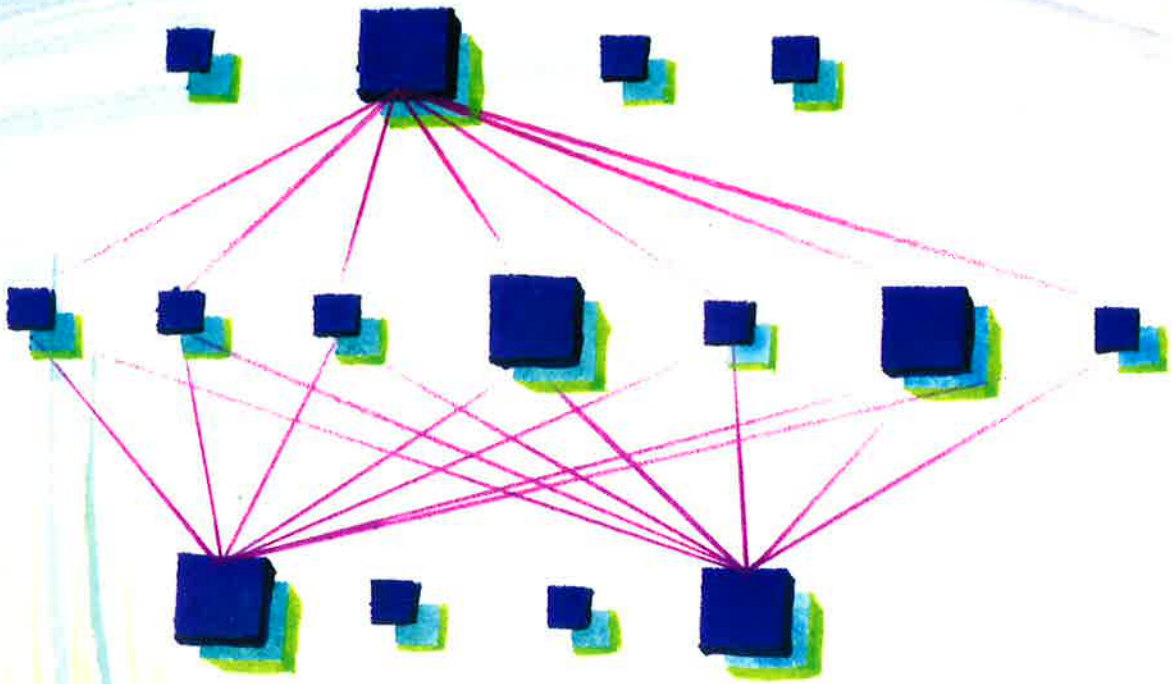
CODEN: LUTFD2/(TFRT-5441)/1–115/(1991)

Artificiella neuronnät – En applikation för linjedetektering

Carl Henrik Petersson

Institutionen för Reglerteknik
Tekniska Högskolan i Lund
September 1991



Examensarbete

Neurala Nätverk

En applikation för linjedetektering

Carl Henrik Petersson

Framsida: Anna Tillman

Telub AB, VÄXJÖ, Augusti 1991

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
September 1991

Document Number
CODEN:LUTFD2/(TFRT-5441)/1-115/(1991)

Author(s)
Carl Henrik Petersson

Supervisor
Rolf Johansson, Christian Fürst (Telub AB)

Sponsoring organisation
Telub AB

Title and subtitle
Artificial neural networks - An application for line segment detection in images

Abstract

This master thesis considers the application of neural network technology for detection of line segments in image processing. Several different neural networks structures have been compared with respect to performance. There has been some emphasis on back-propagation networks as this network type offered good flexibility in problem solving.

It was shown that the representation of input data was important for good performance of the neural networks for our application. Also the threshold properties at the output was of considerable importance to obtain correct categorizations.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
115

Recipient's notes

Security classification

Examensarbete

Neurala Nätverk

En applikation för linjedetektering

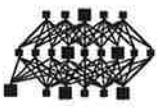
Carl Henrik Petersson
Juni - Augusti 1991

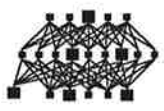
Telub
Avdelning LYM, Växjö

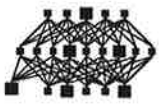
Lunds Tekniska Högskola,
Institutionen för Reglerteknik

Rev (1.224)

INNEHÅLLSFÖRTECKNING







1 FÖRORD

Undertecknad studerar för närvarande på linjen för Teknisk Fysik vid Lunds Tekniska Högskola. I utbildningen ingår att genomföra ett examensarbete omfattande 12 poäng. Examensarbetet är tänkt att ge den studerande en möjlighet att fördjupa sig i något projekt samtidigt som han/hon får en praktisk erfarenhet av industriellt ingenjörsarbete.

Första gången jag hörde talas om neurala nätverk var under arbetsmarknadsdagarna i november 1990, ett årligt återkommande arrangemang vid Lunds Tekniska Högskola. De representerade företagen brukar i exjobbskataloger presentera vilka examensarbeten som man har att erbjuda. Denna gång erbjöd flera företag arbeten som handlade om neurala nätverk. Mitt intresse tilltog efter det att jag fått förklarat för mig, hur dessa nätverk fungerar och vilka användningsområden de har.

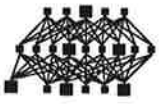
Under våren 1991 tog jag kontakt med Telub AB i Växjö och kort därefter hade vi gjort upp om att jag skulle utföra mitt examensarbete hos dem under sommaren 1991. Vid högskolan i Lund tog jag kontakt med Institutionen för Reglerteknik och knöt mitt arbete dit. Under en vecka, då jag hade påskledigt, satte jag mig in i uppgiften och bekantade mig med programvaran, som man inköpt för att utvärdera och utveckla neurala nätverk. Under tiden juni till augusti fullföljde jag sedan detta arbete.

Att genomföra sitt examensarbete under sommaren har både för- och nackdelar. Man kan i bästa fall förkorta sin studietid med ett par månader. Å andra sidan kan det vara svårt att få hjälp och handledning, då både handledare och andra har sin semester under denna tid. De första veckorna av sommaren, ägnade jag större delen av min tid, till att lära mig hur programvaran fungerar, och till att studera manualer för densamma. Under denna tid uppkom de flesta av de problem jag stött på. Men då semestrarna ännu inte hade infallit, fanns hjälpvilliga handledare tillhands för att klargöra och hjälpa. Senare under juli, när jag var i stort sett ensam, gjorde jag många testkörningar av nätverken, och var då inte i något större behov av hjälp och vägledning. Efter semesterperioden redovisade jag mina resultat och vi diskuterade igenom vad som mer skulle göras. De sista veckorna ägnade jag mestadels åt dokumentation. Dokumentering tar oftast längre tid än man beräknar. Men med en del hemarbete hann jag även få detta gjort. Det för mig gynnsamma kalla vädret i början och slutet av sommaren gjorde sitt till för att komma igång med samt avsluta detta arbete.

Jag vill härmed tacka de som hjälpt mig att genomföra detta arbete, mina handledare vid Telub i Växjö, Christian Fürst och Anders Yngvesson samt min handledare i Lund, Rolf Johansson.

Carl Henrik Petersson

Växjö den 20:e augusti 1991



2 INLEDNING

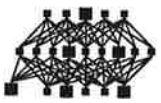
I detta examensarbete har en applikation av neurala nätverk för linjedetektering studerats. Uppgiften var att identifiera och klassificera linjer i tänkta bilder i signal/tidsplanet. Huvudsakligen har programmet NeuralWorks Professional II/PLUS från NeuralWare Inc använts vid studierna. Detta är ett komplext program för generering, upplärning, testning och kompilering av neurala nätverk. För generering av testdata har MATLAB från The Math Works Inc använts, och för analys av utdata har Lotus 1-2-3 nyttjats. De kompilerade nätverken har knutits samman till fungerande exekverbara program skrivna i Borlands Turbo C++. I filförteckningen, se kapitel 11, finns en lista över filer tillgängliga på diskett, för den som vill studera och undersöka på egen hand.

Denna rapport är utskriven med hjälp av ordbehandlaren Lotus Manuscript. Sidantalet kan verka stort, men då många sidor endast är halvt fullskrivna, för tydlighetens skull, och ett stort antal diagram tar upp en del av utrymmet, blir den löpande texten inte så lång, som den först kan verka. De två sista kapitlen innehåller mest data och listningar, intressant för den som vill återskapa och ändra i körningarna.

Neurala nätverk har under de senare åren rönt ett tilltagande intresse. Men som på åtskilliga andra områden existerar ej någon komplett svensk terminologi. En mängd engelska termer används där inte någon bra svensk motsvarighet finns. I kapitel 9 återfinns en sammanställning över en del av de begrepp, som förekommer i texten och i programmet NeuralWorks Professional. I denna presentation görs ej någon grundläggande genomgång av teorin bakom neurala nätverk, utan den som vill veta mer hänvisas till litteraturen på området samt till den mängd av artiklar som nu allt oftare publiceras i tidningar och tidskrifter. Se litteraturreferenser i kapitel 10. Förutom de manualer och artiklar, som nämns där, har ingen särskild litteratur nyttjats.

Hänvisningar till andra kapitel och avsnitt görs med kapitelnummer eller motsvarande, omgärdat av vanliga parenteser, medan hänvisningar till litteraturreferenser görs med omgärdning av hakparenteser.

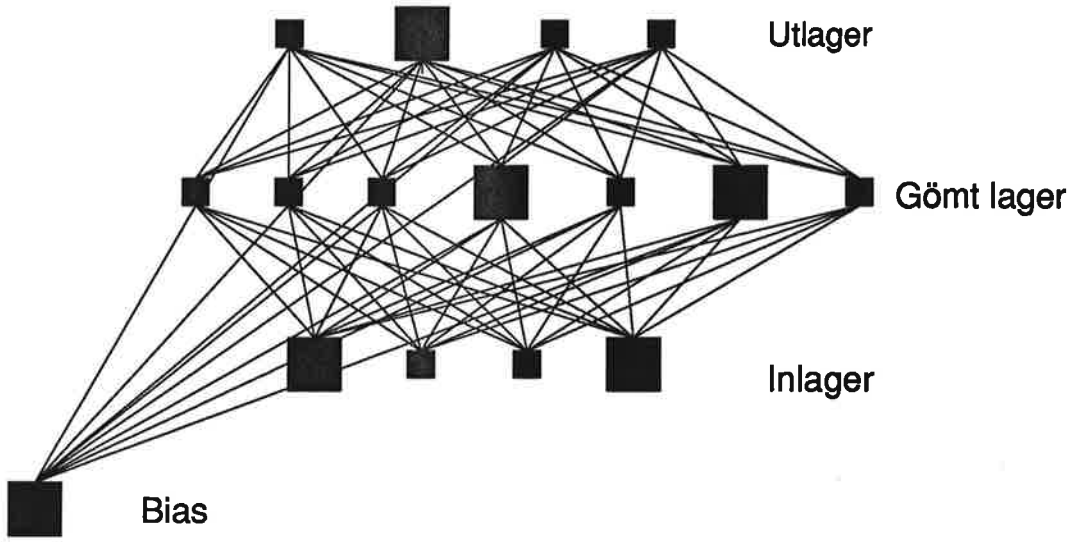
Jonas Samuelsson slutförde sitt examensarbete [1] i juni. I detta har han utvärderat och sammanfattat såväl funktion som användningsområden för ett flertal olika nättyper. För att inte än en gång utföra samma arbete, har i denna rapport endast en kortfattad genomgång av samma områden gjorts.
Efterhand som arbetet fortskridit, har även inriktningen på detta ändrats, med hänsyn till vilka resultat, som då hittills uppnåts.



3 ALLMÄNT OM NEURALA NÄTVERK

3.1 STRUKTUR

Neurala nätverk kan, på ett enkelt sätt, sägas vara ett försök att efterlikna de biologiska nervsystemen. Precis som dessa består de neurala nätverken av celler, kallade neuroner eller noder. I nätverken är dessa organiserade i ett antal lager där neuronerna är sinsemellan förbundna mellan de olika lagren. Det understa lagret brukar benämnas inlagret och det översta utlagret. De däremellan liggande mellanlagren benämns de gömda lagren. Se *figur 1*.



Figur 1.

Innan ett nätverk kan bearbeta information måste det läras upp. Det finns flera olika typer av nätverk. Vissa lärs upp genom att man visar nätet vilka indata och utdata som hör ihop. Andra nättyper kräver inte någon sådan styrning utan har förmågan att själva organisera sig. Ett nät kan kräva tusentals exempel innan det klarar av att göra riktiga associationer.

Nätverken har förmåga att generalisera och ge lösningar till problem som de inte är tränade för. Denna egenskap skiljer dem från andra expertsystem och digitala konstruktioner. En annan skillnad är att näten består av många celler eller processorer, medan konventionella system bara består av en. Genom att de olika cellerna sköter olika uppgifter fungerar näten nästan lika bra även då ett fåtal komponenter har gått sönder. Ett datorsystem havererar då första vitala komponenten gått sönder medan näten då bara gradvis försämrats i prestanda.

Olika nättyper är lämpliga för olika typer av problemlösning. I sitt examensarbete presenterar Jonas Samuelsson en överskådlig sammanställning av olika typer av neurala nätverk [1].

3.2 INFORMATIONSVÄGAR

Informationen, som skall bearbetas av nätet, passerar via inlagret genom de mellanliggande lagren till utlagret där den bearbetade informationen kan avläsas. Varje förbindelse mellan ett par av neuroner har en vikt kopplad till sig. Till var och en av neuronerna går en eller flera inkommande förbindelser. Signalerna från dessa multipliceras med respektive vikt och läggs sedan samman enligt en bestämd summationsfunktion (5.3.2). Summan av de viktade insignalerna ger sedan, efter transformering med hjälp av en överföringsfunktion, neuronens utsignal dess nya värde (5.3.3). Denna överföringsfunktion, som oftast är olinjär, väljs vanligen till tangenshyperbolicus eller sigmoidfunktionen. Den senare definieras enligt

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

Från inlagret vandrar således informationen upp genom lagren. I biologiska nervsystem sker överföringarna parallellt, men i nätverk simulerade i mjukvara måste tyvärr bearbetningen oftast ske sekvensiellt. Det finns dock parallellt arbetande datorer, så kallade transputers, speciellt lämpade för neurala nätverk [19] sid 21-24 samt [24] och [25].

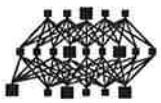
3.3 UPPLÄRNING AV NÄTET

Precis som sin biologiska förebild behöver även de neurala nätverken tränas för att kunna åstadkomma något.

I de flesta nättyper går detta till så att man skickar insignaler till nätet samtidigt som man talar om vilken önskad utsignal man vill att nätet skall ge. Nätet kommer själv att beräkna en utsignal. Denna jämförs med den önskade. Differensen, dvs den önskade utsignalen subtraherad från den beräknade, används till att justera vikterna i nätet så att felet minskar. Det finns olika typer av felfunktioner vilka används i olika nät (5.3.5).

I vissa typer av självorganiserande nätverk sker upplärningen självständigt, utan att näten får några styrsignaler. Dessa nät innehåller nästan alltid ett sk Kohonenlager. Detta är ett tvådimensionellt lager, där indata med liknande egenskaper aktiverar noder inom samma delområde av lagret. Upplärningen kan liknas vid den då ett barn lär sig att gå. Ingen talar om för barnet hur det skall lära sig att hålla balansen och röra på benen. Inlärningen stimuleras istället gynnsamt av positiva händelser, som att barnet märker att det kan komma dit det vill, bara det först lärt sig att gå.

För att inte fastna i lokala minima kan man addera brus till insignalen innan denna transformeras till nodens utsignal (5.3.8).



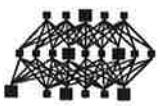
3.4 IN- OCH UTDATA

Ett av de svårare problemen vid design av neurala nätverk är formuleringen av hur in- och utdata skall se ut. Vilka indata är relevanta för nätverket och hur skall dessa bearbetas ? Hur skall man tolka nätens utsignaler ?

Beroende på vilken uppgift, som nätet skall lösa, varierar metoderna för förbearbetningen av data. Man bör dock alltid försöka begränsa indata, och sålla bort de parametrar som är irrelevanta. Om t ex blommor skall kategoriseras efter art och familj, så är egenskaper som färg, storlek, bladform och förökningssätt väsentliga. Däremot är det vid denna kategorisering tämligen oväsentligt om blomman är vacker, om den är en populär krukväxt eller om den är en landskapsblomma någonstans.

Vid bildbehandling kan förbearbetningen bestå av olika transformationer av indata. För linjedetektering finns operatorer såsom Laplace, Canny, Sobel och Prewitt med flera. Vid gråskalemanipulationer används t ex metoder grundade på histogramutjämning, [27].
Vid signalanalys är olika transformeringar till frekvensplanet fördelaktiga.

Då in- och utdata har samma utseende kallas näten för auto-associativa, och då de är olika hetero-associativa. I auto-associativa nät kan utdata återkopplas till indata för att understödja inlärningen. Nättypen 'recirculation' utnyttjar denna metod.
Indata och utdata beskrivs närmare i kapitel 6 och 7.



4 SLUTSATSER OCH PRAKTISKA ERFARENHETER

4.1 UPPGIFT

I detta arbete har neurala nätverk studerats, för en tillämpning där linjer skall identifieras i matrisbilder. Linjerna klassificeras i olika kategorier, beroende av deras lutningar.

4.2 NÄTVERKSTYPER

Även om större delen av tiden, lagts på studier av back-propagation nätverk, kan i efterhand sägas att andra, klassificerande nätverkstyper såsom pnn, counter-propagation och s o m verkar mer lovande, även om de kräver mer förbearbetning av indata, och är svårare att dimensionera rätt.

4.3 ANTAL NODER I DET GÖMDA LAGRET

Lämpligt antal noder i det gömda lagret beror på hur komplext sambandet mellan in- och utdata är. Ju mer komplext förhållande, desto fler noder i detta lager behövs. I fallet back-propagation, existerar dock ett intervall där inlärningen är kraftigt försämrad. En varning utfärdas, för att använda de antal noder, som ligger inom detta intervall, se diagram 6 a). Drs använder ett mindre antal noder i gömda lagret för att uppnå samma resultat som back-propagationnäten.

Generellt kan sägas att ju färre noder, som behövs i det gömda lagret för att uppnå goda resultat, desto bättre generaliserande egenskaper får nätet.

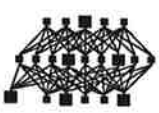
4.4 INDATA

I detta arbete har indata bestått av matriser av storlek 7 gånger 5 element, respektive 13 gånger 9 element. Antalet innoder har därmed blivit i största laget, 35 respektive 117 stycken. Speciellt om kategoriserande nätverk skall användas, tvingas man att på något sätt reducera mängden indata, för att inte få allt för komplexa nätverk. Att använda mindre matriser ger visserligen en mindre indatamängd, men identifieringen blir samtidigt osäkrare då den utgår från mindre information. En variant kan vara att använda små matriser, som element i en större matris, och därmed låta två nätverk arbeta sekvensiellt.

Back-propagation och drs är mindre känsliga för ojämnt fördelade indata, medan pnn och s o m kräver att indata fördelats jämnt i olika klasser.

4.5 UTDATA

Att tolka utdata från nätverk, kräver mer än att bara 'tröskla' utsignalerna vid ett visst värde. Genom att rita histogram över utsignalernas fördelning, kan emellertid



lämpliga trösklingsgränser bestämmas. Backpropagation och drs ger sällan entydiga kategoriseringar, utan indikerar även närliggande kategorier. Pnn, counterpropagation och s o m ger bättre kategoriseringar, men utdata förvrängs om indata inte fördelats jämnt.

4.6 ÖVERINLÄRNING

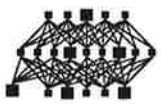
Då back-propagation och drs nätverk har tränats upp på, i storleksordning ett par hundratusen inlärningsexempel, leder fortsatt inlärning till att resultaten försämras. Nätverken mister en del av sin generaliserande förmåga, och går istället över till att minnas de exempel som de tränats på.

4.7 OSÄKRA SLUTSATSER

För att kunna dra säkra slutsatser, måste man göra flera oberoende körningar med samma indata, för att minimera statistiska avvikelser. Då nätverken tar en del tid att köra, och programmet saknar möjligheten att exekvera från kommandofiler, bör man vid en större analys av nätverk, själv försöka simulera kommandofiler. Ett försök har gjorts med ett kort program, Keyfake, som simulerar tangentbordstryckningar, men svårigheter uppstod vid inläsning till programmet. Dessa bör dock gå att övervinna.

4.8 POTENTIAL FÖR NEURALA NÄTVERK

Neurala nätverk har potential för att kunna användas för åtskilliga tillämpningar inom mönsterigenkänning och klassificering. För att nå bra resultat, krävs dock god kännedom om olika parametrars inverkan på nätverken. Denna kunskap går inte att få, enbart genom litteraturstudier, utan kräver en hel del praktiskt laborerande.



5 NEURALWORKS

5.1 ALLMÄNT

Det finns ett antal program på marknaden för att generera neurala nätverk. NeuralWorks Professional verkar vara ett av de mer avancerade. Nackdelen med detta program kan dock vara att mängden parametrar som kan varieras, till en början verkar avskräckande många. Enklare program kan vara lättare att använda, men ger samtidigt inte alla de möjligheter, som de mer avancerade har. Se artiklar [20]-[23].

NeuralWorks finns i två versioner, Explorer och Professional. Explorer är mer tänkt som en introduktion till neurala nätverk, än som ett verktyg för att skapa färdiga användbara applikationer. Professional ger däremot möjlighet att kompilera färdiga upptränade nät till C-kod för integrering i andra program. Efter uppgradering med bland annat ikonhantering kallas denna version numera Professional II/PLUS. NeuralWorks marknadsförs i Sverige av NovaCast AB, Soft Center, S-372 25 Ronneby.

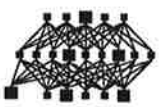
NeuralWorks Professional II, NeuralWorks Explorer, Designer Pack, InstaNet, InstaProbe, FlashCode och NeuralProbe är av NeuralWare Inc inregistrerade varumärken.

5.2 IMPLEMENTERADE NÄTTYPER

I Neuralworks finns följande nättyper implementerade

adaline
art1
back-propagation
bam
Boltzmann
brain-state-in-a-box
cascade correlation
category learning
counterpropagation
delta-bar-delta
extended-delta-bar-delta
digital neural network
directed random search
Hamming
Hopfield
learning vector quantization
madaline
perceptron
probabilistic neural network
recirculation
self organizing map
spatio temporal pattern recognition

Back-propagation är den klart mest använda typen av nätverk. I programmet har implementeringen av just denna nättyp underlättats genom ett speciellt menyval, back-prop-builder.



5.3 NÄTVERKSPARAMETRAR

Följande parametrar kan användaren själv variera.

5.3.1 Antal lager och noder

Bortsett från att vissa typer av nätverk kräver ett visst minimum antal lager, kan antalet lager och noder, i respektive lager, fritt väljas. Till exempel bi-directional-associative-memory, bam, består av inlager, två gömda lager samt ett utlager.

5.3.2 Summationsfunktion

Summationsfunktionen bestämmer på vilket sätt de inkommande signalerna till en nod skall sättas samman. Innan summering multipliceras varje insignal med motsvarande viktsfunktion. Den summerade signalen skrivs I_i, där i står för nod nummer i i det aktuella lagret. Viktsfunktionen mellan nod i det aktuella lagret och nod j underliggande lager skrivs som W_{ij}. Utsignalen från nod j i underliggande lager skrivs som X_j.
Följande alternativ finns

sum — Vanlig summation av insignalerna.
$$I_i = \sum_j W_{ij} X_j$$

cumsum — Summation som ovan men även det gamla summationsvärdet adderas.
$$I_i = I_{iold} + \sum_j W_{ij} X_j$$

maximum — Den största insignalen väljs.
$$I_i = \underset{j}{\mathrm{MAX}} (W_{ij} X_j)$$

minimum — Den minsta insignalen väljs.
$$I_i = \underset{j}{\mathrm{MIN}} (W_{ij} X_j)$$

majority — Antalet insignaler med värde större än noll räknas. Därefter subtraheras antalet insignaler mindre än eller lika med noll.
$$I_i = \sum_j sgn(W_{ij} X_j)$$

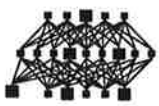
forts

norm1	Olika normaliserande funktioner. Se [18] sid 170-174.
normN	d:o
normScale	d:o
normPolar	d:o
normMult	d:o
product	Produkten av alla insignaler. $I_i = \prod_j W_{ij} X_j$
city_block	Manhattannormen av avståndet mellan viktvektorn och insignalvektorn. $I_i = \sum_j \lvert X_j - W_{ij} \rvert$
spr	Speciella funktioner för respektive nätverk.
dnna	d:o
s o m	d:o

5.3.3 Överföringsfunktion, transfer function

Överföringsfunktionen anger hur insignalen omformas till nodens utsignal. Utsignalen skrivs som T, I enligt ovan. Följande funktioner finns

tanh	Vanliga tangenshyperbolicusfunktionen. $T = \tanh(I)$
sigmoid	Sigmoid funktionen. $T = \frac{1}{1 + e^{-I \times Gain}}$
linear	Linjär överföring. $T = I$
bsb	Brain-state-in-a-box, linjär funktion. $T = I \times Gain$
sine	Vanlig sinusfunktion. $T = \sin(I)$



signum — Signumfunktion.

$$T = \begin{Bmatrix} 1, I > 0 \\ -1, I \le 0 \end{Bmatrix}$$

signum0 — Signum noll.

$$T = \begin{Bmatrix} 1, I > 0 \\ 0, I = 0 \\ -1, I < 0 \end{Bmatrix}$$

stepfunction — Stegfunktion.

$$T = \begin{Bmatrix} 1, I > 0 \\ 0, I \le 0 \end{Bmatrix}$$

bam — Bam även kallad psifunktionen.

$$T = \begin{Bmatrix} 1, I > 0 \\ T, I = 0 \\ -1, I < 0 \end{Bmatrix}$$

perceptron — Perceptron, styckvis linjär.

$$T = \begin{Bmatrix} I, I > 0 \\ 0, I \le 0 \end{Bmatrix}$$

Efter det att transferfunktionen applicerats skalas T enligt

$$T = Scale \times T_{old} + Offset$$

och begränsas enligt

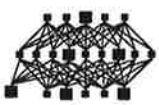
$$T = \begin{Bmatrix} Övre gräns, t > Övre gräns \\ Undre gräns, t < Undre gräns \\ T, för övrigt \end{Bmatrix}$$

Den senare operationen är väsentlig i nät av typen brain-state-in-a-box, bsb.

För sigmoid och tangenshyperbolicusfunktionen adderas ytterligare ett konstant offsetvärde, F' Offset, till derivatan av dessa transferfunktioner. Operationen medför att noder i back-propagationnät kan fortsätta inlärningen, även sedan de blivit mättade.

5.3.4 Utfunktion, output function

Hur vinnande nod väljs i ett konkurrerande lager, bestäms av utfunktionen. Denna typ av lager förekommer i de av Kohonen modellerade self-organizing-maps, [17] sid 307-327. I back-propagationnät utnyttjas inte denna funktion, utan är fördefinierad, "direct", eftersom ingen konkurrens förekommer mellan noderna.



5.3.5 Felfunktion, error function

Felfunktionen anger hur felet mellan önskad och verklig utsignal skall beräknas. Detta fel används vid uppdateringen av vikterna. Följande funktioner är valbara

standard	Ingen omformning.
quadratic	Felet kvadreras men behåller tecknet.
cubic	Felet tas i kub.
tolerant	Använder koefficient 3 i inlärningsschemat (5.3.11) som det felvärde som kan anses vara noll. När detta värde nås avbryts inlärningen.

Egna varianter för hur felet skall beräknas kan också specificeras. Denna möjlighet har inte utnyttjats i detta arbete.

5.3.6 Inlärningsregel, learning rule

Hur vikterna i nätet skall uppdateras, bestäms av vilken inlärningsregel som används. För olika inlärningsregler se [1].

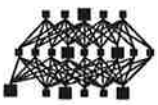
5.3.7 Kontrollstrategi

Informationsflödet mellan de olika lagren bestäms av kontrollstrategin, som definierar hur data skall flyttas under inlärningen. T ex vid nät av typ bam sker inlärningen under ett oscillerande förlopp, där data transporteras fram och tillbaka mellan två lager till dess ett stabilt tillstånd har uppnåts. Hur dessa data förflyttas beskrivs i nätets kontrollstrategi. De olika nättyperna har sina olika strategier. Egna strategier kan skrivas i ett assemblerliknade språk. Här har dock bara de fördefinierade använts.

5.3.8 Brusfunktion, noise

För att komma ur lägen där man fastnat i lokala minima, adderas brus. Man anger om och i så fall vilken typ av brus som skall adderas till nodens insignal, innan denna transformeras till den blivande utsignalen. Möjliga alternativ är

uniform	Till den summerade insignalen adderas ett likafördelat slumptal i intervallet plus minus en procent av temperaturen som anges i inlärningsschemat.
Gaussian	Som ovan men normalfördelat slumptal adderas.
none	Inget brus adderas.



5.3.9 Epochstorlek

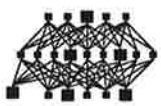
Antalet inlärningsexempel som lärs in mellan varje uppdatering av deltavikterna, beror av epochstorleken. En epoch motsvarar alla exempel som bearbetas före varje uppdatering av vikterna. När vikterna mellan olika noder uppdateras, lagras dessa ändringar av vikterna, i deltavikterna.

5.3.10 Initiering av vikter

Vid initiering av nätverkets vikter, antar dessa slumpmässiga värden mellan övre och undre initieringsgränserna.

