testsæt, ses det at en værdi af $\{\alpha_1; \alpha_2\}$ på ca. $\{0.01; 0.1\}$, nok er det rette kompromis.

5.2 Forsøg med ideelle studenter

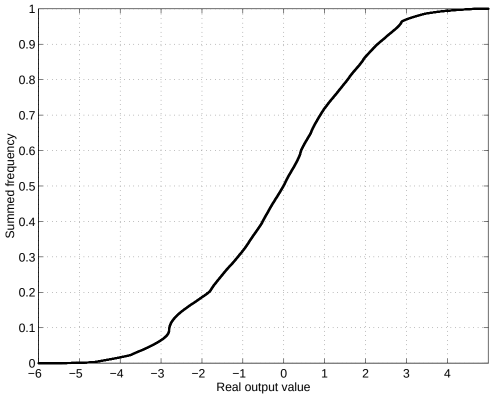
Forsøgene der er beskrevet her, er udført for at vise hvordan self-repair mekanismen virker på ideelle studenter. Med ideelle forstås:

- **Uendelig vægtområde**
  Der er ikke en øvre eller nedre grænse for værdien af studentens vægtværdier. (Maskinens regnenøjagtighed sætter selvfølgelig en øvre og nedre grænse for "uendelig")
- **Back-prop med anden ordens tilnærmelse**
  Der bruges en anden ordens metode til opdatering af vægtene, med et check på at opdateringen faktisk giver et fald i kostfunktionen taget på træningssættet.
- **Støjfrit**
  Der er ikke støj eller offset, på hverken vægte, input, eller output. Der regnes med fuld `double` nøjagtighed.

5.2.1 Forsøg nr I-A



FIGUR 5.8: Frekvensfunktionsplot af outputtet fra et teacher net inden det "klippes". Der er kørt 100,000 tilfældige input eksempler igennem teachernettet, og frekvensværdien af output neuronen er målt. Teacheren er af typen 8:5:1, med uniformt fordelte vægte i intervallet $[-2.0; +2.0]$



FIGUR 5.9: Fordelingsplot af outputtet fra et teachernet inden det "klippes". Tærskelværdien i outputlaget er justeret, således at der klassificeres ligeligt.

Beskrivelse af teacher net til forsøg I

Som det ses af tabel 5.4 er teacher nettet's vægte initialiseret med tilfældige uniformt fordelte tal i intervallet $[-2.0; +2.0]$. Nettet er af typen $8 : 5 : 1$, med tærskelværdier. Tærskelværdien i output laget er afstemt således at teacher nettet klassificerer så ligeligt som muligt. Teachernettets frekvens- og fordelingsplot kan ses på figur 5.8 og 5.9. Teachernettet klassificerer outputtet efter fortegn. Selve klassificeringen sker med værdierne $\{-1;+1\}$.

Dette teachernet bruges også i følgende forsøg: I-B, I-C og I-D.

Beskrivelse af studenter net til forsøg I-A

Der er 7 studenter net i dette forsøg. Nettenes topografi er beskrevet ved: $8:M:1$, hvor M varieres over værdierne $\{3,4,5,6,7,8,9,10\}$. Studenter nettene er af standard typen, dvs at der er tærskelværdier på både de skjulte- og outputneuronen, og de skjulte neuroner har aktiveringsfunktion tanh(), mens outputneuronen er lineær. Vægtene er initialiseret med uniformt fordelte tilfældige tal i intervallet $[-0.1;+0.1]$.

Efter initialisering er studenterne optrænet med 15 epoker af almindelig første ordens backpropagation uden moment. Derefter benyttes der anden ordens metode, som beskrevet i afsnit 3.2.4, på vægtlaget mellem input og skjulte. For vægtene mellem skjulte og output benyttes matrix inversion, som beskrevet i afsnit 3.2.4. Da målet er studenter der tager fejl på forskellige dele af inputrummet, er der valgt ikke at bruge vægthenfald Vægthenfald bevirker at studenterne har tendens til at finde løsninger der ligger tættere, og således bliver mere ens. Inputtet til studenterne er normalfordelte tilfældige tal, med forventningsværdi $\mu = 0$ og spredning $\sigma = 1.0$.

Træningssættet er på 300 eksempler, og der krydsvalideres på 2000 eksempler.

Teacher

Topografi	$8:5:1$
Vægtfordeling	$U(-2,2)$

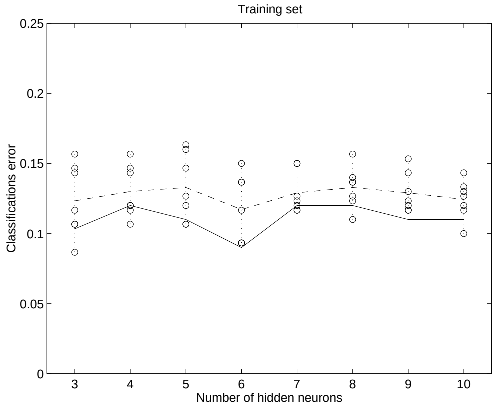
Studenter

Antal	7
Topografi	$8:M:1$
	$M \in \{3,\ldots,10\}$

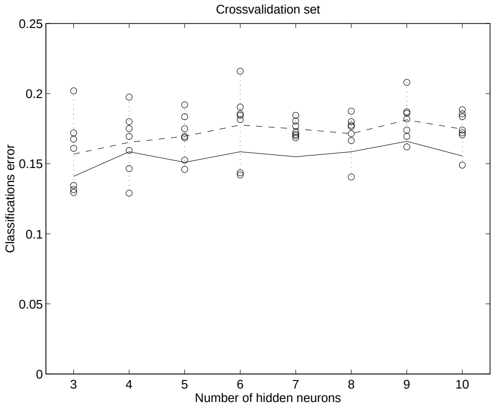
Kørselsbetingelser

Optræningsperiode	300 epoker
Krydsvalideringssæt	2000 eksempler
Træningssæt	300 eksempler

Tabel 5.4: Parametre til forsøg I



FIGUR 5.10: Plottet viser klassifikationsfejlen på træningssættet, som funktion af antallet af skjulte neuroner i studenternettene. Den enkelte students præstation er markeret med cirkel. Gennemsnitspræstationen er markeret med stiplet linie, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 5 studenter. Disse er optrænet med anden ordens opdatering i 100 epoker.



FIGUR 5.11: Dette plot viser klassifikationsfejlen på krydsvalideringssættet, for de samme studenter som afbildet på figur 5.10 som funktion af antallet af skjulte neuroner i studenter nettene. Bemærk at der ikke vindes noget i forhold til generaliseringsevnen, ved at øge antallet af skjulte neuroner. Bemærk også at fejlen på krydsvalideringssættet er en smule højere end for træningssættet.

Resultat af forsøg I-A

Dette forsøg skal afsløre hvordan studenternes præstation afhænger af antallet af skjulte neuroner. Eventuelt skal forsøget afsløre om der er én konfiguration der giver studenterne en bedre præstation end andre. På figur 5.10, og figur 5.11 ses de enkelte studenters præstationer (markeret med cirkler), sammen med gennemsnitspræstationen (markeret med stiplet line) og ensemblets consensus-beslutning (markeret med fuldt optrukket linie) på træningssættet hhv krydsvalideringssættet, som funktion af antallet af skjulte neuroner. Man ser af disse figurer, at fejlen på både træningssættet og krydsvalideringssættet, er stort set upåvirket af antallet af skjulte neuroner, specielt når man betragter variansen af de enkelte nets præstationer.

På dette grundlag skulle man mene at det er underordnet hvor mange skjulte neuroner de enkelte studenternet har. Der er valgt at lave 3 forsøg, med forskellige typer studenter: Et forsøg med studenter der har samme konfiguration som teachernettet, et forsøg hvor studenterne har flere skjulte neuroner, og et forsøg hvor de har færre.

5.2.2 Forsøg nr I-B

I dette forsøg er konfigurationen af studenter og teacher den samme. Således at studenterne har 5 skjulte neuroner.

Teacher

Topografi	8 : 5 : 1
Vægtfordeling	$U(-2,2)$
Klassificeringsværdi	$\{-1.0;+1.0\}$

Studenter

Antal	11
Topografi	8 : 5 : 1
Initial vægte	$U(-0.1;0.1)$
Input	$N(0;1)$

Optræning

Optræningsperiode	500 epoker
Træningssæt	300 eksempler
Krydsvalideringssæt	2000 eksempler
$(\alpha_1;\alpha_2)$	$(0;0)$

Self-repair og vægtdrift

Vægtdriftperiode	9500 epoker
Krydsvalideringssæt	2000 eksempler
Self-repair sæt	100 eksempler
w_{drift}	$N(0.0;3.0\cdot10^{-4})$

Tabel 5.5: Parametre til forsøg I-B

Teacher

Topografi
Teacher nettets topografi er beskrevet ved 8 indgangsenheder, 5 skjulte neuroner, og 1 output neuron. Der er en ekstra, ikke medtalt, enhed i hhv. inputlaget og det skjulte lag. Denne enhed er fast sat til værdien +1 og implementerer tærskelværdien i det efterfølgende lag.

Vægtfordeling
Teacher nettet initialiseres med tilfældige uniformt, i intervallet $[-2.0;+2.0]$, fordelte vægte. Tærskelværdien i output-laget justeres så teacher-nettet klassificerer input-rummet ligeligt i to klasser.

Klassificeringsværdi
Output af teacher-nettet, er sat til $\{-1.0;+1.0\}$.

Studenter

Antal
Der er 11 studenternet i ensemblet.

Topografi
Studenternettene har samme topografi som teacher-nettet.

Initial vægte
Studenternettene er initialiseret med tilfældige tal uniformt fordelt i intervallet $[-0.1;+0.1]$.

Input
Input til studenternettene er vektorer af normalfordelte tilfældige tal.

Optræning

Optræningsperiode
Studenternettene bliver optrænet med gradient metoden i 15 epoker, hvorefter de optrænes med anden ordens metoder indtil 500 epoker.

Træningssæt
Træningssættet er på 300 eksempler. For hver epoke ser den enkelte student alle 300 eksempler én gang.

Krydsvalideringssæt
Der krydsvalideres på 2000 eksempler. Det antages at dette sæt dækker input-rummet, så krydsvalideringsfejlen, er repræsentativ for generalisationsfejlen.

$(\alpha_1; \alpha_2)$
Vægthenfaldsleddene α_1 og α_2 (se evt. formel (3.2), (3.18) og (3.18)) er sat til 0. Der er altså ikke vægthenfald.

Self-repair og vægtdrift

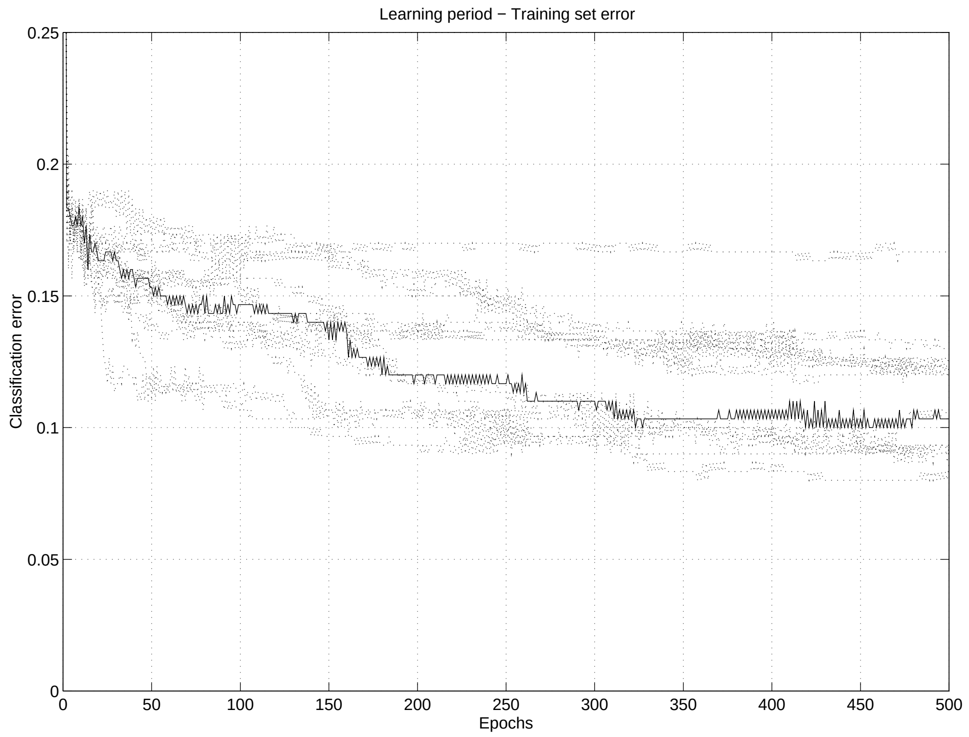
Vægtdriftperiode
Der køres self-repair, og "vægt-drift uden self-repair" i 9500 epoker.

Krydsvalideringssæt
Der krydsvalideres på de samme 2000 eksempler, som under optræningen.

Self-repair sæt
For hver epoke, opdaterer hvert studenternet vægtene på grundlag af en "batch" af 100 nye eksempler fra input-rummet.

Optræning af studenter til forsøg I-B

På figur 5.12 ses studenternettenes præstation på træningssættet under optræning. De enkelte nets præstation er markeret med punkteret linie, og consensus er markeret med fuld optrukket linie. Selvom vægtopdateringsalgoritmen sørger for at vægtopdatering giver et fald i kostfunktionen, sker det undertiden at der kommer en stigning i klassifikationsfejlen, sædvanligvis kun i en kortere periode. Dette skyldes at kostfunktionen og klassifikationsfejlen ikke har helt samme mål. For kostfunktionen er fejlen afhængig af afstanden fra outputtet til den ønskede klassifikationsværdi. For klassifikationen er fejlen afhængig af outputtets fortegn. På figur 5.13 ses studenternettenes præstation på krydsvalideringssættet. Det bemærkes at der er 1 student der er en del bedre end studenternes consensus, og et par stykker der er næsten lige så gode. Dette tyder på at en stor del af de 11 studenter er meget enige. Betragter man figur 5.14 og figur 5.15 kan man få information omkring de 11 nets indbyrdes korrelation. Figur 5.14 viser nettenes indbyrdes fejl-korrelation. Figuren viser et plot med søjler i en matrix. Matricens indices er vist som tal på $x-$ og $y-$aksen. En søjle stående i matricens koordinat (i, j) viser sandsynligheden for at studenternet i og j tager fejl på samme

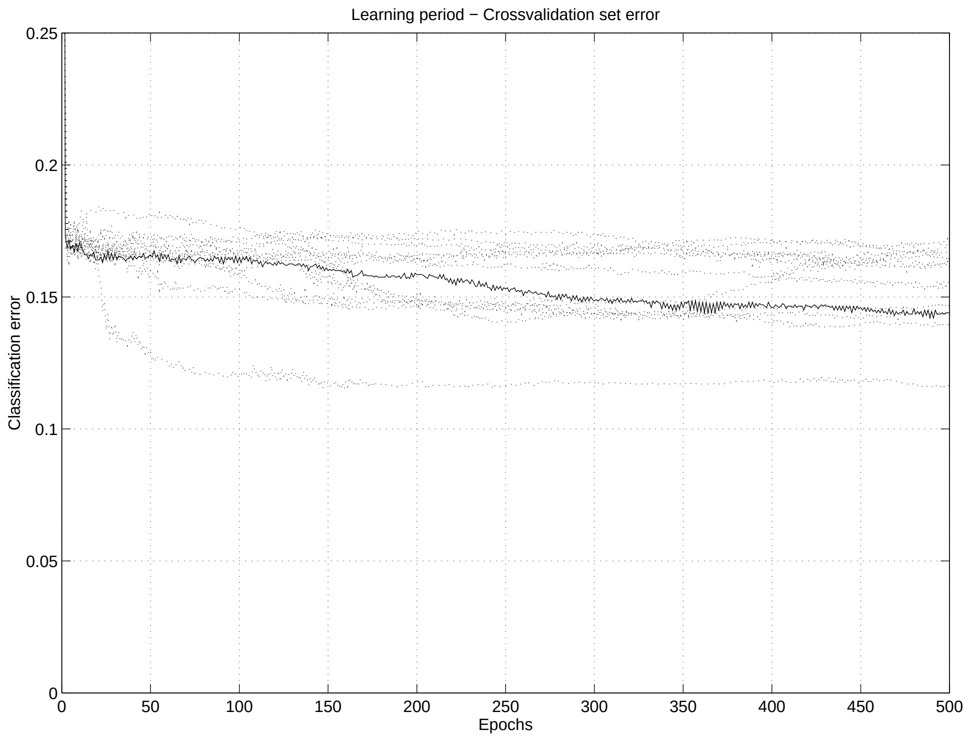


FIGUR 5.12: Plottet viser klassifikationsfejlen på træningssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 11 studenter. Disse er optrænet med ordinær gradient metode i 15 epoker efterfulgt af anden ordens opdatering indtil 500 epoker.

vilkårlige inputmønster. Figuren er derfor symmetrisk omkring diagonalen, og i diagonalen optræder der et estimat for de enkelte nets krydvalideringssfejl. Man ser af figuren at der er ret høj fejl-korrelation mellem alle net. Hvis nettene havde været ukorrelerede med hensyn til fejl, ville niveauet uden for diagonalen været kvadratet på fejlen i diagonalen. Dette indses ved at betragte sandsynligheden for fejl på net A og net B, og antage at de tager fejl uafhængigt af hinanden: [Con84]

$$p(A \cap B) = p(A) \cdot p(B) \tag{5.7}$$

På figur 5.15 ses et $\mu(\theta)$-plot. Af dette plot kan man aflæse et estimat af input-rummets sværhed. Havde nettene været ukorrelerede med hensyn til fejl, og havde hver enkelt net haft en krydsvalideringsfejl på 14.5%, skulle søjlerne have fulgt den fuldt optrukne linie i plottet. Nettene er tydeligvis ikke ukorrelerede. Plottet viser at en stor del af input-rummet let kan læres, 67% af alle input kan alle net klassificere korrekt. Samtidigt er der 5% af input-rummet som ingen net kan klassificere korrekt. Dette antyder at problemet er meget svært at lære med



FIGUR 5.13: Dette plot viser klassifikationsfejlen på krydsvalideringssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 11 studenter. Disse er optrænet med ordinær gradient metode i 15 epoker efterfulgt af anden ordens opdatering indtil 500 epoker.

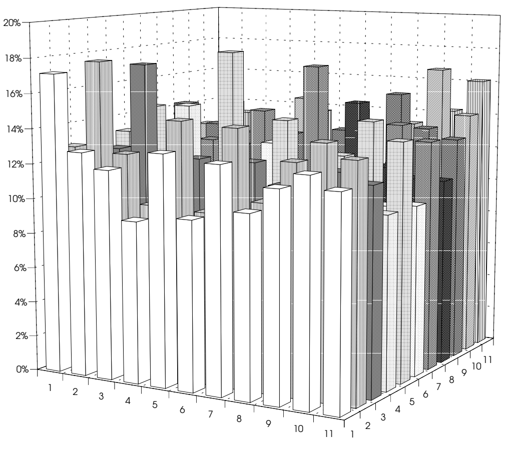
studenternes givne konfiguration.

Vægtdrift og self-repair

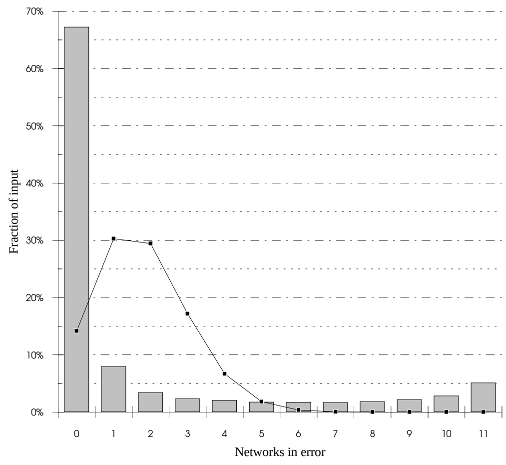
I denne del af forsøget er studenternettene udsat for vægtdrift, og det er undersøgt hvilken effekt self-repair har. Vægtdrift betyder at hver vægt får lagt en værdi (dw_i) til for hver epoke. Denne værdi er fast for den enkelte vægt, og valgt fra en normal fordeling.

På figur 5.18 ses effekten af vægtdriften, når studenterne ikke udfører self-repair. Af figuren kan man se at påvirkningen har meget forskellig effekt på studenterne. Dette kan skyldes at vægtene i studenternettene ikke har en øvre grænse, og at studenterne er optrænet uden vægthenfald. Derved kan nogle studenter have vægte der er ordener større end andre. Det kan også tænkes at driften er i den næsten modsatte retning af gradienten for kostfunktionen.

På figur 5.19 ses studenterne udsat for præcis samme påvirkning, dog med self-repair. På figuren er consensus fra figur 5.18 desuden optegnet med stiplet



FIGUR 5.14: Dette plot viser korrelationen mellem studenterne, efter endt optræning. Hver søjle viser sandsynligheden for at de tilsvarende net samtidigt tager fejl på tilfældigt inputmønster. I diagonalen optræder derfor et estimat af de enkelte nets kryd-svalideringsfejl. Korrelationsplottet er baseret på 100,000 tilfældige inputmønstre.



FIGUR 5.15: Dette plot, er et $\mu(\theta)$-plot, der viser hvor mange net der tager fejl på et tilfældigt input, lige efter optræningen. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 14.5%, og havde været ukorrelerede med hensyn til fejl. Søjlerne er målt over 100,000 tilfældige input.

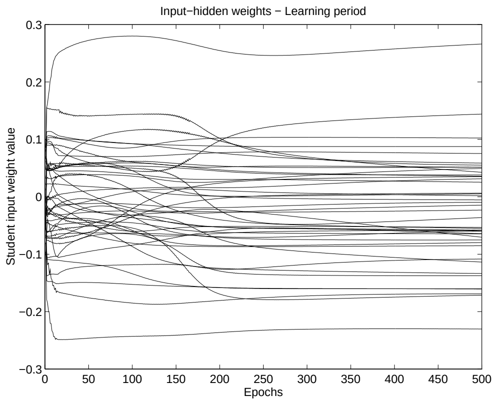
linie, for at lette sammenligningen. Man ser at der er en voldsom aktivitet i den første del af perioden (se figur 5.32), hvorefter nettene svinger ind i stabile kurver. Consensus kurven virker næsten upåvirket af de enkelte nets udsving, og holder sig bemærkelsesværdigt stabil, næsten kostant i de første 2000 epoker, hvor consensus uden self-repair øger fejl raten fra 15% til ≈37%.

Til trods for at consensus efter indsvingningsperioden holder sig næsten konstant, kan de enkelte net ikke opretholde en konstant præstation.

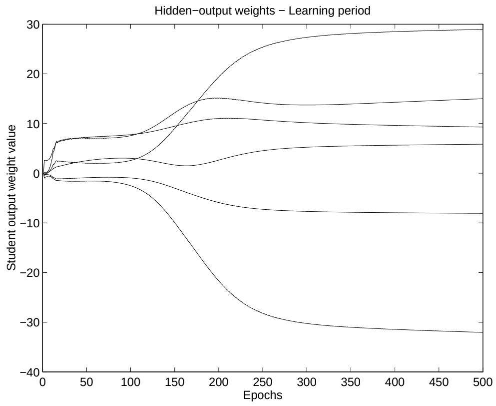
Vægtdriften gør altså at studenterne tager fejl på forskellige dele af inputrummet. De bliver mindre korrelerede.

På figur 5.30 og 5.31 ser man self-repair nettenes fejl-korrelation efter 9500 epokers. Man ser at der nu mindre enighed om klassifikationen. Hvor der før var op mod 70% sandsynlighed for at alle net kunne klassificere et tilfældigt input korrekt, er der nu stort set ingen inputmønstre alle studenter kan klassificere korrekt. På figur 5.31 ser man at $\mu(\theta)$-plottet nu er ret tæt på den teoretiske for ukorrelerede net. Man ser også hvorfor consensus nu er så god. Sandsynligheden for at consensus svarer rigtigt på et tilfældigt input, er summen af søjlerne 0...5.

På figur 5.32 ses et udsnit af figur 5.19. Her ser man overgangen fra optræning til vægtdrift, med self-repair. Den stiplede linie er consensus uden self-repair. Bemærk at consensus kurvernes udgangspunkt ved overgangen (0'te epoke) til vægtdrift er den samme. Dette niveau er ca. 16%. Man ser altså at self-repair

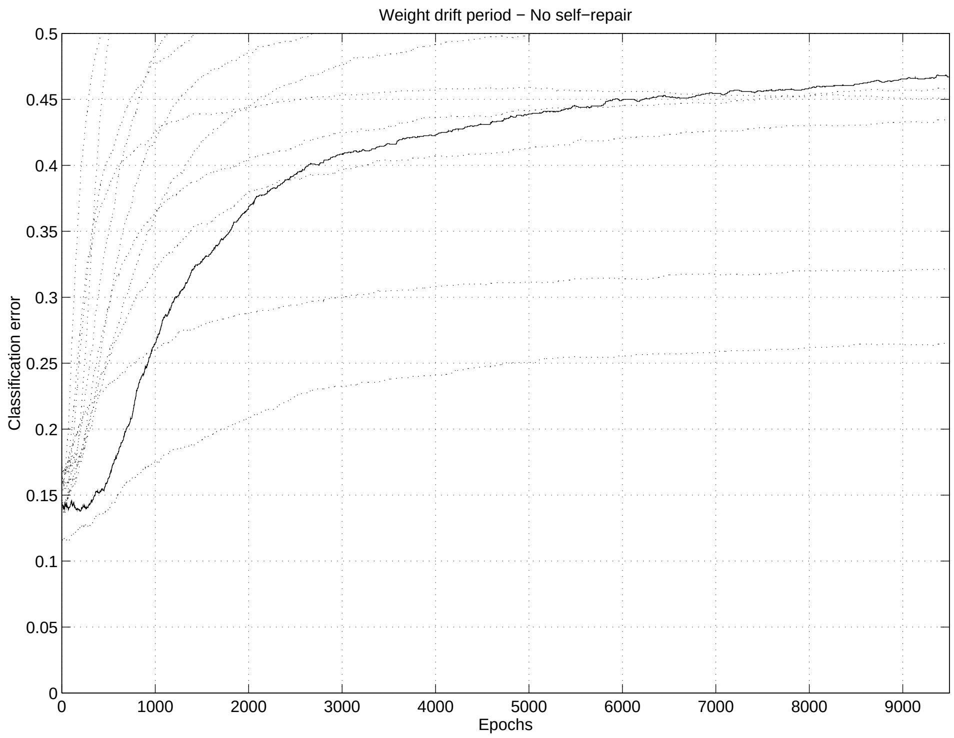


FIGUR 5.16: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under optræningen. Plottet viser et typisk forløb, hvor de fleste af vægtene næsten ikke ændrer sig, og en lille rest laver nogle kraftige udsving tidligt i forløbet. Efter en indsvingningsperiode, finder de en næsten stationær værdi.