5.3.11 Inlärningsscheman, Learn/Recall Schedule (L/R)

Vid inlärningen uppdateras vikterna i nätet så att deras värde antingen ökar eller minskar beroende på det aktuella felet. Om stora ändringar tillåts, går visserligen inlärning fort men instabilitet blir följden. Storleken på ändringarna bestäms av koefficienterna i inlärningsschemat. Genom att börja med större värden på dessa koefficienter och sedan minska dem efterhand fås bättre och snabbare inlärning. Koefficienternas betydelse framgår av [18] sid 181-190.



5.4 ÖVRIGA PARAMETRAR

5.4.1 Skärmparametrar, display style

Skärmparametrarna innehåller information, för hur nätverket skall ritas upp på skärmen. Färger på noder och förbindelser, storlek på noder, bakgrundsfärg etc.

5.4.2 Skalning av in- och utdata, minmaxtabell

In och utdata till näten ligger inte alltid i rätt intervall för att passa summations- och överföringsfunktionerna. En omskalning är för det mesta nödvändig. I minmaxtabellen anges vilka intervall som begränsar in- och utsignaler. Se [17] och [18] för utförligare beskrivning .

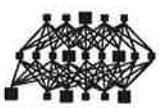
5.4.3 Instrument

För att lättare kunna studera hur inlärningen fortskrider, hur data grupperar sig eller för att se andra samband mellan olika data, finns möjligheten att ansluta instrument till enskilda noder eller lager. Instrumenten är diagram av olika slag, vanliga kurvdiagram, stapeldiagram i en eller två dimensioner, histogram samt två andra typer kallade Hinton respektive Confusion matrix, se [18] sid 118-119. Värden från instrument kan loggas i filer.

5.5 FILHANTERING

5.5.1 Nätverk

Genererade och tränade nätverk kan sparas med alla sina parametrar. Nätverken kan sparas som binärfiler, asciifiler eller kommenterade asciifiler. För att spara plats på lagringsmedia bör större nätverk sparas i binärformat. Nätverksfilerna sparas med suffixet .nnd.



5.5.2 Tabeller

I NeuralWorks finns flera tabeller där parametrar av olika slag får sina värden. Dessa tabeller kan sparas och laddas från filer. Varje tabell som på detta sätt kan sparas har sin egen filtyp och sitt eget unika suffix, enligt nedan.

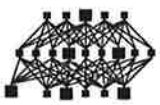
skärmparametrar	.nns	[11.1.5]
inlärningsscheman	.nnt	[11.1.6]
minmaxtabeller	.nnm	[11.1.7]
kontrollstrategier	.nnc	[5.3.7]
felfunktion	.nne	[5.3.5]

Då ett nätverk sparas i en .nnd-fil lagras även alla tabeller enligt ovan.

5.5.3 Datafiler

De data, som läses och genereras vid inlärning och testning, lagras sekvensiellt i filer.

indata	.nna	Asciikodade infiler.
	.nnb	Binärkodade infiler. Indata kan sparas antingen i asciiform eller i binär form. Binärkoden sparar utrymme i minnet och på lagringsmediat, ger snabbare exekvering men är ej möjlig att senare läsa i editor. Asciifiler kan i programmet konverteras till binärfiler [11.1.2].
utdata	.nnr	Då nätverket är upptränat kan det testas med kommandot 'test' eller 'recall'. Härvid läses data från en infil varvid nätverket genererar en utsignal. Den önskade utsignalen samt den genererade skrivs ut på en ny fil med samma namn som infilen, men med suffixet '.nnr'. Eftersom namnet på utfilen blir detsamma som på infilen, måste flera identiska infiler med olika namn användas, då man vill testa flera olika nätverk efter varandra [11.1.3].
instrument	.nnp	Instrumentvärdena kan loggas i filer för att sedan bearbetas i kalkylark eller program typ MATLAB [11.1.4].



6 INDATA, UTSEENDE OCH BEARBETNING

6.1 ALLMÄNT

Indata till neurala nätverk utgörs av vektorer. Antalet element i dessa vektorer är detsamma som antalet noder i inlagret. Indata kan därför sägas spänna upp rymden R^n då inlagret består av n noder. Ofta vill man förprocessa indata för att reducera antalet parametrar in i nätet. En variant av back-propagation, kallad 'functional link', använder sinusfunktioner i inlagret för en slags transformering.
En styrka hos neurala nätverk är att den, som vill bygga upp ett nätverk för att lösa ett visst problem, inte behöver känna till exakt hur lösningen skall formuleras. Man överlåter istället åt nätverket att pröva sig fram för att hitta en bästa lösning. Även om förbearbetning av indata ger ett bättre resultat, missar man en av poängerna med nätverk, om man specificerar problemet allt för detaljerat. I detta arbete har ingen direkt förbearbetning gjorts. De transformerade bilder som utgör indata, är den information som skall analyseras.

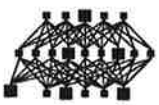
6.2 BAKGRUND

Tillämpningen är avsedd för att bland annat kunna identifiera och klassificera signaler, som registrerats med hydrofoner och andra typer av undervattensdetektorer. Signalerna har efter registrering samplats tidsmässigt, och därefter Fouriertransformerats till digitala bilder i tids-/frekvensplanet. Resultatet kan bli matriser med en bredd av flera hundra pixels. Konventionell bildbehandling bygger ofta på att man sveper ett litet 'fönster' över den aktuella bilden. Varje steg i denna svepning inkluderar en enkel analys. Samma tillvägagångssätt kan användas även i detta fall. Analysen i varje steg låter man nu nätverket utföra. Den del av bilden som nätverket behöver för sin analys, begränsas nu till den del som täcks av det lilla fönstret.

6.3 INDATA TILL NÄTVERKEN

I detta arbete består de flesta indata till nätverken av bildmatriser av storlek 7 gånger 5 pixels. Vid ett mindre antal körningar har matriser av storlek 13 gånger 9 pixels använts. Dessa bildmatriser visar alltså frekvensspektrat som funktion av tiden. Matrisernas storlek är vald genom avvägning mellan upplösning i bilden och antalet noder i nätet. Ett allt för stort antal innoder gör att hela nätstrukturen växer och därmed tar inlärningen avsevärt längre tid.

För att kunna lära nätet att känna igen linjer i dessa bildmatriser måste nätet 'få se' hur linjerna ser ut. I detta fall är linjetjockleken begränsad till 1 pixel och linjernas lutningar är begränsade till vinklar understigande 45 grader, mätt från lodräta axeln. Den senare restriktionen är dock mindre väsentlig. Genom att ändra samplingsfrekvensen, då originalsignalerna samplas, kan upplösningen i bilden, och därmed även linjernas lutningar påverkas. En högre upplösning av bilden längs tidsaxeln får linjerna att räta upp sig mot den lodräta axeln, och en lägre upplösning får dem att närma sig horisontella axeln.



6.4 LINJEBILDER, 7 GÅNGER 5 PIXELS

Rådata består av en fil där 68 stycken binära matriser med inlagda linjer lagrats. I samtliga matriser går linjen genom den centrala punkten (4,3). Den första matrisen innehåller en vertikal linje. De nästföljande 33 innehåller alla tänkbara linjer som lutar åt vänster, några med samma lutning men då med olika translationer. Därefter följer ytterligare en vertikal linjebild följd av 33, på motsvarande sätt, åt höger lutande linjer. Studera gärna bilderna på något avstånd så framträder linjerna tydligare.

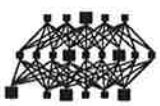
Linjemönster, vertikalt och lutningar vänster

```
00100 01000 00100 01000 00100 01000 01000 00100 01000 01000 01000 01000
00100 00100 00100 01000 00100 00100 01000 00100 01000 00100 01000 01000
00100 00100 00100 00100 00100 00100 01000 00100 00100 00100 00100 01000
00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100
00100 00100 00100 00100 00100 00100 00100 00010 00100 00100 00100 00100
00100 00100 00100 00100 00010 00100 00100 00010 00100 00010 00010 00100
00100 00100 00010 00100 00010 00010 00100 00010 00010 00010 00010 00010

01000 01000 01000 10000 01000 10000 01000 01000 10000 10000 10000 10000
00100 01000 01000 01000 00100 01000 01000 01000 10000 01000 10000 01000
00100 00100 01000 01000 00100 01000 00100 00100 01000 01000 01000 00100
00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100
00010 00010 00100 00100 00010 00100 00010 00010 00100 00010 00100 00010
00010 00010 00010 00100 00010 00010 00010 00001 00010 00010 00010 00001
00010 00010 00010 00010 00001 00010 00001 00001 00010 00001 00001 00001

10000 10000 00000 10000 10000 00000 10000 00000 10000 00000
10000 01000 10000 01000 10000 10000 01000 10000 10000 10000
01000 01000 01000 00100 01000 01000 01000 01000 01000 01000
00100 00100 00100 00100 00100 00100 00100 00100 00100 00100
00010 00010 00100 00010 00010 00010 00010 00010 00010 00010
00010 00001 00010 00001 00001 00010 00001 00001 00001 00001
00001 00001 00001 00000 00001 00001 00000 00001 00000 00000
```

Linjemönster, vertikalt och lutningar höger

```
00100 00010 00100 00010 00100 00010 00010 00100 00010 00010 00010 00010
00100 00100 00100 00010 00100 00100 00010 00100 00010 00100 00010 00010
00100 00100 00100 00100 00100 00100 00010 00100 00100 00100 00100 00010
00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100
00100 00100 00100 00100 00100 00100 00100 01000 00100 00100 00100 00100
00100 00100 00100 00100 01000 00100 00100 01000 00100 01000 01000 00100
00100 00100 01000 00100 01000 01000 00100 01000 01000 01000 01000 01000

00010 00010 00010 00001 00010 00001 00010 00010 00001 00001 00001 00001
00100 00010 00010 00010 00100 00010 00010 00010 00001 00010 00001 00010
00100 00100 00010 00010 00100 00010 00100 00100 00010 00010 00010 00100
00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100 00100
01000 01000 00100 00100 01000 00100 01000 01000 00100 01000 00100 01000
01000 01000 01000 00100 01000 01000 01000 10000 01000 01000 01000 10000
01000 01000 01000 01000 10000 01000 10000 10000 01000 10000 10000 10000

00001 00001 00000 00001 00001 00000 00001 00000 00001 00000
00001 00010 00001 00010 00001 00001 00010 00001 00001 00001
00010 00010 00010 00100 00010 00010 00010 00100 00010 00010
00100 00100 00100 00100 00100 00100 00100 00100 00100 00100
01000 01000 00100 01000 01000 01000 01000 01000 01000 01000
01000 10000 01000 10000 10000 01000 10000 10000 10000 10000
10000 10000 10000 00000 10000 10000 00000 10000 00000 00000
```

I NeuralWorks läses indata i vektorform. På varje rad i indatafilen har alla element i en matris skrivits, rad för rad, och därefter de fyra element som motsvarar utvektorn till denna invektor. Utvektorns fyra element, som var och en antingen har värdet noll eller ett, indikerar i tur och ordning linje eller inte, lutning vänster eller inte, lutning höger eller inte respektive vertikal linje eller inte. Se vidare kapitel 7.



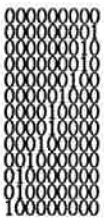
6.5 LINJEBILDER, 13 GÅNGER 9 PIXELS

De 249 olika linjebilderna med matrisstorlek 13 gånger 9 pixels har delats in i 9 klasser beroende på respektive linjes lutning. Första klassen innehåller de linjer som lutar mest åt vänster. Därefter minskar lutningarna i klasserna tills lodräta axeln har passerats, varefter lutningarna åter ökar, denna gång åt höger. Endast första och sista bild i respektive klass har nedan skrivits ut. Matriserna är sparade under olika variabelnamn. Vänsterlutande med namn l124-l1, vertikal med namn v och högerlutande med namn r1-r124. Observera att definitionen på lutningarna tyvärr har blivit omkastade mot de för 7 gånger 5 matriserna.

Linjemönster

Klass 1

l124



l96

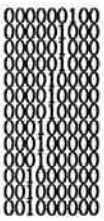
Klass 2

l95

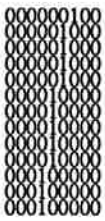
l69

Klass 3

l68



l39



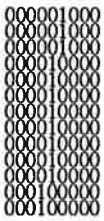
Klass 4

l38

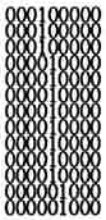
l15

Klass 5

l14

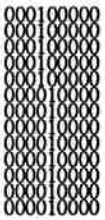


r14

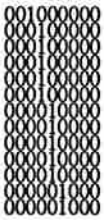


Klass 6

r15

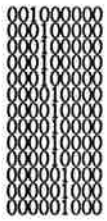


r38



Klass 7

39

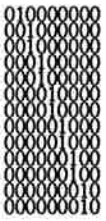


r68

Klass 8

r69

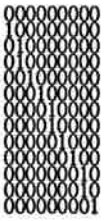
r95



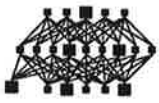
Klass 9

r96

r124



Datafilerna är i detta fall organiserade radvis, som i det tidigare fallet. Utvektorn består dock nu av 9 element, (7.2.2).



6.6 BRUS

I verkligheten kommer naturligtvis inte linjerna att vara lika lätta att hitta i bilderna. En ofrånkomlig källa till problem vid bildanalys är brus. Bruset kommer in på flera nivåer. Detektorer kan uppfatta andra signaler än de önskade, eller signaler från flera källor samtidigt. Vid överföringen av signalerna utsätts dessa för distortion och vid sampling och transformering fås alltid en viss grad av mätfel.

Om bruset från början antages ha normalfördelning med standardavvikelsen σ, blir fördelningen efter Fouriertransformering i *M* intervall, normalfördelad med standardavvikelsen

$$\sigma_{effektiv} = \frac{\sigma}{\sqrt{M}}$$

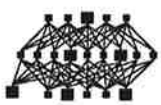
De rena signalerna blir, efter Fouriertransformering, matriser i tids-/frekvensplanet, vilka teoretisk inte har någon utsträckt fördelning. Signal/brusförhållandet, blir omvänt proportionellt mot sigma_effektiv. Se [28] - [30] för teori om signalanalys och signal/brusförhållanden. I fortsättningen nämns inget mer om signal/brusförhållanden, eftersom detta begrepp saknar betydelse i denna tillämpning. Då det talas om brusförhållanden, beskrivs dessa bara i termer av brusets fördelning, karakteriserad av standardavvikelsen sigma enligt nedan. Då man har ett intresse av att jämföra med verkliga signaler, får man räkna baklänges, utgående från det genererade brusets standardavvikelse, för att komma fram till signal/brusförhållandet.

I modellen har brus generats, i form av matriser med samma storlek som linjemönstren, med en standardavvikelse kallad sigma. Mönstermatriserna och brusmatriserna har sedan elementvis lagts ihop, och därefter har summorna kvadrerats. I och med kvadreringen, går man över till en amplitudskala, med enbart positiva värden.

I diagram 1 visas hur matrisen med linjemönster enligt

```
0 1 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 1 0
```

ser ut då brus med olika standardavvikelse adderas.



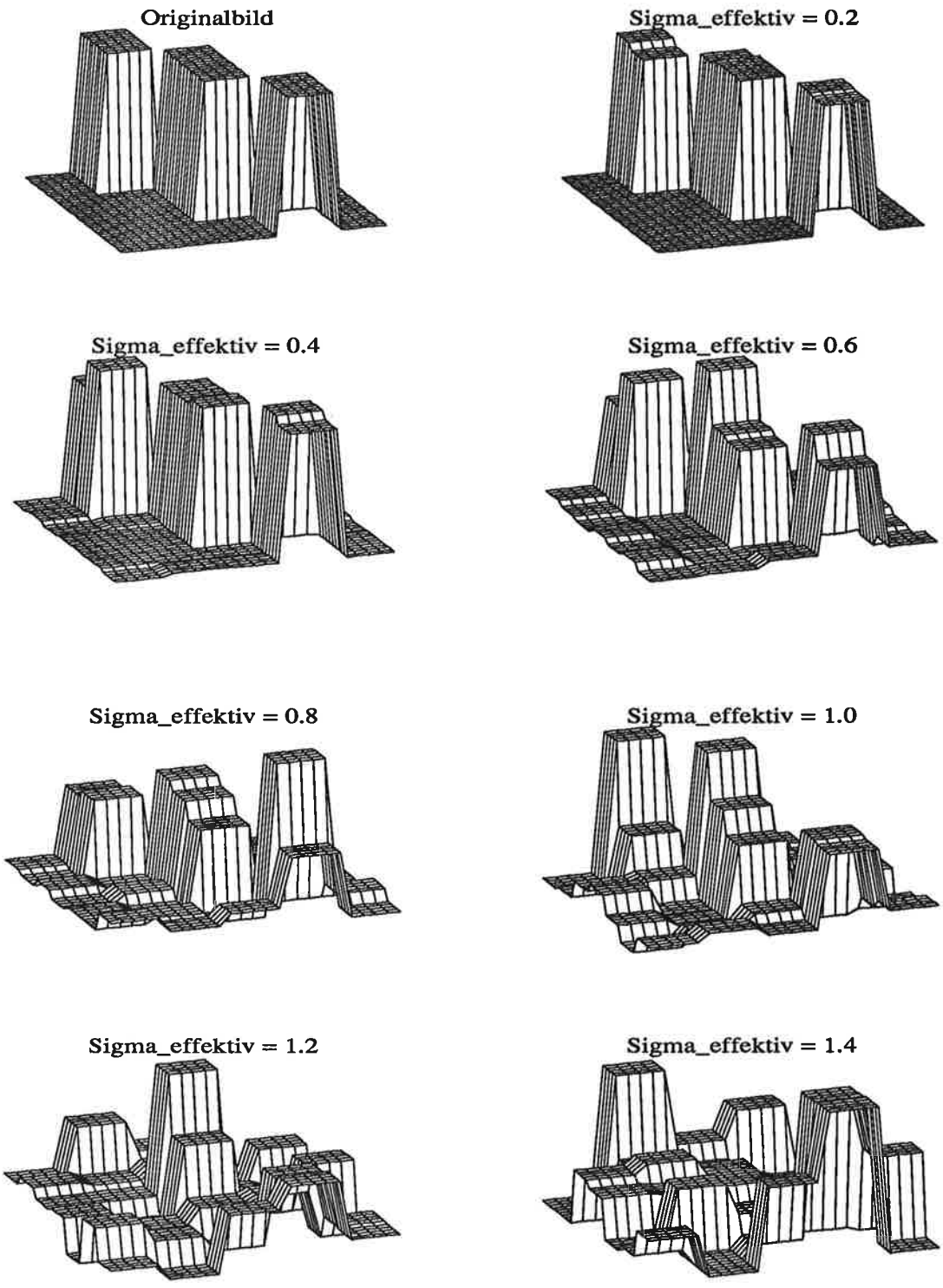


Diagram 1. Originalbild och med brus överlagrade bilder.
Brusets standardavvikelse = sigma_effektiv.

6.7 ANTAL SET AV INDATA

Om bara en linjebild för vardera lutningen inkluderas i indatafilen, blir antalet indata alltför begränsat. Ett litet antal olika inlärningsexempel får nätet att minnas vad det sett. Förmågan att kunna generalisera bland nya okända data uteblir. Rådatafilerna innehåller bara ett mönster för vardera lutningen, dvs ett set av bilder. Ett set består, i fallet 7 gånger 5 pixels, av en bild vardera av de bilder som visas i (6.4), samt 6 stycken bilder med vertikala linjer och 72 bilder utan någon avsiktlig linje alls, dvs totalt 144 olika bilder. I fallet 13 gånger 9 pixels består ett set, på motsvarande sätt, av 249 bilder.

Om man tar flera set av originalmönster, adderar brus till dessa och sparar dem sekvensiellt i samma indatafil, får man ett större urval av data som ger en bättre inlärning för nätverken. Indata med bara en omgång mönster kan visserligen tyckas ge snabbare konvergens då man studerar hur RMS-felet avtar, men detta är skenbar effekt eftersom nätet då troligen bara lärt sig att känna igen just dessa indata, med fixt brus. För bildstorleken 7 gånger 5 pixels, har inte fler än 8 set indata använts, eftersom detta leder till allt för stora datafiler. Asciifilen för 8 set är av storleksordningen 1 megabyte.

7 UTDATA

7.1 ALLMÄNT

Beroende på vilka uppgifter nätverket skall lösa, varierar antalet utnoder. Auto-associativa nätverk har utdata med samma form som indata, och lika många noder i in- och utlager.
Hetero-associativa nätverk saknar denna likhet. I t ex problem av typen klassificering motsvarar vardera kategorin en utnod.

7.2 DEFINIERING AV UTNODER

I denna applikation är uppgiften att identifiera och klassificera linjer i bilder. Enligt vad som nämnts tidigare, har två olika bildstorlekar använts, 7 gånger 5 respektive 13 gånger 9 pixels.

7.2.1 7 gånger 5 pixels

I detta fall består utdata av en vektor med fyra element, som i tur och ordning markerar om en linje finns, om lutningen är åt vänster, om lutningen är åt höger respektive om linjen är vertikal. Vid inlärningen får nätet veta vilka in- och utdata som hör samman. En etta i en utnod innebär att den aktuella bilden har den egenskap, som markeras av denna nod, och en nolla motsatsen. De värden på utsignalerna, som nätet lärs upp på, är således bara ett respektive noll.

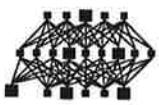
7.2.2 13 gånger 9 pixels

I den lite större bilden har utdata formulerats aningen annorlunda. Nio utnoder svarar mot nio olika lutningar för en linje. Ingen särskild nod indikerar om en bild innehåller en linje eller inte. Om någon nod är aktiverad innebär detta att bilden innehåller en linje. Första noden svarar mot linjen med störst lutning åt vänster, och den sista svarar mot den med störst lutning åt höger.

Nätverk av typ back-propagation och drs har en tendens att förutom den korrekta noden, även aktivera grannarna till denna. Konkurrerande lager i nätverk, såsom i pnn och s o m, ger nätet en bättre selektiv förmåga, att endast aktivera den avsedda utnoden.

7.3 TOLKNING AV UTDATA

Utsignalerna från nätverken blir ingalunda lika distinkta som de signaler på vilka nätet lärts upp. I fallet klassificering när antingen etta eller nolla förväntas som utsignal, borde den verkliga signalen inte vara alltför svår att känna igen. Men att enbart runda av till närmaste heltal, dvs ett eller noll, kan vara förödande. Man kan helt komma att misstolka signaler, som ligger strax under den gräns man satt upp för detektering.



Om en kategori är underrepresenterad i indata får detta till följd, att den nod som skall indikera motsvarande egenskap, ger betydligt svagare signaler. Ett utvärde som borde varit runt 1.0 hamnar kanske istället på 0.4 eller därunder. Hela intervallet tycks i detta fall bli nerskalat. Var man sätter gränserna eller trösklarna är högst väsentligt, för att inte missa de värden man vill kunna indikera.

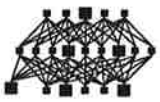
I diagram 2 och 3 ses exempel på hur utdata fördelar sig i histogram med 20 intervall vid bildstorlek 7 gånger 5 pixels. Diagram 2 härrör från från en körning med nätverk 'linje820', vilket är av typ back-propagation, medan diagram 3 kommer från nätverk 'linj1770' vilket är av typ s o m.

Indatafilen till de båda körningarna innehöll 33 bilder med linjer lutande åt vänster, 33 lutande åt höger, 6 vertikala samt 72 helt slumpmässiga bilder utan avsiktliga linjer. Utnoderna indikerar i tur och ordning linje, lutning vänster, lutning höger respektive vertikal linje.

Bilden över utnod 1, i diagram 2, visar att värdena har fördelat sig i två intervall. Värdena i det undre intervallet motsvarar utsignalen noll, och i det övre utsignalen ett. I bilderna för utnod 2 och 3, i samma diagram, har intervallen flutit ihop, men man kan sätta gränsområdet runt 0.65 respektive 0.75. Det är svårare att se gränserna i histogrammet för utnod 4, men en möjlig gräns ses vid 0.75. I de tre första bilderna är andelen utsignaler hörande till värdet ett relativt stort, medan det syns att de vertikala linjerna är underrepresenterade, i utnod 4. Staplarna runt värdet 1.0 är här ganska låga.

I diagram 3 är histogramspektra inte kontinuerliga som ovan, utan bara vissa värden är representerade. Detta gör det svårare att sätta gränser mellan utsignalerna ett och noll. Eftersom ingen av noderna har utsignal runt 0.5, kan gränsen lämpligen sättas vid detta värde i samtliga fall, i diagram 3. Speciellt i s o m nätverk bör data fördelas så att alla kategorier är representerade med lika många exempel. I bilden över utnod 4, ser man att få data klassificerats som utsignal ett, och dessutom att hela värdeintervallet skalats ned, största värde runt 0.65. S o m är känsligare än back-propagation, för underrepresenterade indata.

Diagrammen skall inte tolkas som ett mått på hur bra nätverken lyckas klassificera, utan avser belysa skillnaden i fördelning av utdata, och konsekvensen av underrepresenterade indata. Back-propagation ger utdata i ett kontinuerligt spektrum. Mer än en utnod kan vara aktiverad samtidigt. S o m ger kvantiserade utdata och aktiverar, p g a konkurrens i lagren, bara en utnod åt gången.



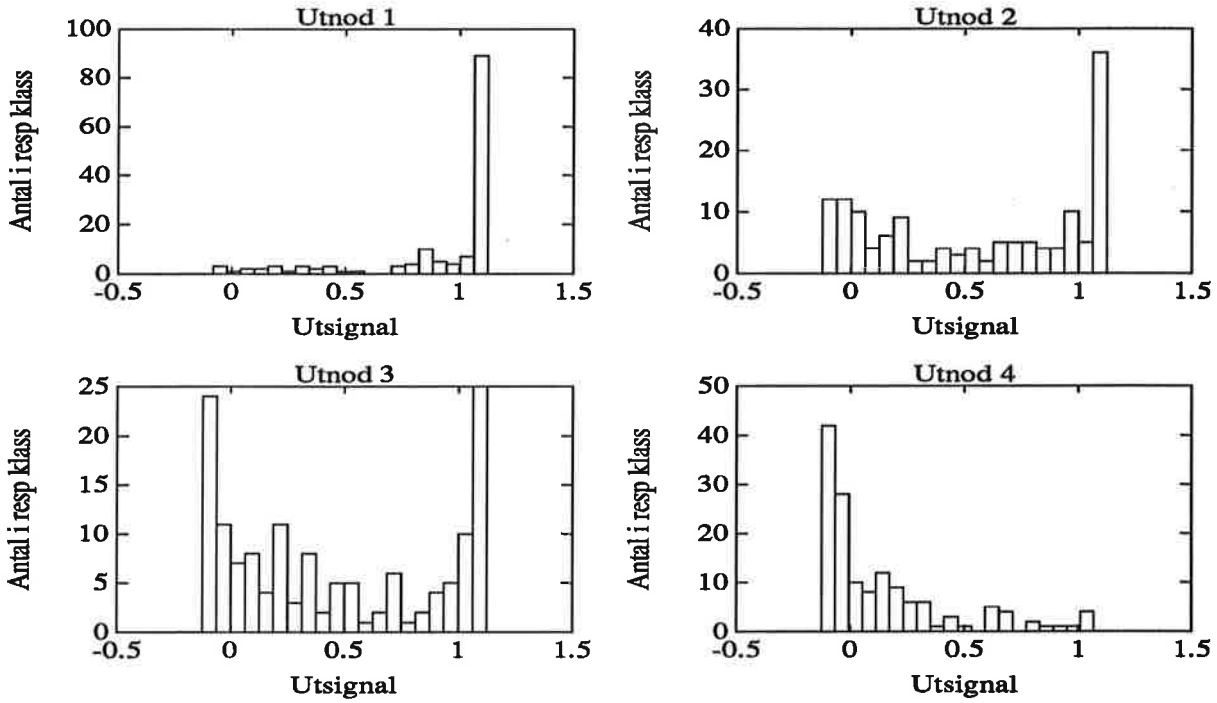


Diagram 2. Fördelningen av utnodernas värden i histogram med 20 intervall, back-propagation. Noderna indikerar i tur och ordning linje, lutning vänster, lutning höger respektive vertikal linje.

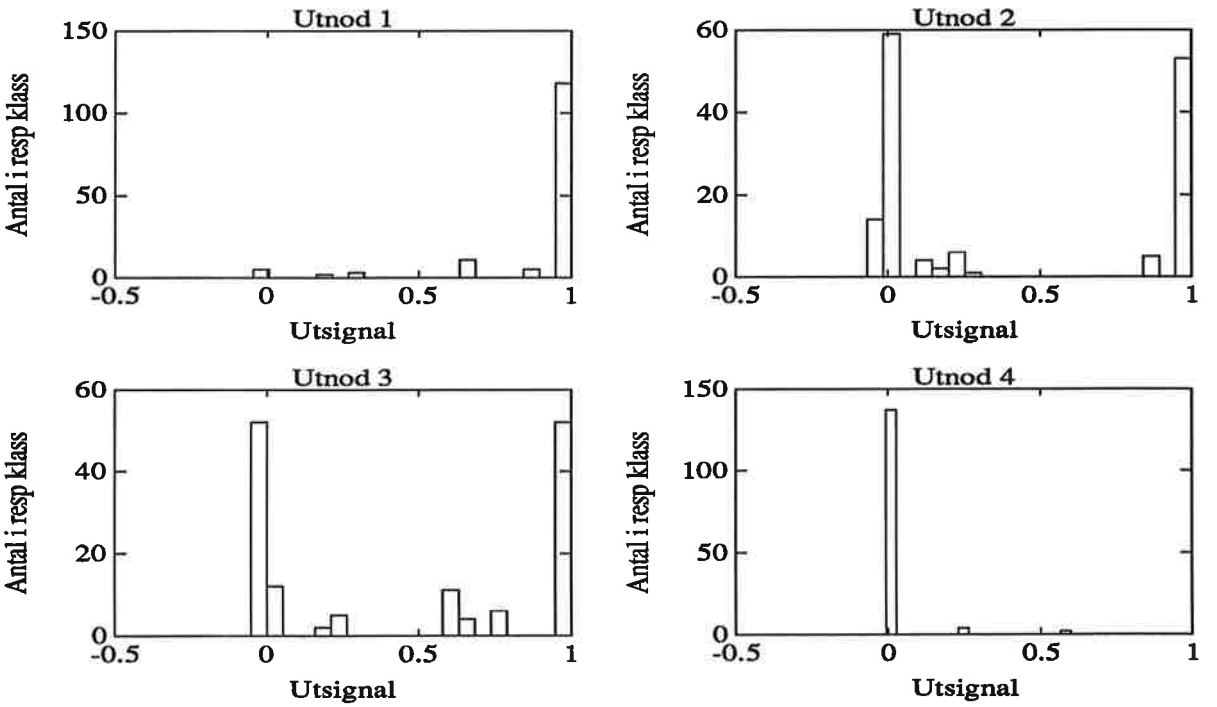


Diagram 3. Fördelningen av utnodernas värden i histogram med 20 intervall, s o m. Noderna indikerar i tur och ordning linje, lutning vänster, lutning höger respektive vertikal linje.

8 RESULTAT AV KÖRNINGAR MED NÄTVERK

8.1 ALLMÄNT

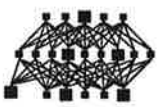
Huvuddelen av körningarna har ägnats åt back-propagationnätverk eftersom detta är den vanligast förekommande typen av nätverk. Andra nätverk som har testats är direct-random-search, probabilistic-neural-network och self-organizing-maps. Namnen på dessa brukar förkortas till drs, pnn respektive s o m.

I flera olika typer av diagram har resultaten av körningarna sammanställts. I de diagram där RMS-felet visas, definieras detta fel som roten ur summan av kvadraterna av felen i utnoderna, dvs skillnaden mellan önskad utsignal och den beräknade utsignalen. RMS-felet beroende av antalet inlärda exempel visar hur inlärningen artar sig. I de flesta fall avtar detta fel långsamt för att så småningom plana ut på en konstant nivå. Observera att skalan längs y-axeln, inte alltid är den samma, inom ett och samma diagram.

Standardavvikelsen av RMS-felet kan sägas vara ett slags mått på stabiliteten vid inlärningen. Denna standardavvikelse beror i högsta grad på hur ofta instrumenten loggar sina data. Eftersom denna loggning inte sker lika ofta, i alla körningar, bör inte diagram från olika körserier jämföras, utan bara de som är ritade i samma diagram.

Vid generering av nätverken anger användaren typ och storlek på nätet, dvs hur många lager och hur många noder i varje lager, som nätet skall bestå av. En mängd parametrar får sina defaultvärden vid genereringen. Andra värden på dessa parametrar kräver att de ändras explicit. I filförteckningen och i tabellerna över defaultvärden anges vilka parametrar som ändrats explicit vid respektive körning. Man inser snabbt att alla nätverk, med olika kombinationer av parametrar, blir en omöjlighet att prova. I detta arbete har huvudsakligen inverkan av olika parametrars värden, på back-propagation nätverk, studerats.

De flesta typer av nätverk kräver vanligen flera tusen inlärningsexempel. Men eftersom många inlärningsexempel medför körningar under lång tid, har i de flesta körningarna 'bara' 3000 inlärningsexempel exekverats. Dessa kortare körningar visar inte nätens fulla prestanda, utan visar vilken betydelse olika parametersättningar får för nätens utresultat. För några nätverk har körningar med betydligt längre inlärningsserier gjorts. Där inget annat anges, har indata med bildstorlek 7 gånger 5 pixels, använts.



8.2 BACK-PROPAGATIONNÄT

8.2.1 Allmänt

Back-propagationnätverk behöver relativt många inlärningsexempel för att visa godtagbar upplärning, [6]. För att minimera inlärningstiden har ett par varianter av denna nättyp utvecklats.

Enligt [19] kan antalet noder i det gömda lagret uppskattas med formeln

$$antal \quad noder = \frac{antal \quad exempel}{10 \times (m+n)}$$

där m betecknar antalet innoder, n antalet utnoder och *antal exempel* antalet inlärningsexempel i indatafilen. Formeln räknar dock alldeles för lågt. I fallet med 7 gånger 5 pixels och 8 set av indata, anger formeln antalet till 2. Vanligtvis rekommenderas att storleksordningen av antalet noder i in- och mellanlagret bör vara den samma. Detta antal ger dock så komplexa nätverk att inlärningen tar väl lång tid, varför antalet noder i de flesta fall har satts till 13. Se även [5].

De parametrar som varierats vid körningarna är

Brusnivåer
Antal noder i gömda lagret
Utlagrets
- Summationsfunktion
- Överföringsfunktion
- Felfunktion
- Brus
- Inlärningsregel
- Utfunktion

Antal förbindelser mellan gömda lagret och utlagret
Antalet set av indata se (6.7).

Övrigt om back-propagation, se [7 och 8].

8.2.2 Fördefinierade värden, default, back-propagation nätverk

*) Defaultvärde som ändrats explicit under körningarna, se vidare i filförteckningen.

Nätverket

Nätverkstyp	Hetero-Associativt	
Kontrollstrategi	Back-propagation	
Inlärningsschema	Back-propagation *)	

Inlagret

Summationsfunktion	Sum	
Transferfunktion	Linjär	
Skalning	Skala	1.0
	Offset	0.0
Gränser	Undre gräns	-9999
	Övre gräns	+9999
Felfunktion	Standard	
Utfunktion	Direkt	
Inlärningsregel	Ingen	
F' Offset	0.0	
Initiering	undre gräns	-0.1
	övre gräns	+0.1

Gömda lagret och utlagret

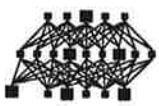
Summationsfunktion	Sum *)	
Transferfunktion	Sigmoid *)	
Skalning	Skala	1.0
	Offset	0.0
Gränser	Undre gräns	-9999
	Övre gräns	+9999
Felfunktion	Standard *)	
Utfunktion	Direkt *)	
Inlärningsregel	Delta regeln *)	
F' Offset	0.0	
Initiering	undre gräns	-0.1
	övre gräns	+0.1

8.2.3 Olika brusnivåers inverkan på nätverkens inlärningsförmåga

Härledning till hur brusnivån definieras återfinns i (6.6). I diagram 4 och ses hur RMS-felet varierar med olika brusnivåer.

Parametrarna i dessa nätverk har sina fördefinierade värden satta, i de fall något fördefinierat värde existerar. Antalet noder i det gömda lagret är 10. Nätverken benämns 'linje050' upp till 'linje086', se filförteckningen kap 11.

För brus med standardavvikelse 0.5 till 0.7 syns klart hur RMS-felet avtar redan efter bara några hundra inlärda exempel. Det snabba avtagandet kan dock delvis förklaras med att bara ett set av indata använts. Om fler set av indata använts skulle nätet troligen ha lärt sig att generalisera bättre, även om RMS-felet då hade varit större. Med bara ett set indata, lär sig nätet snarare att känna igen just dessa data, än att verkligen lära sig att generalisera på ett riktigt sätt bland okända indata. Då sigma > 1.0 blir konvergensen dålig, i de fall den kan noteras. I de senare körningarna har brus med standardavvikelse 0.5 använts, där inget annat värde nämns.



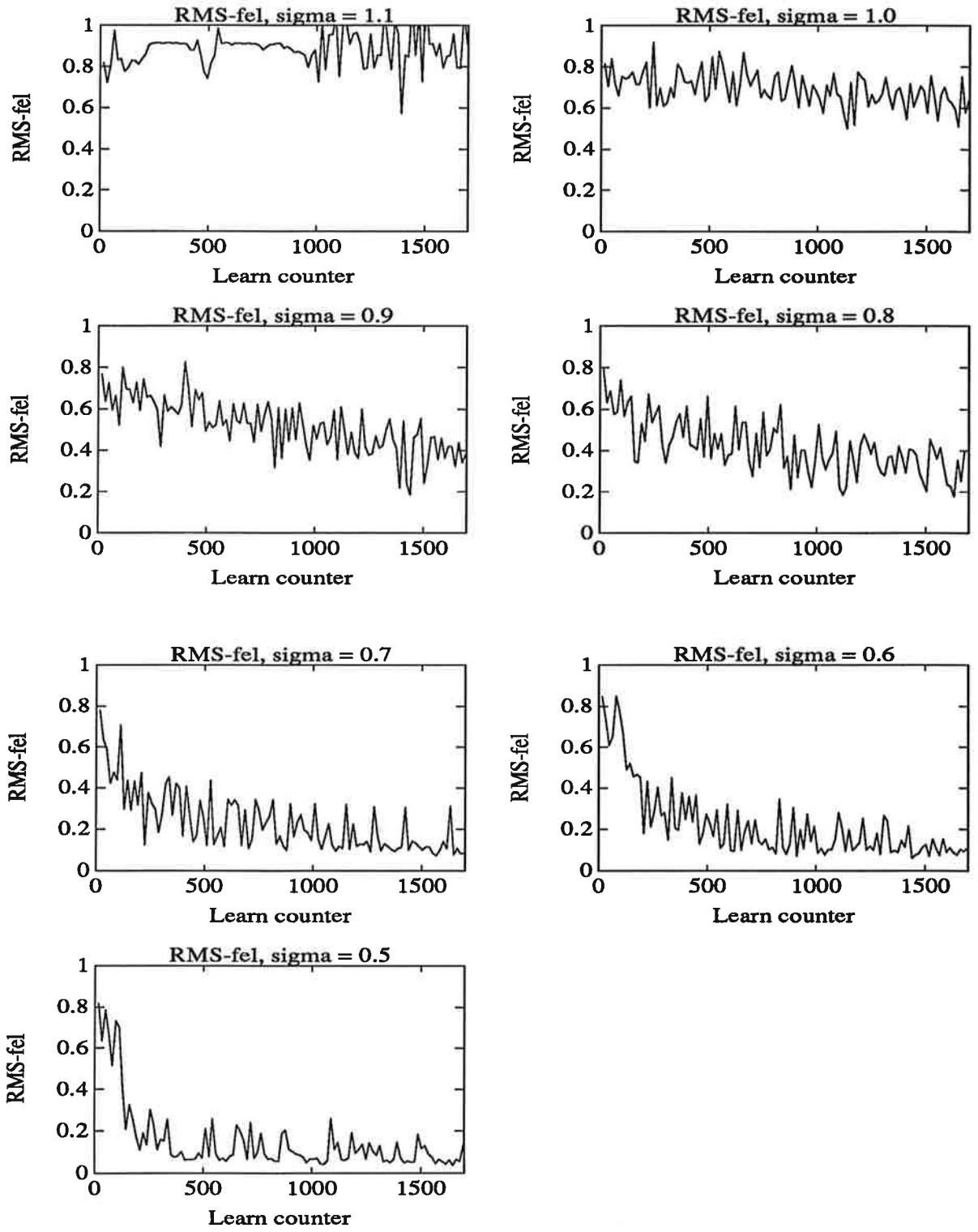


Diagram 4. RMS-felet som funktion antalet inlärda exempel, för olika standardavvikelse, sigma, på det överlagrade bruset.

8.2.4 Antal noder i det gömda lagret

Antalet noder i det gömda lagret bestämmer till viss del hur mycket information som skall passera nätet. Sättes antalet lågt blir nätet informationskomprimerande, man tappar en del av informationen, vilket dock kan vara till nytta vid t ex bild-kompression. I (8.2.1) nämns en formel som är tänkt att ge rätt antal noder i detta lager. I denna tillämpning ger dock formeln värden som ligger åtminstone en tiopotens från det värde som ger någorlunda bra resultat. Alla värden från 1 upp till 40 har prövats, och därefter vart femte värde upp till 60.

Nätverken kallade 'linje100' till 'linje540' har fördefinierade parametervärden, och brus med standardavvikelsen 0.8 har överlagrats på indata.

För vart femte värde från 5 till 60 ses i diagram 5 a) och b) hur RMS-felet avtar med antalet inlärda exempel i respektive fall.
Observera att skalan längs y-axeln inte är den samma i fallet 30 noder i gömda lagret.

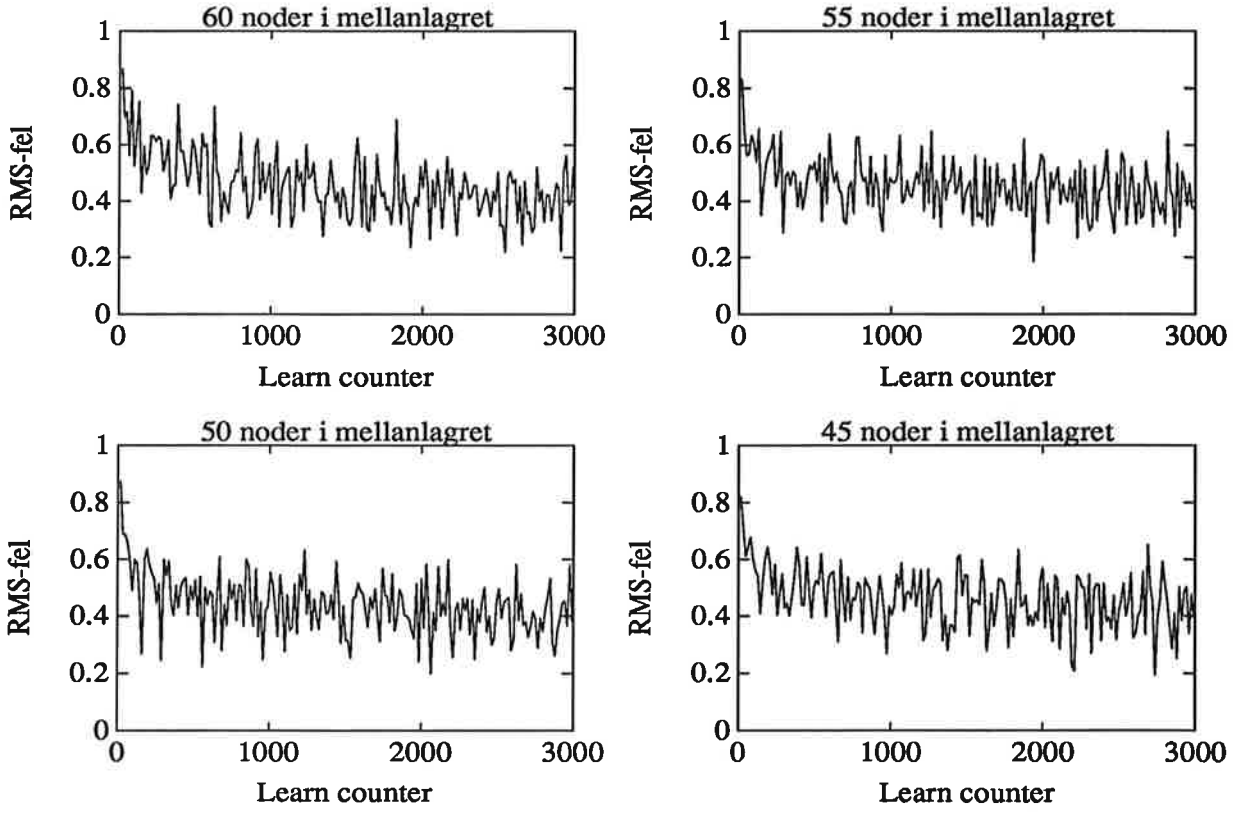


Diagram 5 a) RMS-felet som funktion antalet inlärda exempel, för olika antal noder i det gömda lagret. Brusets standardavvikelse sigma = 0.8

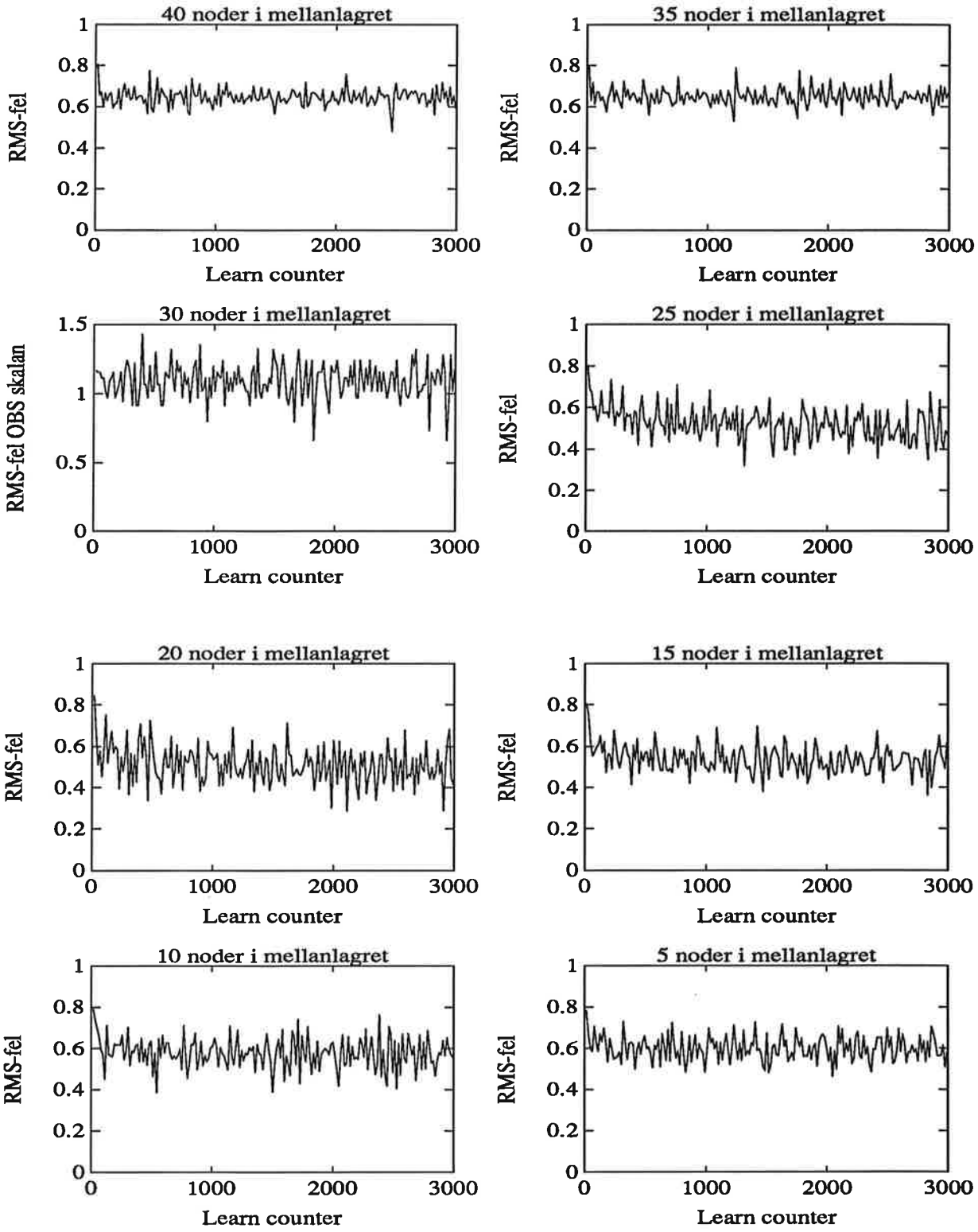


Diagram 5 b) RMS-felet som funktion antalet inlärda exempel, för olika antal noder i det gömda lagret. Brusets standardavvikelse sigma = 0.8

Antalet inlärda exempel vid körningarna ovan var 3000. I diagram 5 a) och b) ser man att RMS-felet avtar snabbare då fler gömda noder använts. Ett medelvärde av RMS-felet, taget över inlärningsexempel 2500 till 3000, ger en mer lättolkad bild av hur antalet noder påverkar inlärningen. I diagram 6 a) och b) är dels detta medelvärde av RMS-felet ritat som funktion av antal gömda noder, dels standardavvikelsen av detta medelvärde ritat på samma sätt. Standardavvikelsen ger en antydan om hur stabilt felkurvan avtar. Vart femte värde, från 5 till 60 noder, samt varje värde, från 1 till 40 noder, är representerade i kurvorna.

En intressant iakttagelse är, att ett slags motsatsförhållande råder mellan felets storlek och stabiliteten i inlärningen. Detta syns tydligast i diagram 6 b). Dessutom ser man att för vissa kritiska värden, 30 respektive 34 gömda noder, fås ett relativt stort RMS-fel, se diagram 6 a). Överst i diagram 6 b) ser man att RMS-felet, utanför intervallet 25 till 45 noder, avtar ganska jämnt med antalet gömda noder.

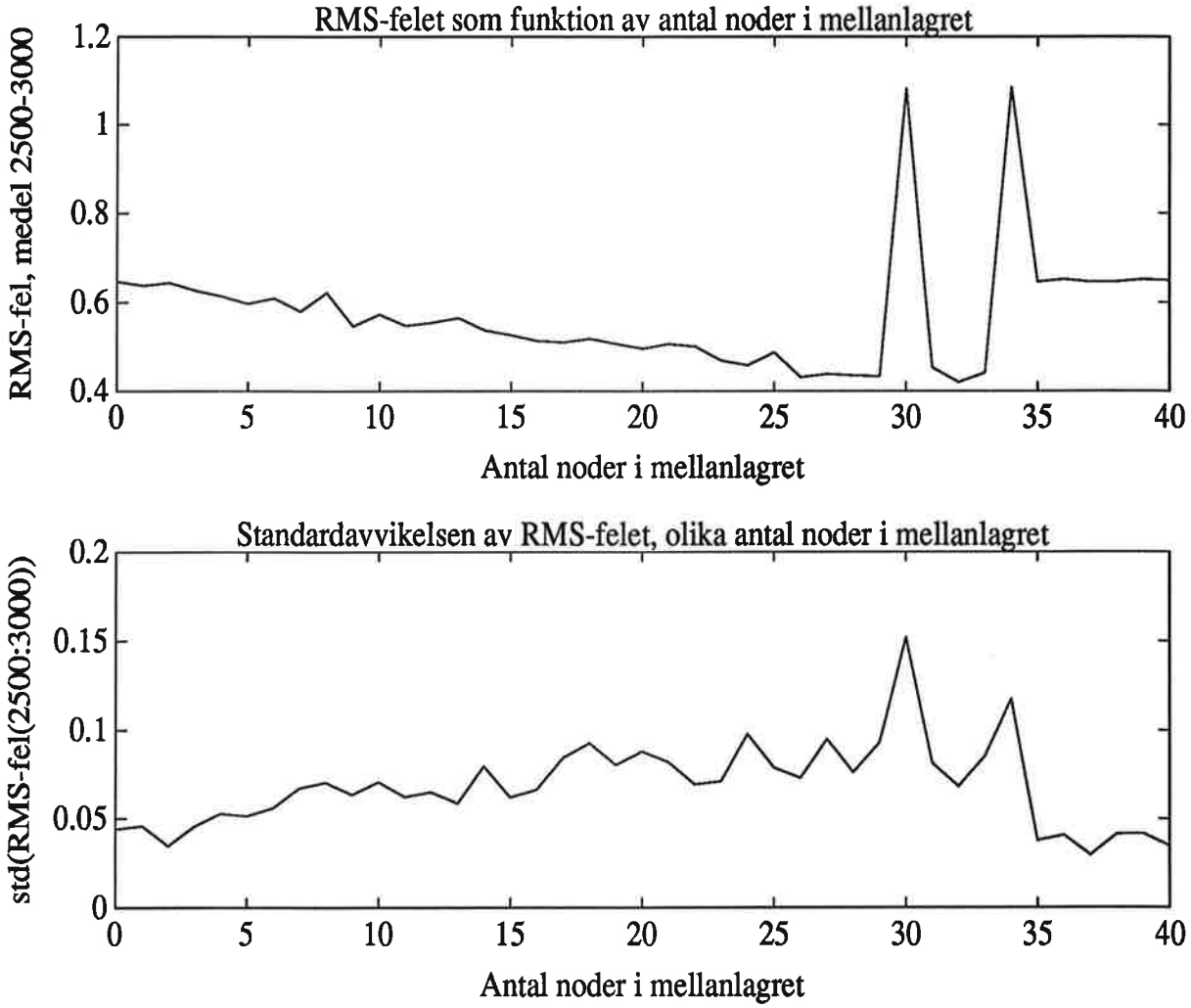


Diagram 6 a) RMS-felet samt dess standardavvikelse som funktion av antal noder i det gömda lagret. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 5.

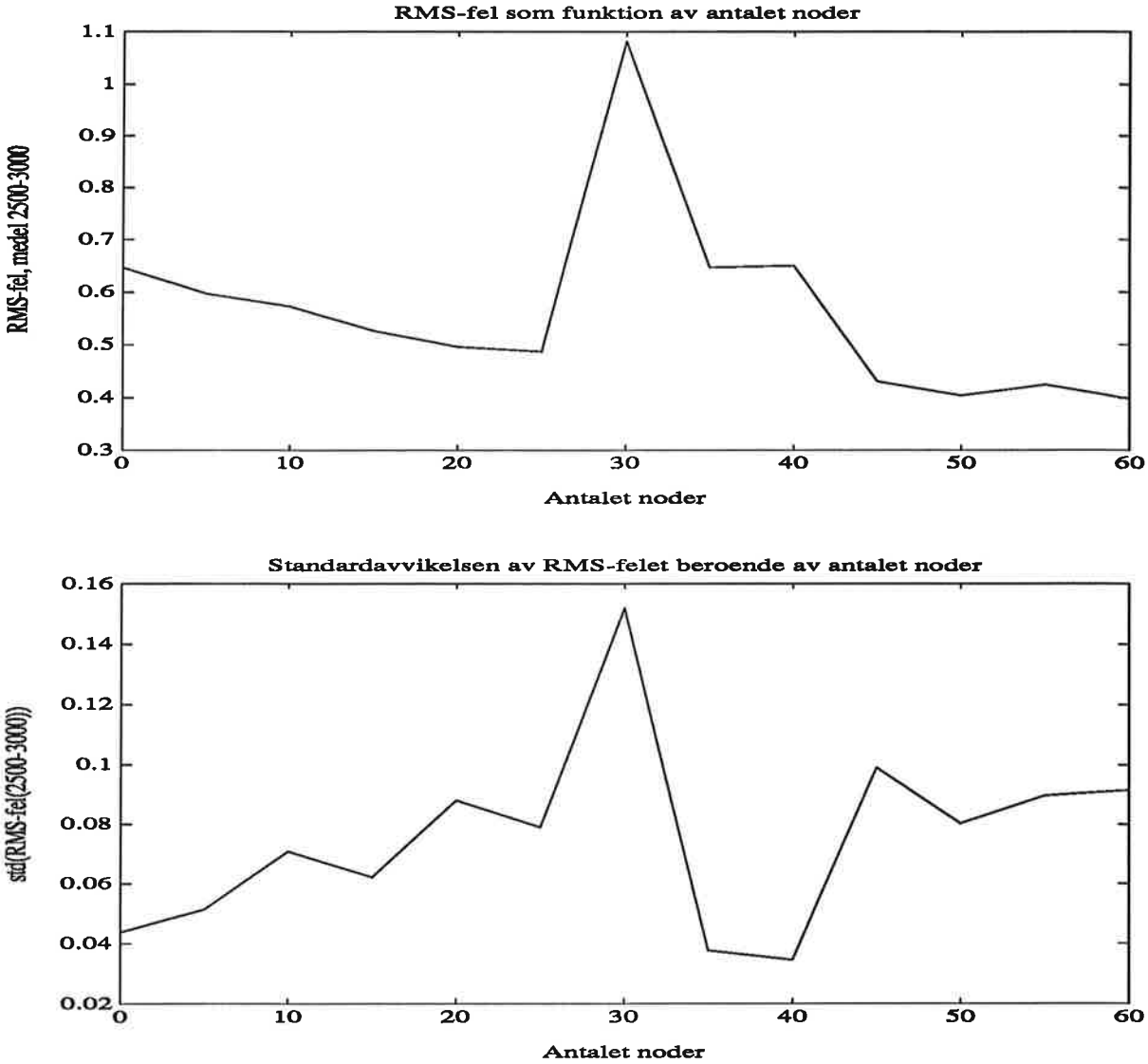
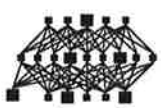


Diagram 6 b) RMS-felet samt dess standardavvikelse som funktion av antal noder i det gömda lagret. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 5.

Orsaken till det instabila området mellan 25 till 45 noder, är inte fastställd. Ett nät med få noder i det gömda lagret, får främst generaliserande förmåga, medan många noder i detta lager gör att nätet istället börjar minnas de data som det tränats på. Om antalet noder i gömda lagret är mindre, än det antal då nätet börjar minnas, och större, än det antal då det bara generaliserar, kan man tänka sig att funktionen blir nedsatt, i detta mellanliggande intervall.

Rita om möjligt diagram, på samma sätt som diagram 6 är ritat, för att upptäcka och därmed undvika dessa olämpliga intervall.



8.2.5 Parametrar i utlagret

8.2.5.1 Allmänt

Bara en parameter åt gången har varierats i utlagret. Antalet kombinationer med olika parametervärden är mycket stort, och därför kan inte alla olika parameteruppsättningar kombineras, eftersom detta skulle ta alldeles för lång tid att utvärdera. Av samma anledning har inte parametervärden i det gömda lagret varierats. I det gömda lagret sker för det mesta linjära summationer och transformationer. Därför har det bedömts som intressantare att studera vad som händer då utlagret modifieras, eftersom operationerna där är mer olinjära och dynamiska.

Vissa alternativ nedan till summations- och transferfunktion är oftast inte speciellt lämpliga, då de är framtagna för speciella nätverkstyper.