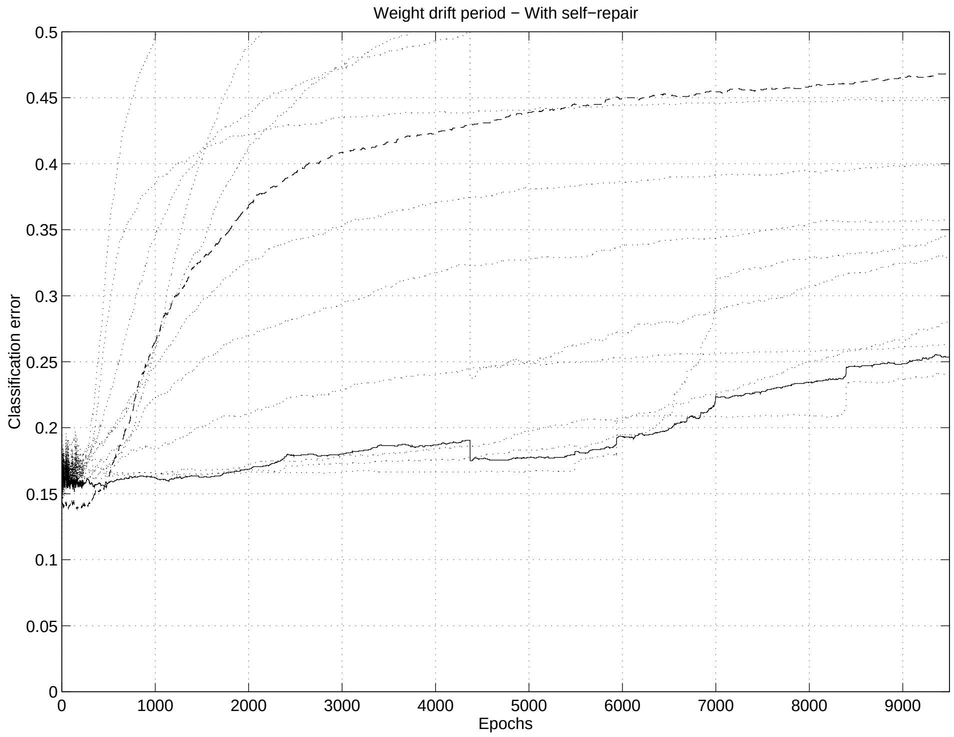


FIGUR 5.17: Skjulte-output vægtværdier for studenten fra figur 5.16, under optræningen. Plottet viser et typisk forløb, hvor de fleste vægte laver nogle kraftige udsving tidligt i forløbet, hvorefter de går over i en mere rolig periode. Det er typisk at værdierne er et par ordener højere end for vægtene i laget mellem input og skjulte.

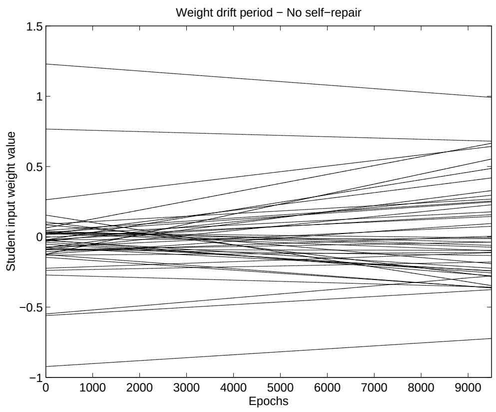
i de første epoker har en indsvingningsperiode, hvor nettene "koordinerer" deres viden. For at undersøge om denne indsvingningsperiode har en indvirkning på korrelationen mellem nettene, er der lavet et fejlkorrelations-plot og et $\mu(\theta)$-plot. Disse ses på figur 5.28 hhv. 5.29. Disse plot skal sammenholdes med de tilsvarende plot for studenterne uden self-repair, på figur 5.24 og figur 5.25.



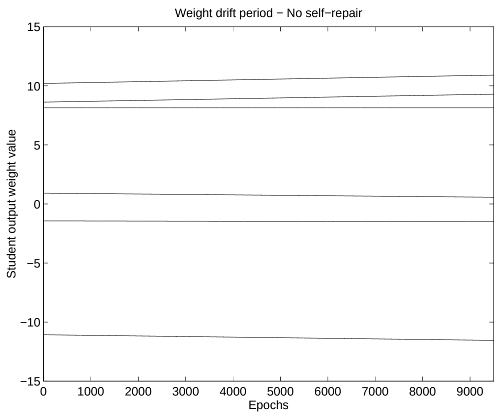
FIGUR 5.18: Plottet viser klassifikationsfejlen på studenternettene, når de ødelægges med en konstant rate, som funktion af epoker. For hver epoke lægges der en konstant værdi til vægtene. Værdien er forskellig fra vægt til vægt og vælges tilfældigt fra en normalfordeling. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie.



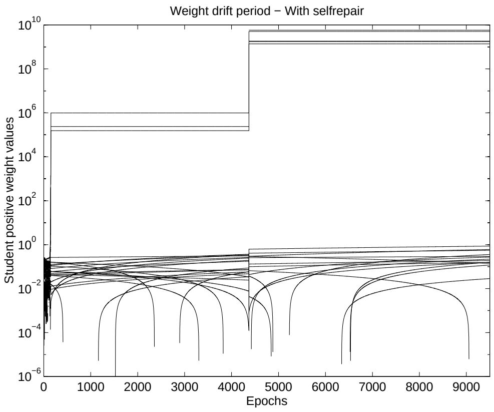
FIGUR 5.19: Plottet viser studenternettene når de udsættes for samme ødelæggelse som på figur 5.18. Her forsøger de dog at rette op på skaden ved at udføre self-repair med ensemblets consensus beslutning. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuldt optrukket linie, og consensus uden self-repair (fra figur 5.18) er markeret med stiplet linie.



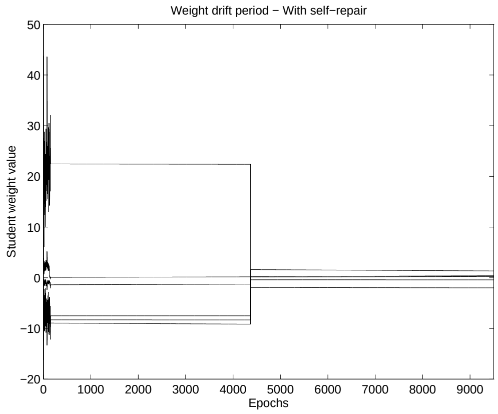
FIGUR 5.20: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under vægtdrift perioden. Der er ikke self-repair. Man ser, som ventet, at vægtene driver med en konstant rate.



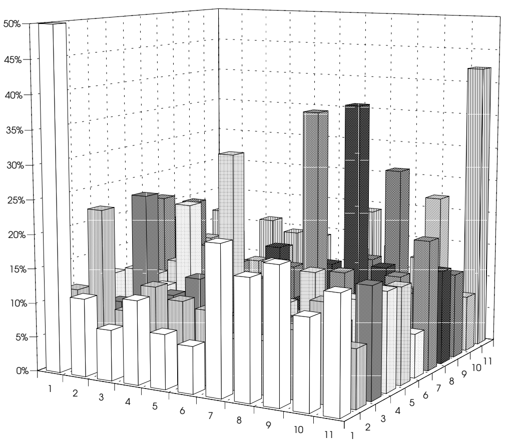
FIGUR 5.21: Der er plottet skjulte-output vægtværdier for studenten fra figur 5.20, under vægtdrift perioden. Man ser at vægtene her tilsyneladende ikke ændrer sig så meget som input vægtene. Dette skyldes at raten er den samme, men outputvægtene er ca. en faktor 10 højere.



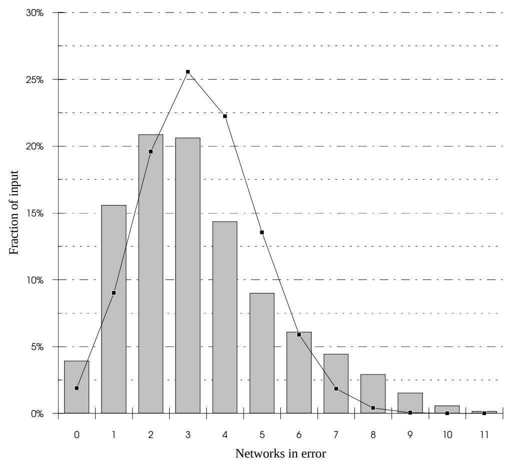
FIGUR 5.22: Der er plottet en typisk students input-skjulte vægtværdier, som funktion af epoker, under self-repair perioden. Plottet er vist med logaritmisk y-akse, og derfor er negative vægte ikke plottet. Det betyder at en positiv vægt der krydser $y = 0$ forsvinder, og omvendt. Det skal bemærkes at det tilsvarende plot for negative vægte har et lignende udseende. Man ser, at vægtene har voldsomme udsving, og at udsvingene sker på forholdsvis kort tid (2-3 epoker).



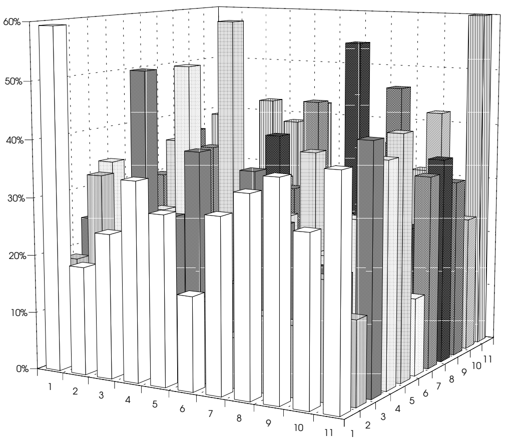
FIGUR 5.23: Skjulte-output vægtværdier for studenten fra figur 5.22, under self-repair perioden. Plottet viser at outputvægtene også har kraftige udsving. Det er ikke typisk at vægtene kun falder. Betragter man andre studenternets outputvægte ser man at disse også kan stige. Bemærk aktiviteten i begyndelsen, og omkring 4400 epoker, der også ses på den modsvarende figur 5.22.



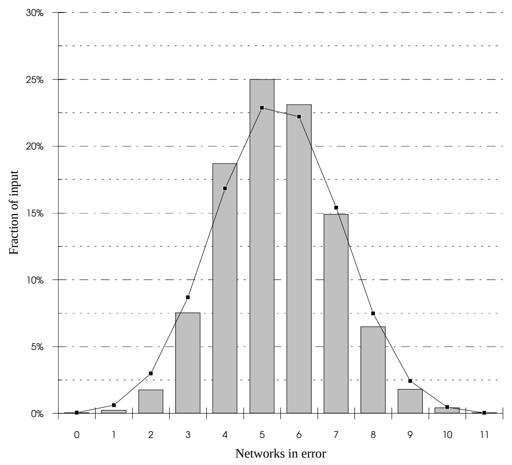
FIGUR 5.24: Korrelationen mellem studenterne, efter 400 epoker vægtdrift. Bemærk forskellen på akserne i forhold til figur 5.14, der er taget lige efter optræningen. Baseret på 100,000 tilfældige inputmønstre.



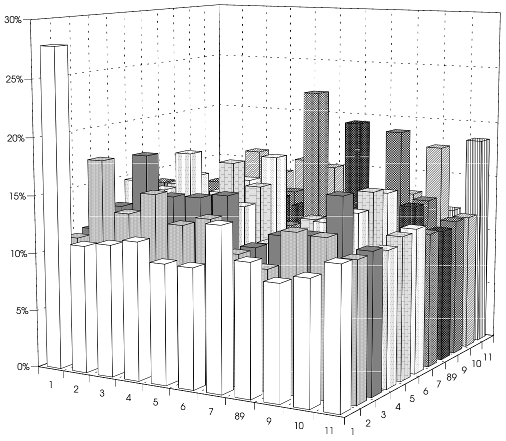
FIGUR 5.25: $\mu(\theta)$-plot, efter 400 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, for en ukorreleret krydsvalideringsfejl på 30% Målt over 100,000 tilfældige input.



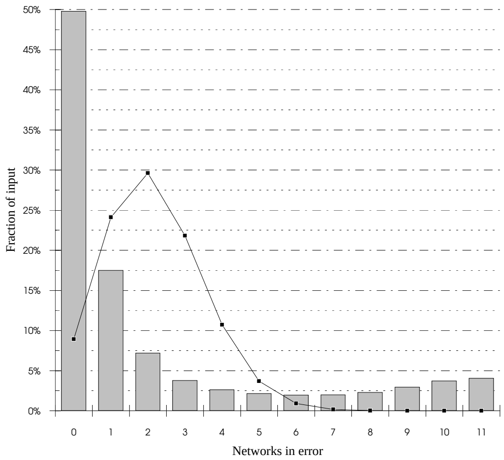
FIGUR 5.26: Korrelationen efter 9500 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



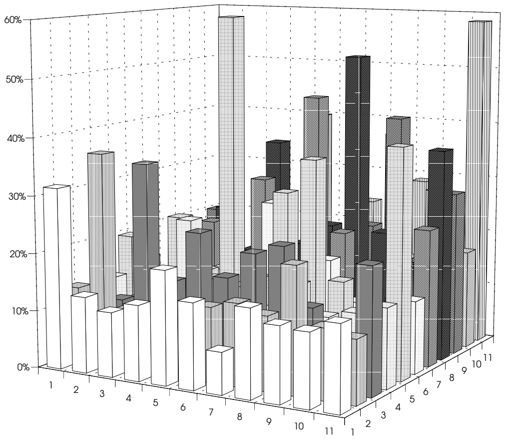
FIGUR 5.27: $\mu(\theta)$-plot, efter 9500 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 49%. Målt over 100,000 tilfældige input.



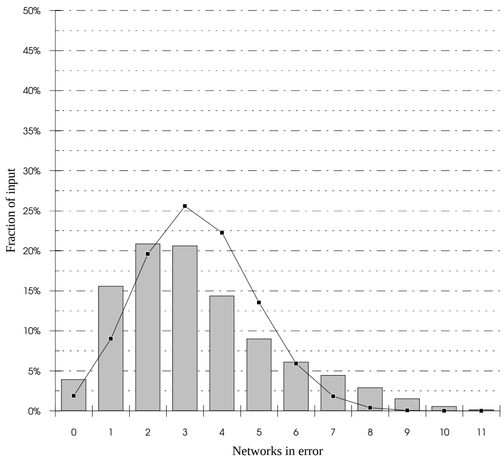
FIGUR 5.28: Korrelationen mellem studenterne, efter 400 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



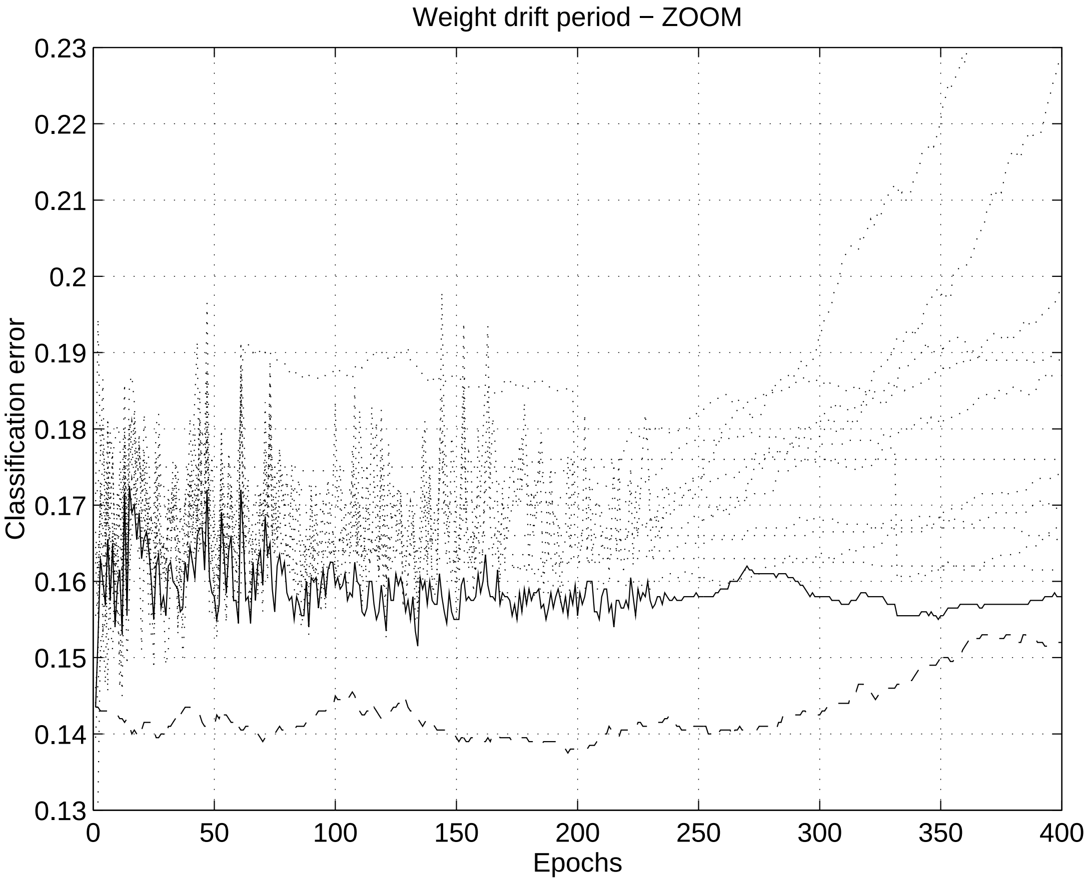
FIGUR 5.29: $\mu(\theta)$-plot, efter 400 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 20%. Målt over 100,000 tilfældige input.



FIGUR 5.30: Korrelationen mellem studenterne, efter 9500 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



FIGUR 5.31: $\mu(\theta)$-plot, efter 9500 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 30%. Målt over 100,000 tilfældige input.



FIGUR 5.32: Zoom af figur 5.19. Bemærk indsvingningsperioden.

5.2.3 Forsøg nr I-C

Det ønskes undersøgt om flere neuroner i det skjulte lag kan forbedre self-repair mekanismen. Der er derfor opsat et eksperiment med 9 studenter, der hver især har 7 skjulte neuroner, overfor teacherens 5 skjulte neuroner. Dette forsøg er sat op på samme måde som forsøg I-B, med mindre ændringer. Disse er:

- **Antal skjulte neuroner**
  Der er i dette forsøg 7 skjulte neuroner i studenternettene, mod 5 skjulte i forsøg I-B
- **Antal studenter**
  Der er i dette forsøg 9 studenter net, mod 11 i forsøg I-B
- **Ødelæggelses rate**
  Ødelæggelses raten i dette forsøg beskrives ved $dw_i \in N(0.0, 3 \cdot 10^{-5})$, mod $dw_i \in N(0.0, 3 \cdot 10^{-4})$ i forsøg I-B

I tabel 5.6 ses en oversigt over opsætningen af parametre i dette forsøg. Vægtopdatering for studenternettene foregår med anden ordens approximation.

Teacher

Topografi	8 : 5 : 1
Vægtfordeling	$U(-2, 2)$
Klassificeringsværdi	$\{-1.0; +1.0\}$

Studenter

Antal	9
Topografi	8 : 7 : 1
Initial vægte	$U(-0.1; 0.1)$
Input	$N(0; 1)$

Optræning

Optræningsperiode	500 epoker
Træningssæt	300 eksempler
Krydsvalideringssæt	2000 eksempler
$(\alpha_1; \alpha_2)$	(0;0)

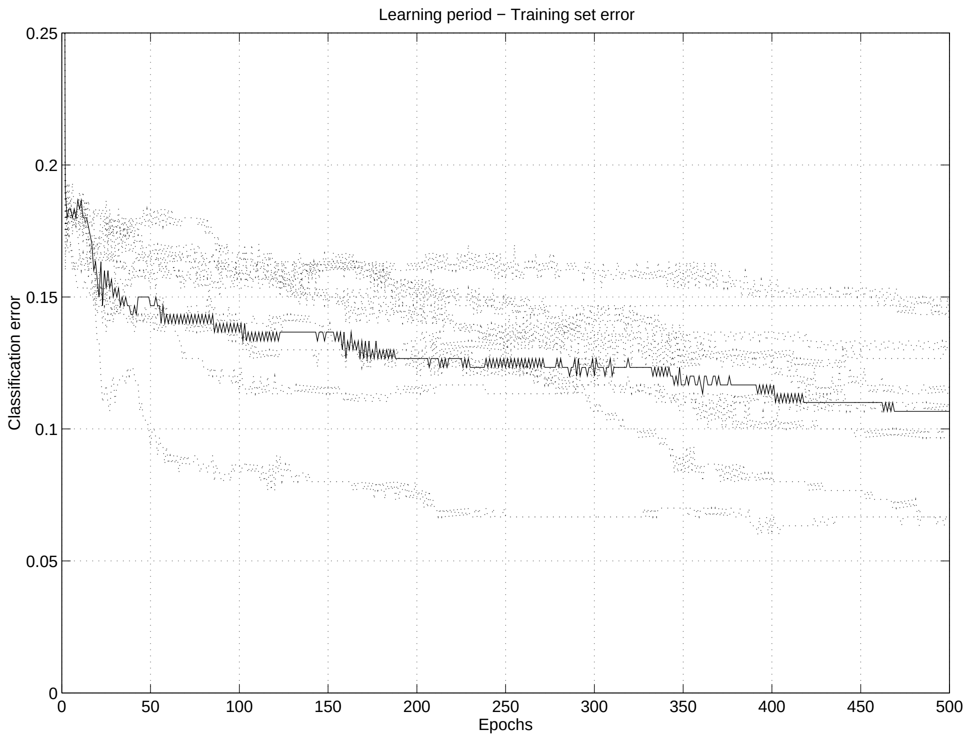
Self-repair og vægtdrift

Vægtdriftperiode	9500 epoker
Krydsvalideringssæt	2000 eksempler
Self-repair sæt	100 eksempler
w_{drift}	$N(0.0; 3.0 \cdot 10^{-5})$

Tabel 5.6: Parametre til forsøg I-C

Optræning af studenter i forsøg I-C

Forløbet af optræningen ses på figur 5.33 og figur 5.34, der viser at der opnås næsten samme klassifikationsfejl på træningssættet, som i forsøg I-B (11-12%). Derimod er klassifikationsfejlen på krydsvalideringssættet en smule højere (14% i I-B mod 17% her). Da teacher, træningseksempler, og krydsvalideringssæt, er de samme, kan den lille forskel kun skyldes initialisering, og størrelsesforskellen på studenternettene. Betragtes korrelations- og $\mu(\theta)$-plottene optaget efter endt optræning, figur 5.35 og 5.36, ses det at studenterne udviser samme symptomatiske enighed omkring hvad der er rigtigt og forkert.



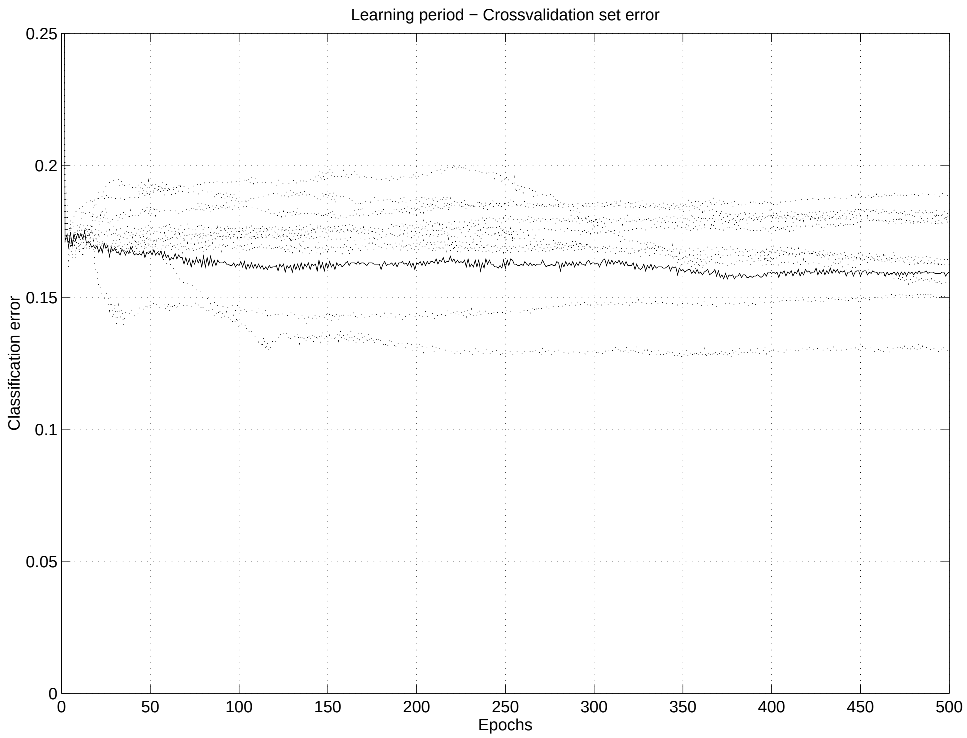
FIGUR 5.33: Plottet viser klassifikationsfejlen på træningssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 15 epoker efterfulgt af anden ordens opdatering indtil 500 epoker.

Vægtdrift og self-repair. Forsøg I-C

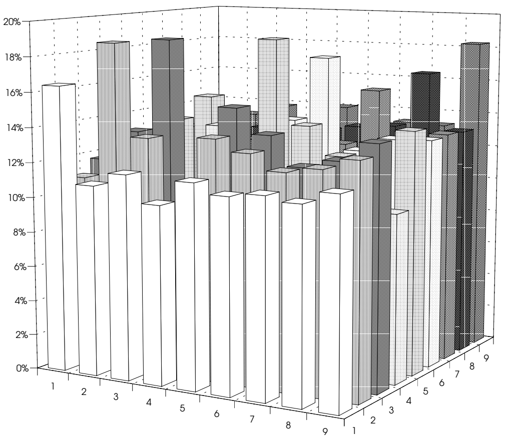
På figur 5.39 ses studenterne udsat for vægtdrift.