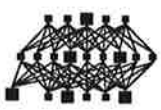
8.2.5.2 Summationsfunktion

Körning med nätverken 'linje820' till 'linje960', fördefinierade värden på parametrar, 13 noder i gömda lagret och 8 set av indata.

I diagram 7 a) och b) visas RMS-felet under inlärningen för olika summationsfunktioner. Observera att skalan längs y-axeln inte är densamma p g a allt för olika utsignaler. De funktioner som ger användbara resultat är sum, majority, normScale, normPolar och normMult. Sum är den funktion som från början är satt som defaultvärde. Funktionerna spr, dnna och s o m som är direkt avsedda för respektive nätverkstyp ger inte speciellt bra resultat i back-propagationnätverk. I diagram 8 är medelvärdet av RMS-felet samt dess standardavvikelse ritat för de olika summationsfunktionerna. Medelvärdet och standardavvikelsen har beräknats över exempel 2500 till 3000. Jämför med diagram 7 a) och b).

I diagram 7 a) syns att cumsum, norm1 och normN har planat ut alltför tidigt, på en hög felnivå. I diagram 8 syns visserligen att dessa funktioner är stabila, men de är tämligen ointressanta, eftersom de blir statiska efter ett kort insvängningsförlopp. Orsaken till detta, kan vara att vikterna låst fast sig i lokala felminima, som är svåra att komma ur. Norm1 och normN är dessutom bara avsedda att användas, då det gömda lagret innehåller en nod.

Den fördefinierade funktionen sum, är tillsammans med normScale, normPolar och normMult, de funktioner som rekommenderas vid back-propagation.



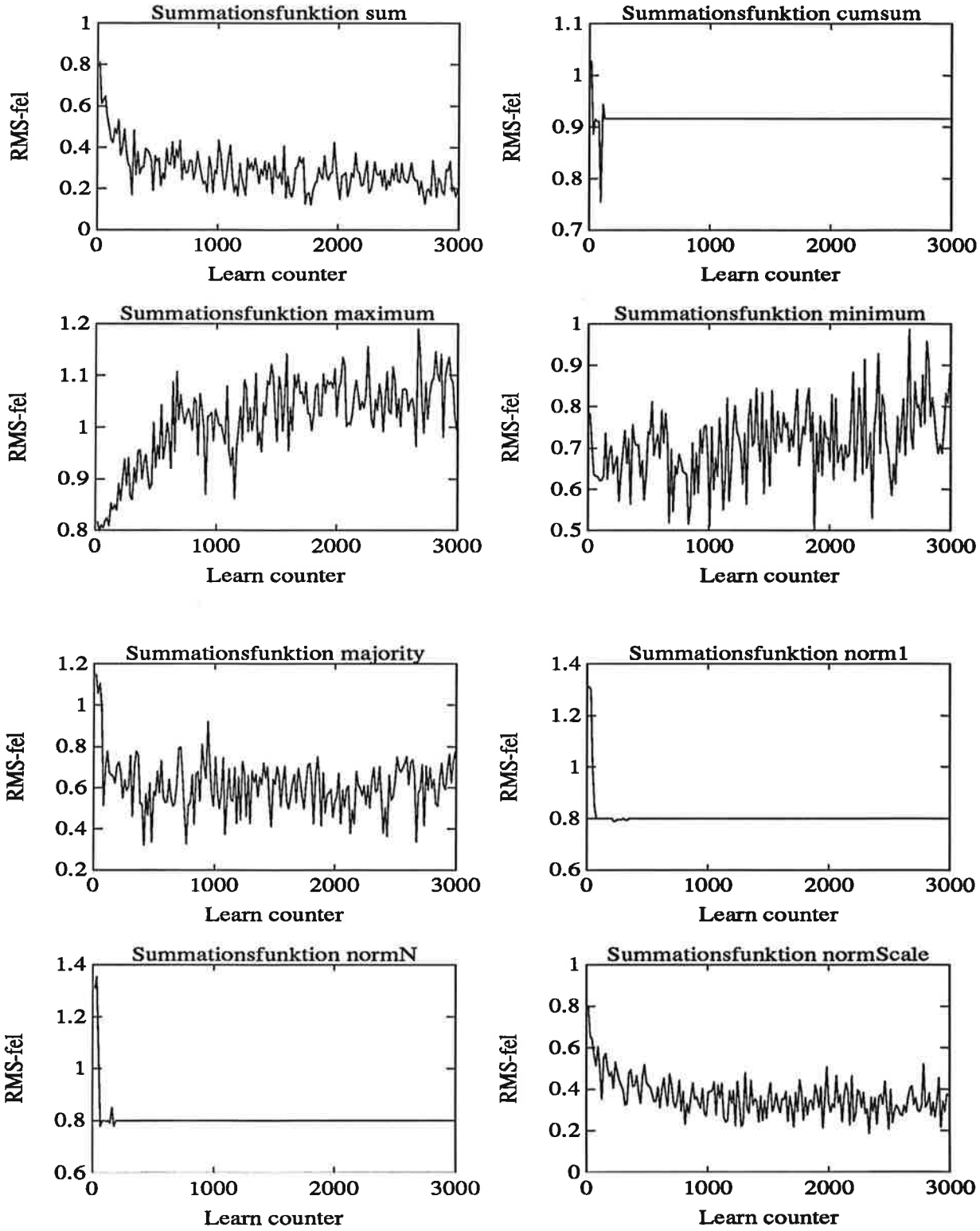


Diagram 7 a) RMS-felet som funktion av antal inlärda exempel för olika summationsfunktioner, 13 noder i gömda lagret, 8 set av indata.

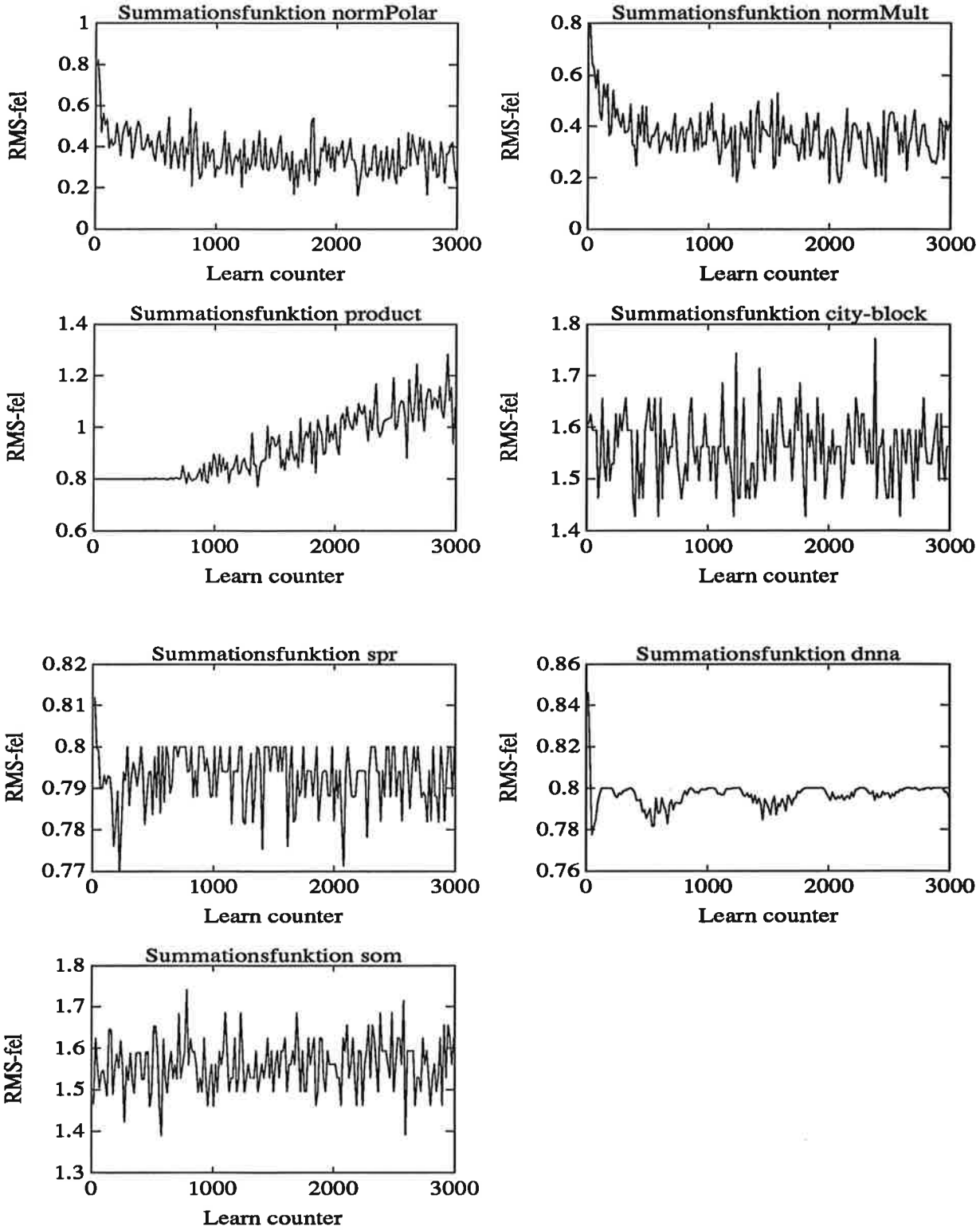


Diagram 7 b) RMS-felet som funktion av antal inlärda exempel för olika summationsfunktioner, 13 noder i gömda lagret, 8 set av indata.

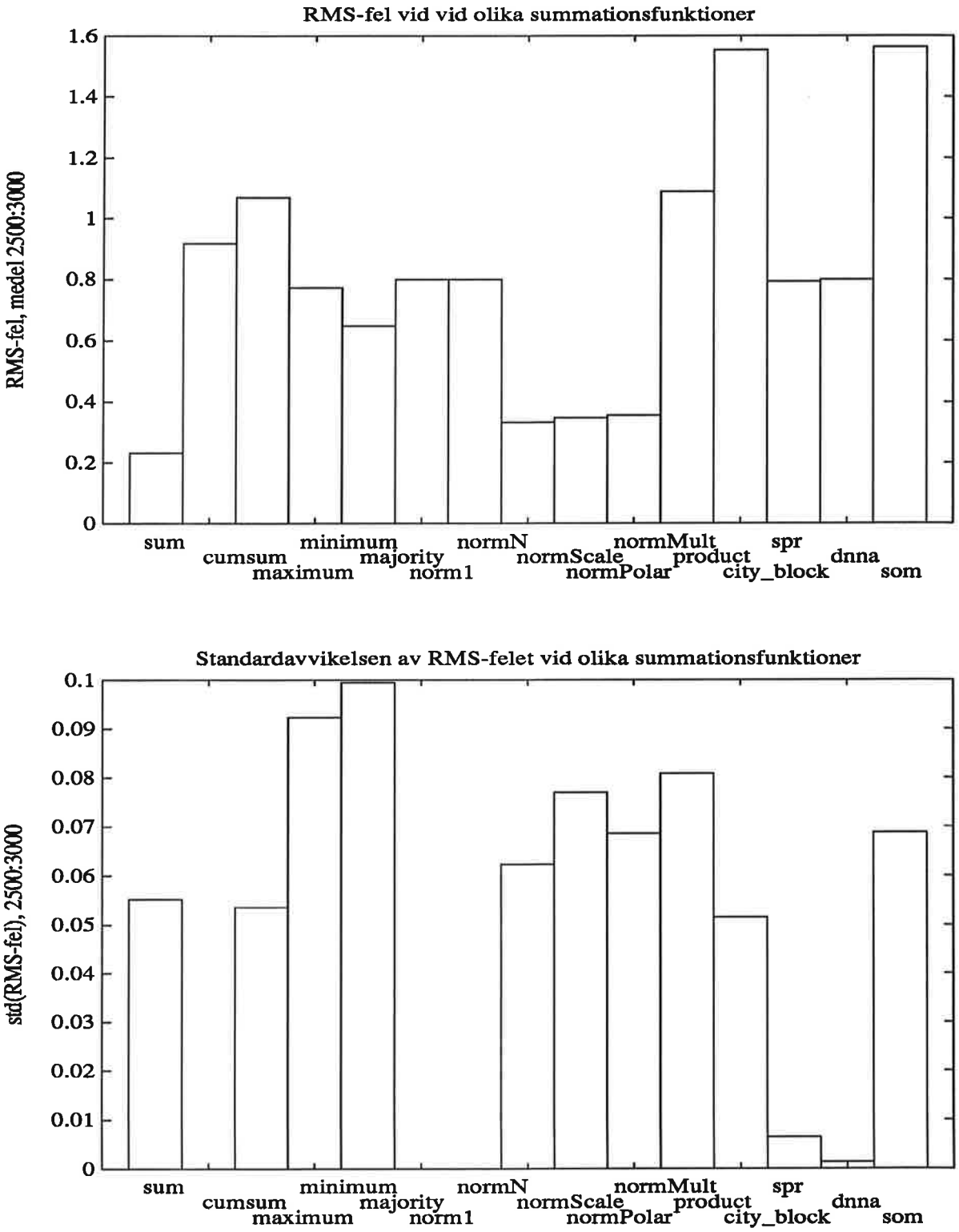


Diagram 8. RMS-felet samt dess standardavvikelse som funktion av summationsfunktionen. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 7 a) och b).

8.2.5.3 Överföringsfunktion, transfer function

Olika överföringsfunktioner har undersökts i nätverken 'linje720' till 'linje810'. Parametrarna, förutom överföringsfunktionen i utlagret, är satta till sina fördefinierade värden, gömda lagret innehåller 13 noder och indata består av 8 set.

I diagram 9 a) och b) visas resultatet av inlärningen. Observera att skalan längs y-axeln, inte är densamma i alla bilder. De två vanligaste överföringsfunktionerna i back-propagationnät, tangenshyperbolicus och sigmoidfunktion, visar bra resultat. Sigmoidfunktionen verkar ge en jämnare felkurva men planar å andra sidan ut på en högre nivå. Tangenshyperbolicus rekommenderas i manualerna till Neural-Works, som den mest användbara överföringsfunktionen i back-propagation nätverk. Diagram 10 visar medelvärde och standardavvikelse för RMS-felet, mätt över inlärningsexemplen 2500 till 3000. Här syns tydligare att även funktionerna bsb, linear och sine, ger bra resultat. Stegfunktionen (stepfunction) ser ut att vara stabil, men i diagram 9 syns att denna funktion planar ut alldeles för snabbt för att verkligen vara bra. Bam- och perceptronfunktionerna, speciellt lämpade för dessa nätverkstyper, ger inte heller användbara resultat.

I back-propagation rekommenderas att använda någon av överföringsfunktionerna tanh, linear, sine eller bsb. Prova i respektive tillämpning att funktionen verkligen ger hyfsade resultat. Beroende på andra parametrar kan vissa funktioner vara olämpliga.

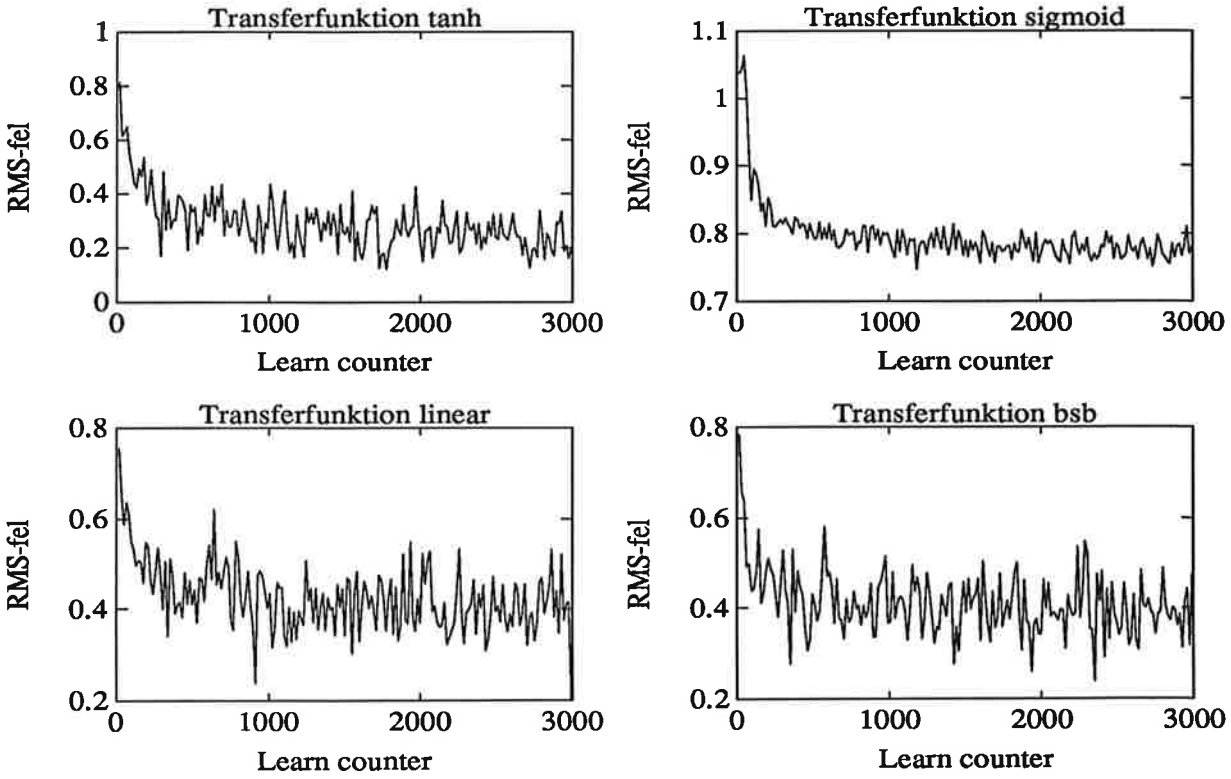


Diagram 9 a) RMS-felet som funktion av antal inlärda exempel för olika överföringsfunktioner, 13 noder i gömda lagret, 8 set av indata.

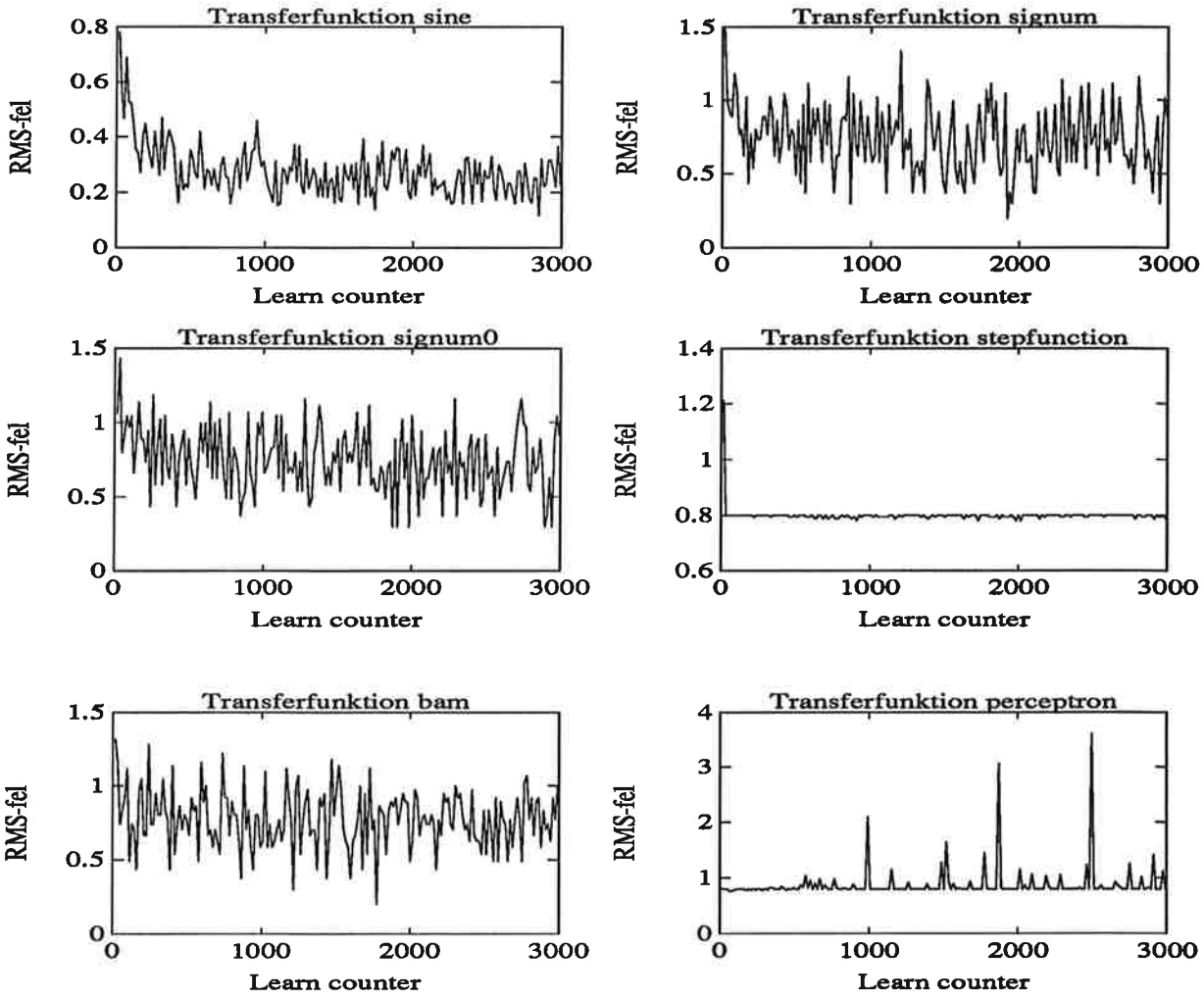


Diagram 9 b) RMS-felet som funktion av antal inlärda exempel för olika överföringsfunktioner, 13 noder i gömda lagret, 8 set av indata.

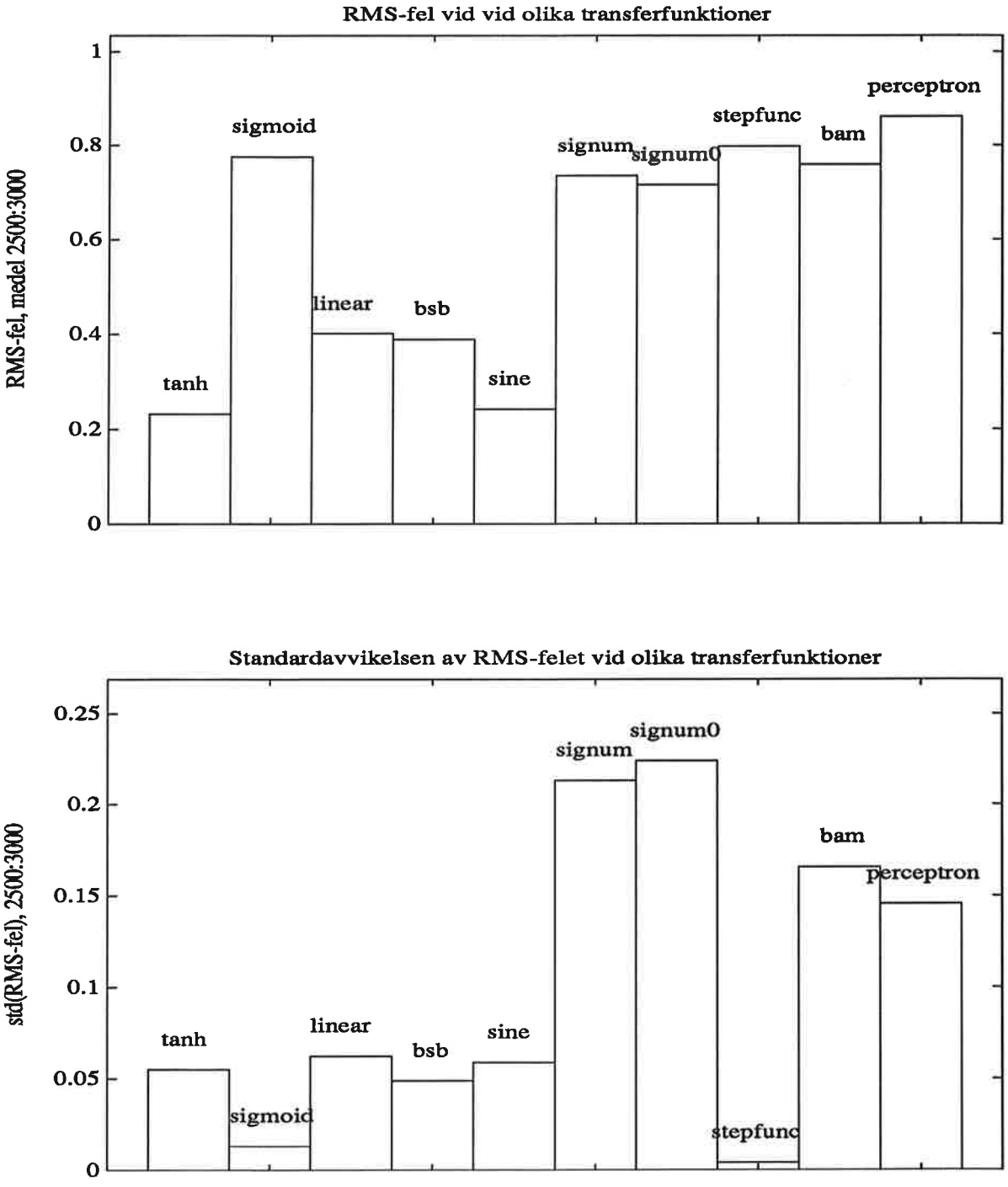
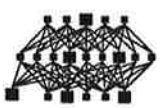


Diagram 10 RMS-felet samt dess standardavvikelse som funktion av överföringsfunktionen. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 9 a) och b).



8.2.5.4 Felfunktion, error function

Back-propagation bygger på att felet beräknas i utlagret och sedan skickas vidare bakåt. Detta fel kan beräknas på olika sätt. I körningarna 'linje970'till 'linj1000' har parametrarna satts till sina fördefinierade värden, utom felfunktionen som varierats. Antalet noder i gömda lagret sattes till 13 och indata bestod av 8 set.

I diagram 11 är RMS-felet under inlärningen uppritad för olika felfunktioner. Diagram 12 visar medelvärdet och standardavvikelsen av RMS-felet, beräknat över exemplen 2500 till 3000. Standard och quadratic är ungefär likvärdiga. Cubic ger ett mindre fel men är samtidigt inte lika stabilt. Tolerant som har nästan samma låga fel som cubic, är dessutom den mest stabila funktionen, varför tolerant nog får anses ge det bästa resultatet av felfunktionerna.

Standard är den fördefinierade felfunktionen. Denna, samt även tolerant rekommenderas som lämpligaste felfunktioner i back-propagation nätverk.

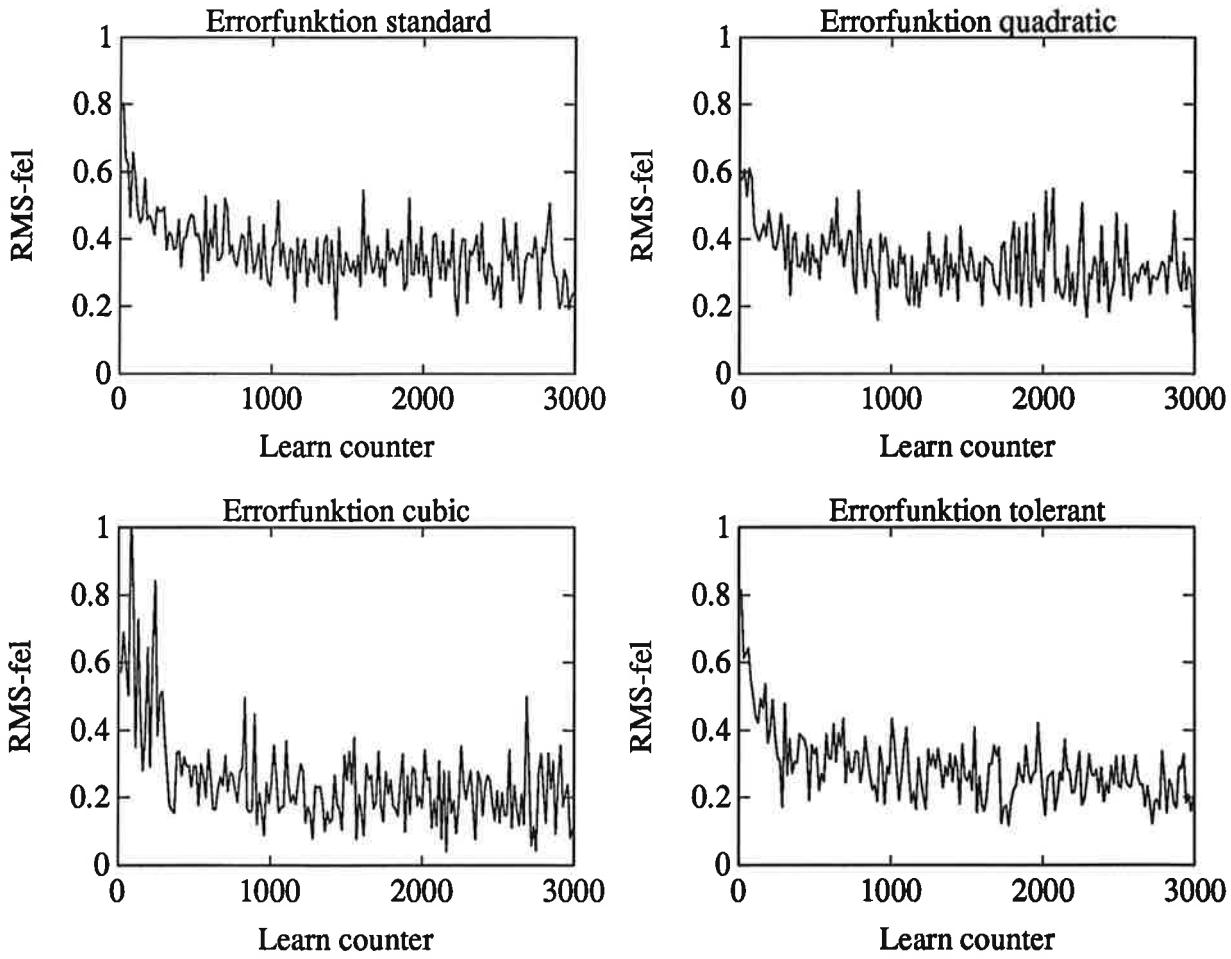


Diagram 11. RMS-felet som funktion av antal inlärda exempel för olika felfunktioner, 13 noder i gömda lagret, 8 set av indata.

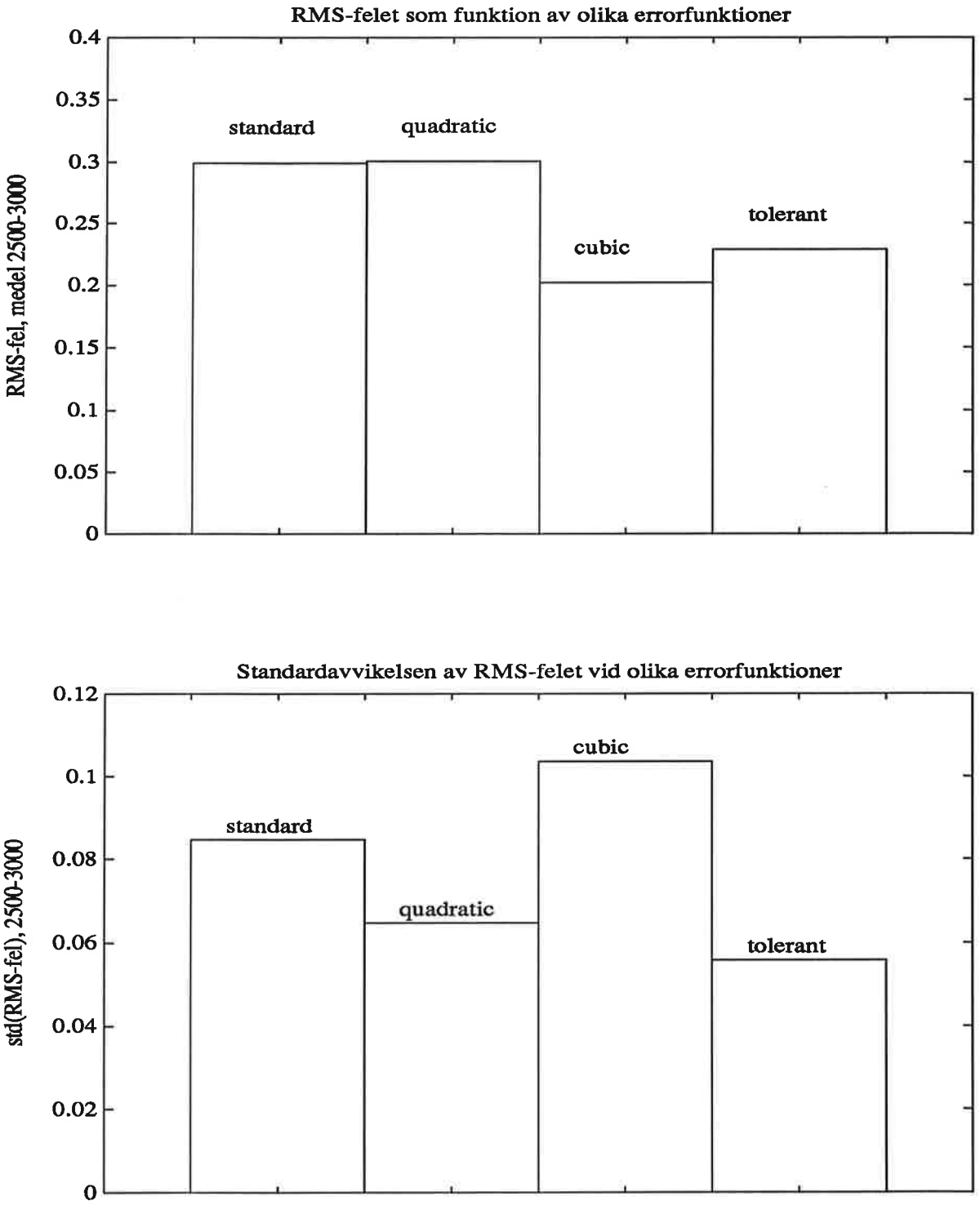


Diagram 12. RMS-felet samt dess standardavvikelse som funktion av felfunktionen. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 11.

8.2.5.5 Brus, noise function

För att undvika att fastna i lokala minima då felet skall minimeras adderar man brus till vikterna vid uppdatering. Nätverken 'linje970' , 'linj1010' och 'linj1020' har olika brusfunktion, men övriga parametrar satta till sina fördefinierade värden. Antal gömda noder är 13 och 8 set av indata har använts.

RMS-felets avtagande under inlärningen kan studeras i diagram 13, och dess medelvärde och standardavvikelse i diagram 14. Standard och none är stort likvärdiga, medan gaussian ger ett lägre RMS-fel. Gaussian innebär att normalfördelat brus adderats till vikterna. Notera att indata till programmet redan har överlagrats med just normalfördelat brus.

Skillnaderna mellan de olika alternativen är små. Eftersom gaussian ger normalfördelat brus, vilket är vanligt i många sammanhang, bör denna funktion vara den som lämpar sig bäst för back-propagation.

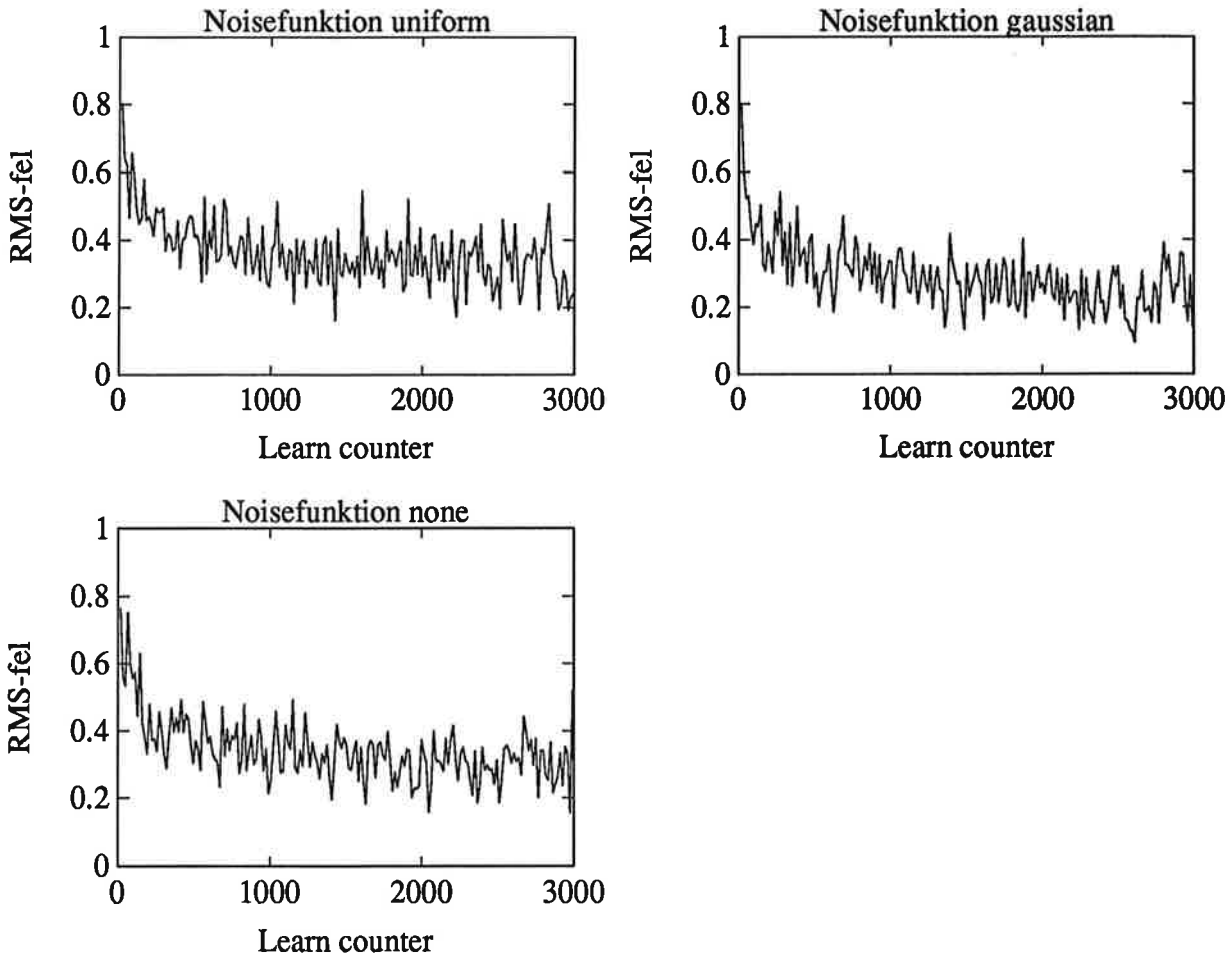


Diagram 13. RMS-felet som funktion av antal inlärda exempel för olika brusfunktioner, 13 noder i gömda lagret, 8 set av indata.

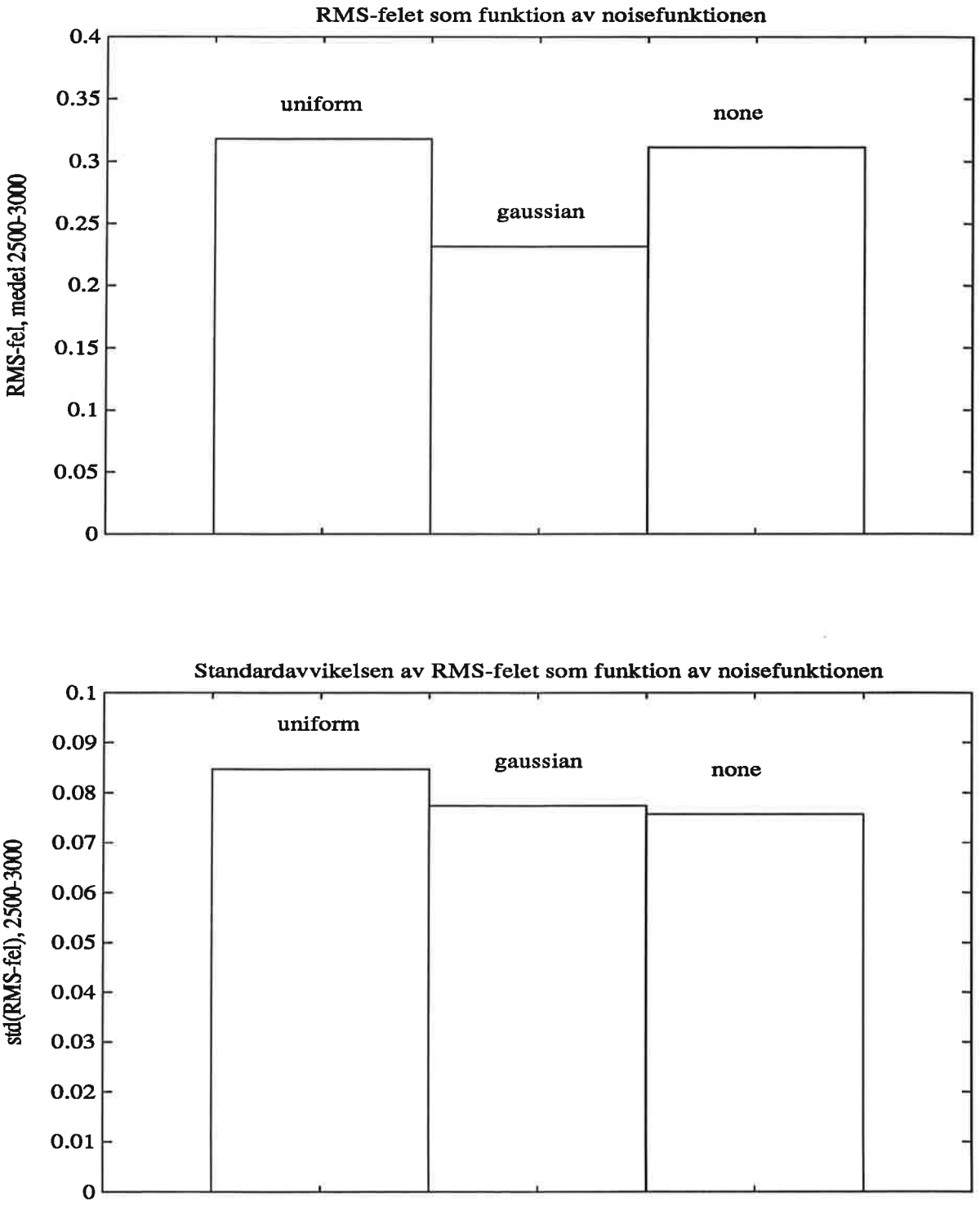


Diagram 14. RMS-felet samt dess standardavvikelse som funktion av brusfunktionen. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 13.

8.2.5.6 Inlärningsregel, learn rule

De flesta nätverk har sin egen speciella inlärningsregel. Till back-propagationnät brukar rekommenderas att använda delta-rule, cumulative-delta-rule eller norm-delta-rule. I näten 'linj1120' till 'linj1330' har olika inlärningsregler, learn rules, provats.
Parametrar utom inlärningsregeln, är fördefinierade. 13 noder i gömda lagret och 8 set av indata har använts.

I diagram 15 a), b) och c) ses hur RMS-felet avtar under inlärning. P g a stor spridning i RMS-felen, är skalan i y-led inte den samma i de olika bilderna. Medelvärdet av RMS-felet samt dess standardavvikelse i intervallet 2500 till 3000 inlärda exempel åskådliggörs i diagram 16 a) och b). Som förutspått ger de olika deltareglerna bästa inlärningen. Bsb Widrow-Hoff och bsb-Hebb, som också är varianter av delta-regeln, ger även de bra reultat. Dessa inlärningsregler är dock främst tänkta att användas ihop med nät av typen brain-state-in-a-box, bsb. Adaline ger ett högre fel men är å andra sidan mer stabil. Klart olämpliga är Hebb-, Kohonen-, art- och Boltzmannreglerna. Den sistnämnda blir snabbt statisk, efter ett kort insvängningsförlopp, och ser därför ut att vara stabil, utgående från den låga standardavvikelsen för RMS-felet.

De inlärningsfunktioner som är avsedda för andra nätverkstyper, bör inte användas i back-propagation. Istället rekommenderas de olika deltareglerna, inklusive bsb-varianterna.

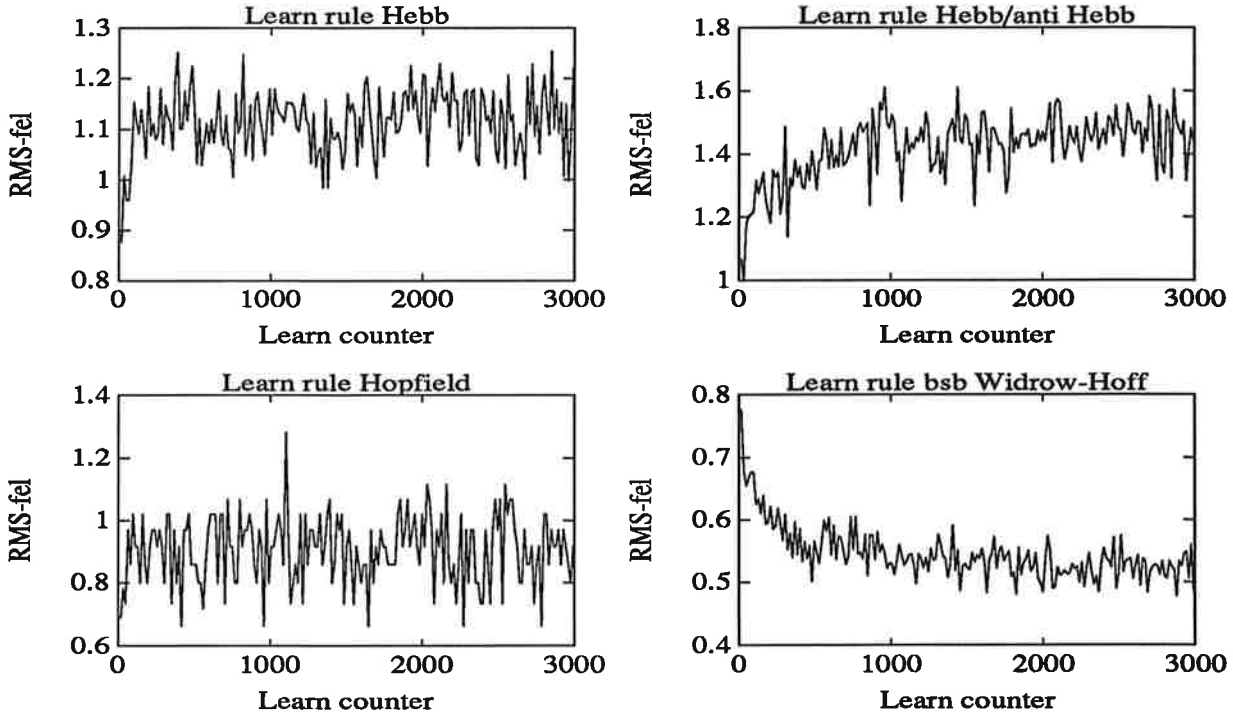


Diagram 15 a) RMS-felet som funktion av antal inlärda exempel för olika inlärningsregler, 13 noder i gömda lagret, 8 set av indata.

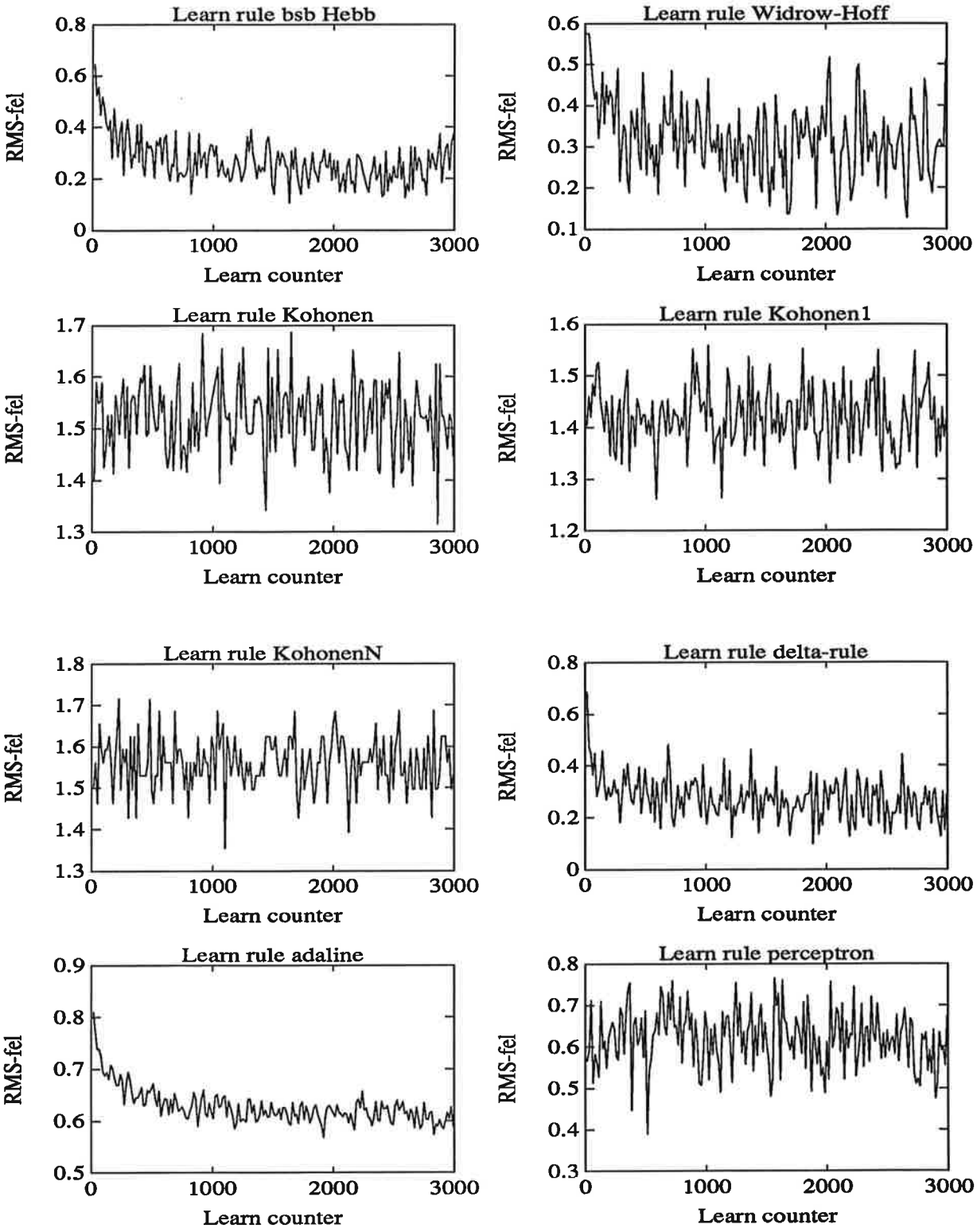


Diagram 15 b) RMS-felet som funktion av antal inlärda exempel för olika inlärningsregler, 13 noder i gömda lagret, 8 set av indata.

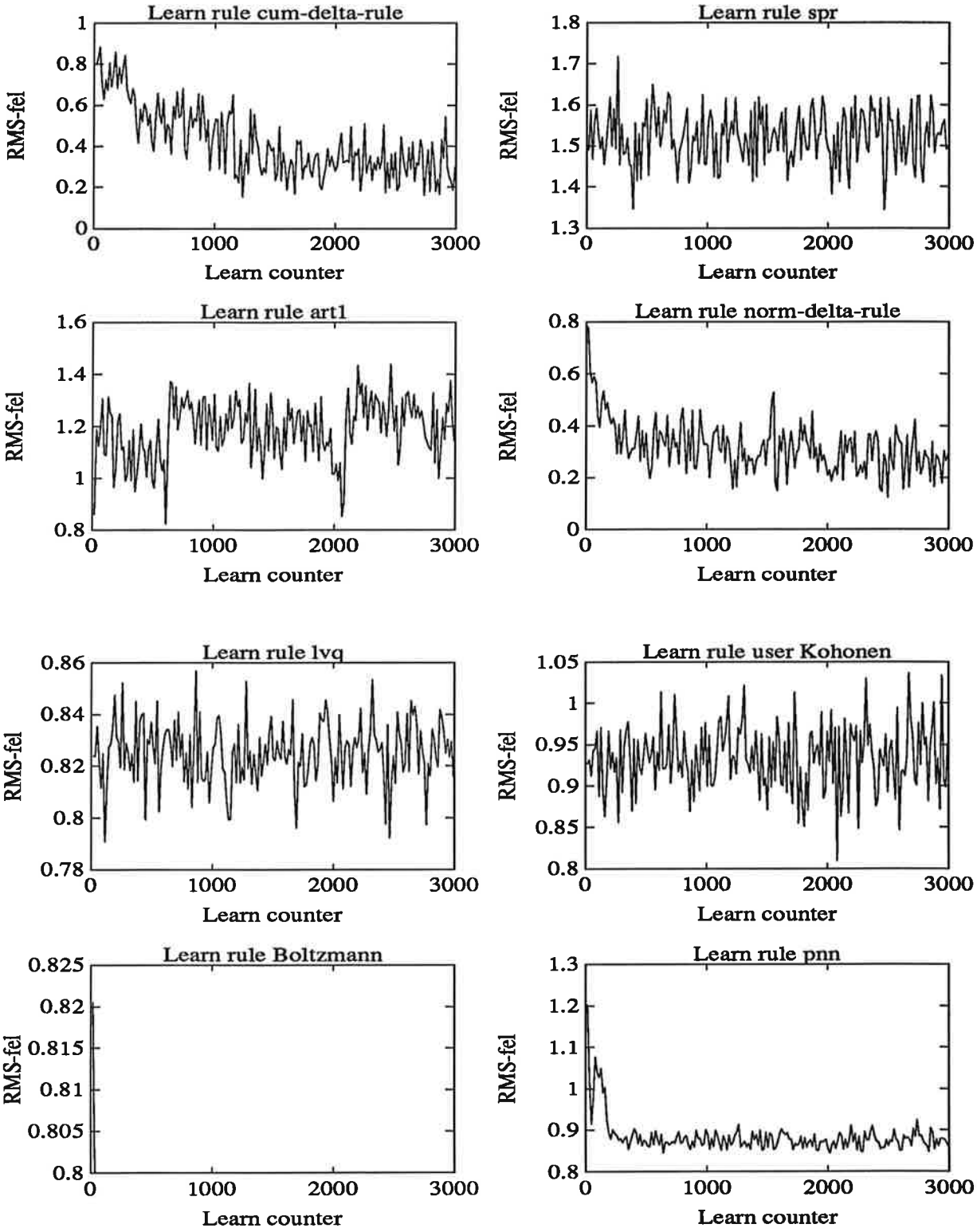
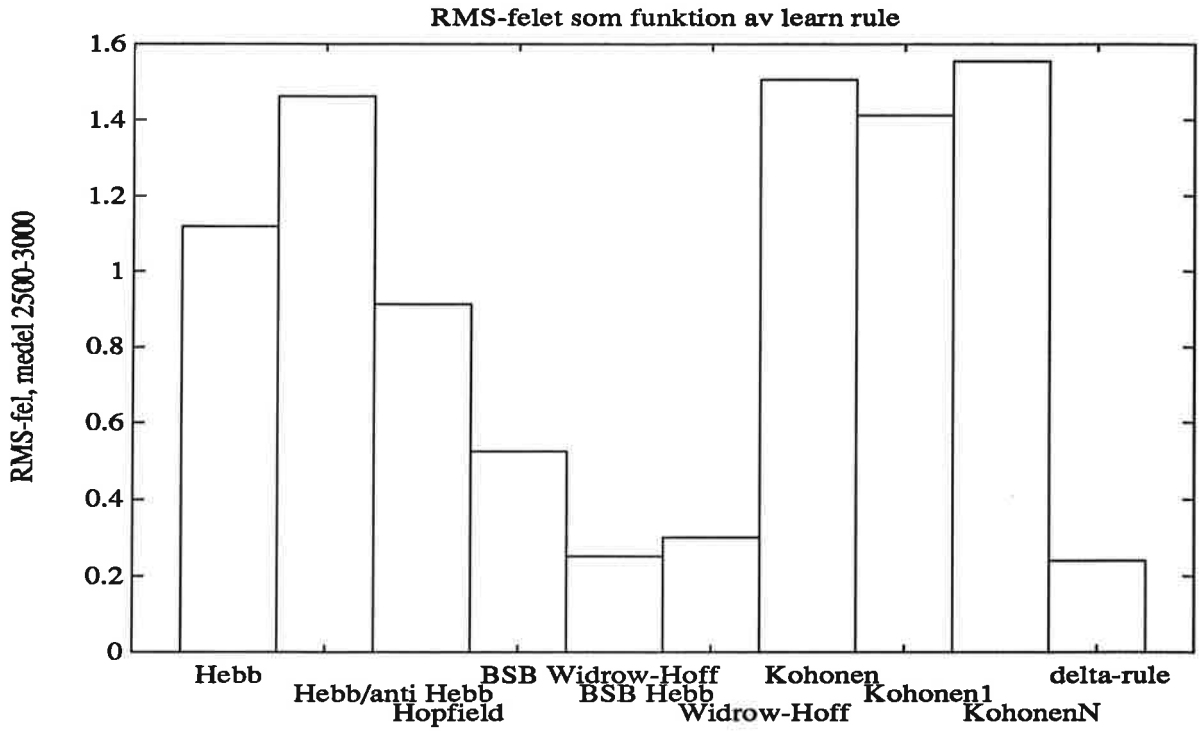


Diagram 15 c) RMS-felet som funktion av antal inlärda exempel för olika inlärningsregler, 13 noder i gömda lagret, 8 set av indata.