I forhold til forsøg I-B ses det at præstationsfaldet er trukket ud over en længere periode, men forløbet er af samme karakter. Self-repair (figur 5.40) udviser også, i store træk, det samme mønster som for forsøg I-B. Man ser forløbet delt op i perioder med karakteristisk flade, næsten konstante præstationer, med trinvis skift mellem to perioder. Til trods for at ødelæggelsesraten er sat ned med en faktor 10, ender man ikke med væsentlig bedre præstation for self-repair.

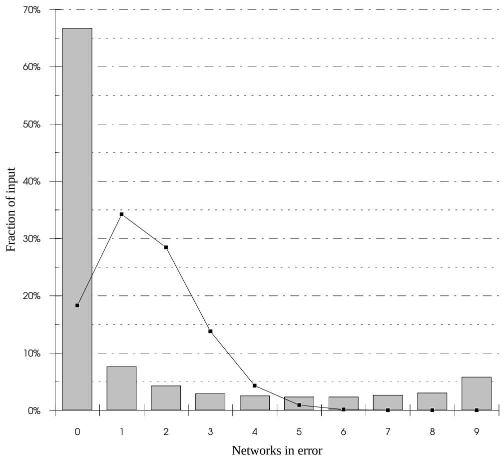
Den mest bemærkelsesværdige lighed mellem forsøg I-C og I-B er den karakteristiske indsvingningsperiode på ca. 250 epoker. Plottene for indsvingningsperioden for forsøg I-B (figur 5.32), og forsøg I-C (figur 5.51), er vist i samme skala, og udviser tydeligt samme slags forløb. I denne periode bliver nettene ret forskellige, hvis de ikke udfører self-repair (se $\mu(\theta)$-plottene figur 5.25 og 5.44). Modsat hvis de udfører self-repair, efter samme periode er nettene stadig meget enige om hvad der er sandt og falsk (se $\mu(\theta)$-plottene figur 5.29 og 5.48).



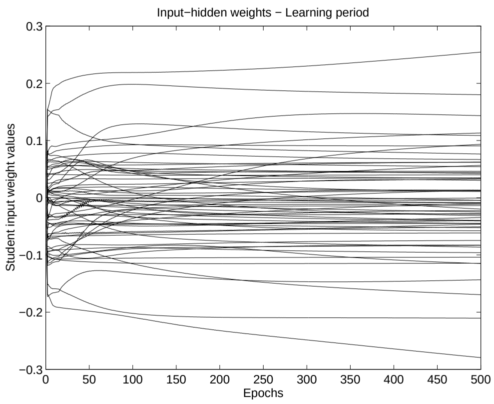
FIGUR 5.34: Dette plot viser klassifikationsfejlen på krydsvalideringssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 15 epoker efterfulgt af anden ordens opdatering indtil 500 epoker.



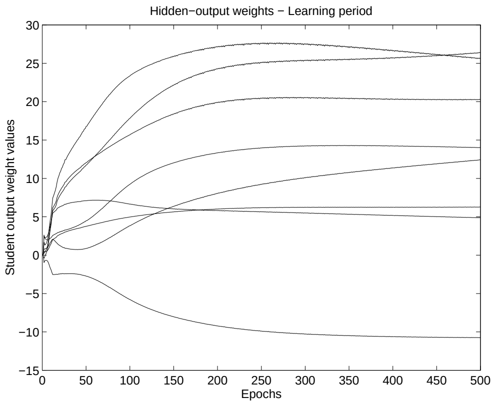
FIGUR 5.35: Dette plot viser korrelationen mellem studenterne, efter endt optræning. Hver søjle viser sandsynligheden for at de tilsvarende net samtidigt tager fejl på tilfældigt input-mønster. I diagonalen optræder derfor et estimat af de enkelte nets krydsvalideringsfejl. Korrelationsplottet er baseret på 100,000 tilfældige inputmønstre.



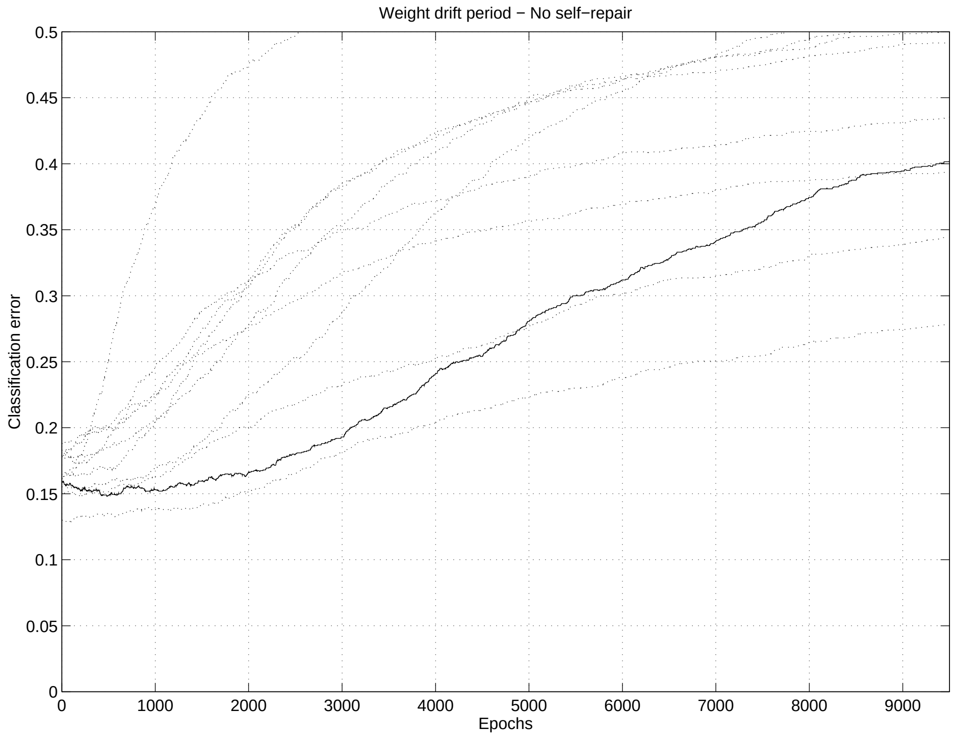
FIGUR 5.36: Dette plot, er et $\mu(\theta)$-plot, lige efter optræningen. Plottet viser sandsynligheden for at et vilkårligt input fejlklassificeres af n studenternet. $n = 0$ svarer derfor til sandsynligheden for, at et input klassificeres korrekt af alle studenter, og $n = N = 9$ svarer til sandsynligheden for, at et input fejlklassificeres af alle net. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 17%, og havde været ukorrelerede med hensyn til fejl. Søjlerne er målt over 100,000 tilfældige input.



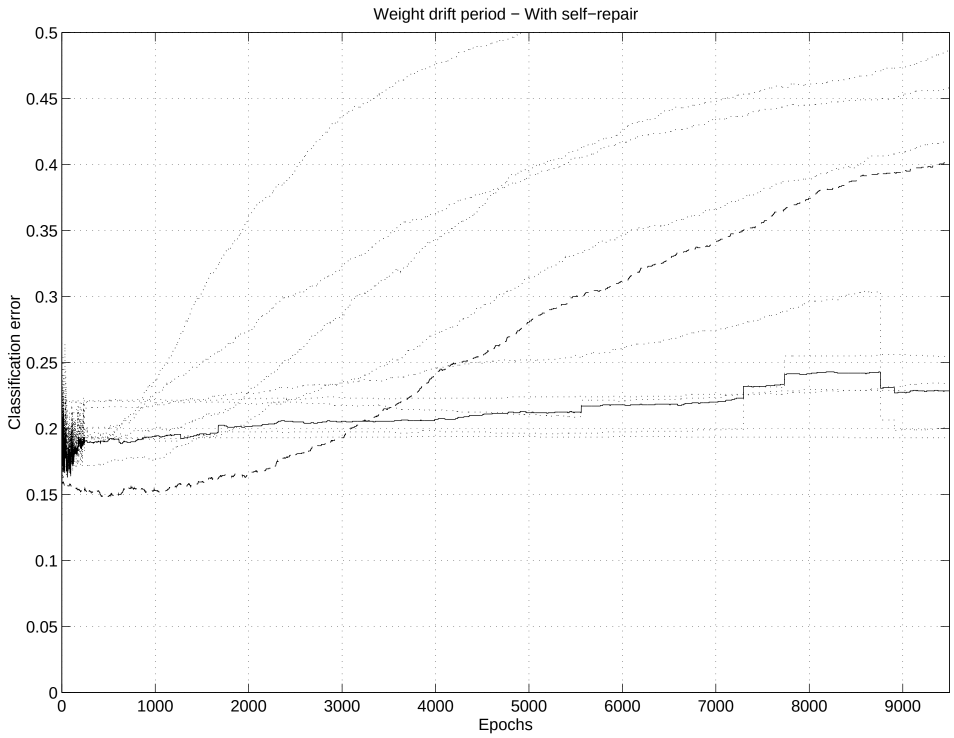
FIGUR 5.37: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under optræningen. Plottet viser et typisk forløb, hvor de fleste af vægtene næsten ikke ændrer sig, og en lille rest laver nogle kraftige udsving tidligt i forløbet. Efter en indsvingningsperiode, finder de en næsten stationær værdi.



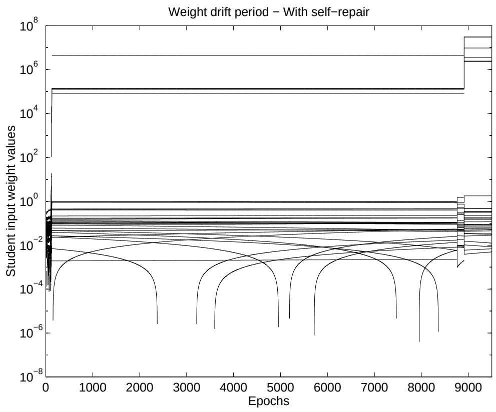
FIGUR 5.38: Skjulte-output vægtværdier for studenten fra figur 5.37, under optræningen. Plottet viser et typisk forløb, hvor de fleste vægte laver nogle kraftige udsving tidligt i forløbet, hvorefter de går over i en mere rolig periode. Det er typisk at værdierne er et par ordener højere end for vægtene i laget mellem input og skjulte.



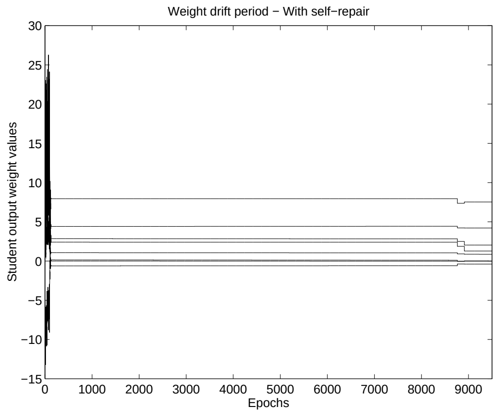
FIGUR 5.39: Plottet viser klassifikationsfejlen på krydsvalideringssættet, som funktion af epoker. Studenter nettene ødelægges med en konstant rate. For hver epoke lægges der en konstant værdi til vægtene. Værdien er forskellig fra vægt til vægt og vælges tilfældigt fra en normalfordeling. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie.



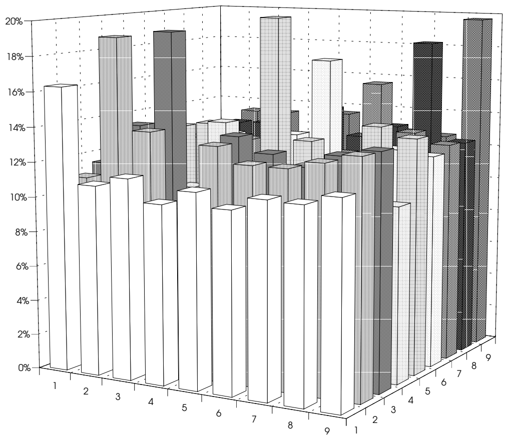
FIGUR 5.40: Plottet viser klassifikationsfejlen på krydsvaliderings-sættet, som funktion af epoker. Studenter nettene udsættes for samme ødelæggelse som på figur 5.39, men forsøger at rette op på skaden ved at udføre self-repair med ensemblets consensus beslutning. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie. Ensemblets consensus beslutning uden self-repair markeret med stiplet linie.



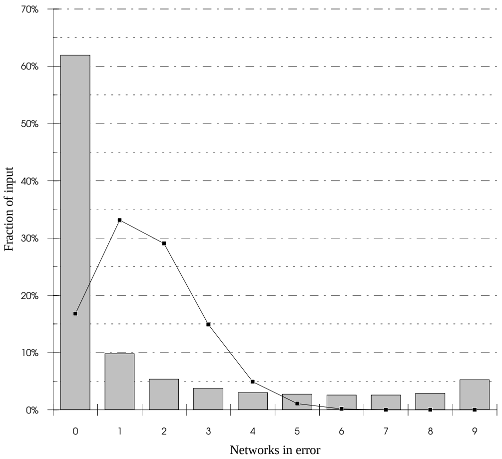
FIGUR 5.41: Der er plottet en typisk students input-skjulte vægtværdier, under self-repair perioden. Plottet er vist med logaritmisk y-akse, og derfor er negative vægte ikke plottet. Det betyder at en positiv vægt der krydser $y = 0$ forsvinder, og omvendt. Det skal bemærkes at det tilsvarende plot for negative vægte har et lignende udseende.



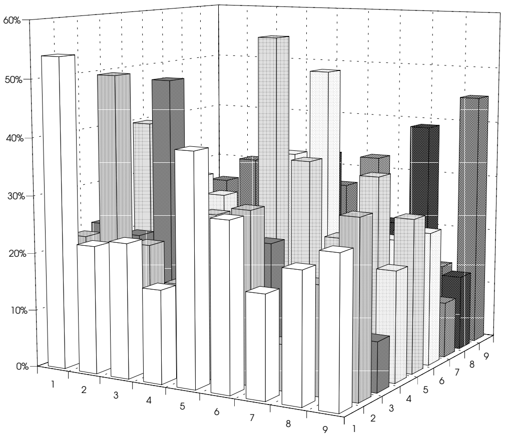
FIGUR 5.42: Skjulte-output vægtværdier for studenten fra figur 5.41, under selfrepair perioden.



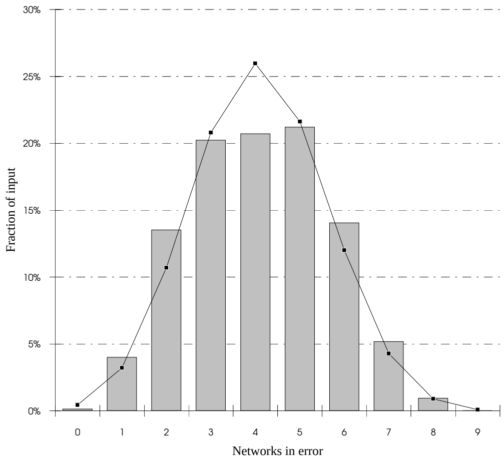
FIGUR 5.43: Korrelationen mellem studenterne, efter 300 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



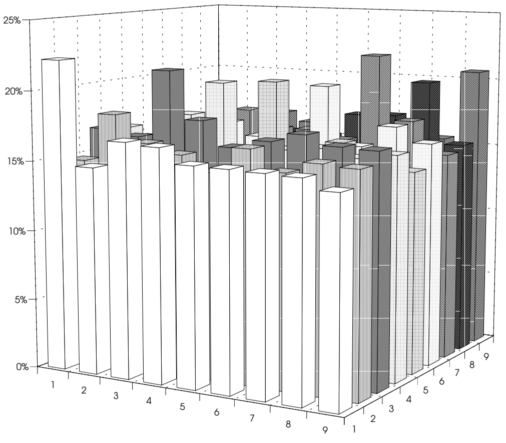
FIGUR 5.44: $\mu(\theta)$-plot, efter 300 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, for en ukorreleret krydsvalideringsfejl på 18% Målt over 100,000 tilfældige input.



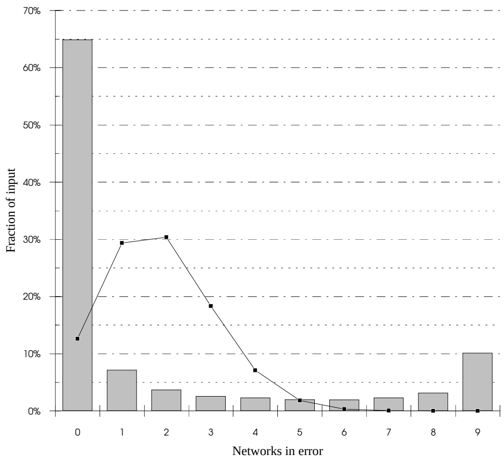
FIGUR 5.45: Korrelationen efter 9500 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



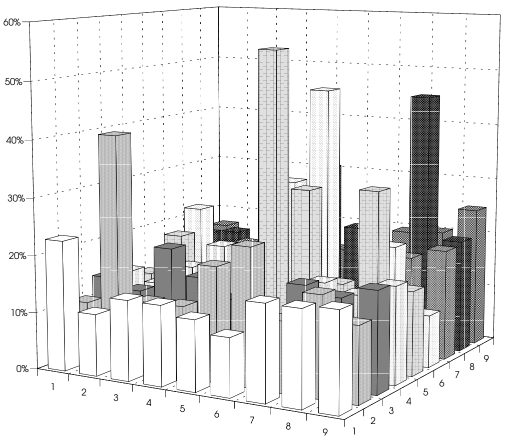
FIGUR 5.46: $\mu(\theta)$-plot, efter 9500 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 45%. Målt over 100,000 tilfældige input.



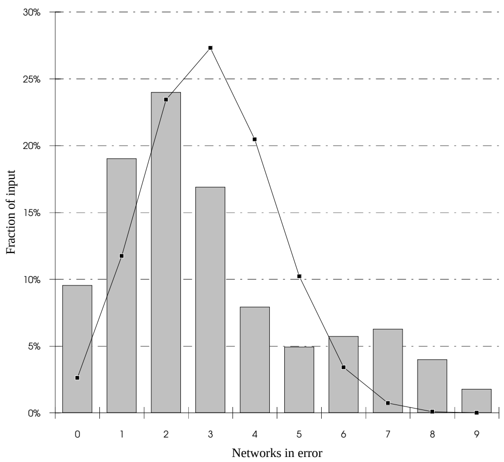
FIGUR 5.47: Korrelationen mellem studenterne, efter 300 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



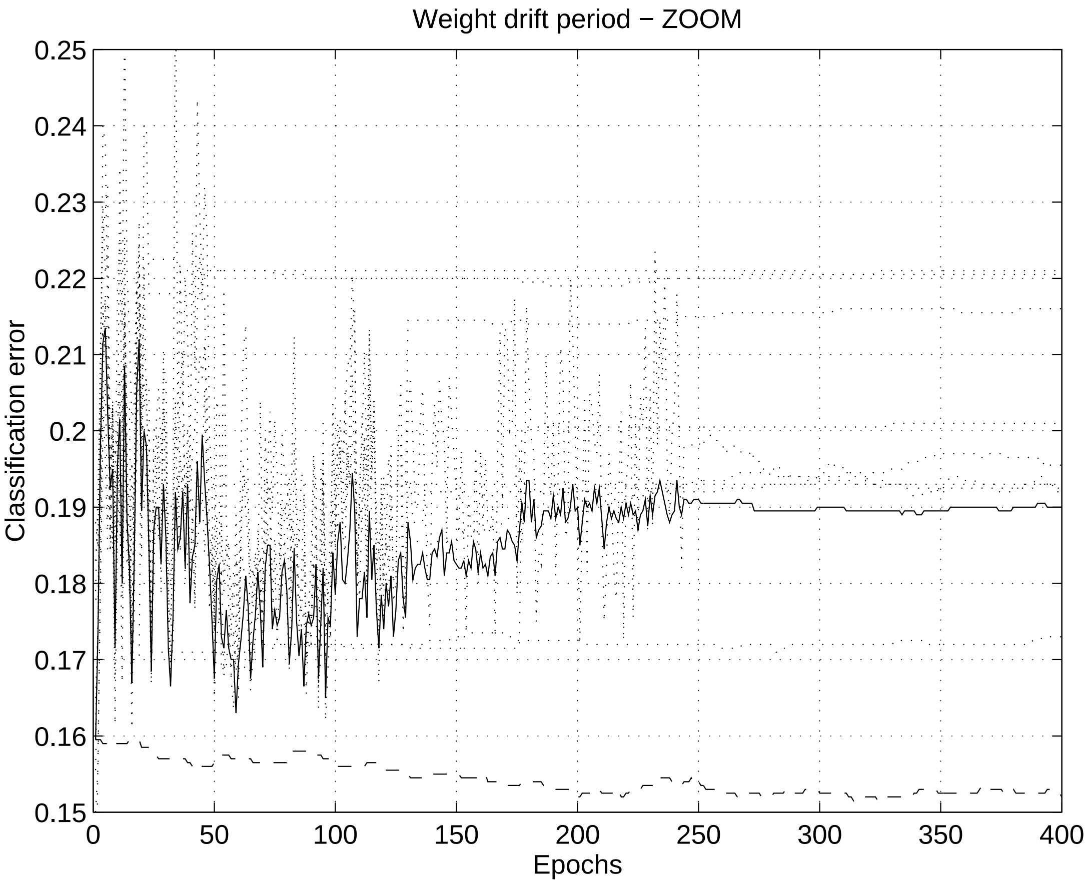
FIGUR 5.48: $\mu(\theta)$-plot, efter 300 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 21%. Målt over 100,000 tilfældige input.



FIGUR 5.49: Korrelationen mellem studenterne, efter 9500 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



FIGUR 5.50: $\mu(\theta)$-plot, efter 9500 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 33%. Målt over 100,000 tilfældige input



FIGUR 5.51: Zoom af figur 5.40. Bemærk indsvingningsperioden.

5.2.4 Forsøg nr I-D

I dette forsøg har studenterne 3 skjulte neuroner. Da teachernettet er det samme som i de tre foregående forsøg, med 5 skjulte neuroner, er studenterne altså "underdimensionerede".

Teacher

Topografi	8 : 5 : 1
Vægtfordeling	$U(-2, 2)$
Klassificeringsværdi	$\{-1.0; +1.0\}$

Studenter

Antal	9
Topografi	8 : 3 : 1
Initial vægte	$U(-0.1; 0.1)$
Input	$N(0; 1)$

Optræning

Optræningsperiode	500 epoker
Træningssæt	300 eksempler
Krydsvalideringssæt	2000 eksempler
$(\alpha_1; \alpha_2)$	(0;0)

Self-repair og vægtdrift

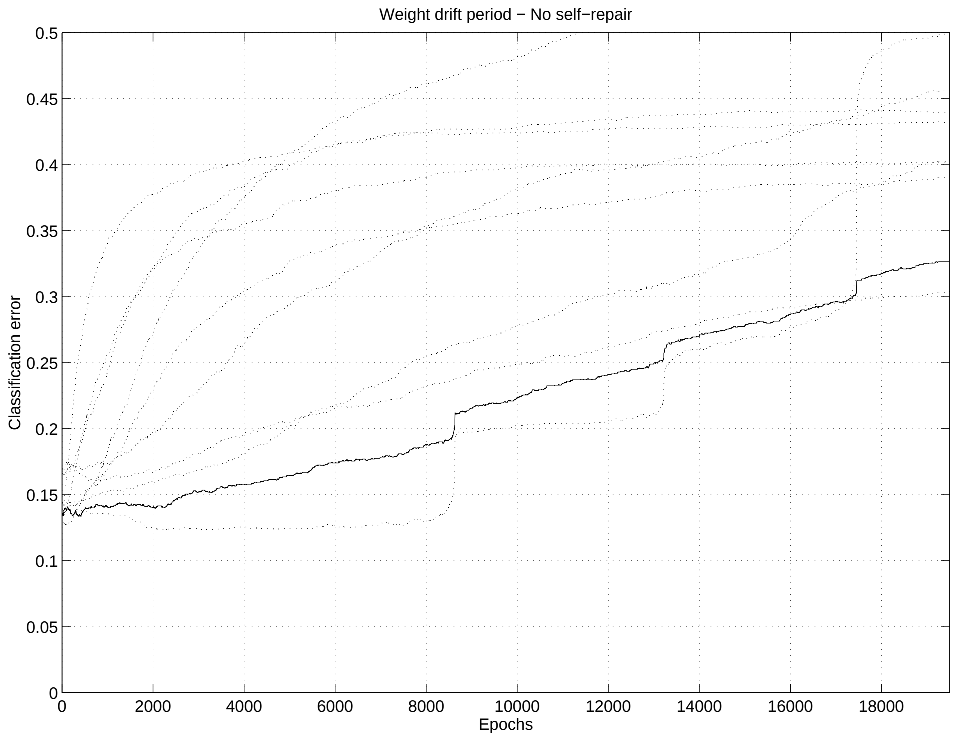
Vægtdriftperiode	9500 epoker
Krydsvalideringssæt	2000 eksempler
Self-repair sæt	100 eksempler
w_{drift}	$N(0.0; 3.0 \cdot 10^{-4})$

Tabel 5.7: Parametre til forsøg I-D

Opsætningen af forsøget ses af tabel 5.7. Der er her kun medtaget de vigtigste plot. For fuldstændighedens skyld er alle plot til dette forsøg listet i appendiks C.

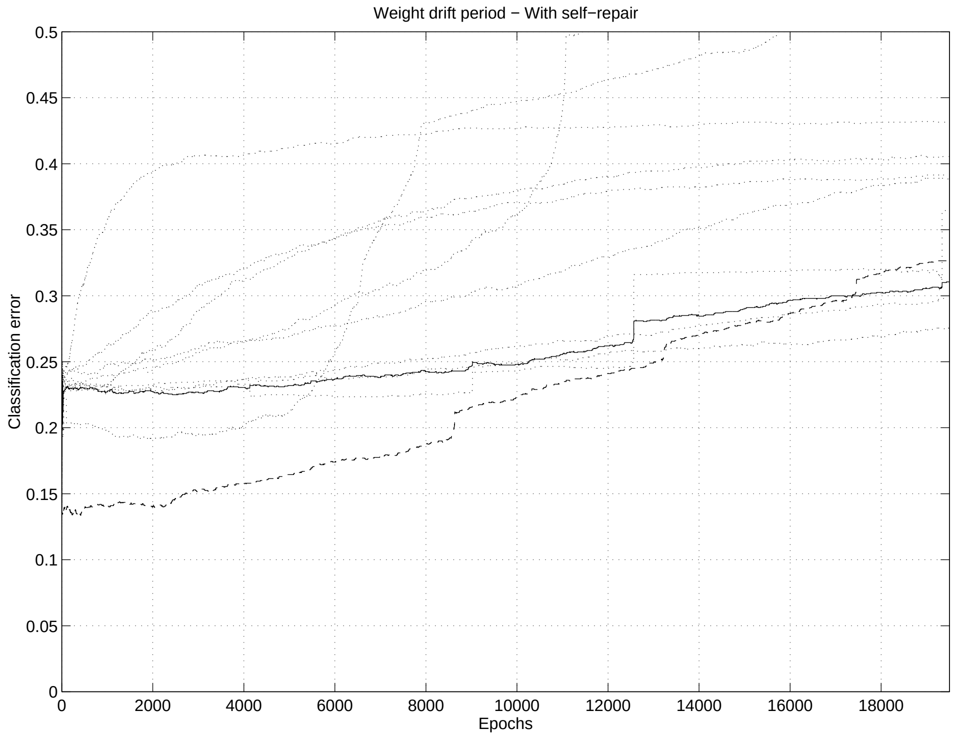
Der viser sig i dette forsøg ingen væsentlig forringelse af indlæringskapaciteten under optræningen. Studenterne opnår en consensus på ca. 13-14% fejl på krydsvalideringssættet, og det er mindst lige så godt som studenterne med 5, og 7 skjulte neuroner. Når de derefter udsættes for vægthenfald med self-repair, sker der et drastisk fald i studenternes præstationsevne. På 5-10 epoker øges krydsvalideringsfejlen til 22-23% (se figur 5.54). For at kunne forklare dette spring, må man sammenholde figur 5.55, og figur 5.56. Den første figur viser hvordan nettene er enige om et vilkårligt mønster lige efter endt optræning, den anden efter 60 epoker self-repair. Der er en meget tydelig forskel på de to plot, der går ud på at den smule inputmønstre studenterne var uenige om efter endt optræning, svarende til søjlerne $1 \ldots 8$, næsten helt forsvinder. Der sker en slags koordinering af viden så næsten alle net ved det samme. Denne koordinering sker også for studenter med 5, og 7 skjulte neuroner, men ikke nær så dramatisk.

Betragter man plottet over self-repair perioden, bemærker man udover det meget kraftige spring i præstation først i perioden, at plottet strækker sig over 19500 epoker. Til sammenligning, er de tilsvarende plot for studenter med 5, og 7 skjulte neuroner, kun over 9500 epoker. Man bemærker også at præstationen af studenterne uden self-repair er næsten lineær, med karakteristiske spring, i modsætning til de S-formede kurver for studenter med 5, og 7 skjulte neuroner. Betragter man figur 5.52, der viser studenternes præstationen, under vægtdrift perioden uden self-repair, ser man at der er én student der i de første 8000 epoker ikke mister præstation, og to andre hvis præstationer kun stiger ganske langsomt.

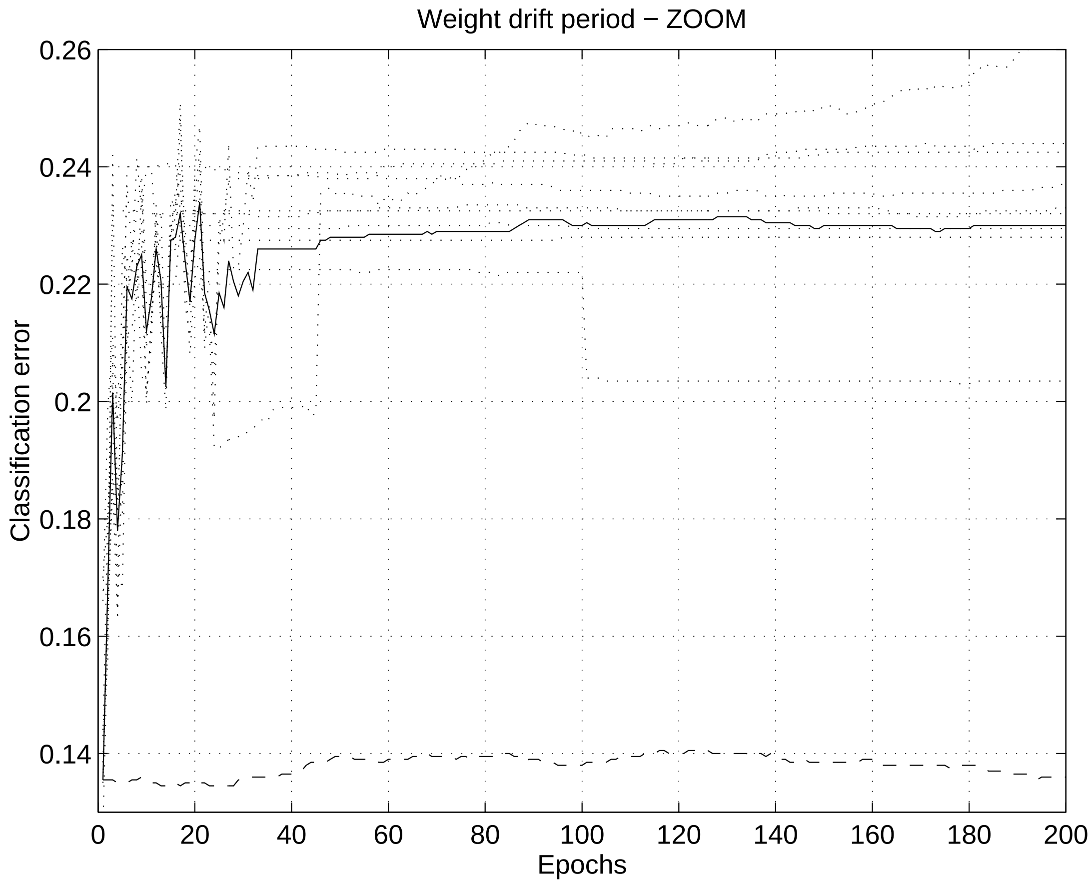


FIGUR 5.52: Plottet viser klassifikationsfejlen på studenternettene, når de ødelægges med en konstant rate, som funktion af epoker. For hver epoke lægges der en konstant værdi til vægtene. Værdien er forskellig fra vægt til vægt og vælges tilfældigt fra en normalfordeling. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie.

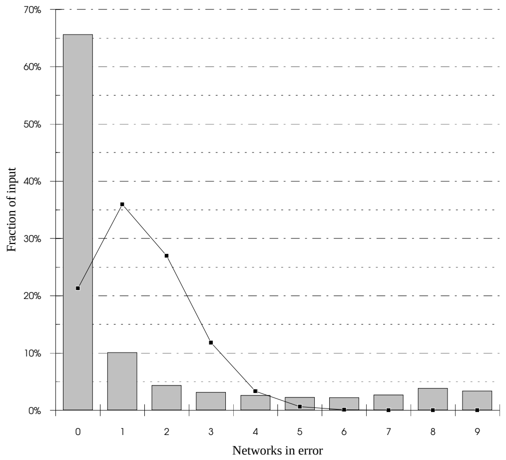
Da nettene under vægtdrift uden self-repair, bliver mere og mere ukorrelerede med hensyn til fejl, kan disse tre net tilsyneladende dække den del af inputrummet, som resten af nettene mister præstation for. Plottet viser, med stor tydelighed, styrken i ensembler, men gør det samtidigt vanskeligt at sammenligne netop denne kurve med de tilsvarende kurver for studenter med 5, og 7 skjulte neuroner.



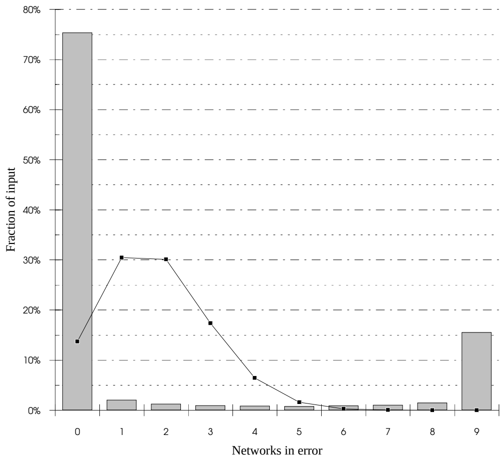
FIGUR 5.53: Plottet viser studenternettene når de udsættes for samme ødelæggelse som på figur 5.52. Her forsøger de dog at rette op på skaden ved at udføre self-repair med ensemblets consensus beslutning. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuldt optrukket linie, og consensus uden self-repair (fra figur 5.52) er markeret med stiplet linie.



FIGUR 5.54: Zoom af figur 5.53. Bemærk indsvingningsperioden.



FIGUR 5.55: $\mu(\theta)$-plot, efter endt optræning. Plottet viser sandsynligheden for at et vilkårligt input fejlklassificeres af n studenternet. $n = 0$ svarer derfor til sandsynligheden for at alle net korrekt klassificerer et vilkårligt input, og $n = N = 9$ er sandsynligheden for at et input fejlklassificeres af alle net. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 16%, og havde været ukorrelerede med hensyn til fejl. Søjlerne er målt over 100,000 tilfældige input.



FIGUR 5.56: $\mu(\theta)$-plot, efter 60 epoker self-repair. Plottet viser sandsynligheden for at et vilkårligt input fejlklassificeres af n net, hvor n er afbildet på x-aksen. Bemærk at nettene har koordineret deres viden således at næsten alle input falder i de 2 kategorier, hvor de enten klassificeres korrekt af alle net, eller fejlklassificeres af alle net (hhv. $n = 0$ og $n = N = 9$). Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde haft ukorreleret en krydsvalideringsfejl på 20%. Målt over 100,000 tilfældige input.

5.2.5 Diskussion af forsøg med idelle studenter

Der er i det indledende forsøg I-A vist at studenternes præstation efter 500 epokers optræning, er næsten uafhængig af antallet af den enkelte students skjulte neuroner.

Der er kørt 3 forsøg der skulle belyse hvordan antallet af neuroner påvirker ensemblets consensus under self-repair perioden.

Det er karakteristisk at nettene udviser en indsvingningsperiode i de første epoker af self-repair perioden. Under denne "indsvingningsperiode" stiger generalisationsfejlen markant. Indsvingningsperioden opstår formentlig når studenterne i begyndelsen af self-repair perioden udveksler information så de bliver mere ens. På figur 5.55 og 5.56 ser man et tydelig eksempel på dette fænomen.

Efter "indsvingningsperioden" viser forsøgene at studenter der udfører self-repair under vægtdriftperioden er i stand til at holde en næsten konstant consensus over en periode, hvor de enkelte net mister præstationsevne. Den stabile consensus skyldes at studenterne bliver mere ukorrelerede pga. vægtdriften.

Nedenfor ses en sammenligning af nogle karakteristika fra de 3 forsøg. Consensus er målt lige inden vægtdrift og self-repair sættes igang (epoke 0). Consensus er igen målt efter 300 epoker self-repair. Efter 300 epoker self-repair, har consensus stabiliseret sig i alle 3 forsøg. Den procentvise stigning i consensusfejl er vist i sidste kolonne.

Forsøg	epoke 0	epoke 300	stigning
I-B	14.5%	16%	10%
I-C	16%	19%	18%
I-D	14%	23%	64%

Forsøgene viser at indsvingningsperioden, har mindst effekt på ensemblet af studenter der har samme konfiguration som teacheren.

5.3 Hardware ANN self-repair

I dette afsnit beskrives forsøg udført med simuleringer af ensembler af hardware implementerede ANN. Studenternettene er her specielt karakteriseret ved:

- **Begrænset vægtområde**
  Modellen for et hardware ANN, foreskriver at der er en øvre og en nedre grænse for de værdier en vægt kan antage. Disse grænser er i de efterfølgende forsøg sat til: $(w_{\min}; w_{\max}) = (-1.0; +1.0)$.

- **Ordinær back-propagation**
  Til vægtopdatering bruges en hardware version af første ordens back-propagation (se evt. afsnit 4). Vægtopdatering efter mere sofistikerede algoritmer, er almindeligvis ikke så lette at implementere i hardware [Leh94].

- **Støj på vægtene**
  Der er støj på vægtene. Dette begrænser også nøjagtigheden (opløsningen) af vægtene.

5.3.1 Forsøg nr II-A

Teacher

Topografi	5 : 3 : 1
Vægtfordeling	$U(-2,2)$
Klassificeringsværdi	$\{-0.98; +0.98\}$

Studenter

Antal	9
Topografi	5 : 3 : 1
Initial vægte	$U(-0.1;0.1)$
Input	$N(0,1)$

Self-repair og vægtdrift

Vægtdriftperiode	7000 epoker
Krydsvalideringssæt	1000 eksempler
Self-repair sæt	100 eksempler
$(\eta_1; \eta_2)$	$(10^{-4}; 10^{-5})$
Δt	$5 \cdot 10^{-3} s$

Optræning

Optræningsperiode	1000 epoker
Træningssæt	300 eksempler
Krydsvalideringssæt	1000 eksempler
$(\alpha_1; \alpha_2)$	$(0; 0)$
$(\eta_1; \eta_2)$	$(10^{-3}; 10^{-4})$
β	0
k_1	$N(1.0; 6.0 \cdot 10^{-4})$
k_2	$N(2.0; 6.0 \cdot 10^{-4})$
k_3	$N(1.2; 6.0 \cdot 10^{-4})$
V_{offset}	$N(0.0; 9.6 \cdot 10^{-3})$
w_{offset}	$N(0.0; 9.6 \cdot 10^{-3})$
w_{noise}	$N(0.0; 1.2 \cdot 10^{-3})$
w_{drift}	$N(0.0; 3.0 \cdot 10^{-4}) \cdot s^{-1}$
Δt	$1 \cdot 10^{-4} s$
$w_{\text{min,max}}$	$\{-1.0; +1.0\}$

Tabel 5.8: Parametre til forsøg II-A

Parametrene til forsøg II-A kan ses i tabel 5.8. Det bemærkes at opdelingen er en smule anderledes end for ideelle studenter. Først gennemgås betydningen af felterne i tabellen:

Teacher

Topografi

Teacher nettets topografi er beskrevet ved 5 indgangsenheder, 3 skjulte neuroner, og 1 output neuron. Der er en ekstra, ikke medtalt, enhed i hhv. inputlaget og det skjulte lag. Denne enhed er fast sat til værdien +1 og implementerer tærskelværdien i det efterfølgende lag.

Vægtfordeling

Teachernettets vægte initialiseres med tilfældige tal uniformt fordelt i intervallet $[-2.0; +2.0]$. Tærskelværdien i output-laget justeres så teacher-nettet klassificerer input-rummet ligeligt i to klasser.

Klassificeringsværdi

Output af teacher-nettet, er sat til $\{-0.98; +0.98\}$. Denne værdi er med vilje valgt lavere end 1.00, idet der er offset på hardware studentens output-neuron. Hvis neuronen har et negativt offset, vil output aldrig kunne nå en klassifikationsværdi på +1.00.

Studenter

Antal

Der er 9 studenternet i ensemblet.

Topografi

Studenternettene har samme topografi som teacher-nettet.

Initial vægte

Studenternettene er initialiseret med tilfældige tal uniformt fordelt i intervallet $[-0.1; +0.1]$.

Input

Input til studenternettene er vektorer af normalfordelte, $N(0; 1)$, tilfældige tal.

Optræning

Optræningsperiode

Studenternettene bliver optrænet med gradient metoden i 1000 epoker.

Træningssæt

Træningssættet er på 300 eksempler. For hver epoke ser den enkelte student alle 300 eksempler én gang.

Krydsvalideringssæt

Krydsvalideringssættet er på 1000 eksempler.

$(\alpha_1; \alpha_2)$

Vægthenfaldsleddene α_1 og α_2 (se evt. formel (3.2), (3.18) og (3.18)) er sat til 0. Der er altså ikke vægthenfald.

$(\eta_1; \eta_2)$

Under optræningen er skridtlængden i vægtopdateringsalgoritmen 4.5

sat til $1 \cdot 10^{-3}$ for vægtene imellem input og skjulte neuroner. Skridtlængden for vægtene mellem skjulte- og outputneuronen er sat til $1 \cdot 10^{-4}$.

β
: Momentet β er sat til 0. Der er altså ikke moment.

k_1
: Konstanten k_1, i formlen til beregning af neuron-værdien, formel (4.2), er for hver neuron i studenternettet sat til et tilfældigt tal, valgt fra en normalfordeling, med parametre: $N(1.0; 6.0 \cdot 10^{-4})$

k_2
: Konstanten k_2, i formlen til beregning af neuron-værdien, formel (4.2), er for hver neuron i studenternettet sat til et tilfældigt tal, valgt fra en normalfordeling, med parametre: $N(2.0; 6.0 \cdot 10^{-4})$

k_3
: Konstanten k_3, i formlen til beregning af neuron-værdien, formel (4.2), er for hver neuron i studenternettet sat til et tilfældigt tal, valgt fra en normalfordeling, med parametre: $N(1.2; 6.0 \cdot 10^{-4})$

V_{offset}
: Konstanten V_{offset}, i formlen til beregning af neuron-værdien, formel (4.2), er for hver neuron i studenternettet sat til et tilfældigt tal, valgt fra en normalfordeling, med parametre: $N(0.0; 9.6 \cdot 10^{-3})$

w_{offset}
: I formlen for beregning af en vægts resulterende værdi, formel (4.3), indgår der et vægtoffset. Hver vægt har et fast offset. Dette offset vælges som et tilfældigt tal, valgt fra en normalfordeling med parametre: $N(0.0; 9.6 \cdot 10^{-3})$.

w_{noise}
: Der er støj på vægtene. Hver gang en vægt skal bruges i en beregning adderes et støjbidrag til den ideale vægt. Støjen vælges hver gang som et nyt tilfældigt tal, valgt fra en normalfordeling med parametre: $N(0.0; 1.2 \cdot 10^{-3})$

w_{drift}
: Vægten driver med en rate der er konstant og fast for hver enkelt vægt. Raten vælges som et tilfældigt tal valgt fra en normalfordeling med parametre: $N(0.0; 3.0 \cdot 10^{-4}) \cdot s^{-1}$.

Δt
: Δt er tidsafstanden mellem to opdateringer af vægtene. Vægtdrift implementeres ved at addere et konstant bidrag til hver vægt. Dette bidrag findes som produktet af Δt og w_{drift}.

$w_{\text{min,max}}$
: Hver vægt har en nedre og øvre grænse. Denne grænse er sat til

$\{-1.0; +1.0\}$.

Self-repair og vægtdrift

Vægtdriftperiode
Der køres self-repair, og "vægt-drift uden self-repair" i 7000 epoker.

Krydsvalideringssæt
Der krydsvalideres på 1000 eksempler. Det antages at dette sæt dækker input-rummet, så krydsvalideringsfejlen, er repræsentativ for generalisationsfejlen.

Self-repair sæt
For hver epoke, gennemløbes hvert studenternet af 100 eksempler fra input-rummet.

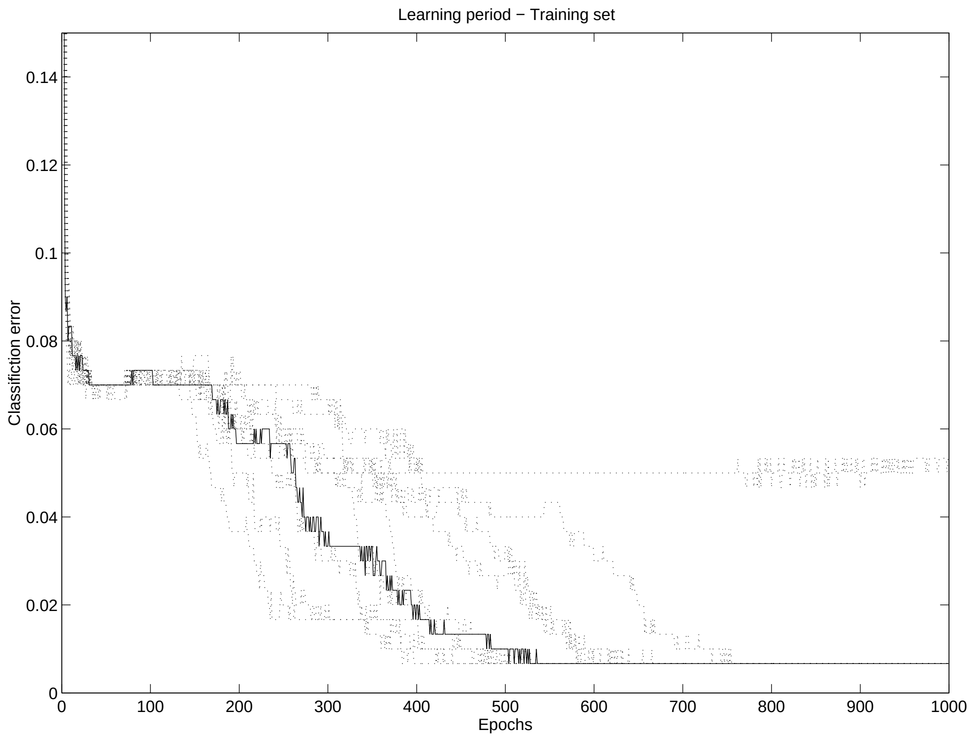
$(\eta_1; \eta_2)$
I self-repair perioden er skridtlængden i vægtopdateringsalgoritmen, formel (4.5) sat til $1 \cdot 10^{-4}$ for vægtene imellem input og skjulte neuroner. Skridtlængden for vægtene mellem skjulte- og outputneuronen er sat til $1 \cdot 10^{-5}$.

Δt
Tiden mellem to opdateringer af en vægt, er sat til $5 \cdot 10^{-3} s$. Ved at multiplicere dette tal, med drift-raten w_{drift} fås driften siden sidste opdatering.

Optræning af studenter til forsøg II-A

Studenterne er optrænet i 1000 epoker. Studenternes præstation på hhv. træningssæt og krydsvalideringssæt ses på figur 5.57 og figur 5.58. På figurerne er de enkelte studenters præstation optegnet med punkteret linie, og ensemblets consensus er optegnet med fuldt optrukket linie. Man ser tydeligt at ensemblet ikke er bedre end studenterne. Faktisk er der på de to figurer en overordentlig høj enighed mellem studenterne om hvordan "verden ser ud". Dette bekræftes også af korrelations- og $\mu(\theta)$-plottet på figur 5.59 og 5.60. Man ser her at studenterne er utroligt enige. Med undtagelse af én student er korrelationsplottet en "flad kasse". Studenterne opnår en rimelig god generaliseringsevne, gennemsnitlig 4% fejl, men når det er de samme 4% alle nettene tager fejl på, er der ingen gevinst ved at bruge ensemblet.

På figur 5.63 og 5.64 vises vægtudviklingen for student nr. 3. Plottet er typisk for et hardware studenternet under optræning. Under optræningen sker det at en vægt når til ydergrænsen for værdi-området. Man ser da ofte at der er andre vægte der "skifter retning", for derved at kompensere for vægtbegrænsningen.

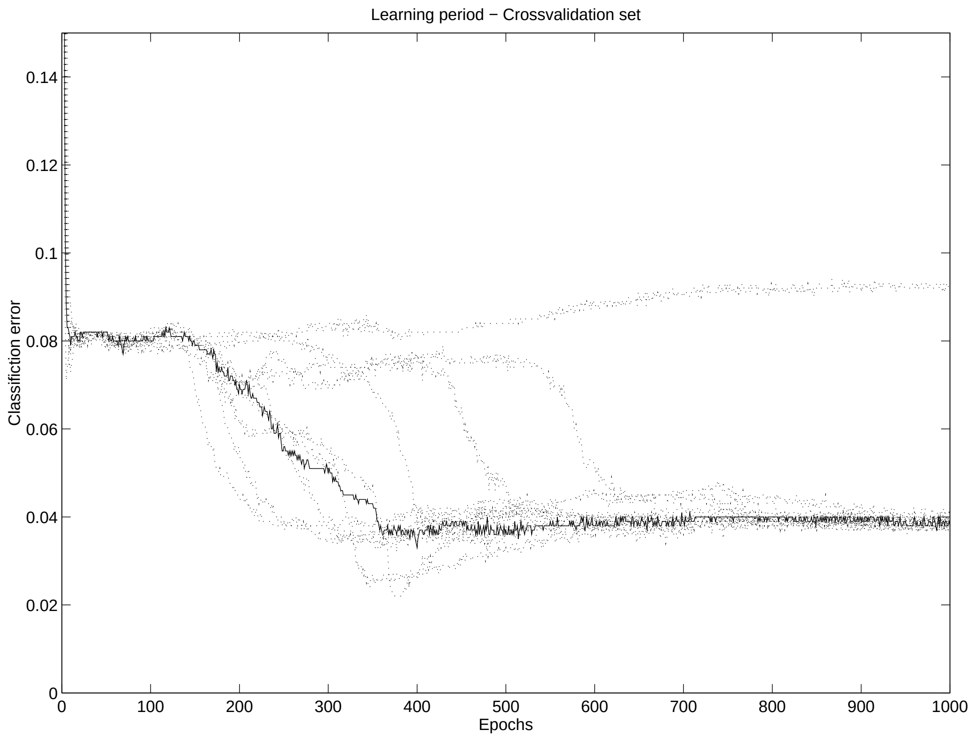


FIGUR 5.57: Plottet viser klassifikationsfejlen på træningssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 1000 epoker.

Vægtdrift og self-repair

Studenternes præstation under "vægtdrift uden self-repair", ses på figur 5.61. Figuren viser de typiske karakteristika for ensemblers præstation, når ensemblets net udsættes for vægtdrift. Ensemblets consensus holder sig stabilt et godt stykke, selv om de enkelte nets præstationer bliver markant dårligere i samme tidsrum. Det er også ret typisk at de enkelte net ikke bliver dårligere i et konstant tempo, men tager nogle ryk over en kortere periode. Efter et stykke tid er de enkelte studenternet så ødelagte at det går ud over ensemblets præstation. Ved ideelle studenter ser man ofte at der er perioder hvor præstationen stort set ikke ændrer sig. Dette ses sjældent med studenterne i hardware simuleringen.