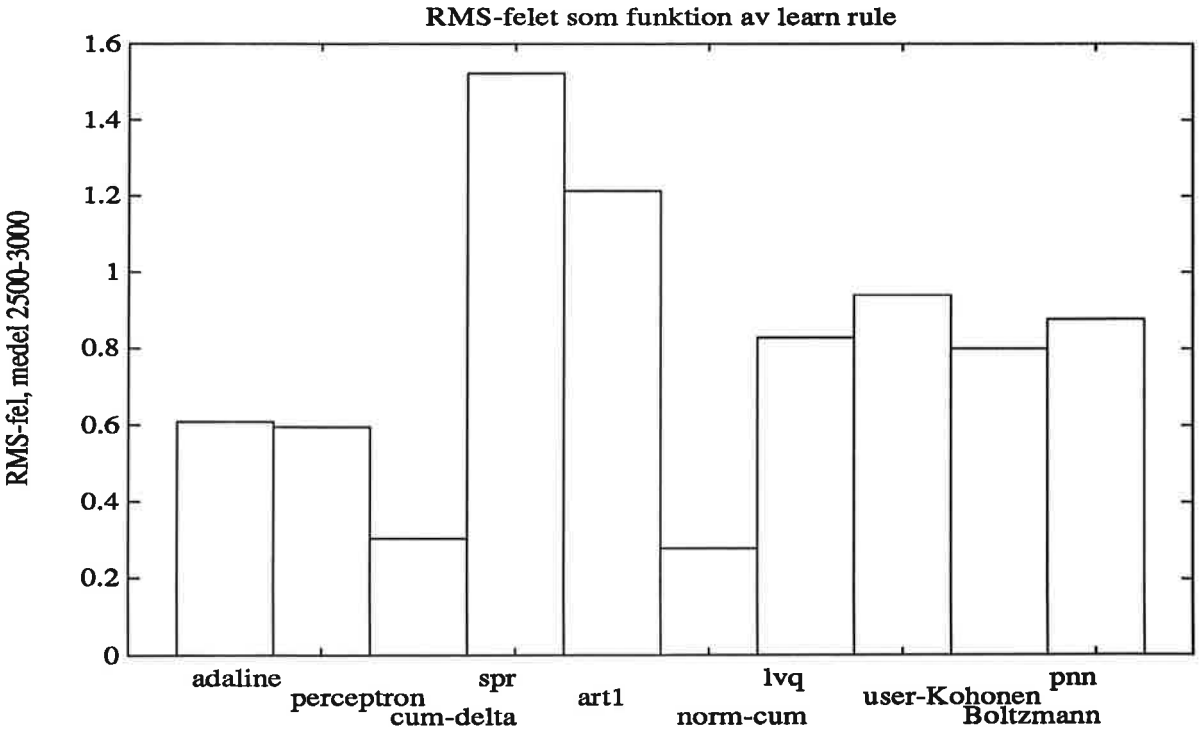
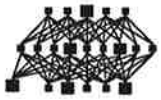
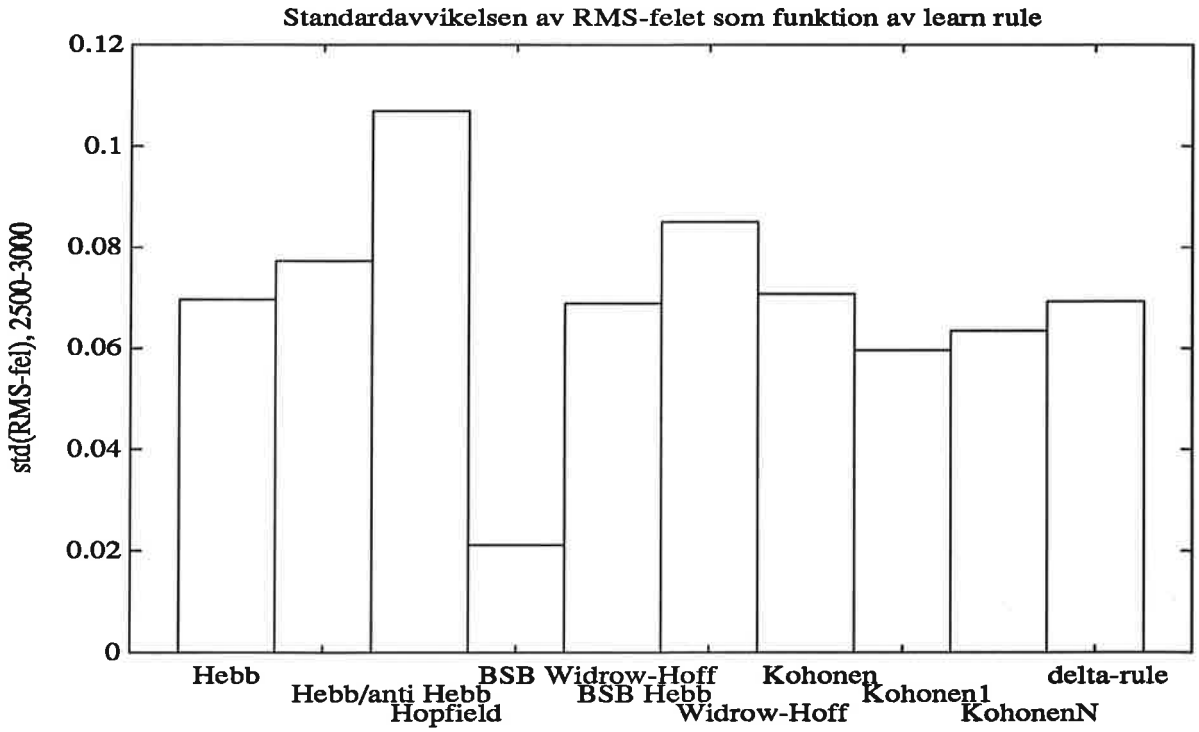


Diagram 16 a) RMS-felet som funktion av learn rule. Medelvärde mätt över de sista 500 exemplen enligt diagram 15.





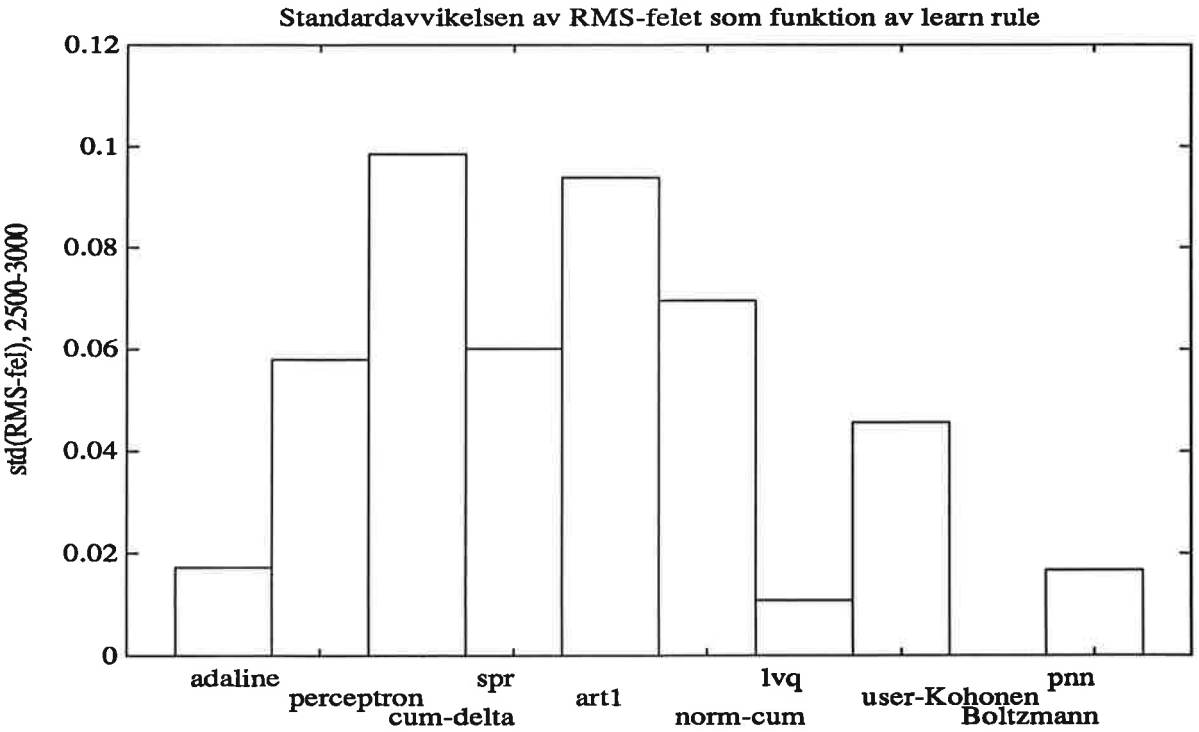


Diagram 16 b) standardavvikelsen av RMS-felet som funktion av learn rule. Standardavvikelsen beräknad på de sista 500 exemplen enligt diagram 15.

8.2.5.7 Utfunktion, output function

I back-propagationnät nyttjas för det mesta inga konkurrerande lager. Därför är utfunktionen 'direct' den mest använda. I klassificerande nätverk används däremot olika konkurrerande alternativ. I nätverken 'linje970' samt 'linj1030' till 'linj1110' har utfunktionen varierats medan övriga parametrar hållits konstanta, satta till sin fördefinierade värden. 13 noder i gömda lagret samt 8 set av indata.

RMS-felets avtagande under inlärning illustreras i diagram 17 a) och b), och dess medelvärde och standardavvikelse i diagram 18. Av de generella funktionerna ger 'direct' klart bättre inlärning än de där endast en eller två utnoder aktiveras. Bland de speciella funktionerna, avsedda för andra nätverkstyper, ger lvq och pnn bra resultat. Dessa båda är av typen 'vinnaren tar allt', dvs endast en nod åt gången ger utsignal.

Då det gömda lagret inte är ett Kohonenlager, bör utfunktionen direct användas. I back-propagation förekommer sällan Kohonenlager, varför direct är den funktion som bör användas.

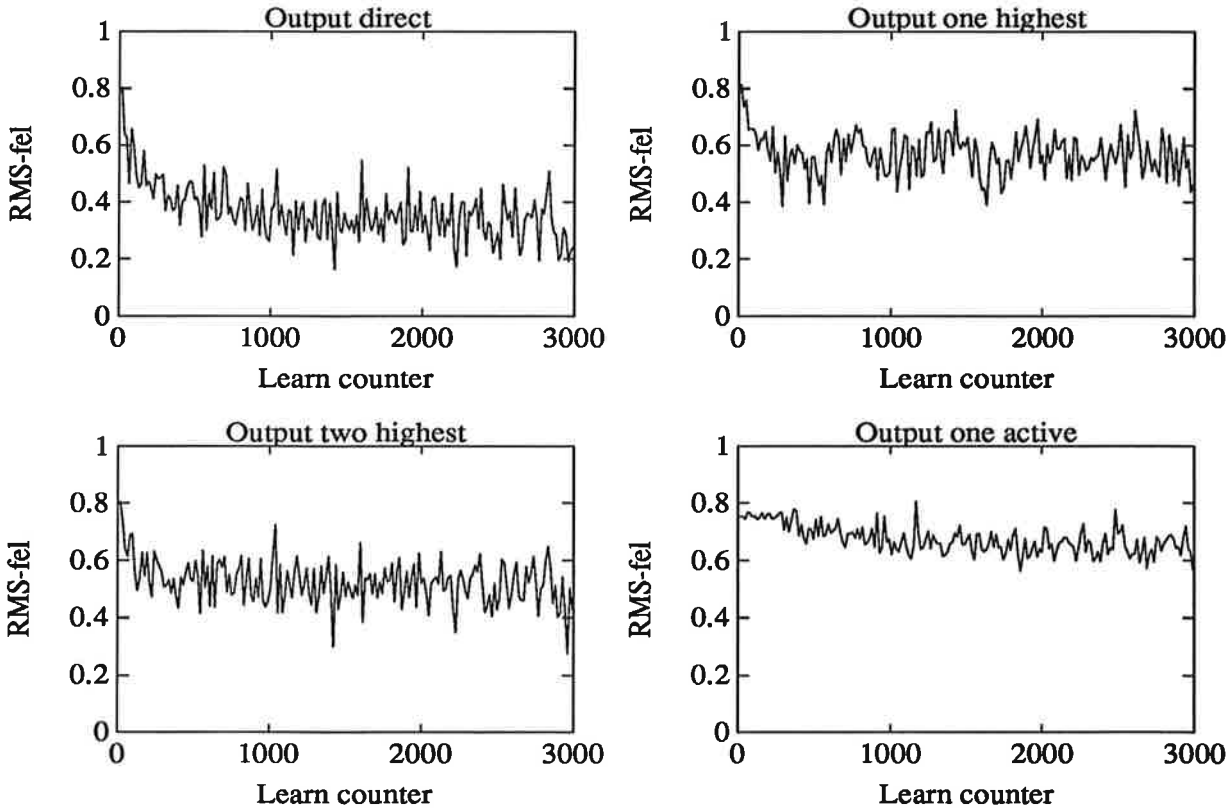


Diagram 17 a) RMS-felet som funktion av antal inlärda exempel för olika utfunktioner, 13 noder i gömda lagret, 8 set av indata.

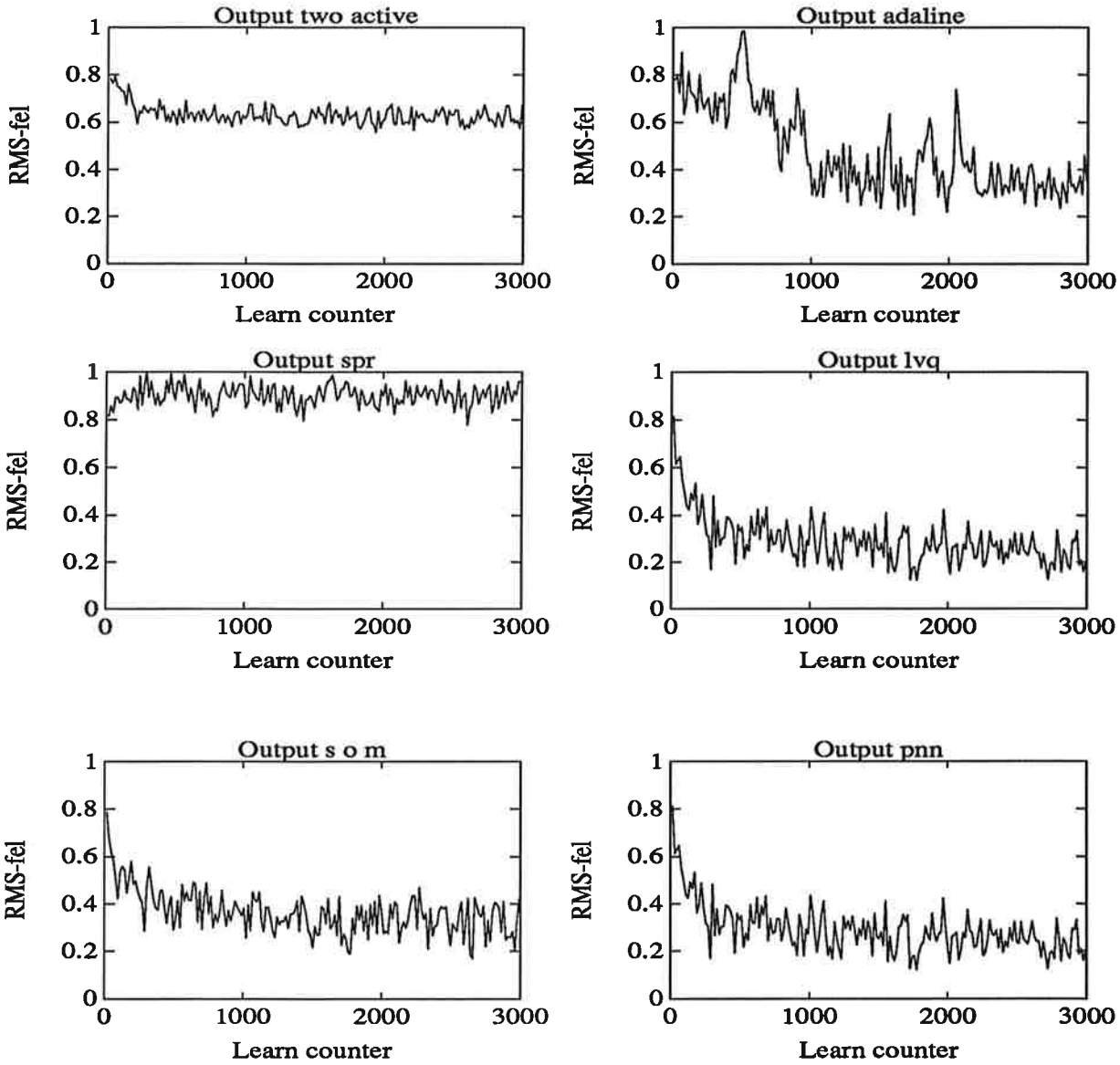


Diagram 17 b) RMS-felet som funktion av antal inlärda exempel för olika utfunktioner, 13 noder i gömda lagret, 8 set av indata.

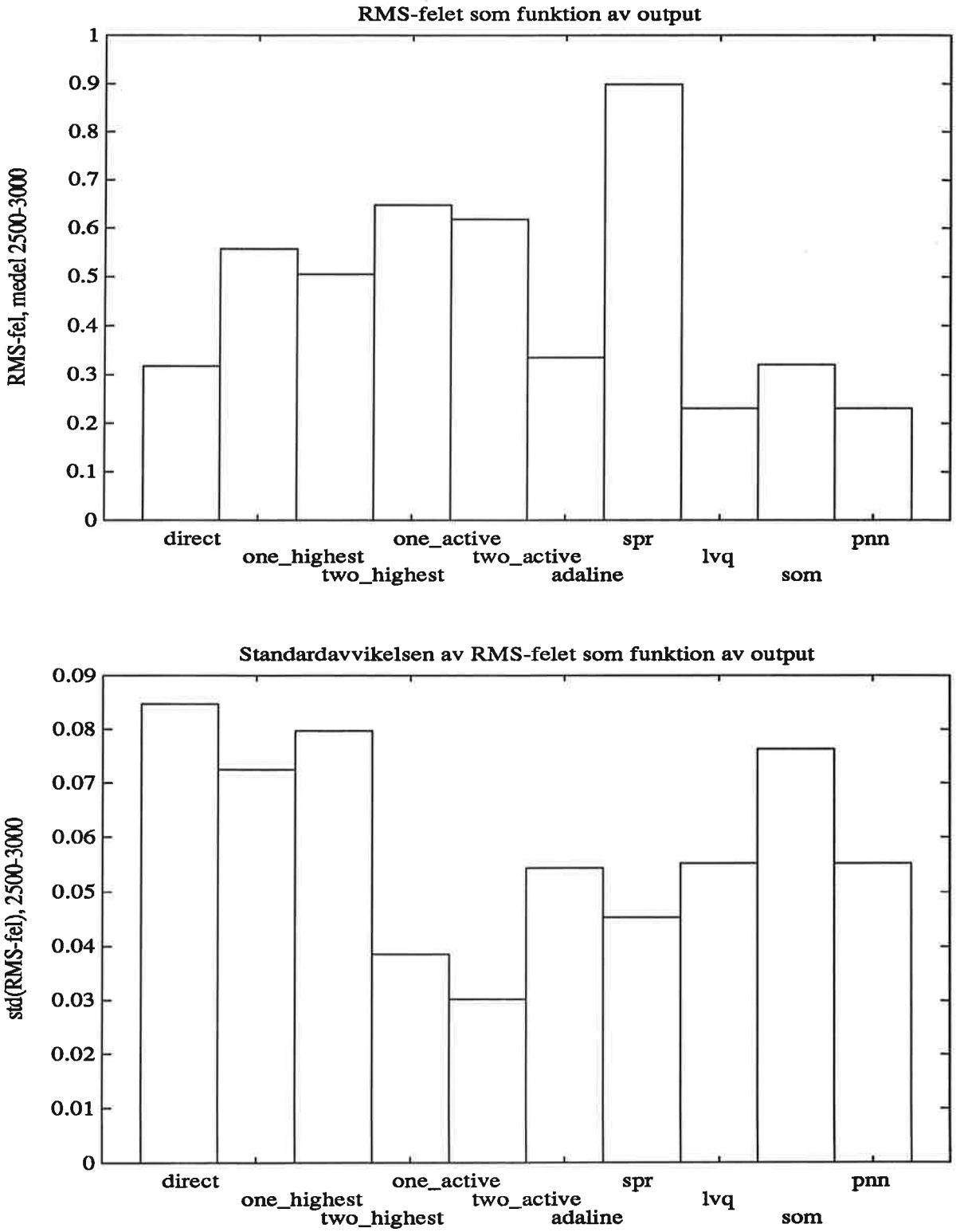


Diagram 18. RMS-felet samt dess standardavvikelse som funktion av utfunktionen. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 17.

8.2.6 Antal set av indata

Urvalet av indata spelar viss roll för hur bra ett nätverk kan generalisera. För få data gör att nätet lär sig att känna igen just de data på vilket det blivit upplärt, med risk för överinlärning. I näten 'linje550' till 'linje640' har 1 till 10 set av indata provats. Övriga parametrar satta till sina defaultvärden, och 13 gömda noder har använts.

Diagram 19 visar hur RMS-felet minskar under inlärningen. Eftersom avtagande ser ungefär likadant ut för de olika fallen, har inte alla tagits med. Medelvärdet och standardavvikelsen av RMS-felet ses i diagram 20, mätt över inlärningsexempel 2500 till 3000. Indatafilen för 10 set av indata blir av storleksordningen 1 megabyte. Större filer än så blir svårhanterliga och utrymmeskrävande. Två körningar har gjorts för vardera storleken på indata, men bara den senare är lagrad och dokumenterad. Den första gick förlorad p g a utrymmesbrist. Då för få testdata är tillgängliga, kan bara antaganden göras om vad resultaten verkligen innebär. RMS-felets storlek verkar öka samtidigt som inlärningen blir instabilare, med ökat antal set av indata. I båda körningarna gick dock felet ner vid 8 set av indata. Liksom i körningarna med varierat antal gömda noder, tycks även här vissa oförklarliga resonanser inträffa. Orsaken kan vara att antalet set ligger på gränsen, mellan det antal som gör att nätet memorerar, och det antal som ger generaliserande egenskaper, jämför (8.2.4). Längre körningar med fler antal set av indata, borde utvisa att detta ger en bättre inlärning, och inte tvärtom såsom här verkar vara fallet.

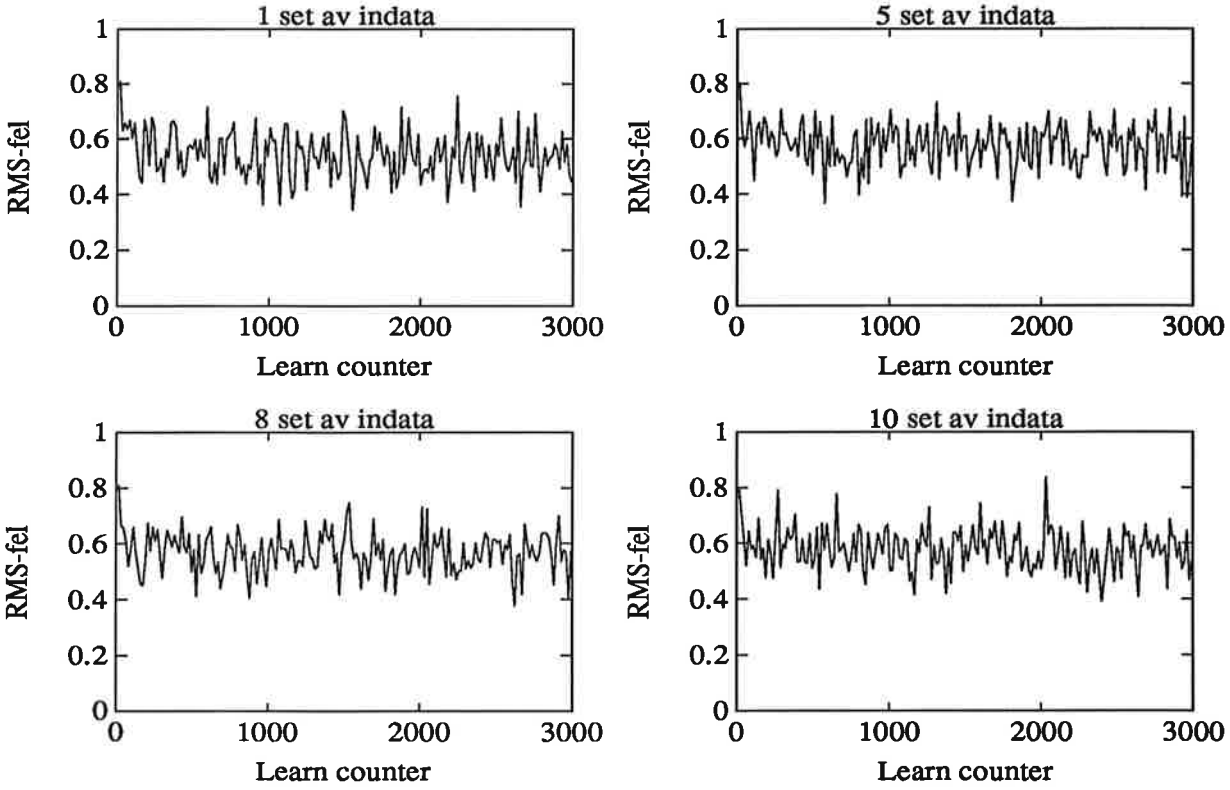


Diagram 19. RMS-felet som funktion av antal inlärda exempel för olika antal set av indata, 13 noder i gömda lagret, 8 set av indata.

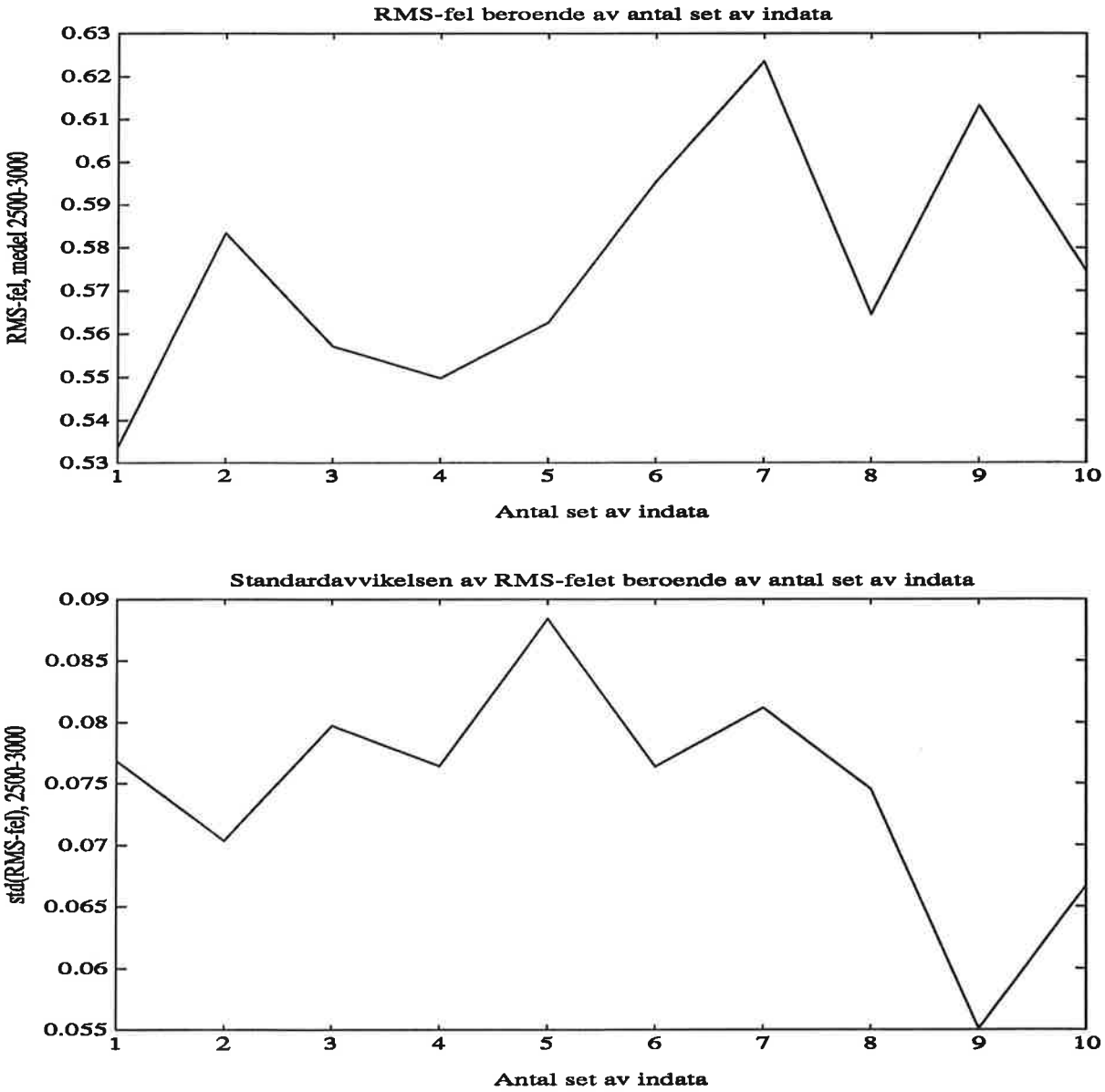


Diagram 20. RMS-felet samt dess standardavvikelse som funktion av antal set av indata. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 19.

8.2.7 Antal förbindelser mellan gömda lagret och utlagret, komplexitet

8.2.7.1 Allmänt

Komplexiteten i ett nätverk beror av antalet förbindelser i detta. Eftersom antalet innoder är betydligt större än antalet utnoder, i denna tillämpning, blir komplexiteten störst mellan inlager och mellanlager. Mellan dessa två lager blir det mer väsentligt att begränsa antalet förbindelser. Förbindelserna mellan gömda lagret och utlagret är inte lika många, varför samtliga kopplas upp på vanligt sätt.

Då back-propagationnät genereras i NeuralWorks är standardmetoden att förbinda samtliga noder i två närliggande lager med varandra. Som alternativ kan man antingen låta varje nod förbindas med en viss andel av alla de noder, som är tänkbara i det närliggande lagret, eller låta varje tänkbar förbindelse ha en viss sannolikhet för att existera. I det förra fallet vet man precis hur många förbindelser varje nod har med noder i det andra lagret, i det senare fallet vet man bara statistiskt sett, hur många förbindelser som verkligen existerar mellan en nod och noderna i det andra lagret. Fördelningen av förbindelserna blir jämnare i det första fallet än i det andra.

8.2.7.2 Förbindelser, andel uppkopplade

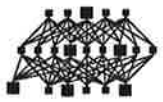
I näten 'linj1340' till 'linj1513' har antalet förbindelser, mellan inlagret och det gömda lagret, hållits konstant vid 250, 500 respektive 750. Andelen existerande förbindelser mellan dessa lager, av det totala antalet möjliga, samt antalet noder i det gömda lagret har varierats. Övriga parametrar enligt default samt 8 set av indata.