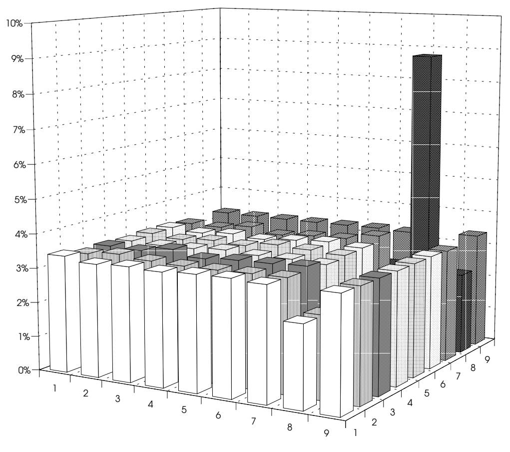
Der er optaget et korrelations- og $\mu(\theta)$-plot efter 2000 epoker "vægtdrift uden self-repair". På figur 5.69 og 5.70 ses disse plot, bemærk det voldsomme skift i akserne i forhold til figur 5.59 og 5.60. Man ser her at studenterne i ensemblet tager fejl på forskellige dele af input-rummet. Der er nu næsten ingen input, som alle net tager fejl på (Søjle $n = 9$ i figur 5.70 er "væk"). Hvor det før var 90% af input-rummet der korrekt blev genkendt af alle net, er det nu kun



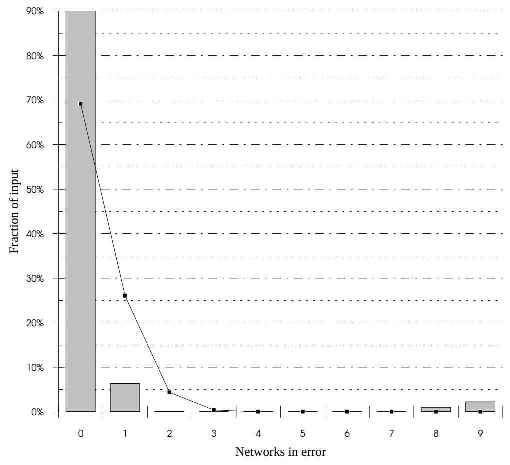
FIGUR 5.58: Plottet viser klassifikationsfejlen på krydsvalideringssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 1000 epoker.

$\approx$ 32%. Efter 7000 epoker med vægtdrift er stort set al information om det oprindelige problem forsvundet (figur 5.73 og 5.74). Ensemblets præstation er her ret lig den gennemsnitlige præstation (46%), selv om studenterne er næsten perfekt ukorrelerede mht. fejl. Dette skyldes at der nu er 2 studenter der tager fejl i mere end halvdelen af input-rummet. Når nogle af studenterne tager fejl i mere end halvdelen af input-rummet bryder den første af de to forudsætninger sammen bag ideen om at ensemblet er bedre end gennemsnittet.

Betragter man studenternes præstation, når de udfører self-repair (figur 5.62), er det meget iøjnefaldende, at studenterne stort set er ens. Dette bekræftes af korrelations- og $\mu(\theta)$-plottet optaget efter 2000 og 7000 epoker self-repair (figur 5.71, 5.72, 5.75 og 5.76). Man ser at self-repair ikke forhindrer studenterne i at blive dårligere, det nedsætter blot hastigheden. Self-repair forhindrer også at én student bliver meget værre end de andre. Den enlige student der havde en højere krydsvalideringsfejl, efter endt optræning, er “bragt tilbage i folden” inden 2000 epoker. På figurerne 5.65, 5.66, 5.67 og 5.68 ses vægtudviklingen for student nr 3, under “vægtdrift uden self-repair”, og “vægtdrift med self-repair”. For



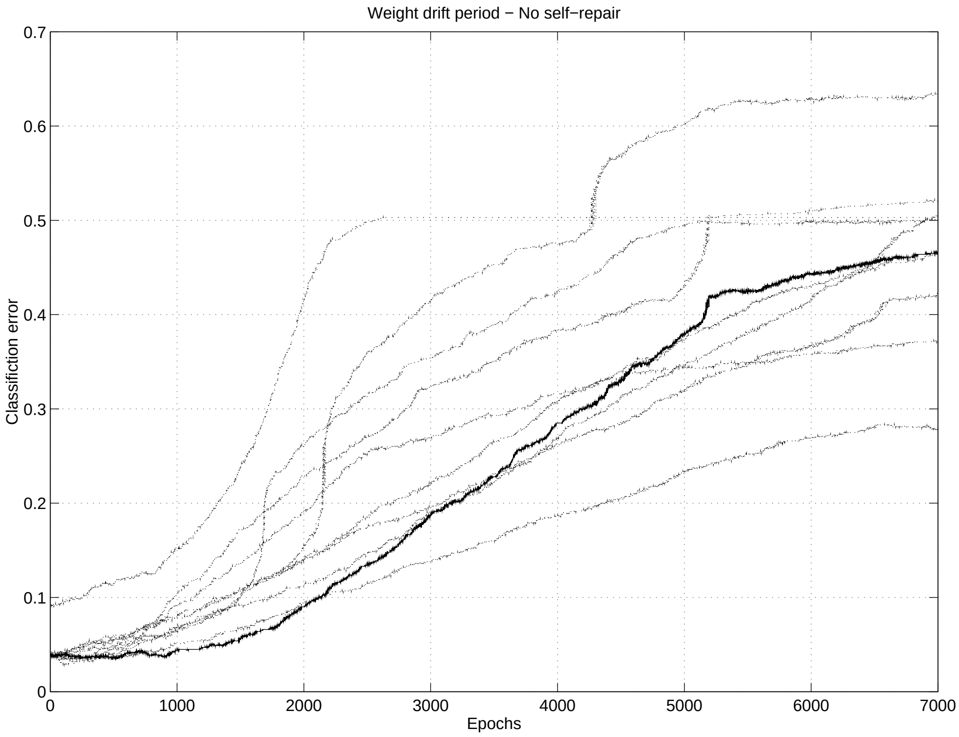
FIGUR 5.59: Dette plot viser korrelationen mellem studenterne, efter endt optræning. Hver søjle viser sandsynligheden for at de tilsvarende net samtidigt tager fejl på tilfældigt input-mønster. I diagonalen optræder derfor et estimat af de enkelte nets krydsvalideringsfejl. Korrelationsplottet er baseret på 100,000 tilfældige inputmønstre.



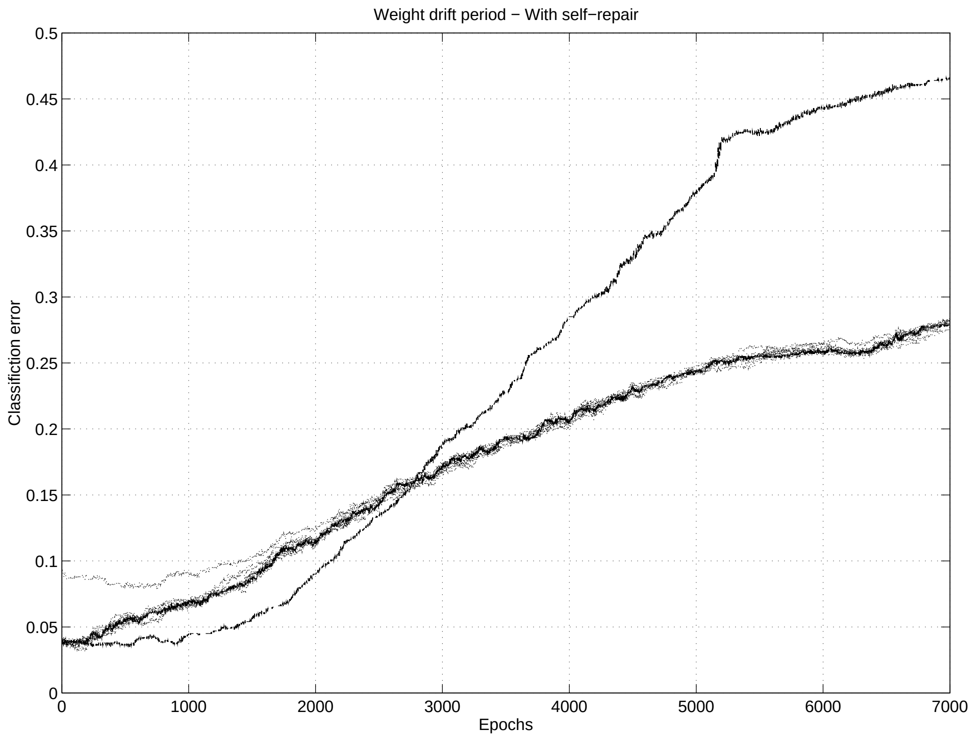
FIGUR 5.60: $\mu(\theta)$-plot der viser sandsynligheden for at et vilkårligt input fejlklassificeres af n studenternet. $n = 0$ svarer derfor til sandsynligheden for, at et input klassificeres korrekt af alle studenter, og $n = N = 9$ svarer til sandsynlingheden for, at et input fejlklassificeres af alle net. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 4%, og havde været ukorrelerede med hensyn til fejl. Søjlerne er målt over 100,000 tilfældige input.

output-vægtenes vedkommende er der stort set ingen forskel, for input-vægtene er det ikke helt så entydigt. Det ser dog ud som om at vægtene driver under self-repair perioden. Dette kan skyldes at man med gradient metoden specificerer en learning-rate. Derved er vægtopdateringerne proportionale med gradienten, med en fast proportionalitetsfaktor. Hvis denne rate er for lav, vil man ikke kunne rette op på vægtdriften. Er den for høj "skyder man over målet".

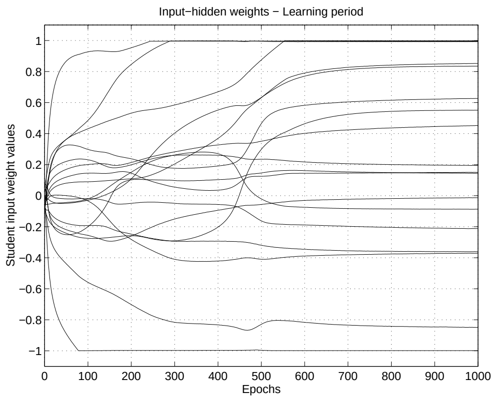
For at belyse denne egenskab, er der sat 2 ekstra delforsøg op. Udgangspunktet for hver af disse 2 forsøg er de studenter, der her er opnået efter endt optræning (se figur 5.58). I det ene forsøg, er learning-rate sat op med en faktor 10 for både input- og outputvægte, i det andet er al støj og offsetværdier på vægte og neuroner fjernet. Resultatet ses på figur 5.77. Man ser af denne figur at hvis learning-rate sættes op fås en markant dårligere præstation. Hvis man fjerner støj og offset får man en bedre præstation, men studenterne bliver alligevel dårligere med tiden.



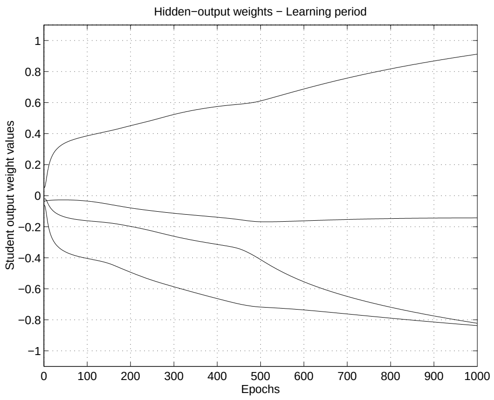
FIGUR 5.61: Plottet viser klassifikationsfejlen på krydsvalideringssættet som funktion af epoker. Studenternettene ødelægges med en konstant rate. For hver epoke lægges der en konstant værdi til vægtene. Værdien er forskellig fra vægt til vægt og vælges tilfældigt fra en normalfordeling. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuldt optrukket linie.



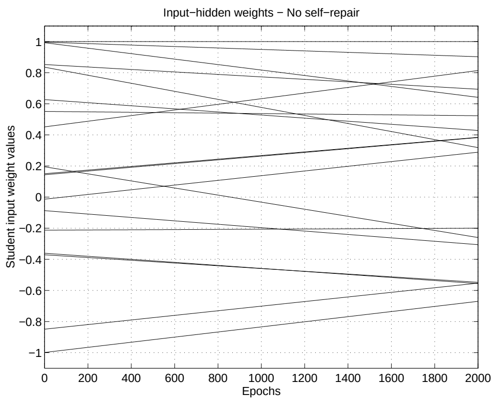
FIGUR 5.62: Plottet viser klassifikationsfejlen på krydsvalideringssættet som funktion af epoker. Studenternettene udsættes for samme ødelæggelse som på figur 5.61, men forsøger at rette op på skaden ved at udføre self-repair med ensemblets consensus beslutning. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuldt optrukket linie. Ensemblets consensus uden self-repair er markeret med stiplet linie.



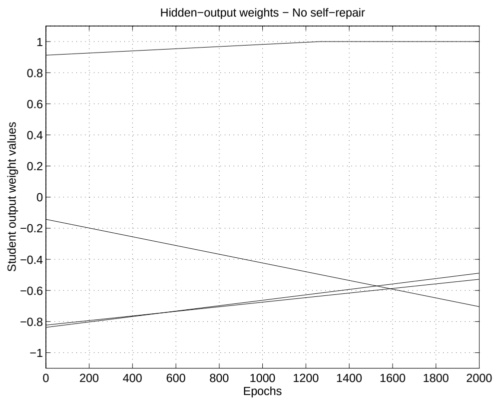
FIGUR 5.63: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under optræningen.



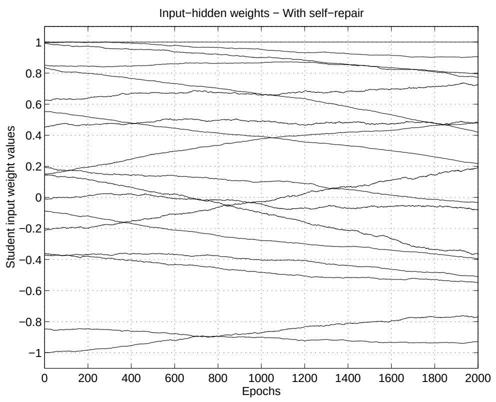
FIGUR 5.64: Skjulte-output vægtværdier for studenten fra figur 5.63, under optræningen.



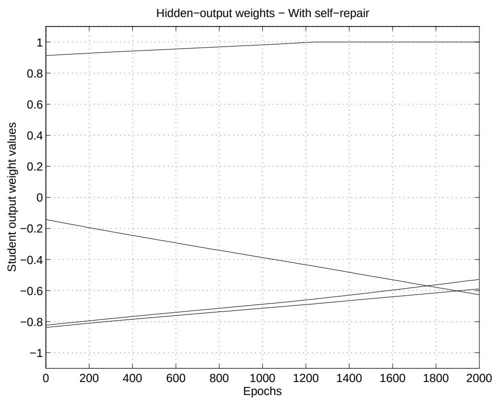
FIGUR 5.65: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under de første 2000 epoker af vægtdrift perioden. Der er ikke self-repair. Man ser, som ventet, at vægtene driver med en konstant rate.



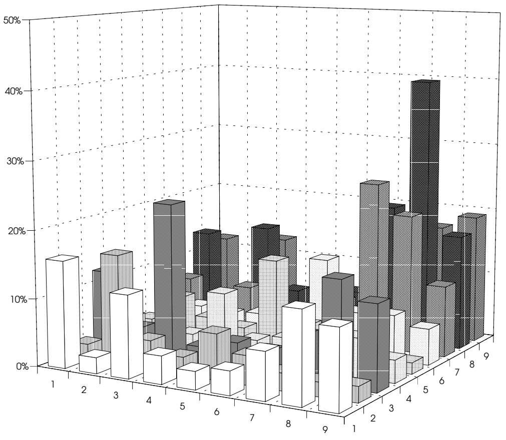
FIGUR 5.66: Der er plottet skjulte-output vægtværdier for studenten fra figur 5.65, under de første 2000 epoker af vægtdrift perioden.



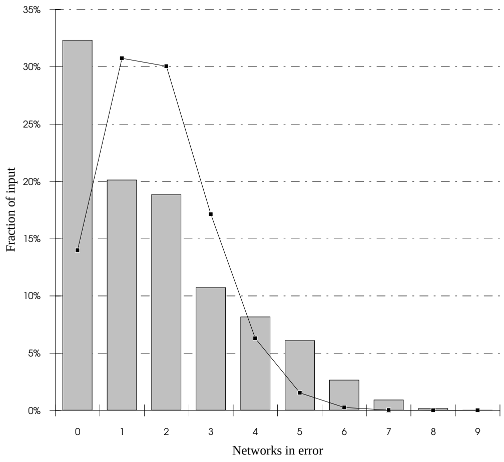
FIGUR 5.67: Der er plottet en typisk students input-skjulte vægtværdier, som funktion af epoker, under de første 2000 epoker af self-repair perioden.



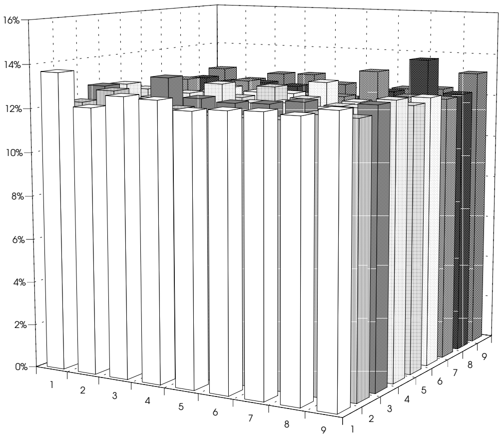
FIGUR 5.68: Skjulte-output vægtværdier for studenten fra figur 5.67, under de første 2000 epoker af self-repair perioden.



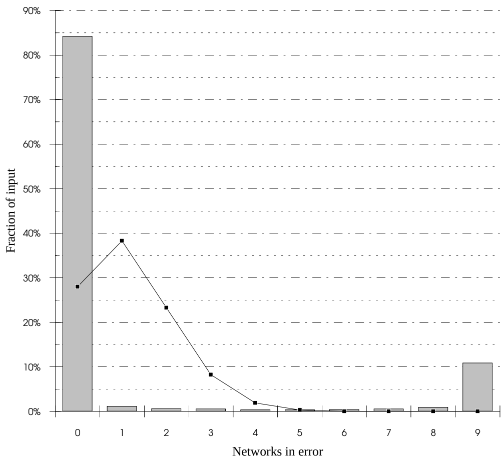
FIGUR 5.69: Korrelationen efter 2000 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



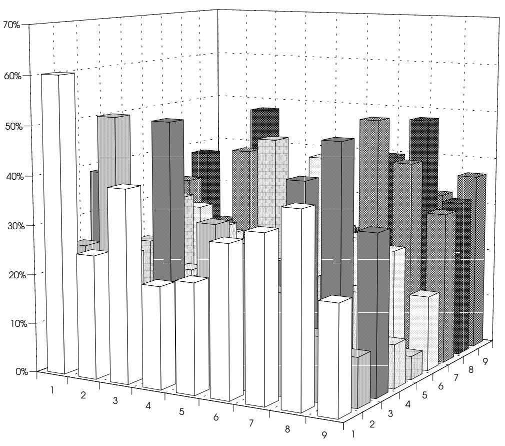
FIGUR 5.70: $\mu(\theta)$-plot, efter 2000 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 20%. Målt over 100,000 tilfældige input.



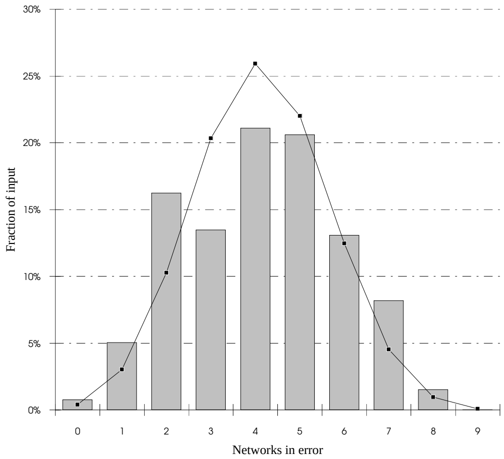
FIGUR 5.71: Korrelationen mellem studenterne, efter 2000 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



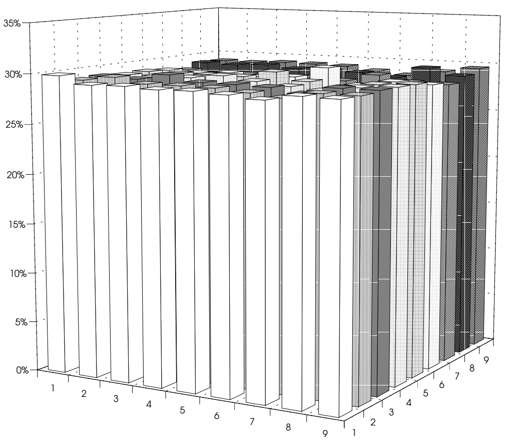
FIGUR 5.72: $\mu(\theta)$-plot, efter 2000 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 13%. Målt over 100,000 tilfældige input



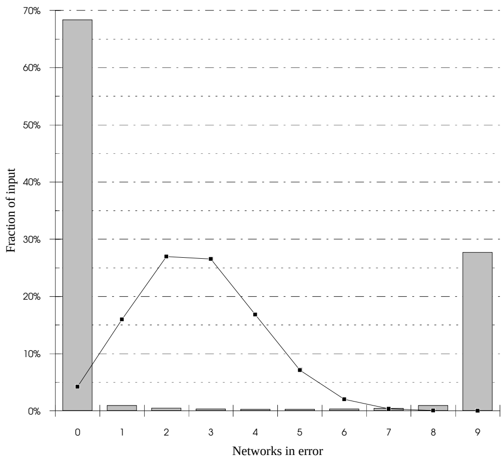
FIGUR 5.73: Korrelationen mellem studenterne, efter 7000 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



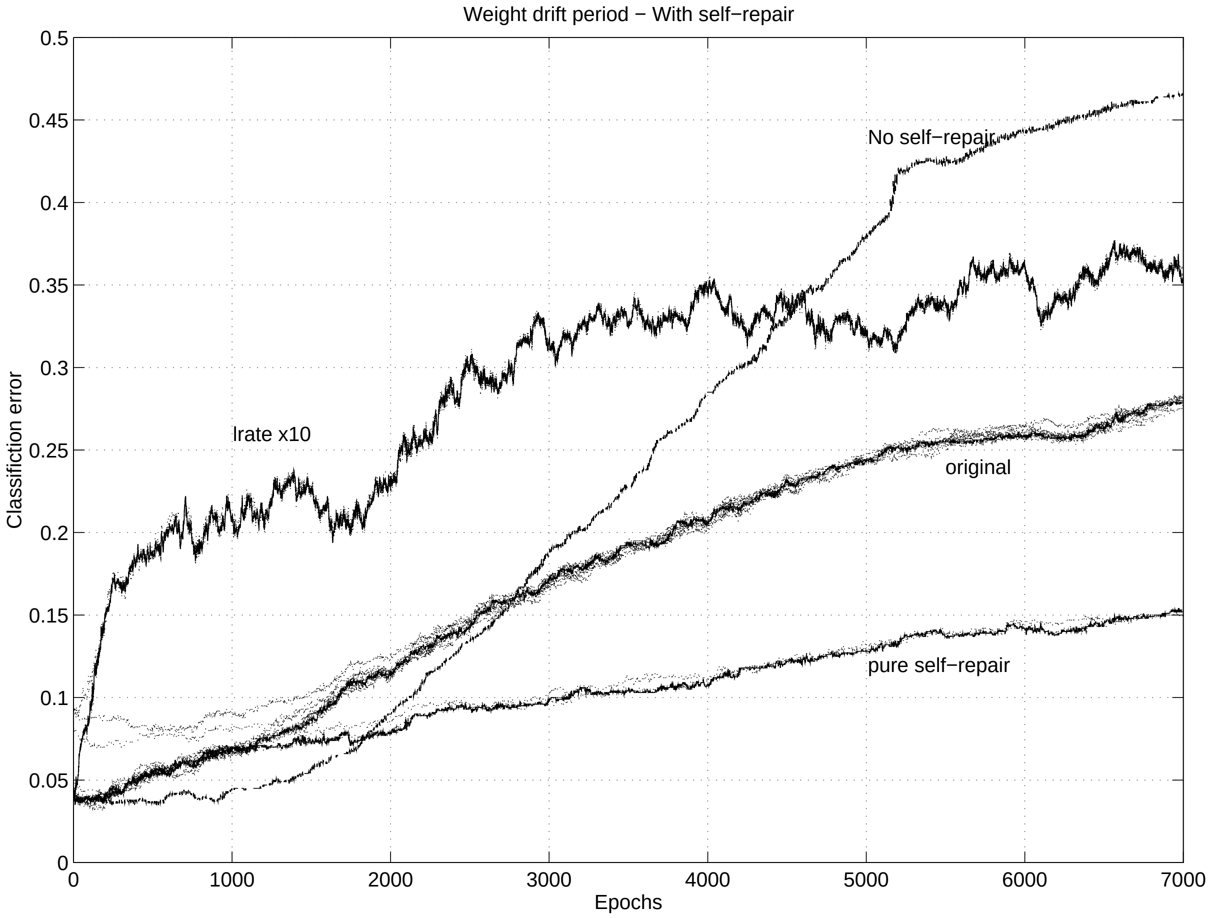
FIGUR 5.74: $\mu(\theta)$-plot, efter 7000 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, for en ukorreleret krydsvalideringsfejl på 46% Målt over 100,000 tilfældige input.



FIGUR 5.75: Korrelationen mellem studenterne, efter 7000 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



FIGUR 5.76: $\mu(\theta)$-plot, efter 7000 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 30%. Målt over 100,000 tilfældige input.



FIGUR 5.77: Der er plottet krydsvalideringsfejlen for 3 forskellige delforsøg. Forsøgene går ud på at udsætte de optrænede studenter fra forsøg II-A for forskellige påvirkninger under self-repair perioden. Det første forsøg (mærket `original`) er reference præstationen fra forsøg II-A. I det andet forsøg (mærket `lrate x10`) er “learning-rate” konstanterne $(\eta_1; \eta_2)$ sat op med en faktor 10. I det tredie forsøg (mærket `pure self-repair`) er vægtdrift, støj og offset på vægte og neuroner sat til 0. Nettenes præstation uden self-repair er mærket `No self-repair`.

5.3.2 Forsøg nr II-B

Teacher

Topografi	5 : 3 : 1
Vægtfordeling	$U(-2, 2)$
Klassificeringsværdi	$\{-0.98 ; +0.98\}$

Studenter

Antal	9
Topografi	5 : 3 : 1
Initial vægte	$U(-0.1 ; 0.1)$
Input	$N(0, 1)$

Self-repair og vægtdrift

Vægtdriftperiode	7000 epoker
Krydsvalideringssæt	1000 eksempler
Self-repair sæt	100 eksempler
$(\eta_1 ; \eta_2)$	$(10^{-4} ; 10^{-5})$
Δt	$5 \cdot 10^{-3} s$

Optræning

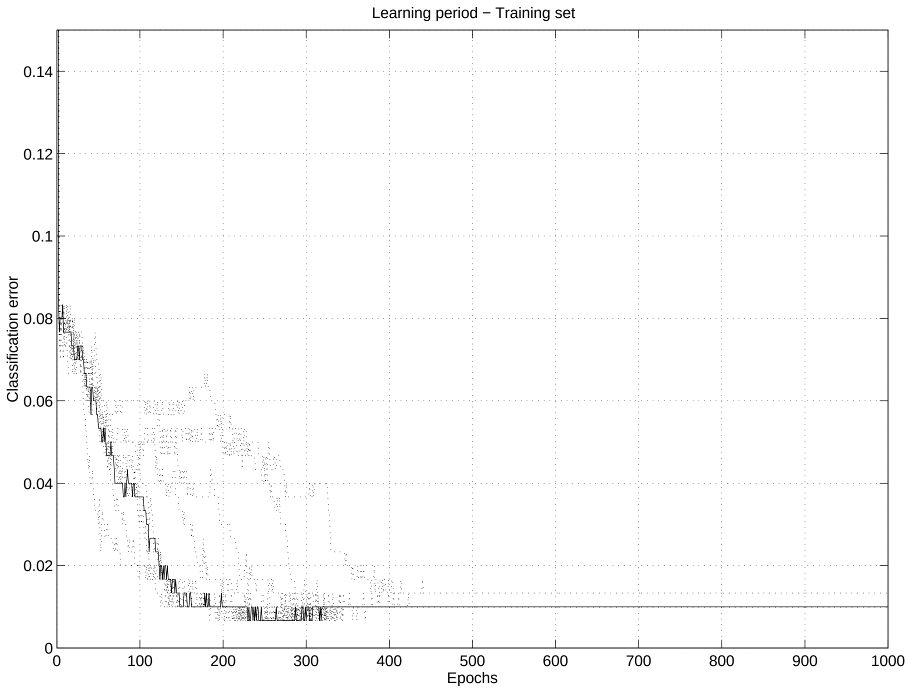
Optræningsperiode	1000 epoker
Træningssæt	300 eksempler
Krydsvalideringssæt	1000 eksempler
$(\alpha_1 ; \alpha_2)$	$(0 ; 0)$
$(\eta_1 ; \eta_2)$	$(10^{-3} ; 10^{-4})$
β	0
k_1	$N(1.0 ; 6.0 \cdot 10^{-4})$
k_2	$N(4.0 ; 6.0 \cdot 10^{-4})$
k_3	$N(1.2 ; 6.0 \cdot 10^{-4})$
V_{offset}	$N(0.0 ; 9.6 \cdot 10^{-3})$
w_{offset}	$N(0.0 ; 9.6 \cdot 10^{-3})$
w_{noise}	$N(0.0 ; 1.2 \cdot 10^{-3})$
w_{drift}	$N(0.0 ; 3.0 \cdot 10^{-4}) \cdot s^{-1}$
Δt	$1 \cdot 10^{-4} s$
$w_{\text{min,max}}$	$\{-1.0 ; +1.0\}$

Tabel 5.9: Parametre til forsøg II-B

Parametrene til forsøg II-B kan ses i tabel 5.9. I dette forsøg er parametrene præcis de samme som for forsøg II-A, med undtagelse af k_2. I dette forsøg er k_2 sat op med en faktor 2 i forhold til forsøg II-A. For ANN implementeret i hardware kan man ikke umiddelbart ændre grænserne for en vægts værdiområde. Så for at udvide, eller indskrænke det effektive vægtområde kan man "skrue" på k_2-faktoren. Man gør simpelthen aktiveringsfunktionen tanh() stejlere, så en neurons output bliver mere følsom for vægtændringer. Man skal her bemærke at samtidig med at man udvider vægtområdet, forstærker man støjbidraget.

Optræning af studenter til forsøg II-B

Studenternes præstation på hhv. træningssæt og krydsvalideringssæt ses på figur 5.78 og figur 5.79. På figurerne er de enkelte studenters præstation optegnet med punkteret linie, og ensemblets consensus er optegnet med fuldt optrukket linie. Studenternes præstationer er også i dette forsøg næsten sammenfaldende. Der er dog den forskel at den ene student der i forsøg II-A, er lidt dårligere end resten, er blevet en smule bedre. Vægtudviklingen under optræningen, for student nr. 3, ses på figur 5.80 og 5.81. Sammenlignes disse to figurer med de tilsvarende (samme student) for forsøg II-A, figur 5.63 og 5.64 på side 94, ser man at vægtene nu hurtigere finder et stabilt forløb. Specielt for output-vægtene ser man at vægtene har lavere værdier. Dette må tilskrives den højere k_2 værdi.

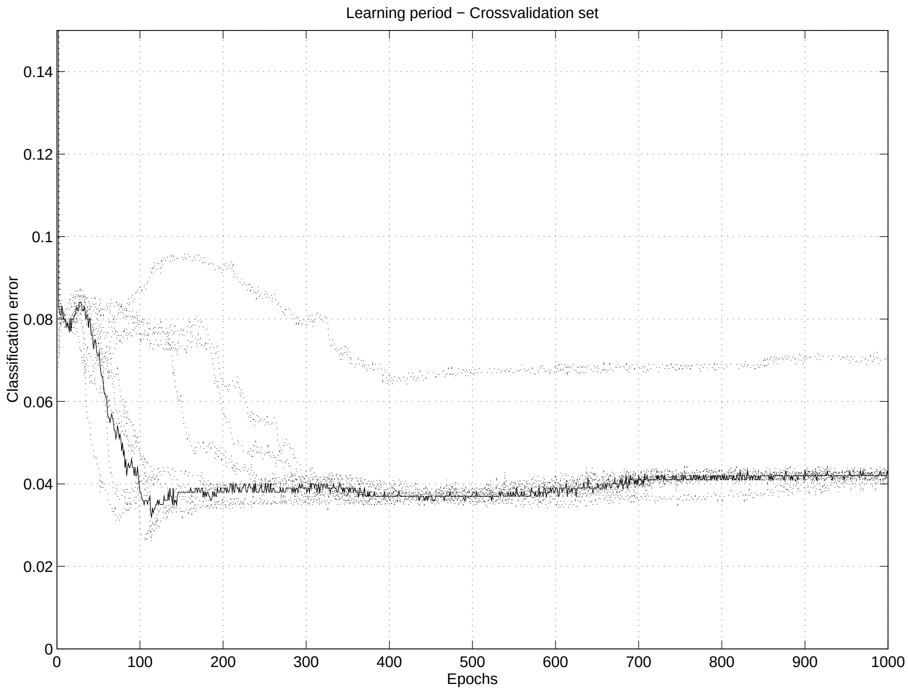


FIGUR 5.78: Plottet viser klassifikationsfejlen på træningssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 1000 epoker.

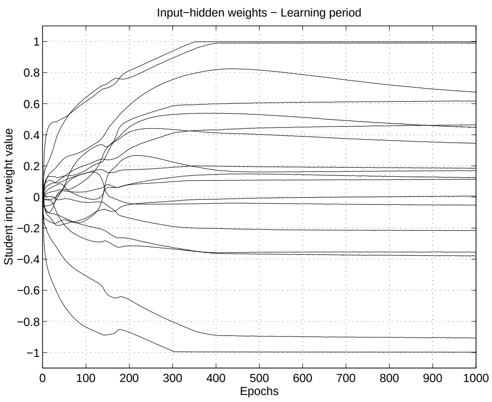
Vægtdrift og self-repair

Studenternes præstation under "vægtdrift uden self-repair", ses på figur 5.82. Sammenlignes med figur 5.61, ser man at vægtdriften hurtigere ødelægger studenterne. Consensus udviser samme karakteristika, som for forsøg II-A. Man ser endog at i de første 500-700 epoker falder consensus en smule, som følge af at studenterne bliver mere ukorrelerede mht. fejl, samtidig med at de enkelte studenter stadig er rimeligt gode. På figur 5.84 ses et udsnit af figur 5.82. På denne figur er de enkelte nets præstationer markeret med fuldt optrukket linie, og consensus med punkteret linie.

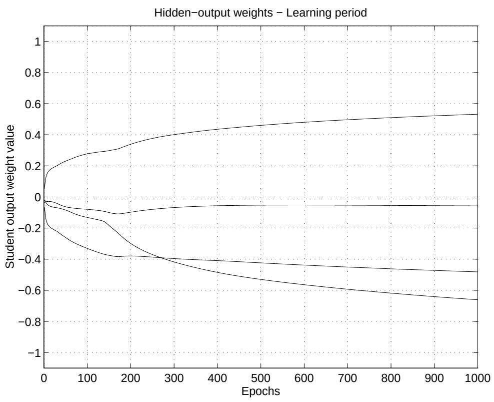
Betragter man studenternes præstation, når de udfører self-repair (figur 5.83), er det meget iøjnefaldende, at studenterne stort set er ens. Sammelignes figuren med den tilsvarende for forsøg II-A, figur 5.62, ser man at studenterne når 20% klassifikations fejl ved ca. 2000 epoker, mod 20% fejl ved ca. 4000 epoker i forsøg II-A. Efter 7000 epoker er fejlniveauet dog næsten det samme for begge forsøg.



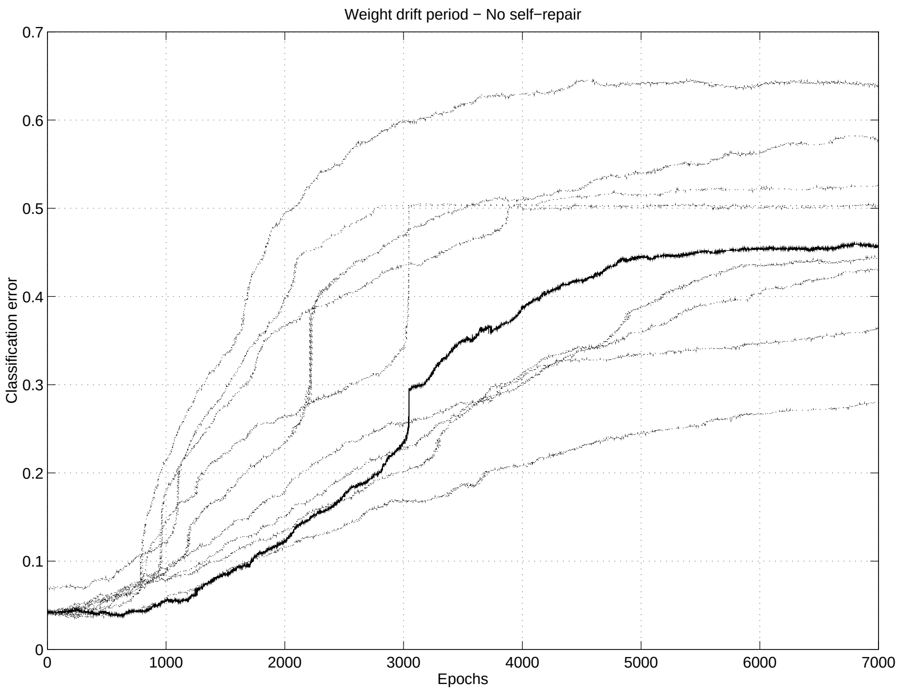
FIGUR 5.79: Plottet viser klassifikationsfejlen på krydsvalideringssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 1000 epoker.



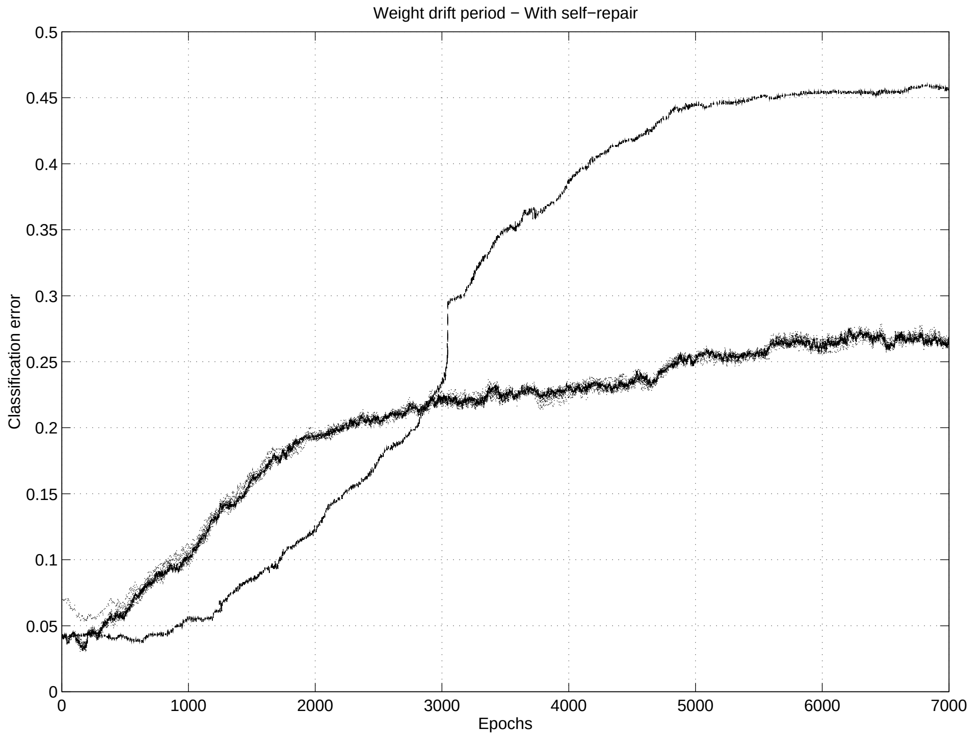
FIGUR 5.80: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under optræningen.



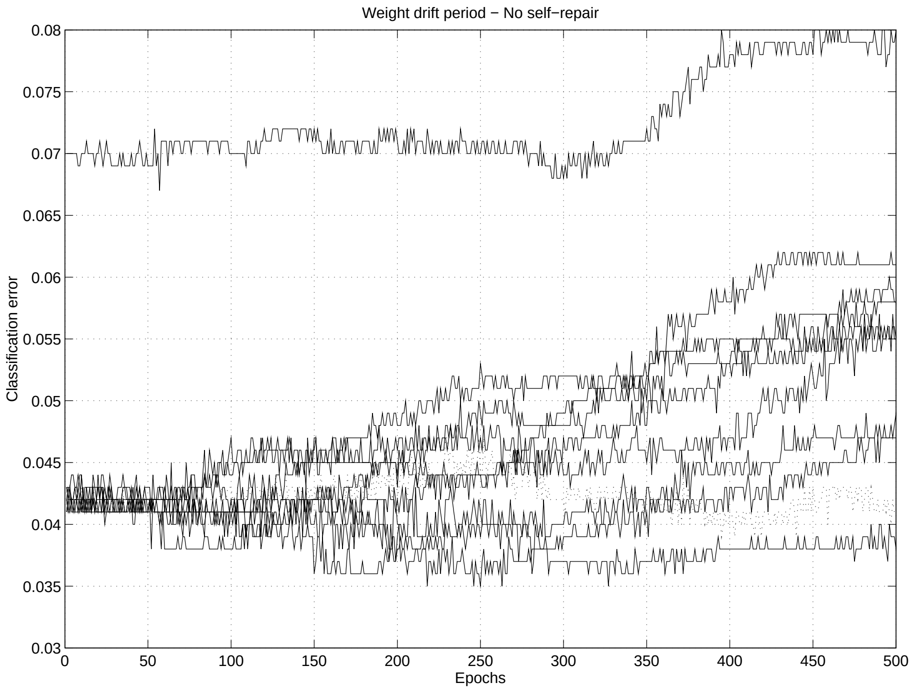
FIGUR 5.81: Skjulte-output vægtværdier for studenten fra figur 5.80, under optræningen.



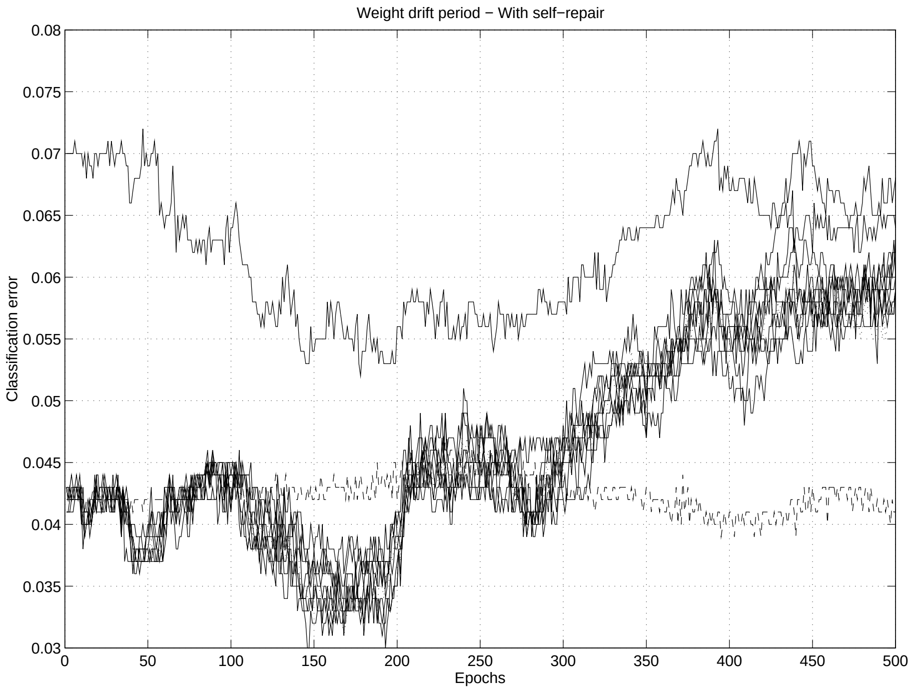
FIGUR 5.82: Plottet viser klassifikationsfejlen på krydsvalideringssættet som funktion af epoker. Studenter nettene ødelægges med en konstant rate. For hver epoke lægges der en konstant værdi til vægtene. Værdien er forskellig fra vægt til vægt og vælges tilfældigt fra en normalfordeling. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie.



FIGUR 5.83: Plottet viser klassifikationsfejlen på krydsvalideringssættet som funktion af epoker. Studenter nettene udsættes for samme ødelæggelse som på figur 5.82, men forsøger at rette op på skaden ved at udføre self-repair med ensemblets consensus beslutning. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie. Ensemblets consensus uden self-repair er markeret med stiplet linie.



FIGUR 5.84: Figuren viser et udsnit af figur 5.82. Bemærk at der er omvendt signatur i forhold til de øvrige figurer. De enkelte studenters præstationer er markeret med fuldt optrukket linie. Ensemblets consensus er markeret med punkteret linie.



FIGUR 5.85: Figuren viser et udsnit af figur 5.83. Bemærk at der er omvendt signatur i forhold til de øvrige figurer. De enkelte studenters præstationer er markeret med fuldt optrukket linie. Ensemblets consensus er markeret med punkteret linie.

5.3.3 Diskussion af forsøg med hardware studenter

Det har været stort set umuligt at optræne studenterne, så de tog fejl på forskellige dele af input-rummet. I alle de forsøg der er foretaget med simulering af hardware net (dermed menes også forsøg der ikke er rapporteret), er studenterne utroligt enige efter endt optræning. Denne enighed synes uafhængig af opsætningen af forsøget, inklusiv om antallet af skjulte neuroner varierer fra student til student.

Studenternes enighed overtræder en af forudsætningerne, for at et ensemble kan få en bedre consensus beslutning end den bedste af de individuelle net, og forsøgene viser da også at det enkelte nets beslutning er lige så gode som consensus.

Betragter man et frekvensplot af et teachernet (se afsnit 3.6) bemærker man at teachernettet selv, faktisk ikke er løsningen til "den opgave teacheren stiller". Havde teacheren været løsningen, ville frekvensplottet have haft to toppe, en top ved hver af de to klassificeringsværdier $\{-1, +1\}$. Hvis en student tilfældigvis havner i samme vægtkonfiguration som teacheren, vil studenten "lære væk fra denne konfiguration", simpelt hen fordi studenten, der har tanh() som aktiveringsfunktion på output-neuronen, ikke "klipper så hårdt" som teacheren.

I forsøg II-A er der i figur 5.77 indsat resultatet af to delforsøg. I det ene af disse delforsøg, udfører studenterne self-repair uden støj, offset, eller vægtdrift. Til trods herfor bliver præstationen af nettene dårligere med tiden. Dette tilskrives vægtopdateringsalgoritmen. Gradient descent metoden udført for hver nyt eksempel, udgør i sig selv en trussel mod "levetiden" af et ensemble. Hvis nettene et stykke tid kun har set eksempler fra et "lille" område af input-rummet, vil information om resten af input-rummet blive glemt (lidt). Eller omvendt, hvis nettene et stykke tid *ikke* har set eksempler fra et område af input-rummet, vil nettene miste lidt information om dette område.

Det andet af delforsøgene viser at sætter man learning-rate op med en faktor 10, får man en dårligere præstation. Dette viser igen at man skal være meget omhyggelig når man vælger parametre til forsøg med hardware studenter.

Det må konkluderes af forsøgene at hardware implementerede neurale net, der optrænes, og consensus-trænes med ren gradient descent metode, ikke generelt kan forvente en længere levetid af ensemblet.

Kapitel 6

Konklusion

Der er i denne rapport set på følgende emner:

- Solplet serien.
- Teacher net.
- Ensembler af neurale net.
- En model for et hardware implementeret analogt neuralt net.
- Ensembler af hardware neurale net.
- Levetidsforlængelse ved consensus træning.

Som et indledende forsøg er der kørt en kort forsøgsrække med solplet serien, og her er det belyst at vægthenfald leder til øget generaliseringsevne. Der er fundet et sæt parametre for det optimale vægthenfald, og for denne sammensætning fundet trænings- og testsæt fejl der er i samme størrelsesorden som rapporteret i litteraturen.

Der er implementeret et teachernet, der kan generere data af kendt kompleksitet, og i rigelig mængde. Dette teachernet afbilder vektorer af reelle tal over i to klasser. Det er vist hvordan teachernettet afstemmes så de to klasser er ligeligt repræsenteret.

Der er skrevet et program, hvor man kan oprette ensembler af neurale net. Nettene kan optrænes med teachernet, og med consensus beslutning. Vægtopdatering kan udføres med bl.a. 2. ordens metoder, og matrix inversion. Ensembler af disse net er i en række forsøg blevet undersøgt. Det er vist at udsættes de enkelte net for vægtdrift kan consensus holde en konstant præstationsevne over en længere periode end det enkelte net. Det er også vist at de enkelte net mister præstationsevne pga. vægtdrift, således at consensus ikke kan holde evigt. Der er rapporteret om en "indsvingningsperiode" i de første epoker af self-repair perioden. Under denne "indsvingningsperiode" stiger generalisationsfejlen markant. Det er vist at ensembler af net med samme kompleksitet som teachernettet, påvirkes mindst af "indsvingningsperioden".

Der er opstillet en model for et hardware implementeret analog neuralt net. Denne model er blevet implementeret i et program, sammen med muligheden for at oprette ensembler af sådanne net. Disse ensembler er blevet undersøgt i en række forsøg. Specielt er det undersøgt om ensemblet kan forlænge sin levetid, hvis nettene optrænes på ensemblets consensus beslutning.

Det har vist sig ret umuligt at optræne hardware net, så de tog fejl på forskellige dele af input-rummet. Derved opnås ikke en bedre consensus beslutning, end det bedste af nettene.

Det er vist at de mange parametre man kan sætte for et hardware net, skal vælges med stor omhu, for at opnå en rimelig præstation.

Det må konkluderes at forsøgene med simulering af hardware implementerede neurale net, der optrænes, og consensus-trænes med ren gradient descent metode, ikke generelt kan forvente en længere levetid af ensemblet.

Litteraturhenvisninger

[Bar91] Simon. A. Barton. A Matrix Method for Optimization a Neural Network. *Neural Computation*, 3:450–459, 1991. ©Massachusetts Institute of Technology.

[BH89] E. B. Baum and D. Haussler. What Size Net Gives Valid Generalization. *Neural Computation*, 1:151–160, 1989. ©1989 Massachusetts Institute of Technology.

[CBD+92] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. 1992.

[CDS90] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal Brain Damage. *Proceedings of the Neural Information Processing System*, 2:589–605, 1990.

[Con84] Knuth Conradsen. *En introduktion til statistik*, volume 1A. IMSOR, 1984. ©Knuth Conradsen.

[DSW+87] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, and L. Jackel. Large Automatic Learning, Rule Extraction, and Generalization. *Complex Systems*, 1:877–922, 1987.

[Fla91] Bryan Flaming. *Turbo C++ A Self-Teaching Guide*, volume 1. Therese A. Zak, 1991. ©1991 John Wiley & Sons, Inc.

[GHAKW] J. Gorodkin, L.K. Hansen, C. Svarer A. Krogh, and O. Winther. A Quantitative Study of Pruning by Optimal Brain Damage. SUPER-revised, June 1993.

[HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to The Theory of Neural Computing*, volume 1. Addison-Wesley Publishing Company, 1991. ©Addison-Wesley Publishing Company.

[HS] Lars Kai Hansen and Peter Salamon. Self–Repair in Neural Network Ensembles.

[HS90] Lars Kai Hansen and Peter Salamon. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):193–209, oct 1990. ©IEEE 1990.

[HSKT92] David Haussler, H. Sebastian Seung, Michael Kearns, and Naftali Tishby. Rigorous Learning Curve Bounds from Statistical Mechanics. 1992.

[HSW92] Babak Hassibi, David G. Stork, and Gregory J. Wolff. Optimal Brain Surgeon and general network pruning. 1992.

[Inc92] The MatWorks Inc. MatLab External Interface Guide. ©1984-92 The MatWork Inc., 1992.