Antal noder	Totalt förbindelser	% vid 250	% vid 500	% vid 750
30	30*35 = 1050	23.81	47.62	71.43
40	40*35 = 1400	17.86	35.71	53.57
50	50*35 = 1750	14.29	28.57	42.86
60	60*35 = 2100	11.90	23.81	35.71
70	70*35 = 2450	10.20	20.41	30.61
80	80*35 = 2800	8.93	17.86	26.79
90	90*35 = 3150	7.94	15.87	23.81
100	100*35 = 3500	7.14	14.29	21.43

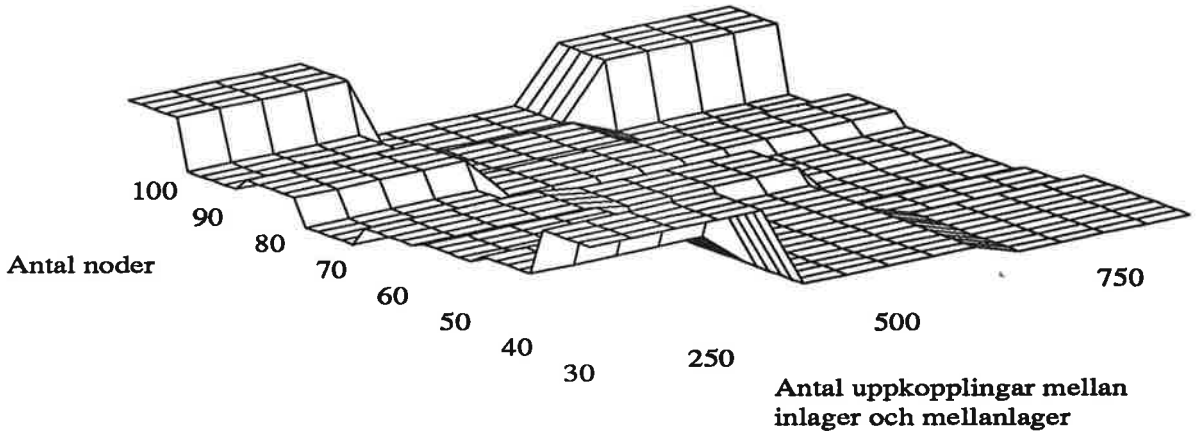
Antal noder står för antalet noder i gömda lagret. *Totalt förbindelser* står för det totala antalet förbindelser mellan inlager och mellanlager. *% vid 250, 500 resp. 750* står för hur stor andel, av det totala antalet förbindelser, som är uppkopplade vid 250, 500, resp. 750 existerande förbindelser.

I diagram 21 a) och b) åskådliggörs, på olika sätt, hur RMS-felet och dess standardavvikelse, mätt över inlärningsexempel 2500 till 3000, varierar med de två parametrarna antal gömda noder och antal förbindelser mellan inlagret och det gömda lagret. Diagram 21 a) och b) redovisar samma data, men presenterat på olika sätt.

Med ändrad komplexitet, dvs olika antal förbindelser mellan dessa två lager, fås felminima för olika antal noder i det gömda lagret. För 250 förbindelser fås minimum vid 70 gömda noder. Vid 500 fås bästa resultat någonstans i intervallet 60 till 100 gömda noder, och för 750 förbindelser i intervallet 50 till 90 gömda noder.



RMS-felet som funktion av antalet noder i mellanlagret och antalet uppkopplingar mellan detta lager och inlagret



Standardavvikelsen av RMS-felet som funktion av antalet noder i mellanlagret och antal uppkopplingar mellan detta lager och inlagret

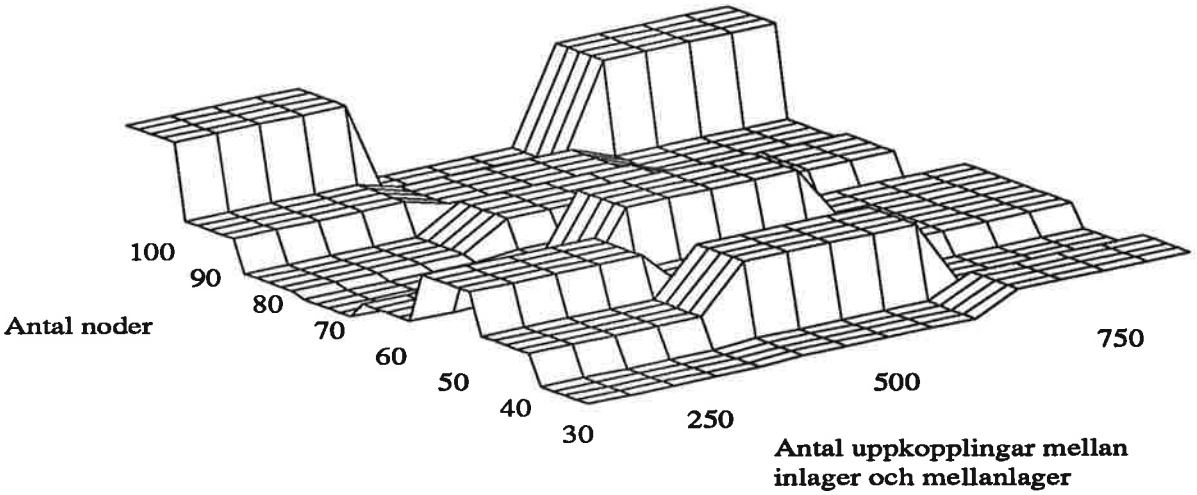
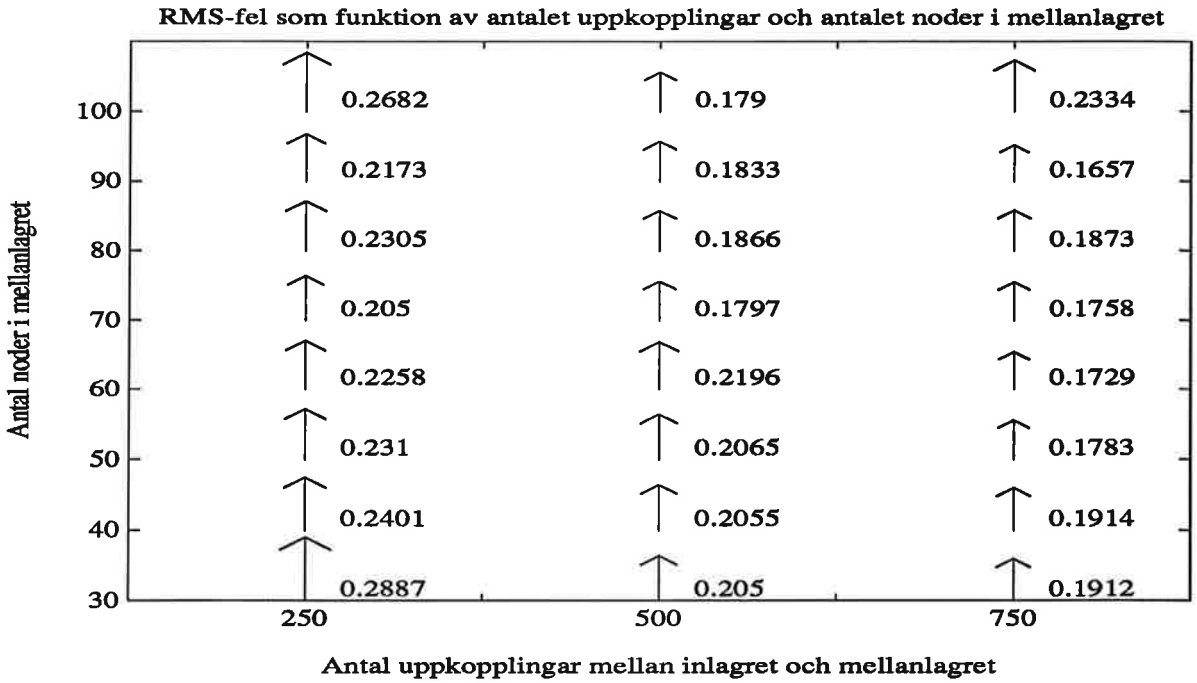


Diagram 21 a) RMS-felet samt dess standardavvikelse som funktion, dels av antal noder i det gömda lagret, dels av antalet uppkopplade förbindelser mellan inlagret och det gömda lagret. Medelvärde och standardavvikelse mätt över inlärningsexempel 2500 till 3000.



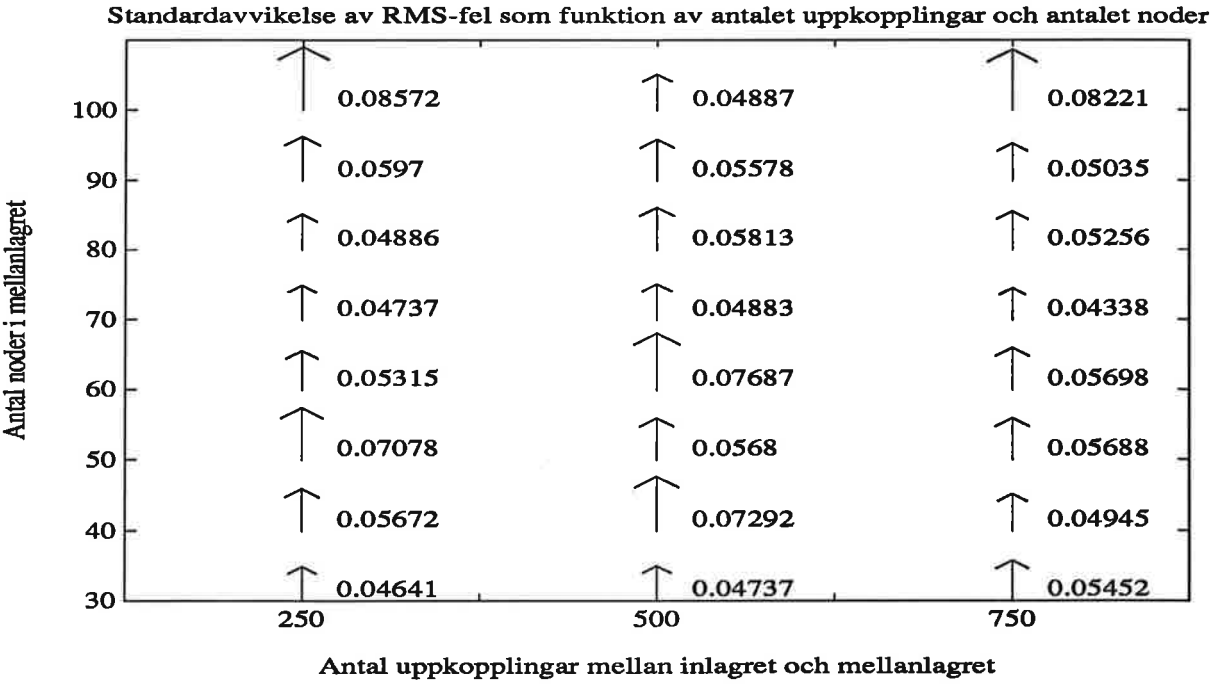


Diagram 21 b)RMS-felet samt dess standardavvikelse som funktion, dels av antal noder i det gömda lagret, dels av antalet uppkopplade förbindelser mellan inlagret och det gömda lagret. Medelvärde och standardavvikelse mätt över inlärningsexempel 2500 till 3000. Samma data som i diagram 21 a).

8.2.7.3 Förbindelser, sannolikhet för uppkoppling

Nätverken 'linj1520' till 'linj1560' representerar körningar där sannolikheten, för att respektive förbindelse mellan inlagret och det gömda lagret existerar, har varierats. 50 noder i det gömda lagret och 8 set av indata användes, övriga parametrar satta till sina defaultvärden. Sannolikheterna har valts till sådana värden att antalet förbindelser blir detsamma som i föregående körning, (8.2.7.2).

I diagram 22 har en jämförelse av hur RMS-felet och dess standardavvikelse, beter sig för olika komplexiteter, i de båda möjliga fallen av uppkopplade förbindelser.

Ingen större skillnad kan iakttagas beroende på vilket sätt noderna förbundits, uppkopplats, mellan de olika lagren.

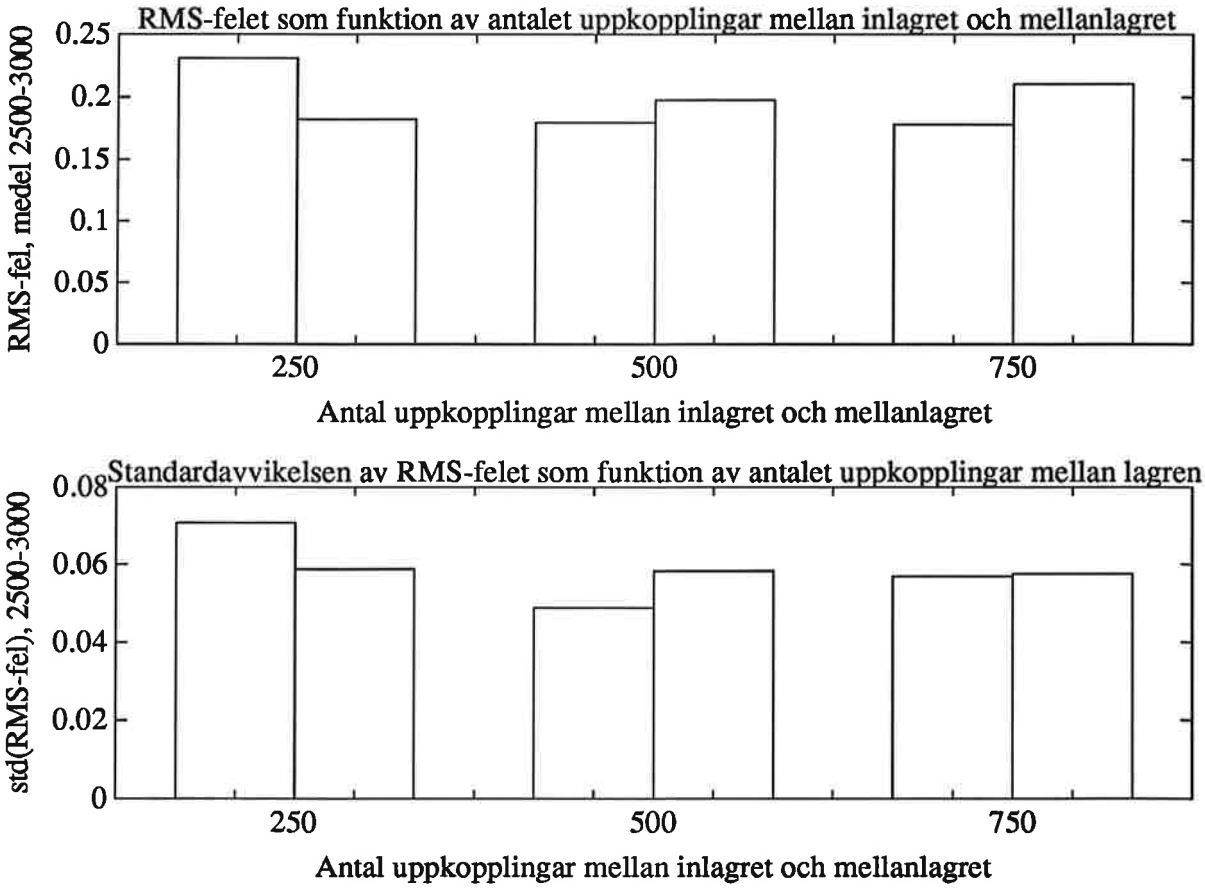
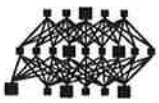


Diagram 22. RMS-felet samt dess standardavvikelse som funktion av antalet uppkopplade förbindelser mellan inlagret och det gömda lagret. För varje värde har två staplar ritats, den vänstra motsvarande förbindelser uppkopplade till 'en viss andel', den högra stapeln motsvarande uppkoppling grundad på 'lika sannolikhet för alla förbindelser'. Gömda lagret innehåller 50 noder. Medelvärde och standardavvikelse mätt över inlärningsexempel 2500 till 3000.



8.2.8 Epochstorlek

Epochstorleken har betydelse främst då de olika deltareglerna används som inlärningsfunktion. Då vikterna uppdateras, lagras ändringarna i deltavikter, där information om åt vilket håll ändringarna skedde, sparas. Om epochstorleken sätts till ett, kommer nätet inte att minnas hur sista uppdateringen gick till. De kommande uppdateringarna blir i så fall helt oberoende av varandra. Näten 'linje650' till 'linje710' har epochstorleken satt till värden mellan 8 och 32, i steg om 4. Övriga parametrar fördefinierade, 13 noder i gömda lagret och 8 set av indata.

Diagram 24 visar RMS-felet under inlärningen. Att RMS-felet svänger upp och ner med olika frekvens, beror på att instrumentet som loggar RMS-värdena, sparar dessa data i jämna multipler av epochstorleken. Skillnaden i svängningarnas frekvens är alltså helt oväsentlig. Som en följd härav blir standardavvikelsen av RMS-felet inget bra jämförelsemått på stabiliteten mellan de olika körningarna. Svängningarnas amplitud är däremot intressantare. Större epochstorlek leder till mindre amplituder i svängningarna. Detta kan förklaras genom att felet ackumuleras över samtliga exempel som ingår i en epoch. Då flera värden adderas, minskar den statistiska spridningen, och inlärningskurvan kommer att se jämnare ut. Medelvärdet av RMS-felet för de olika epochstorlekarna, räknat över inlärningsexempel 2500 till 3000, visas i diagram 23.

Få mätvärden gör det svårt att tolka resultatet, men lämpligt värde på epochstorleken kan vara 24.

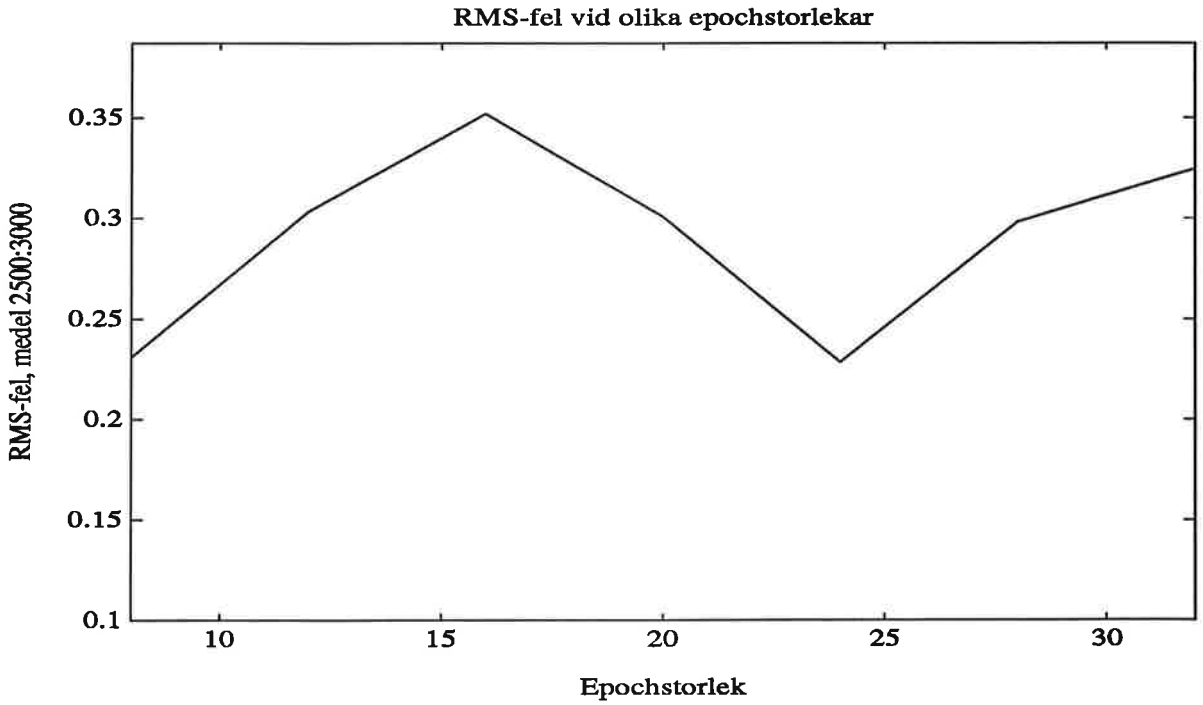


Diagram 23. RMS-felet som funktion av epochstorleken. Medelvärde mätt över de sista 500 exemplen enligt diagram 21.

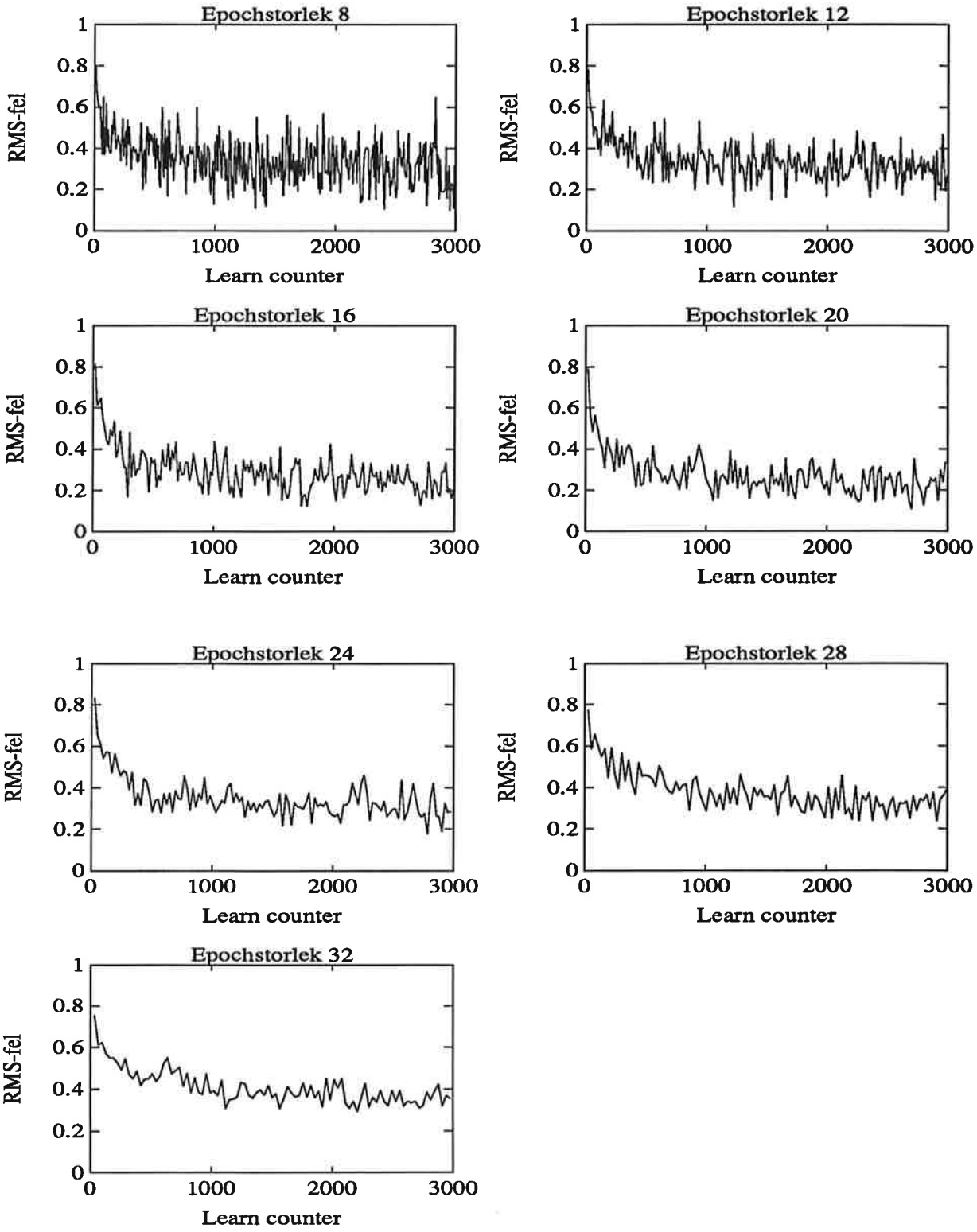


Diagram 24. RMS-felet som funktion av antal inlärda exempel för olika antal set av indata, 13 noder i gömda lagret, 8 set av indata.

8.3 DIRECT-RANDOM-SEARCH NÄTVERK, DRS

8.3.1 Allmänt

Drs bygger på helt slumpmässiga ändringar av vikternas värden. Om ändringarna av vikternas värden ger ett mindre fel, mätt över alla inlärningsexempel, i utlagrets noder behåller vikterna dessa värden, som då ger det hittills bästa resultatet. Blir felet större prövas en ändring med motsatt tecken. Ger denna ett bättre resultat sparas detta på motsvarande sätt som det hittills bästa. Eftersom felet mäts över alla exempel i indatafilen skall antalet inlärda exempel sättas till en jämn multipel av antalet data i indatafilen.

Jämfört med andra inlärningsregler, där t ex derivator beräknas, krävs här inga beräkningar förutom av felen. Detta gör att inlärningen går relativt fort.

Man riskerar heller inte att fastna i lokala minima då felen minimeras, ett problem som annars kräver sin lösning. Om indata består av vektorer X_i med n element kan dessa sägas spänna upp den n-dimensionella rymden, $X_i \in R^n$. Vidare är konvergensen under inlärning alltid garanterad, så länge indata är ett kompakt sammanhängande område i indatarymden, R^n.

För drs nätverk har olika antal noder i det gömda lagret testats, samt skillnaden då sigmoid eller tangenshyperbolicusfunktionen använts som överföringsfunktion i utlagret.

8.3.2 Fördefinierade värden, default, direct-random-search nätverk

*) Defaultvärde som ändrats explicit under körningarna, se vidare i filförteckningen.

Nätverket

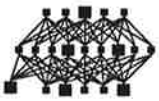
Kontrollstrategi	drs
Inlärningsschema	drsprop *)

Inlagret

Summationsfunktion	Sum
Transferfunktion	Linjär
Felfunktion	Standard
Utfunktion	Direkt
Inlärningsregel	none
Inlärningsschema	Net global

Gömda lagret

Summationsfunktion	Sum
Transferfunktion	Sigmoid
Felfunktion	Standard
Utfunktion	Direkt
Inlärningsregel	Drs-delta
Inlärningsschema	Net global



Utlagret

Summationsfunktion	Sum
Transferfunktion	Sigmoid *)
Felfunktion	drserr
Utfunktion	Direkt
Inlärningsregel	Drs-delta
Inlärningsschema	Net global

8.3.3 Antal noder i gömda lagret

För back-propagationnät anges i [19] en formel för antal noder i det gömda lagret, men för drs finns ingen motsvarande. I nätverken 'linj1570' till 'linj1690' har antalet noder i gömda lagret varierats, medan övriga parametrar satts till fördefinierade värden. Indatafil bestående av 8 set av indata har använts.

I diagram 25 a) och b) visas hur RMS-felet avtar under inlärningen. Beroende på inlärningsregeln blir inlärningen jämnare än för back-propagation, och eftersom varje inlärningsexempel går betydligt fortare att lära in, kan ett större antal exempel gås igenom, här 115 200 stycken, vilket innebär att hela indatafilen använts 100 gånger. Till skillnad från back-propagation, där indata väljs i slumpmässig ordning, läser drs alla indata i samma ordning som de lagrats i filen. I diagram 26 redovisas RMS-felet och dess standardavvikelse, mätt över inlärningsexmpel 90 000 till 115 200, beroende av antalet gömda noder. Tendensen är inte lika klar som i fallet för back-propagation, se (8.2.4), men 8 noder ger både lågt fel och stabil inlärning. I diagram 25 b) ser man emellertid att 7 noder ger lägst fel i slutet av inlärningssekvensen, varför detta antal bör väljas.

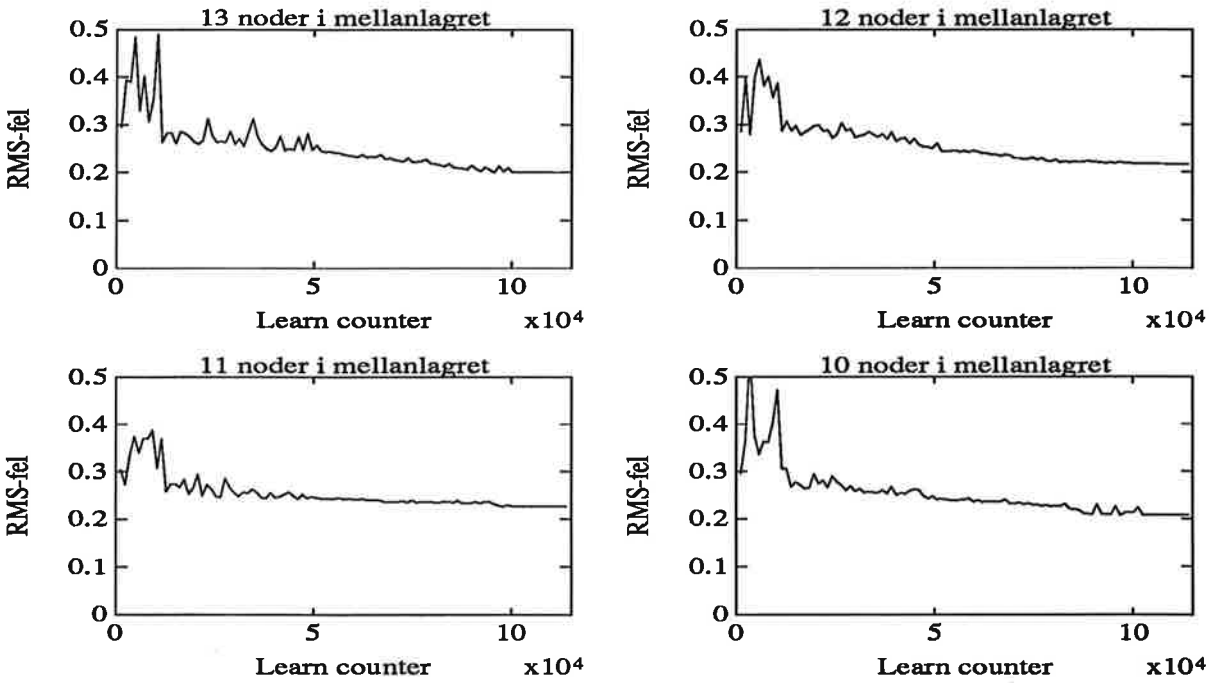


Diagram 25 a)RMS-felet som funktion av antal inlärda exempel för olika antal noder i det gömda lagret, 13 noder i gömda lagret, 8 set av indata.