[KH] Anders Krogh and John A. Hertz. A Simple Weight Decay Can Improve Generalization. Blegdamsvej 17, DK-2100 København.

[Leh93] T. Lehmann. Novel Weight Refreshing Scheme For Analogue Integrated Neural Networks. ©IEE 1993, dec 1993.

[Leh94] Torsten Lehmann. *Hardware Learning in Analogue VLSI Neural Networks*. PhD thesis, Technical University of Denmark, 1994. Electronics Institute.

[LL] John A. Lansner and Torsten Lehmann. Private Communication. Electronics Institute, DTU, Denmark.

[LL93] John A. Lansner and Torsten Lehmann. An Analog CMOS Chip Set for Neural Networks with Arbitrary Topologies. *IEEE Transactions on Neural Networks*, 4(3):441–444, may 1993. ©IEEE 1993.

[Moo91] John E. Moody. Note on Generalization, Regularization, and Architecture Selection in Nonlinear Learning Systems. *IEEE Workshop on Neural Nets for Signal Processing*, 1991. ©IEEE 1991.

[NH92] Steven J. Nowlan and Geoffrey E. Hinton. Adaptive Soft Weight Tying using Gaussian Mixtures. *Advances in the Neural Information Processing System*, 4, 1992.

[PC92] Michael P. Perrone and Leon N. Cooper. When Network Disagree: Ensemble Methods for Hybrid Neural Networks. to appear in Neural Networks for Speech and Image processing, oct 1992.

[PFTV90] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C : The Art Of Scientific Computing*, volume 1. Press Syndicate of the University of Cambridge, 1990. ©Cambridge Uniersity Press 1988.

[PU94] Dimitris C. Psichogios and Lyle H. Ungar. SVD-NET: An Algoritm that Automatically Selects Network Structure. *IEEE Transactions on Neural Networks*, 5(3):441–444, may 1994. ©IEEE 1993.

[SHL] Claus Svarer, Lars Kai Hansen, and Jan Larsen. On Design and Evaluation of Tapped-Delay Neural Network Achitectures. CONNECT, Electronics Institute, Technical University of Denmark.

[SR87] T. J. Sejnowski and C. R. Rosenberg. Parallel Networks that Learn to Pronounce English Text. *Complex Systems*, 1:145–168, 1987.

[SSSD90] D. B. Schwartz, V. K. Samalam, S. A. Solla, and J. S. Denker. Exhaustive Learning. *Neural Computation*, 2:374–385, 1990. ©1990 Massachusetts Institute of Technology.

[WHR90] Andreas S. Weigend, Bernardo A Huberman, and David E. Rumelhard. Predicting The Future: A Connectionist Approach. *International Journal of Neural Systems*, 1(3):193–209, 1990. ©World Scientific Publishing Company.

[WRH91] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by Weight-Elimination with Application to Forecasting. 1991.

Appendiks A

Nomenklatur liste

I rapporten bruges følgende notation.

ξ_k	:	*input enhed*
ζ	:	*ønsket output*
V_j	:	*skjult neuron*
O_i	:	*output neuron*
w_{jk}	:	*vægt mellem input enhed, og skjult neuron*
W_j	:	*vægt mellem hidden neuron, og output neuron*
w_{j0}	:	*tærskelværdi for den j'te skjulte neuron*
W_0	:	*tærskelværdi for output neuron*
α_1	:	*vægthenfaldsled for vægte mellem input og skjulte*
α_2	:	*vægthenfaldsled for vægte mellem skjulte og output*
η	:	*learning-rate i gradient descent vægtopdatering.*
β	:	*moment i gradient descent vægtopdatering.*
g_h	:	*aktiveringsfunktion for skjulte neuroner*
g_o	:	*aktiveringsfunktion for output neuronen*
E	:	*kostfunktion*
μ	:	*ved superscript: indikation af eksempel nummer*
$\mu(\theta)$	:	*input-sværhedsgrad (se evt afsnit 3.5.4)*

Appendiks B

Hardware simulerings programmel

Parametre			
Variabel	Type	Default	Bemærkning
gt_teacher	int	1	Sætter den generator-type der fylder vægtene på teacher nettet. 1 = uniformt fordelte; 2 = normalfordelte.
gt_students	int	1	Sætter den generator-type der initialiserer vægtene på studenter nettene. (se ovenfor.)
gt_data	int	2	Sætter den generatortype der genererer input-værdier.
teacher_p1	float	-1	Parameter nr. 1 til teachernet-generatoren. Hvis generatoren er uniform, vil denne parameter være minimum i fordelingen. Hvis fordelingen er normal vil denne parameter være forventningsværdien (μ).
teacher_p2	float	+1	Parameter nr. 2 til teachernet-generatoren. Hvis generatoren er uniform, vil denne parameter være maximum i fordelingen. Hvis fordelingen er normal vil denne parameter være spredningen (σ).
student_p1	float	-0.1	Se teacher_p1
student_p2	float	+0.1	Se teacher_p2
data_p1	float	0.0	Se teacher_p1
data_p2	float	1.0	Se teacher_p2
teacher_seed	int	10	Seed værdi til teacher-net generatoren. Random generatoren til teachernettet skal bruge en seed værdi. I rutinen listet i tabel 3.1 på side 22, hedder denne seed værdi ran_init.
student_seed	int	11	Seed værdi til studenter-net generatoren.
training_data_seed	int	12	Seed værdi til den generator, der laver træningssættet.
cross_valid_seed	int	13	Seed værdi til den generator, der laver krydsvalideringssættet.
w_droop_seed	int	14	Seed værdi til den generator, der bestemmer vægt driften.
w_noise_seed	int	15	Seed værdi til den generator, der bestemmer støjen på en vægt.
w_offset_seed	int	16	Seed værdi til den generator, der bestemmer offsettet på en vægt.
n_offset_seed	int	17	Seed værdi til den generator, der bestemmer offsettet på en neuron.

Parametre

Variabel	Type	Default	Bemærkning
`w_offset_my`	`float`	0.00	Forventningsværdi, for vægt offset værdien. Modellen for en hardware vægt er givet ved: $$w = w_{\text{ideal}} + w_{\text{offset}} + w_{\text{noise}} + \Delta t \cdot w_{\text{droop}} \quad \text{(B.1)}$$ hvor Δt er tiden siden sidste opfriskning af vægten. Offset værdien er specifik, og konstant for hver vægt i netværket, og modelleres ved en normalfordeling.
`w_offset_sigma`	`float`	$9.64 \cdot 10^{-3}$	Spredning på vægt offset værdierne.
`n_offset_my`	`float`	0.00	Forventningsværdi for neuron offset værdierne. Se formel (B.2).
`n_offset_sigma`	`float`	$9.64 \cdot 10^{-3}$	Spredning på neuron offset værdierne.
`w_noise_my`	`float`	0.00	Forventningsværdi for støjen på vægtene. Støjen er ikke konstant, og beregnes for hver gang vægtene skal bruges. Se formel (B.1).
`w_noise_sigma`	`float`	$1.20 \cdot 10^{-3}$	Spredning på støjen på vægtene.
`w_droop_my`	`float`	0.00	Forventningsværdi for vægtdriften. Denne faktor beskriver hvor stærkt vægtene i netværket driver. Vægtdriften er specifik, og konstant for den enkelte vægt.
`w_droop_sigma`	`float`	$3.01 \cdot 10^{-4}$	Spredning på vægtdriften
`w_max`	`float`	1	Maximum værdi for vægtene i netværket. I et hardware implementeret ANN, vil vægtene være implementeret som en differens spænding, på to kondensatorer. Fysisk er der da en øvre grænse for en vægt i et hardware implementeret ANN.
`w_min`	`float`	1	Minimum værdi for vægtene i netværket. Se ovenstående.
`t_sample`	`float`	$1 \cdot 10^{-4}$	Tidsafstanden mellem to målinger. Dette er samtidig det Δt der indgår i formlen (B.1).
`t_update`	`float`	$1 \cdot 10^{-4}$	
`samples_per_update`	`int`	1	
`w_decay_a1`	`float`	0.01	Vægthendfaldsleddet α_1 der indgår i back-propagation algoritmen. Leddet forsøger at begrænse størrelsen på vægtene i lag 1, laget mellem input, og skjulte neuroner.
`w_decay_a2`	`float`	0.01	Vægthendfaldsleddet α_2 der indgår i back-propagation algoritmen. Leddet forsøger at begrænse størrelsen på vægtene i lag 2, laget mellem skjulte , og output neuroner.
`lrate1`	`float`	$1 \cdot 10^{-4}$	Learning rate parameteren (η^1) for vægtene mellem input og skjulte neuroner. Se formel (B.3).
`lrate2`	`float`	$1 \cdot 10^{-4}$	Learning rate parameteren (η^2) for vægtene mellem skjulte neuroner og outputneuronen. Se formel (B.3).
`moment`	`float`	0.9	Parameter der angiver momentet, ved optræning af netværket med algoritmen "gradient descent med moment". Se evt. afsnit 3.2.4, hvor momentet er betegnet med β i formel (3.19).
`max_epoch`	`int`	100	Hvis der skal køres en indlæringsrunde, bruges `max_epoch` til at angive antallet af epoker der skal trænes over.
`max_itt`	`int`	1000	Hvis der skal køres en "self-repair", eller "decay"-runde, bruges `max_itt` til at angive hvor mange epoker der køres over. Se også forklaring af `data_set_size`

Parametre

Variabel	Type	Default	Bemærkning
`k1_seed`	`int`	18	Seed værdi til den generator, der bestemmer k1 værdien i modellen for hardware neuronen beskrevet i afsnit 4.2.
`k2_seed`	`int`	19	Seed værdi til den generator, der bestemmer k2 værdien i modellen for hardware neuronen beskrevet i afsnit 4.2.
`k3_seed`	`int`	20	Seed værdi til den generator, der bestemmer k3 værdien i modellen for hardware neuronen beskrevet i afsnit 4.2.
`k1_my`	`float`	1	Forventningsværdien for `k1`. Denne faktor indgår i formlen for hardwareneuronen. Hvis den j'te hardwareneuron i lag l betegnes V_j^l, er formlen for beregning af neuronværdien som følger: $$V_j^l = k_1 \tanh\left(k_2 \sum_i w_{ji}^{l-1} V_i^{l-1}\right) + V_{j,\text{offset}}^l \qquad \text{(B.2)}$$ Denne værdi er specifik og konstant for den enkelte neuron, men modellen beskriver en "vis" variation, mellem de enkelte neuroners `k1`-værdier. Denne variation modelleres med en normalfordeling, med forventningsværdi `k1_my` og spredning `k1_sigma` Se også afsnit 4.
`k1_sigma`	`float`	$6.024 \cdot 10^{-4}$	Spredning for `k1`. Se ovenstående forklaring.
`k2_my`	`float`	0.5	Forventningsværdi for `k2`. Denne faktor indgår i formlen for hardwareneuronen, se formel (B.2). Denne værdi er specifik og konstant for den enkelte neuron, men modellen beskriver en "vis" variation, mellem de enkelte neuroners `k2`-værdier. Denne variation modelleres med en normalfordeling, med forventningsværdi `k2_my` og spredning `k2_sigma` Se også afsnit 4.
`k2_sigma`	`float`	$6.024 \cdot 10^{-4}$	Spredning for `k2`. Se ovenstående forklaring.
`k3_my`	`float`	1.2	Forventningsværdien for `k3`. Denne faktor indgår i algoritmen for hardware-gradient descent. Hvis den j'te hardwareneuron i lag l betegnes V_j^l, er algoritmen til beregning af vægtopdateringsværdien som følger: $$\Delta w_{jk}^l = \eta^l \delta_j^l V_k^{l-1} \qquad \text{(B.3)}$$ hvor $$\delta_j^l = k_1 k_2 (k_3 - (V_j^l)^2) \sum_i w_{ij}^l \delta_i^{l+1} \qquad \text{(B.4)}$$ med $$\delta_i^L = \zeta_i - O_i$$ Hvor L er maksimalt lagindeks Bemærk at input-neuroner har lagindeks 0, mens det første lag af vægte har lagindeks 1. Bemærk endvidere at algoritmen starter i laget med højest indeks (outputlaget), og at for dette lag skal summationen i (B.4) erstattes af differensen mellem "output-ønsket-værdi", og "output-opnået-værdi". Se afsnit 4. `k3`-værdien er specifik og konstant for den enkelte neuron, men modellen beskriver en "vis" variation, mellem de enkelte neuroners `k3`-værdier. Denne variation modelleres med en normalfordeling, med forventningsværdi `k3_my` og spredning `k3_sigma` Se også afsnit 4.
`k3_sigma`	`float`	$6.024 \cdot 10^{-4}$	Spredning for `k2`. Se ovenstående forklaring.

Parametre

Variabel	Type	Default	Bemærkning
`max_consensus_error`	`float`	0.5	Hvis krydsvalideringfejlen på ensemblet af net i "self-repair", eller "decay"-runde, overstiger `max_consensus_error` stopper "runden".
`cross_valid_size`	`int`	1000	Antallet af eksempler i datasæt, der skal måles generaliseringsevne på. Jo større `cross_valid_size` jo mere nøjagtig måling af generaliseringsevnen, men på bekostning af en længere beregningstid.
`data_set_size`	`int`	300	Hvis der skal køres en indlæringsrunde, bruges `data_set_size` til at angive hvormange træningseksempler nettene skal læres op på. Under indlæring er træningseksemplerne fastlåste. Hvis der skal køres en "self-repair", eller "decay"-runde, angiver `data_set_size` en slags *batch*-størrelse. I hver epoke køres der `data_set_size` eksempler igennem nettene. Eksemplerne er ikke fastlåste, dvs at der genereres nye eksempler for hver epoke.
`teacher_N`	`int`	8	Topografien på teacher-nettet, er **N:M:1**, dvs `teacher_N` input enheder, `teacher_M` skjulte neuroner, og 1 output neuron.
`teacher_M`	`int`	5	Se ovenstående.
`classify_value`	`float`	1.0	De skjulte neuroner er sigmoide, med tanh() som aktiveringsfunktion, og outputneuronen er "klippet". Dette betyder at dersom teacheren for et givet input får et positivt output, da svarer teacheren +`classify_value`, dersom teacheren får et negativt output svares −`classify_value`
`students_hidden_squash`	`int`	1	Denne værdi angiver om de skjulte neuroner på studenternetværk'erne, skal være lineære (`students_hidden_squash` = 0), eller sigmoide hvilket betyder at tanh() benyttes som aktiveringsfunktion (`students_hidden_squash` = 1).
`students_output_squash`	`int`	1	Denne værdi angiver om output neuronen på studenternetværk'erne skal være lineære, eller sigmoide. Se forklaring på `students_hidden_squash`.
`students_M`	`int`	5	Studenternes topografi er bestemt af antallet af input enheder på teacher-nettet,og en output neuron, men antallet af skjulte neuroner kan vælges frit.
`number_of_students`	`int`	3	Antallet af ANN, der skal evalueres samtidigt. Hvis tallet er over 3, vil det være muligt at evaluere consensus-beslutning.
`output_filename`	`string`	`t1_`	Når programmet skal udskrive de forskellig måleresultater, sker det i filer hvor første del udgøres af "`output_filename`", og anden del er et fortløbende 3-cifret nummer startende med `001`, og sidste del er en extension der afhænger af parameteren `matFmt` se denne.
`write_every`	`int`	10	Angiver hvormange epoker der skal køres inden en resultatfil udskrives. Resultatfiler nummereres med 3 cifre, startende med `001`. Se `output_filename`.

<table>
<tr><th colspan="4">Parametre</th></tr>
<tr><th>Variabel</th><th>Type</th><th>Default</th><th>Bemærkning</th></tr>
<tr><td><code>matFmt</code></td><td><code>int</code></td><td>-1</td><td>Bestemmer formatet, og dermed extension på resultatfilerne.
<table>
<tr><th>matFmt</th><th>Betydning</th></tr>
<tr><td>-1</td><td>ASCII format</td></tr>
<tr><td>0</td><td>IEEE Little Endesian (PC, 386, 486, DEC Risc)</td></tr>
<tr><td>1</td><td>IEEE Big Endesian (Macintosh, SPARC, Apollo, SGI, HP9000/300, other Motorola)</td></tr>
<tr><td>2</td><td>VAX D-float</td></tr>
<tr><td>3</td><td>VAX G-float</td></tr>
<tr><td>4</td><td>Cray</td></tr>
</table>
Hvis der vælges <code>matFmt</code>=-1, er extension på resultatfilerne ".m", ellers er extension ".mat". Bemærk at <code>matFmt</code>=-1, er et universielt format, mens de andre formater er maskinafhængige (se [Inc92]). Det betyder at hvis simuleringsprogrammet compileres på en PC, og man ønsker at lagre resultater direkte i MatLab format, skal man vælge <code>matFmt</code>=0. Det er her en forudsætning at MatLab også køres på PC. Hvis man ønsker at bruge resultater genereret på en type maskine (f.eks PC) i et MatLab program der kører på en anden type maskine (f.eks HP9000) skal man bruge ASCII format.</td></tr>
</table>

Appendiks C

Grafer til forsøg I-D

Teacher

Topografi	$8:5:1$
Vægtfordeling	$U(-2,2)$

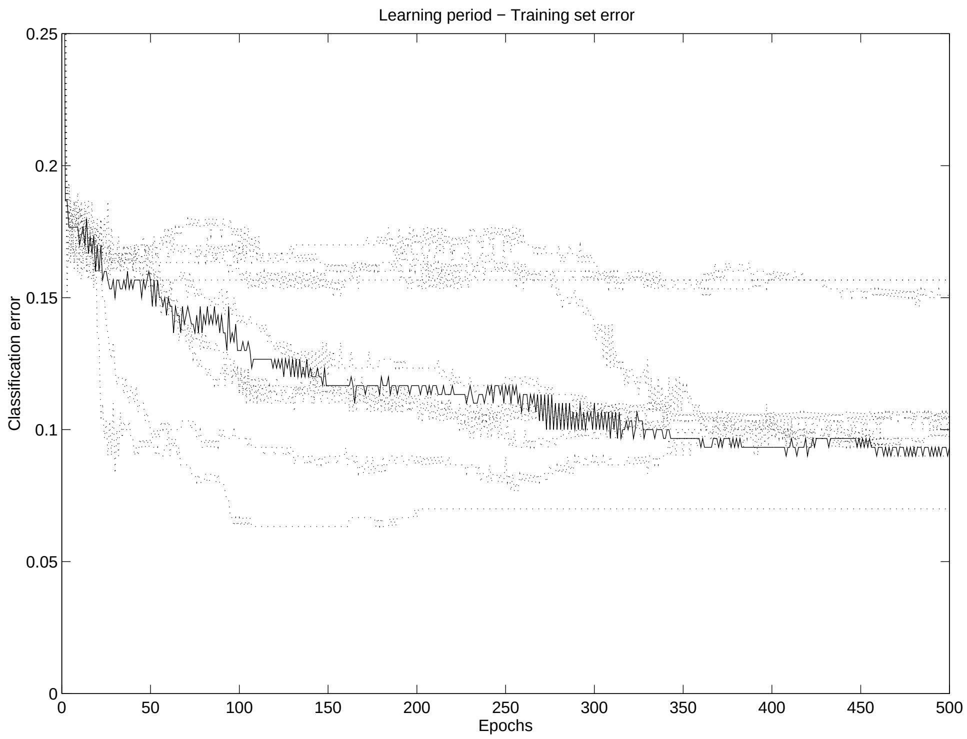
Studenter

Antal	9
Topografi	$8:7:1$
α_1	0
α_2	0
μ_{drift}	0
σ_{drift}	$\simeq 3 \cdot 10^{-5}$

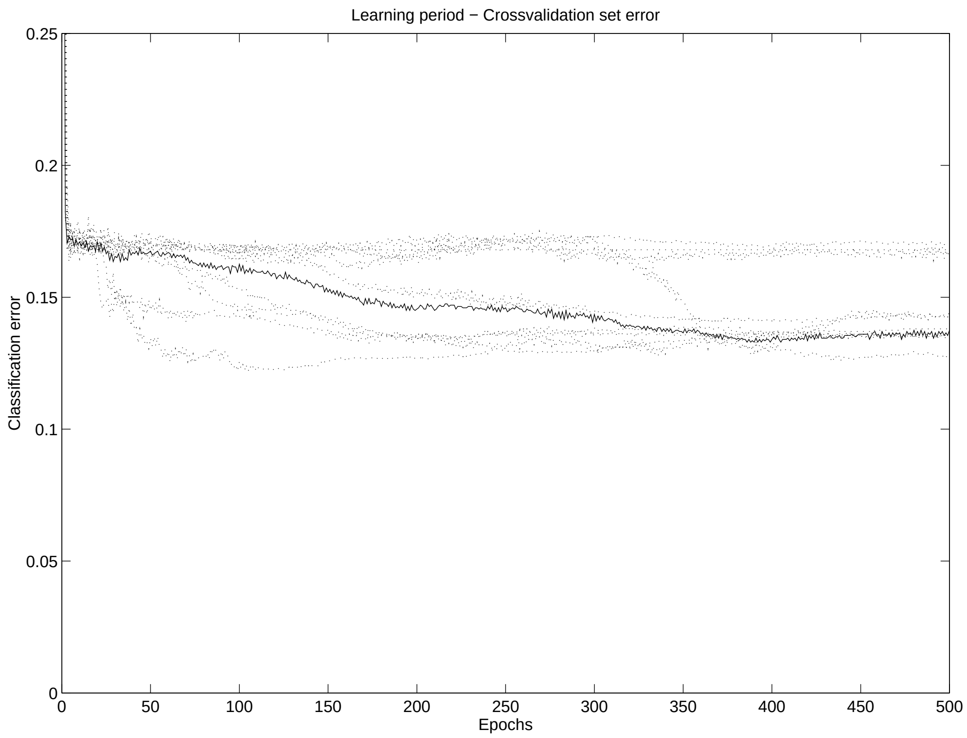
Kørselsbetingelser

Optræningsperiode	500 Epoker
Vægtdriftperiode	9500 Epoker
Krydsvalideringssæt	2000 eksempler
Træningssæt	300 eksempler
Self-repair sæt	100 eksempler

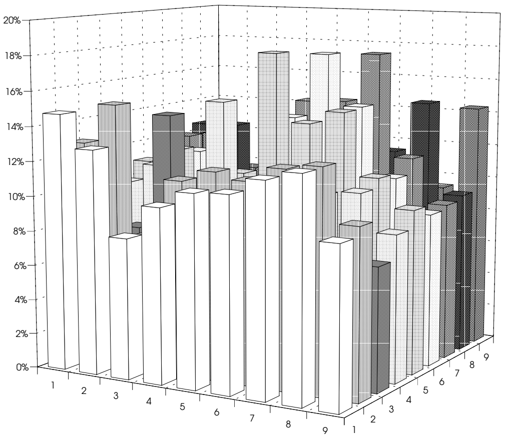
Tabel C.1: Parametre til forsøg I-D



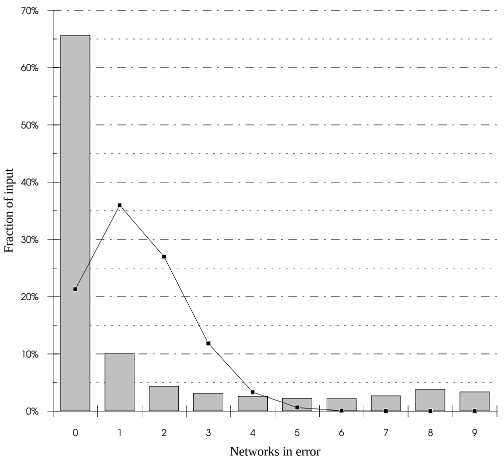
FIGUR C.1: Plottet viser klassifikationsfejlen på træningssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 15 epoker efterfulgt af anden ordens opdatering indtil 500 epoker.



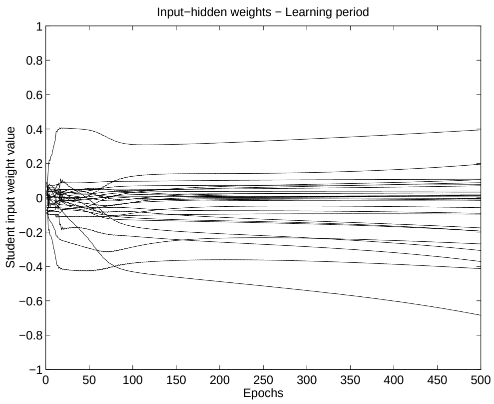
FIGUR C.2: Plottet viser klassifikationsfejlen på krydsvalideringssættet, som funktion af epoker. De enkelte studenters præstationer er markeret med punkterede linier, og ensemblets præstation er markeret med fuldt optrukket linie. Der er brugt 9 studenter. Disse er optrænet med ordinær gradient metode i 15 epoker efterfulgt af anden ordens opdatering indtil 500 epoker.



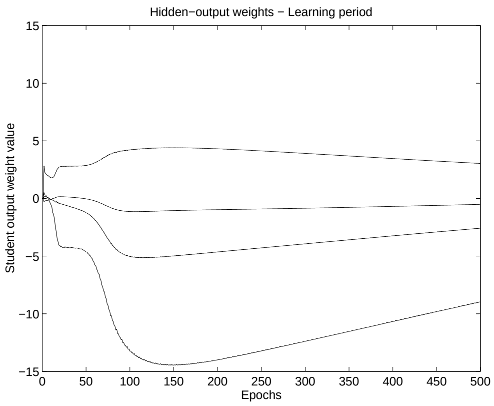
FIGUR C.3: Dette plot viser korrelationen mellem studenterne, efter endt optræning. Hver søjle viser sandsynligheden for at de tilsvarende net samtidigt tager fejl på tilfældigt input-mønster. I diagonalen optræder derfor et estimat af de enkelte nets krydsvalideringsfejl. Korrelationsplottet er baseret på 100,000 tilfældige inputmønstre.



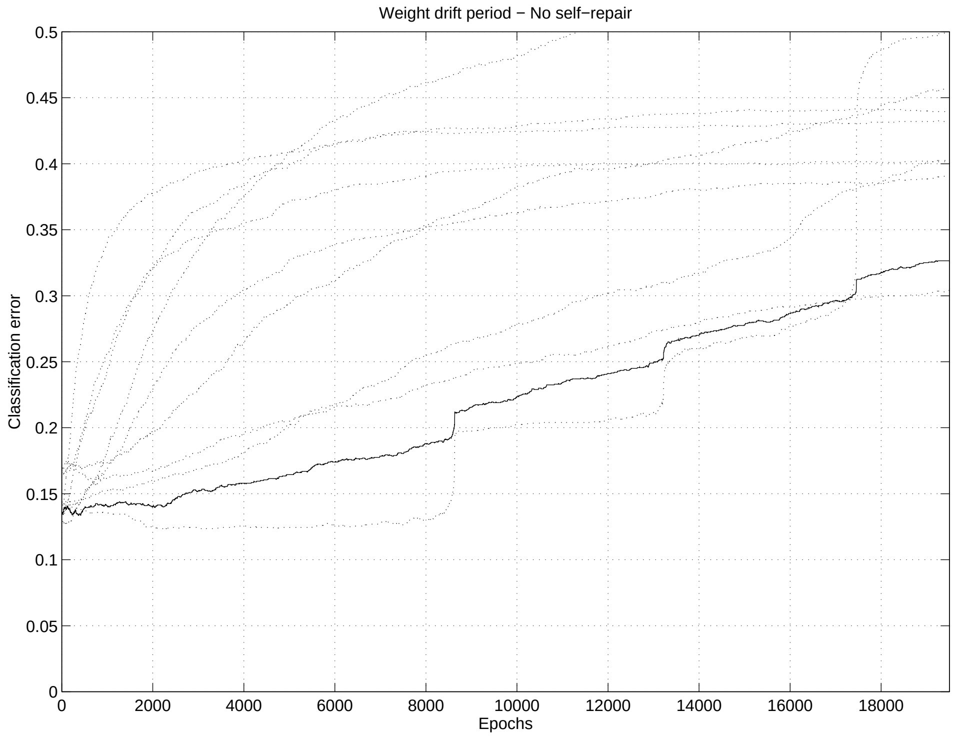
FIGUR C.4: $\mu(\theta)$-plot der viser sandsynligheden for at et vilkårligt input fejlklassificeres af n studenternet. $n = 0$ svarer derfor til sandsynligheden for, at et input klassificeres korrekt af alle studenter, og $n = N = 9$ svarer til sandsynlingheden for, at et input fejlklassificeres af alle net. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 16%, og havde været ukorrelerede med hensyn til fejl. Søjlerne er målt over 100,000 tilfældige input.



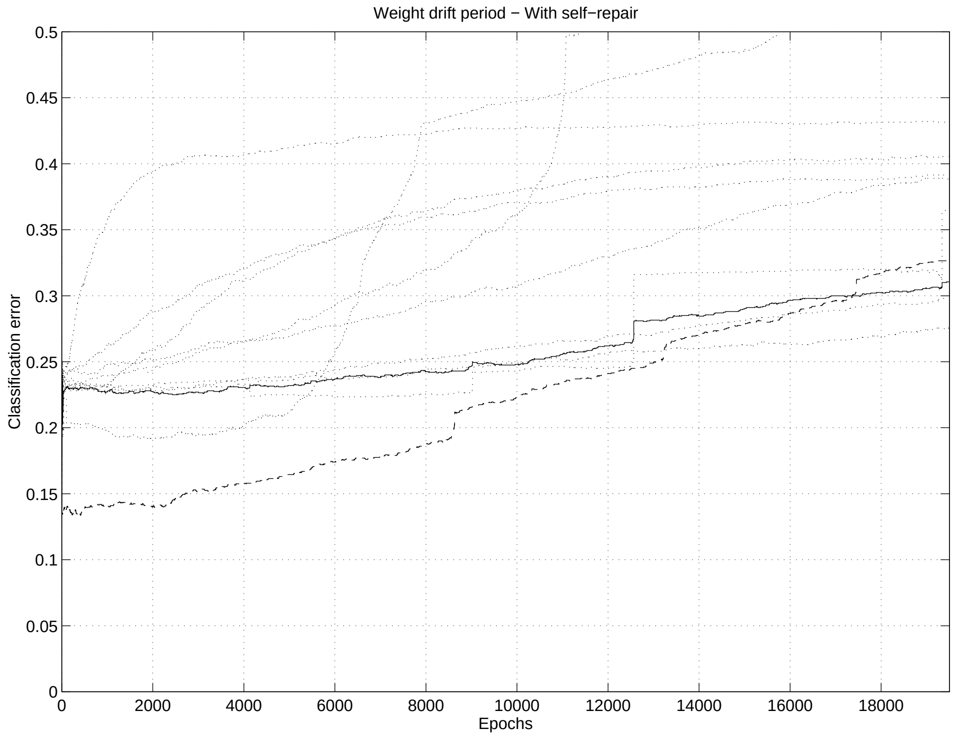
FIGUR C.5: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under optræningen. Plottet viser et typisk forløb, hvor de fleste af vægtene næsten ikke ændrer sig, og en lille rest laver nogle kraftige udsving tidligt i forløbet. Efter en indsvingningsperiode, finder en næsten stationær værdi.



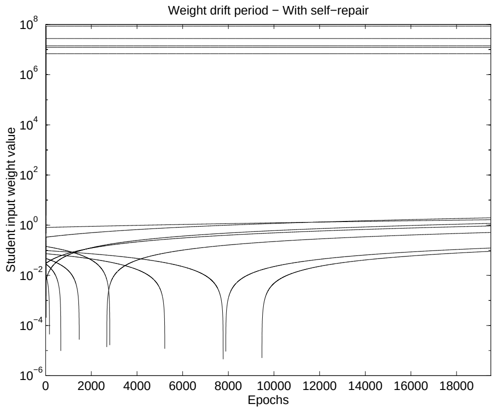
FIGUR C.6: Skjulte-output vægtværdier for studenten fra figur C.5, under optræningen. Plottet viser et typisk forløb, hvor de fleste vægte laver nogle kraftige udsving tidligt i forløbet, hvorefter de går over i en mere rolig periode. Det er typisk at værdierne er en del højere end for vægtene i laget mellem input og skjulte.



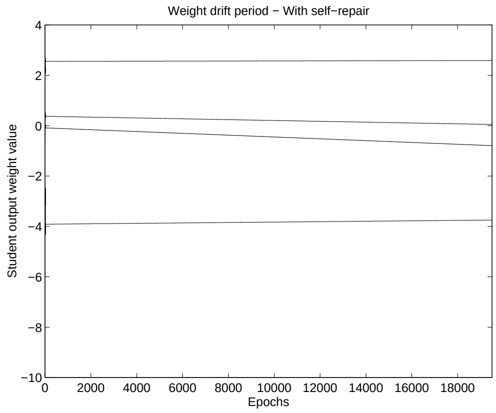
FIGUR C.7: Plottet viser klassifikationsfejlen på krydsvalideringssættet som funktion af epoker. Studenter nettene ødelægges med en konstant rate. For hver epoke lægges der en konstant værdi til vægtene. Værdien er forskellig fra vægt til vægt og vælges tilfældigt fra en normalfordeling. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie.



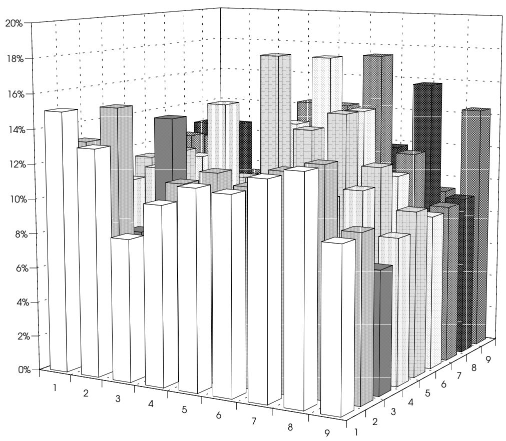
FIGUR C.8: Plottet viser klassifikationsfejlen på krydsvalideringssættet som funktion af epoker. Studenter nettene udsættes for samme ødelæggelse som på figur 5.52, men forsøger at rette op på skaden ved at udføre self-repair med ensemblets consensus beslutning. De enkelte studenters præstationer er markeret med punkteret linie. Ensemblets consensus er markeret med fuld optrukket linie. Ensemblets consensus uden self-repair er markeret med stiplet linie.



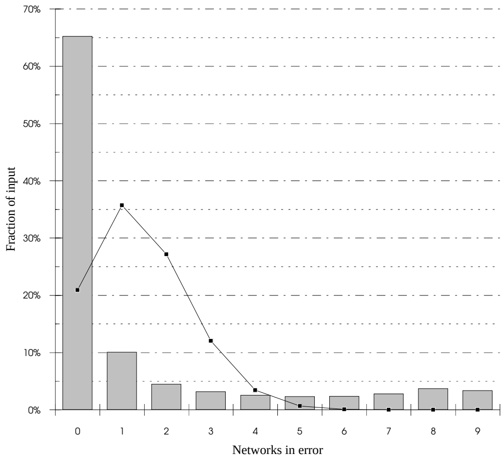
FIGUR C.9: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under self-repair perioden. Plottet er vist med logaritmisk y-akse, og derfor er negative vægte ikke plottet. Det betyder at en positiv vægt der krydser $y = 0$ forsvinder, og omvendt. Det skal bemærkes at det tilsvarende plot for negative vægte har et lignende udseende. Man ser, at vægtene har voldsomme udsving, og at udsvingene sker på forholdsvis kort tid (2-3 epoker).



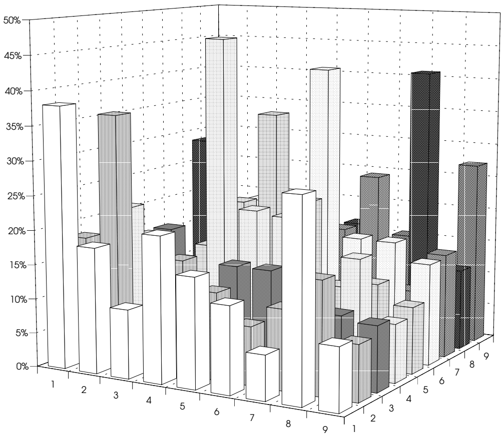
FIGUR C.10: Skjulte-output vægtværdier for studenten fra figur C, under selfrepair perioden.



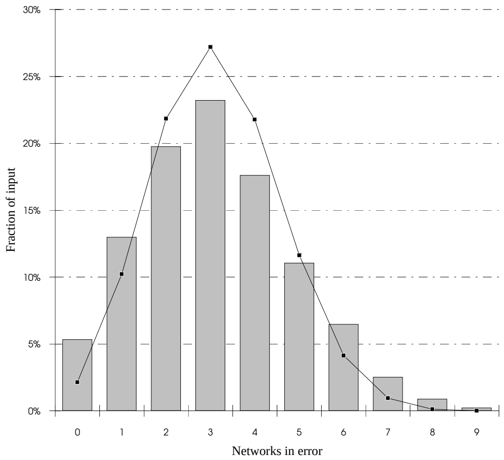
FIGUR C.11: Korrelationen mellem studenterne, efter 60 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



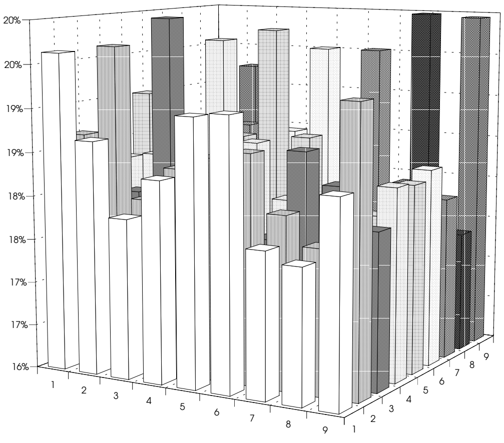
FIGUR C.12: $\mu(\theta)$-plot, efter 60 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, for en ukorreleret krydsvalideringsfejl på 16% Målt over 100,000 tilfældige input.



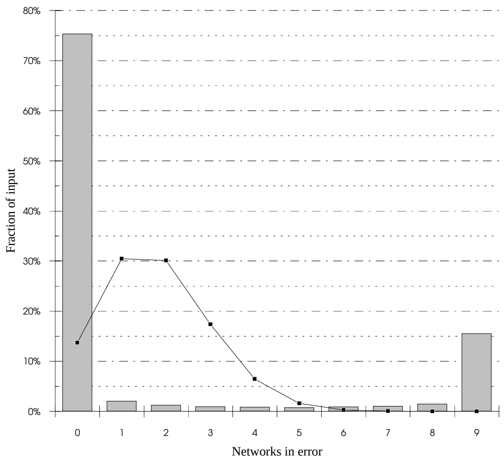
FIGUR C.13: Korrelationen efter 9500 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



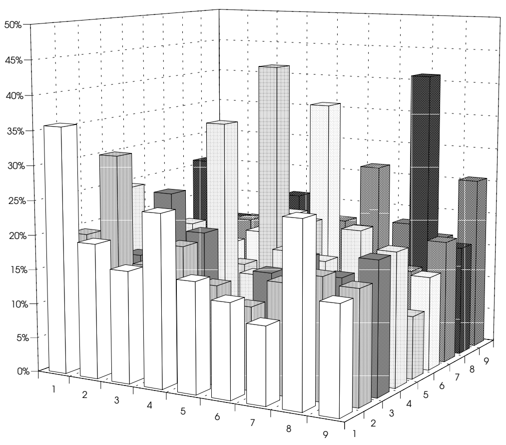
FIGUR C.14: $\mu(\theta)$-plot, efter 9500 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 35%. Målt over 100,000 tilfældige input.



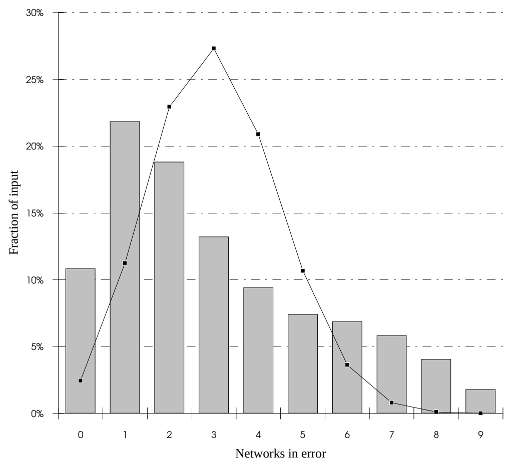
FIGUR C.15: Korrelationen mellem studenterne, efter 60 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



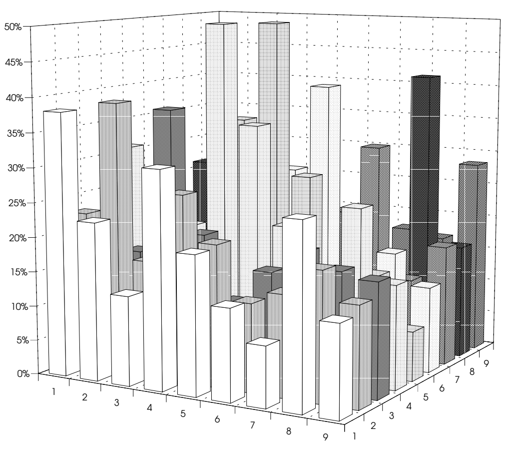
FIGUR C.16: $\mu(\theta)$-plot, efter 60 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 20%. Målt over 100,000 tilfældige input.



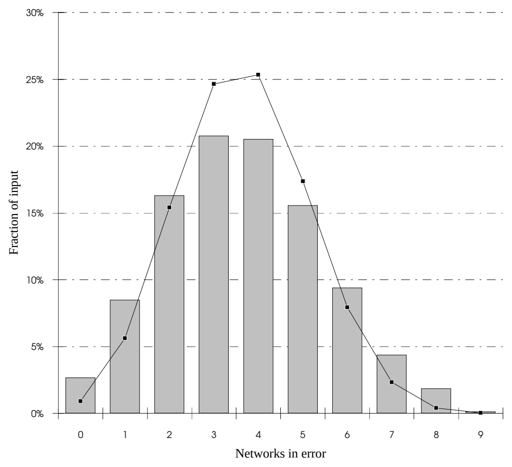
FIGUR C.17: Korrelationen mellem studenterne, efter 9500 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



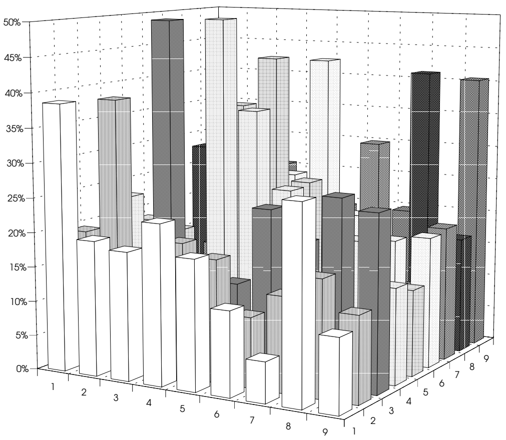
FIGUR C.18: $\mu(\theta)$-plot, efter 9500 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 34%. Målt over 100,000 tilfældige input



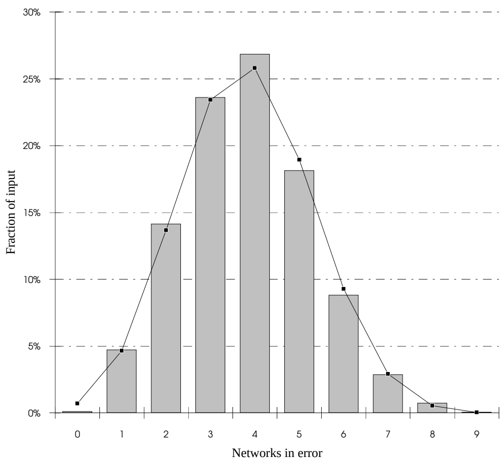
FIGUR C.19: Korrelationen mellem studenterne, efter 19500 epoker self-repair. Baseret på 100,000 tilfældige inputmønstre.



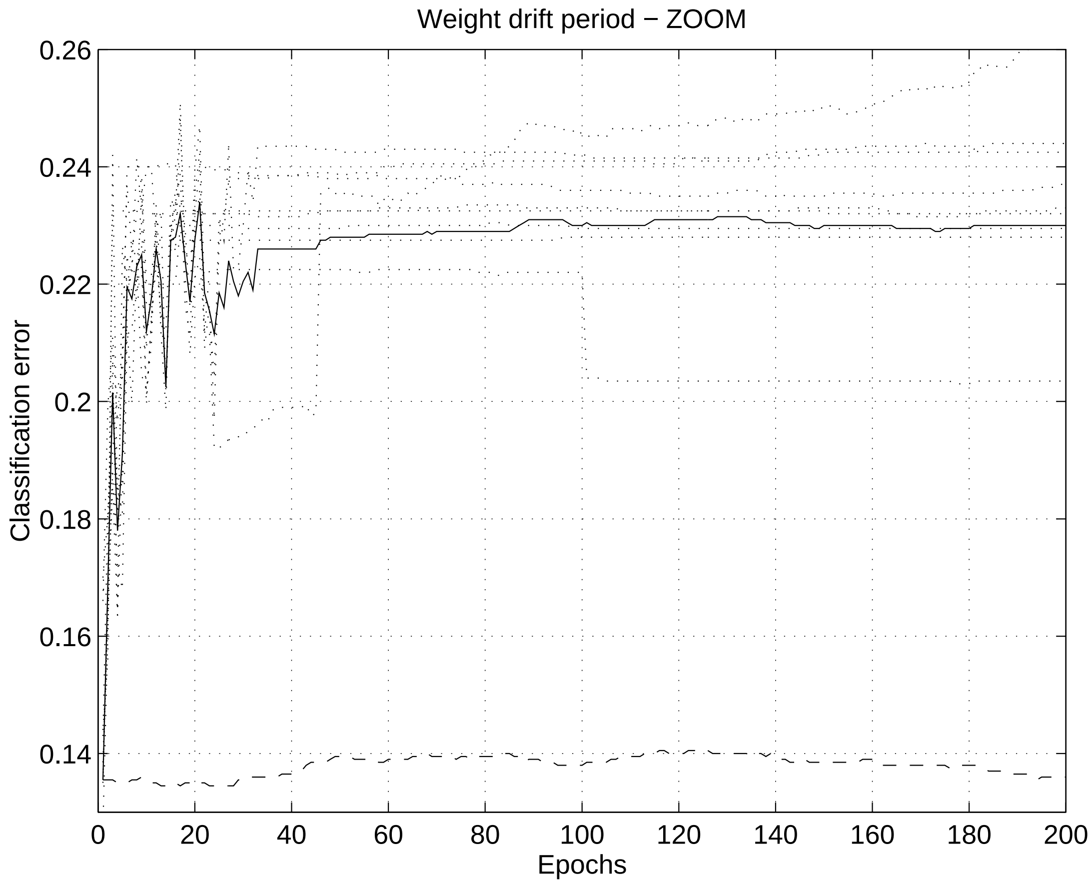
FIGUR C.20: $\mu(\theta)$-plot, efter 19500 epoker self-repair. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 41%. Målt over 100,000 tilfældige input



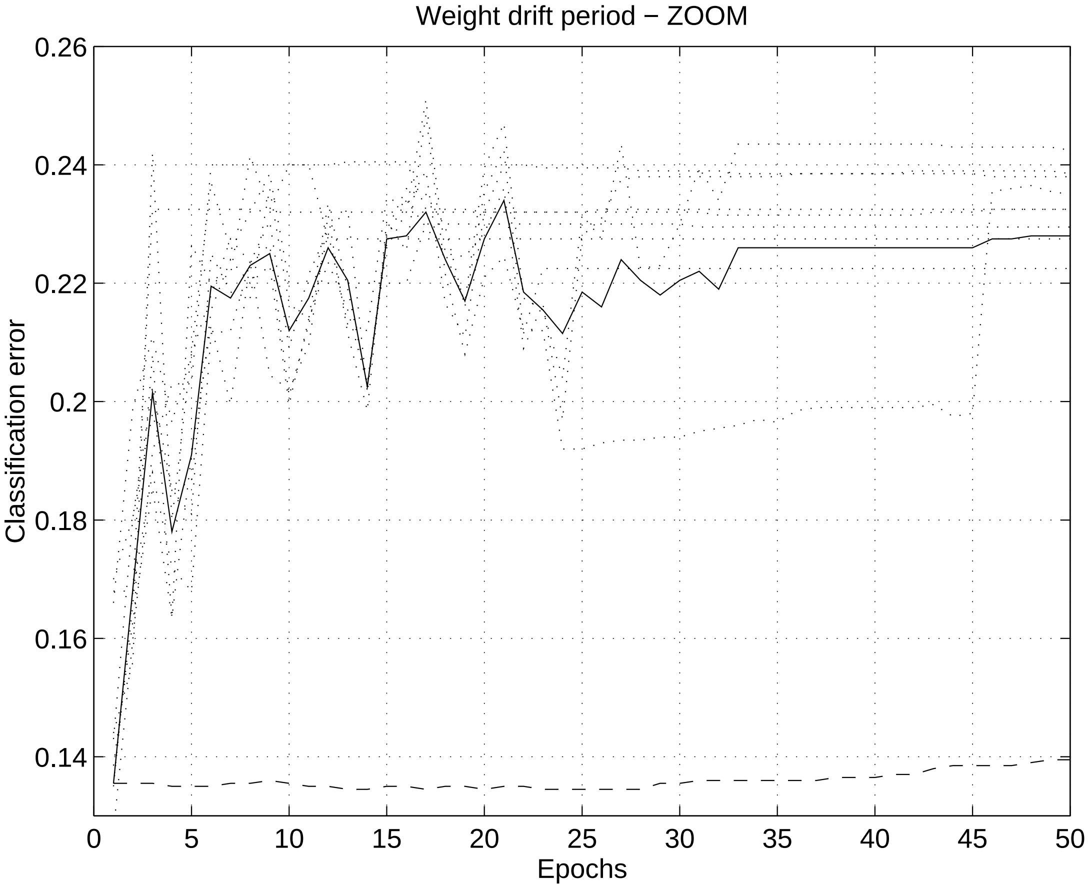
FIGUR C.21: Korrelationen efter 19500 epoker vægtdrift. Baseret på 100,000 tilfældige inputmønstre.



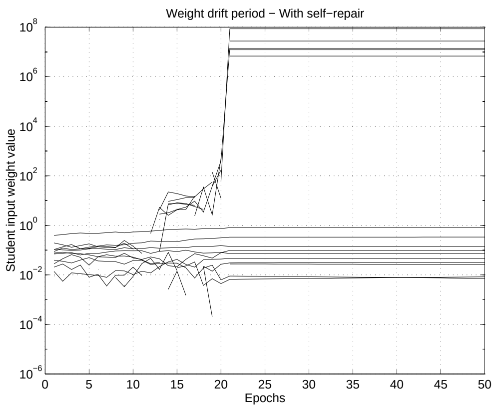
FIGUR C.22: $\mu(\theta)$-plot, efter 19500 epoker vægtdrift. Den fuldt optrukne kurve viser det teoretiske forløb, hvis nettene havde en krydsvalideringsfejl på 42%. Målt over 100,000 tilfældige input.



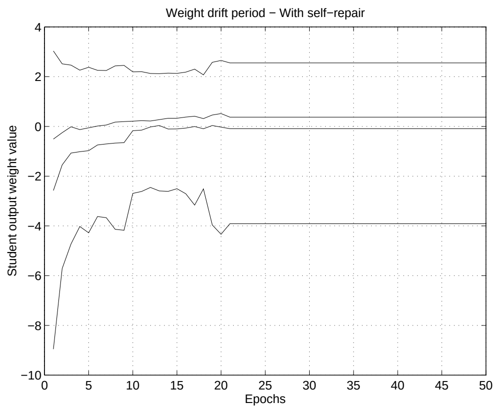
FIGUR C.23: Zoom af figur 5.53. Bemærk indsvingningsperioden.



FIGUR C.24: Zoom af figur 5.53. Bemærk indsvingningsperioden.



FIGUR C.25: Der er plottet en students input-skjulte vægtværdier, som funktion af epoker, under de første 50 epoker af self-repair perioden. Plottet er vist med logaritmisk y-akse, og derfor er negative vægte ikke plottet. Det betyder at en positiv vægt der krydser $y = 0$ forsvinder, og omvendt. Plottet viser at indsvingningsperioden sætter tydelige spor i vægtudviklingen.



FIGUR C.26: Skjulte-output vægtværdier for studenten fra figur C, under de første 50 epoker af self-repair perioden. Bemærk den tydelige overensstemmelse mellem de to figurer, om hvornår indsvingningsperioden stopper.