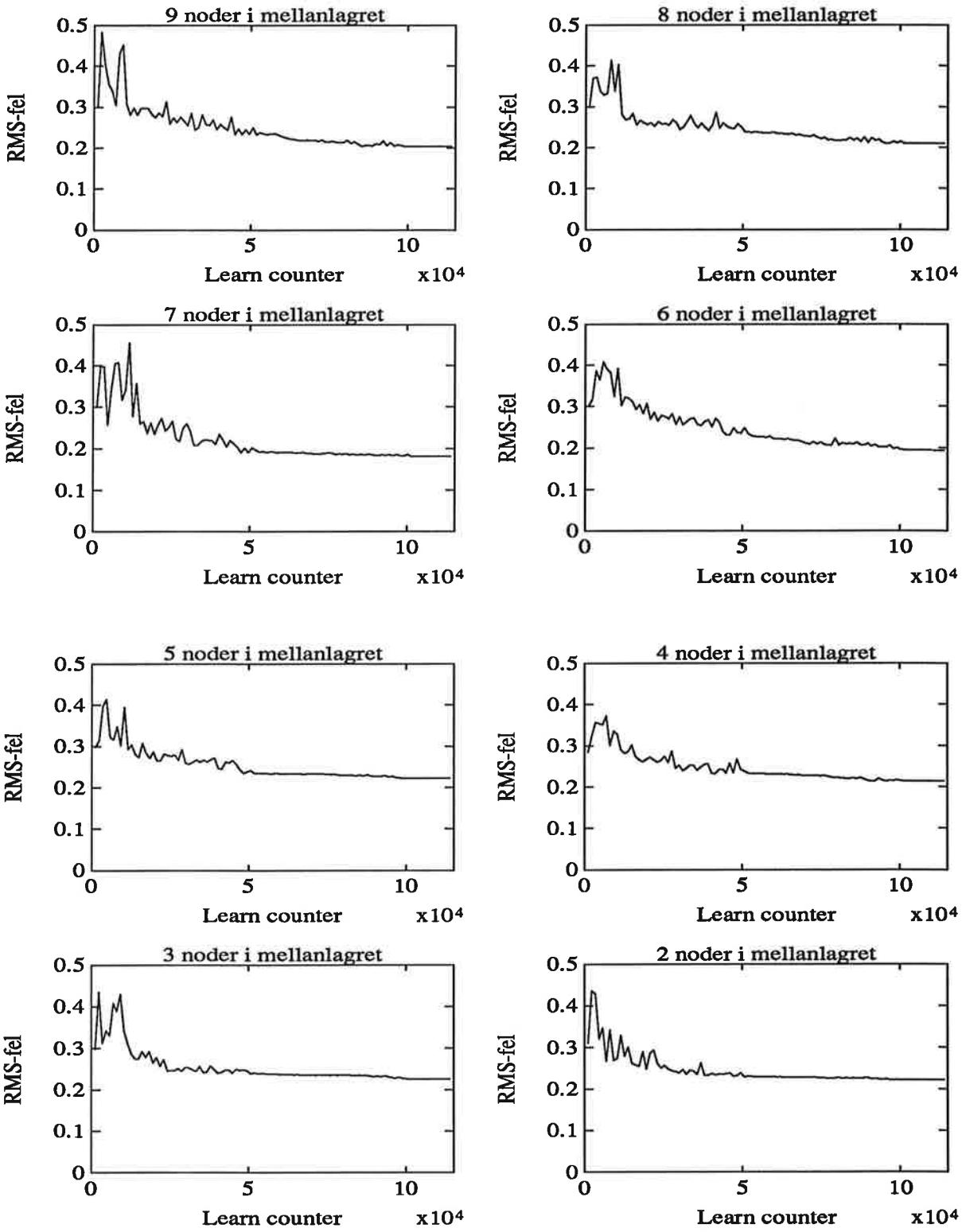


Diagram 25 b) RMS-felet som funktion av antal inlärda exempel för olika antal noder i det gömda lagret, 13 noder i gömda lagret, 8 set av indata.

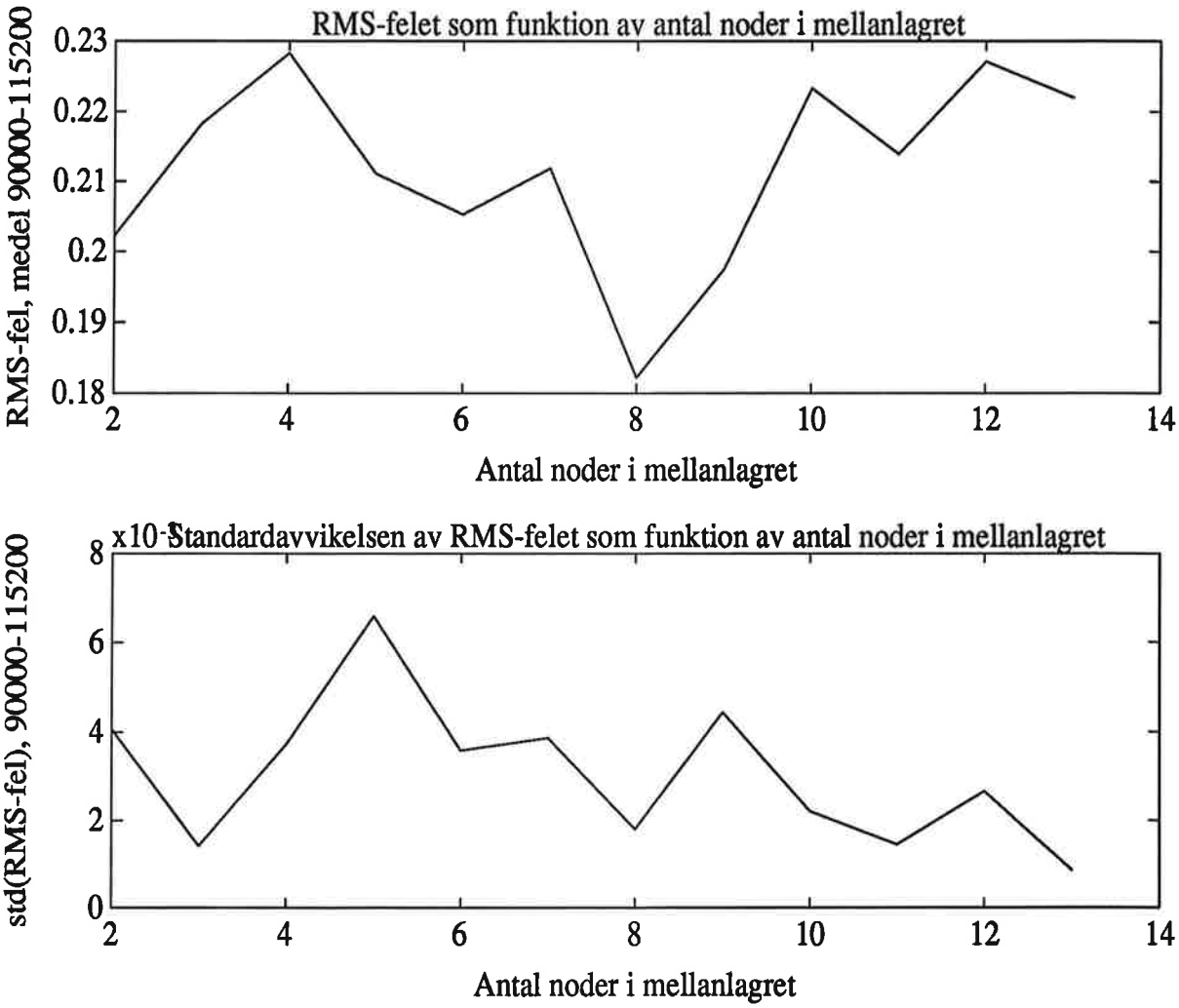


Diagram 26. RMS-felet samt dess standardavvikelse som funktion av antal noder i det gömda lagret. Medelvärde och standardavvikelse mätt över de sista 500 exemplen enligt diagram 25.

8.3.4 Överföringsfunktion i utlagret

Näten 'linj1570' och 'linj1580' har olika överföringsfunktion, sigmoid respektive tangenshyperbolicus. Övriga parametrar är fördefinierade. 13 noder i gömda lagret och 8 set av indata.

Diagram 27 visar dels hur RMS-felet avtar i de två fallen, dels dess medelvärde och standardavvikelse, mätt över exemplen 90 000 till 115 200.

Sigmoidfunktionen ger både lägre fel och stabilare inlärning, till skillnad från back-propagationnät, där tanh gav bäst resultat. För drs nätverk rekommenderas därför att använda överföringsfunktionen sigmoid.

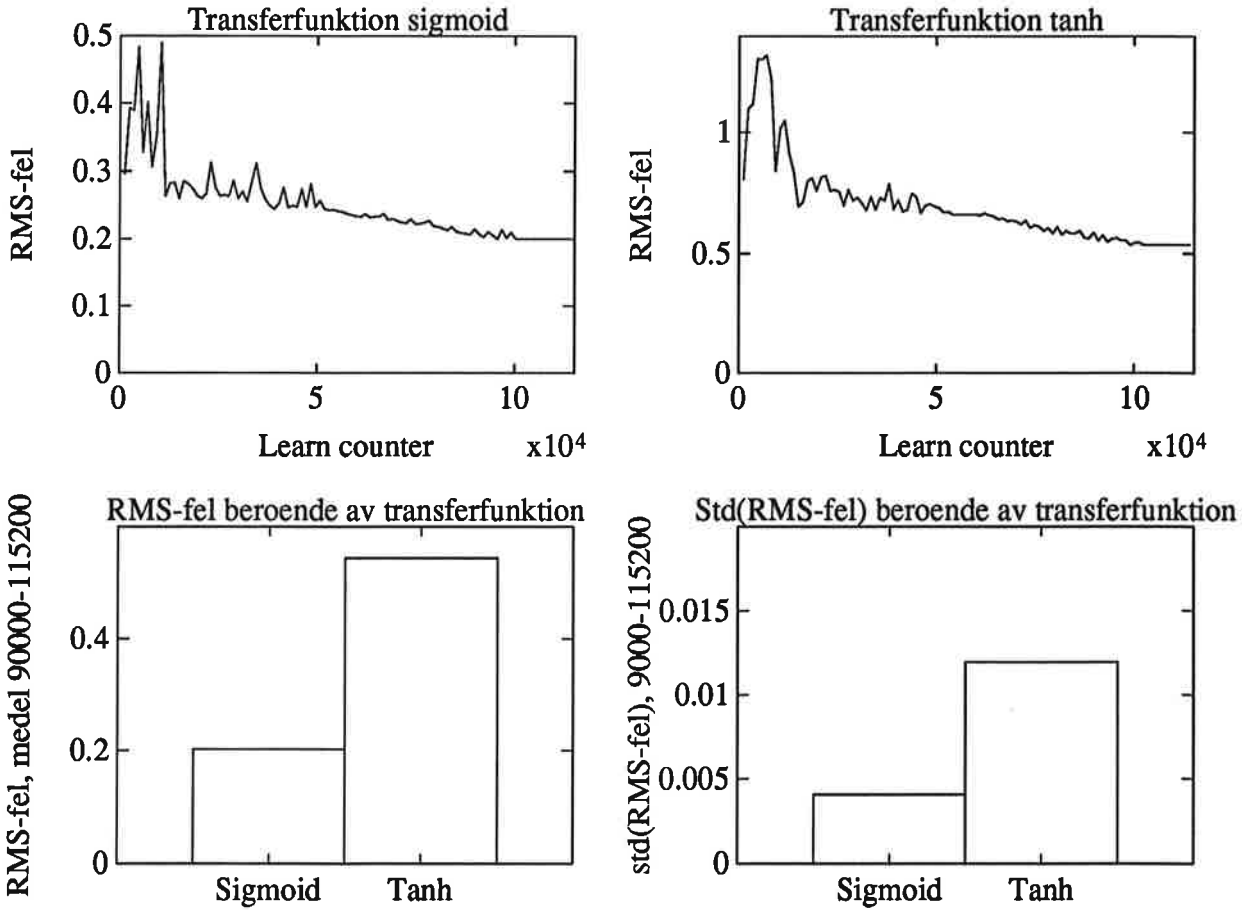


Diagram 27. RMS-felet, som funktion av antal inlärda exempel, för olika överföringsfunktioner i det gömda lagret, samt dess medelvärde och standardavvikelse, mätt över inlärningsexemplen 90 000 till 115 200. 13 noder i gömda lagret, 8 set av indata, 7 x 5 pixels.

8.4 PROBABILISTIC-NEURAL-NETWORKS, PNN

8.4.1 Allmänt

Pnn är ett klassificerande nätverk, som bygger på sannolikheter, för att givna indata skall tillhöra en viss kategori. Det gömda lagret är av Kohonentyp, och antalet noder i detta lager bör motsvara antalet data i indatafilen. Inlärningen sker i två steg. Först lärs Kohonenlagret upp, under ett visst antal inlärningsexempel. Detta tvådimensionella lager organiserar indata, så att data med liknande egenskaper, aktiverar noder i samma delområde av detta lager. Under denna fas är utlagrets noder passiva, dvs inlärningen sker utan styrning mot givna utdata. Därefter lärs utlagrets noder upp. Kohonenlagrets noder är nu redan uppträndade. I utlagret väljs vilken nod, som skall aktiveras, utgående från sannolikheten att en viss egenskap, tillhör en given kategori. Pnn väljer den mest sannolika kategorin för varje invektor. I denna fas styrs upplärningen, mot de utsignaler som är en del av indata.

P g a strukturen hos indata i denna tillämpning, skulle antalet noder i detta lager blivit alltför stort, om varje indata motsvarade en nod i Kohonenlagret. Då färre noder än det rekommenderade används, blir organiseringen av indata inte fullständig. Genom att ända köra nätverket för ett mindre antal gömda noder, fås en möjlighet att studera nätets funktion, även om denna inte blir helt optimerad.

8.4.2 Fördefinierade värden, default, probabilistic-neural-networks, pnn

Nätverket

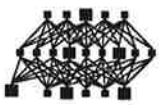
Nätverkstyp	Hetero-Associativt	
Kontrollstrategi	pnn	

Inlagret

Summationsfunktion	Sum	
Transferfunktion	Linjär	
Skalning	Skala	1.0
	Offset	0.0
Gränser	Undre gräns	-9999
	Övre gräns	+9999
Felfunktion	Standard	
Utfunktion	Direkt	
Inlärningsregel	Default	

Normaliserande lager

Summationsfunktion	NormPolar	
Transferfunktion	Linjär	
Skalning	Skala	1.0
	Offset	0.0
Gränser	Undre gräns	-9999
	Övre gräns	+9999
Felfunktion	Standard	
Utfunktion	Direkt	
Inlärningsregel	Ingen	
Inlärningsschema	Default	



Mönsterlager

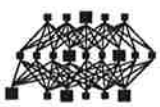
Summationsfunktion	Sum	
Transferfunktion	Pnn	
Skalning	Skala	1.0
	Offset	0.0
Gränser	Undre gräns	-9999
	Övre gräns	+9999
Felfunktion	Standard	
Utfunktion	Pnn	
Inlärningsregel	User Kohonen	
Inlärningsschema	Pnn	
Initiering	undre gräns	-0.0
	övre gräns	+0.0

Summationslager

Summationsfunktion	Sum	
Transferfunktion	Linjär	
Skalning	Skala	1.0
	Offset	0.0
Gränser	Undre gräns	-9999
	Övre gräns	+9999
Felfunktion	Standard	
Utfunktion	Direkt	
Inlärningsregel	Pnn	
Inlärningsschema	Default	
Initiering	undre gräns	-1.0
	övre gräns	+1.0

Klassificerande lager, (utlager)

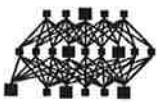
Summationsfunktion	Sum	
Transferfunktion	Linjär	
Skalning	Skala	1.0
	Offset	0.0
Gränser	Undre gräns	-9999
	Övre gräns	+9999
Felfunktion	Standard	
Utfunktion	One-Active-Highest	
Inlärningsregel	Ingen	
Inlärningsschema	Default	
Initiering	undre gräns	-1.0
	övre gräns	+1.0



8.4.3 Testkörning

Vid testkörning med 'linj2300' användes 120 noder i det gömda lagret. Om kriteriet för antal noder skulle varit uppfyllt, borde detta antal ha varit 996, lika många som antalet indatavektorer i filen monst101. Så många noder klarar emellertid inte programmet NeuralWorks av att hantera. Under körning visade det sig, att RMS-felet var konstant under hela inlärningen, med värdet 0.2957. Under tiden inlärningen pågick, kunde på skärmen konstateras att kategoriseringen stämde för i stort sett varenda indatavektor, betydligt bättre än för vad som iakttagits med andra nätverkstyper. Då layouten på skärmen ändrades, blev emellertid resultaten helt olika, något som inte alls borde inträffa. Programvaruleverantören avfärdade det hela, med att antalet noder i det gömda lagret inte var tillräckligt många, vilket dock inte verkade vara den riktiga orsaken.

Rent subjektivt, RMS-felets konstanta värde talar för motsatsen, verkar pnn vara mer lovande än de andra nätverkstyper som undersökts. För en bättre analys fodras dock att indata komprimerats så pass, att antalet noder i gömda lagret kan sättas lika med antalet olika indatavektorer.



8.5 SELF-ORGANIZING-MAPS NÄTVERK, S O M

8.5.1 Allmänt

S o m är en nätverkstyp som används för mönsterigenkänning och kategorisering. Det gömda lagret är av Kohonentyp. Utlagret kan ha olika egenskaper. Man skiljer på tre skilda typer av s o m. Standard s o m, kategoriserande s o m och predikterande s o m. I denna tillämpning har den kategoriserande varianten valts. Denna är i funktionssättet lik pnn, men bygger till skillnad från pnn, inte på sannolikheter för de olika kategorierna. Inlärningen sker, som i fallet med pnn, i två steg. Först lärs Kohonenlagret upp och sedan utlagret, se pnn (8.4.1). Pnn kräver, enligt [17], lika många gömda noder, som antalet invektorer in indatafilen. Även om s o m, till funktionen är likt pnn, nämns inget om detta krav för s o m nätverk, i [17]. Resultaten nedan, (8.5.3), tyder inte heller på att så är fallet.

8.5.2 Fördefinierade värden, default, self-organizing-maps

Nätverket

Kontrollstrategi	s o m

Inlagret

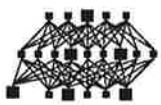
Summationsfunktion	Sum
Transferfunktion	Linjär
Felfunktion	Standard
Utfunktion	Direkt
Inlärningsregel	none
Inlärningsschema	Net global

Gömda lagret, (Kohonen)

Summationsfunktion	S o m
Transferfunktion	Linjär
Felfunktion	Standard
Utfunktion	S o m
Inlärningsregel	S o m
Inlärningsschema	S o m

Utlagret

Summationsfunktion	Sum
Transferfunktion	Linjär
Felfunktion	Standard
Utfunktion	Direkt
Inlärningsregel	Widrow-Hoff
Inlärningsschema	som-cat



8.5.3 Antal noder i Kohonenlagret, 7 gånger 5 pixels

Kohonenlagret är ett tvådimensionellt lager. Vid uppbyggnaden av nätverket anges bredd och höjd, uttryckt i antal noder, för detta lager. I näten 'linj1750' till 'linj1810' har antalet noder, här lika många i bredd som i höjdled, varierats, medan övriga parametrar hållits konstanta, satta till sina fördefinierade värden.

Diagram 28 a) och b) visar RMS-felets avtagande, vid olika storlekar på Kohonenlagret. Notera att fas ett av inlärningen, organiseringen av Kohonenlagret, pågår fram till exempel 2000. Under denna fas är utsignalerna, från nätet, helt intetsägande, eftersom upplärningen av utlagret då ännu ej har börjat. 6 gånger 6 noder i Kohonenlagret, ger det bästa upplärningsbeteendet. Felkurvan har periodvis helt planat ut på felvärdet noll. Emellanåt har dock knyckar i kurvan uppstått, vilka kan tänkas bero på extrema indatavärden.

Då antalet noder är runt 36 stycken, fås bästa resultatet. Bara de fall då Kohonenlagret är kvadratiskt har studerats. Andra förhållanden, mellan höjd och bredd i detta lager, bör också vara intressanta att undersöka.

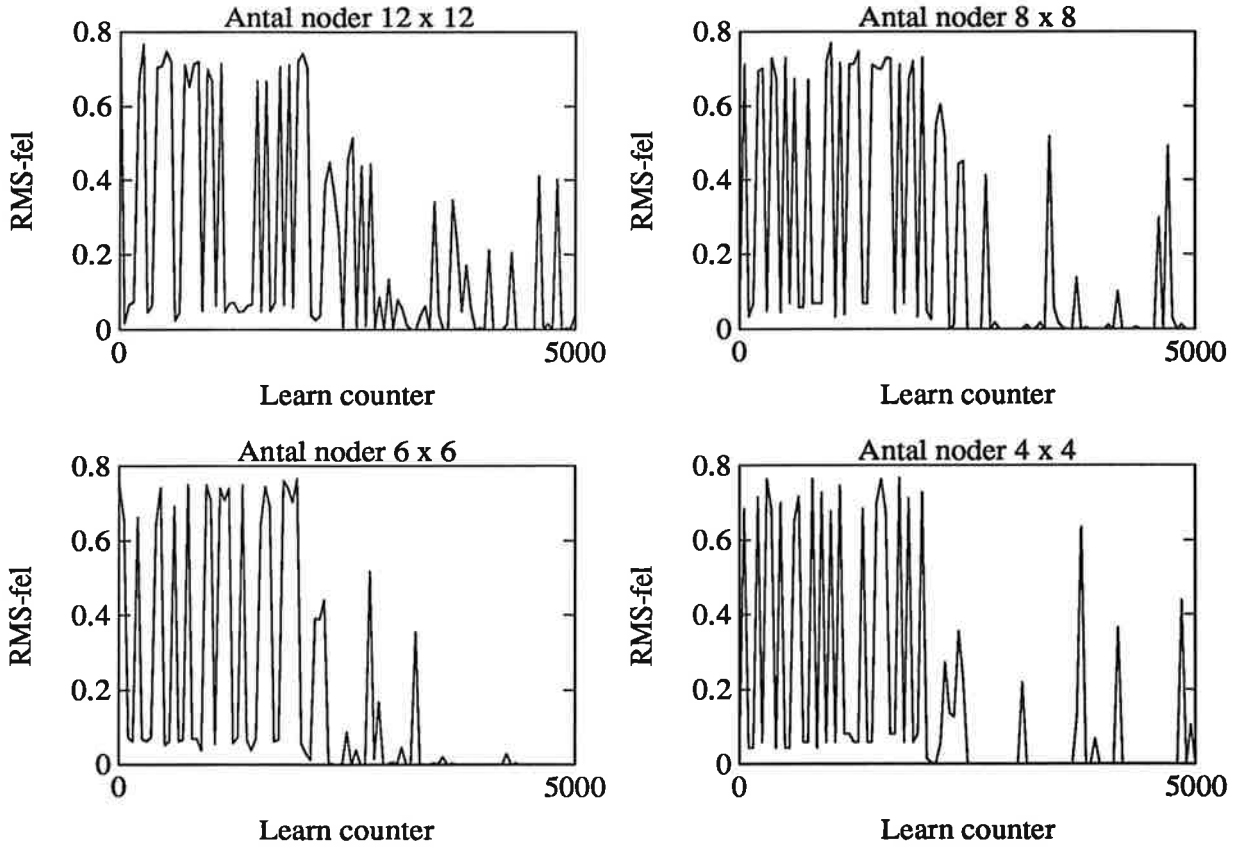
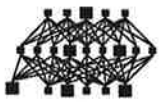


Diagram 28 a)RMS-felet som funktion antalet inlärda exempel, för olika antal noder i Kohonenlagret. 8 set av indata, 7 gånger 5 pixels.



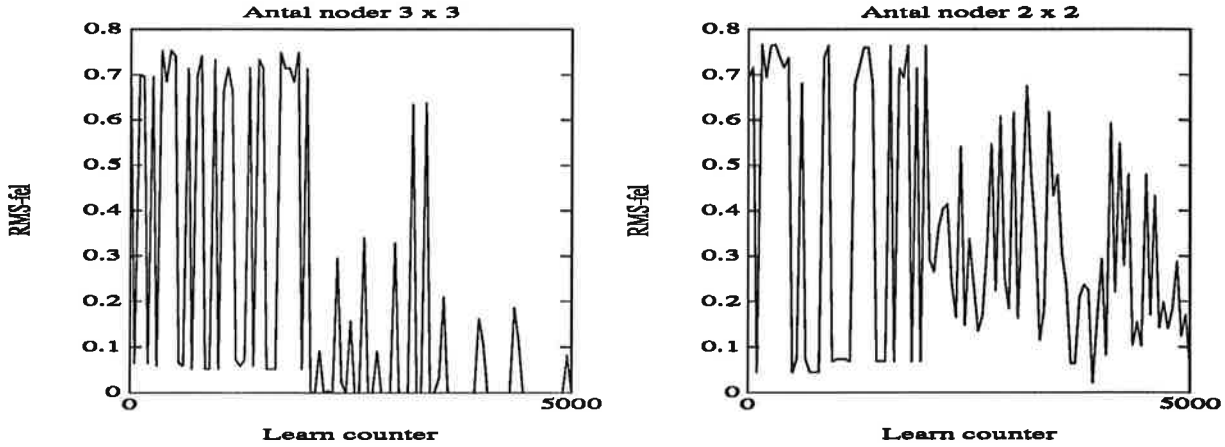


Diagram 28 b) RMS-felet som funktion antalet inlärda exempel, för olika antal noder i Kohonenlagret. 8 set av indata, 7 gånger 5 pixels.

8.5.4 Brus, olika nivåer, 7 gånger 5 pixels

I (8.5.3) kördes näten på indata, vars brus var fördelat med sigma = 0.5, se (6.6). De lyckade resultatet då Kohonenlagret bestod av 6 gånger 6 noder, i nät 'linj1770', beror delvis på högt signal/brusförhållande. 'linj1790' är identiskt uppbyggt, men har tränats på indata med sigma = 0.8, ett lägre signal/brusförhållande. RMS-felen för de två körningarna syns i diagram 29. En betydligt sämre konvergens för RMS-felet, blir följden av ett minskat signal/brusförhållande.

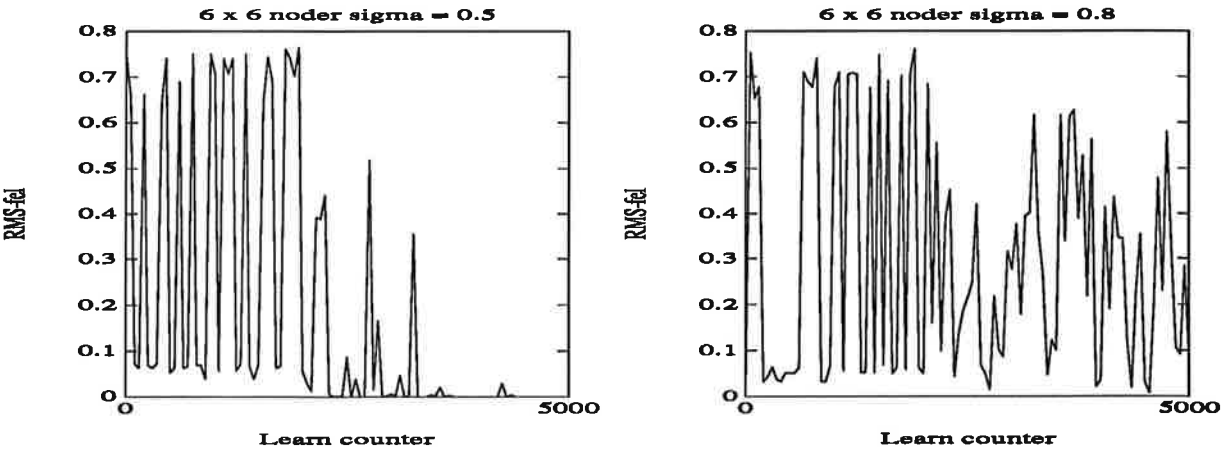


Diagram 29. RMS-felet som funktion antalet inlärda exempel, för olika brusnivåer. 6 gånger 6 noder i Kohonenlagret, 8 set av indata, 7 gånger 5 pixels.

8.5.5 Antal noder i Kohonenlagret, olika brus, 13 gånger 9 pixels

För bildstorlek 13 gånger 9 pixels, har två storlekar på Kohonenlagret, och två olika brusnivåer prövats, i nätverken 'linj1900' till 'linj1930'. P g a att fler olika indata finns för 13 gånger 9 pixels, har endast 4 set av indata använts. Indatafilen blir trots detta av storleken 2 megabyte, i sin asciivariant.

I diagram 30 åskådliggörs RMS-felet under inlärningen. Det större Kohonenlagret ger även här bäst resultat. Förmodligen skulle ett ännu större lager, ge ytterligare förbättring. Oavsett storleken på detta lager, tycks det vara svårt att detektera linjer i en brusmiljö då sigma = 0.8 .

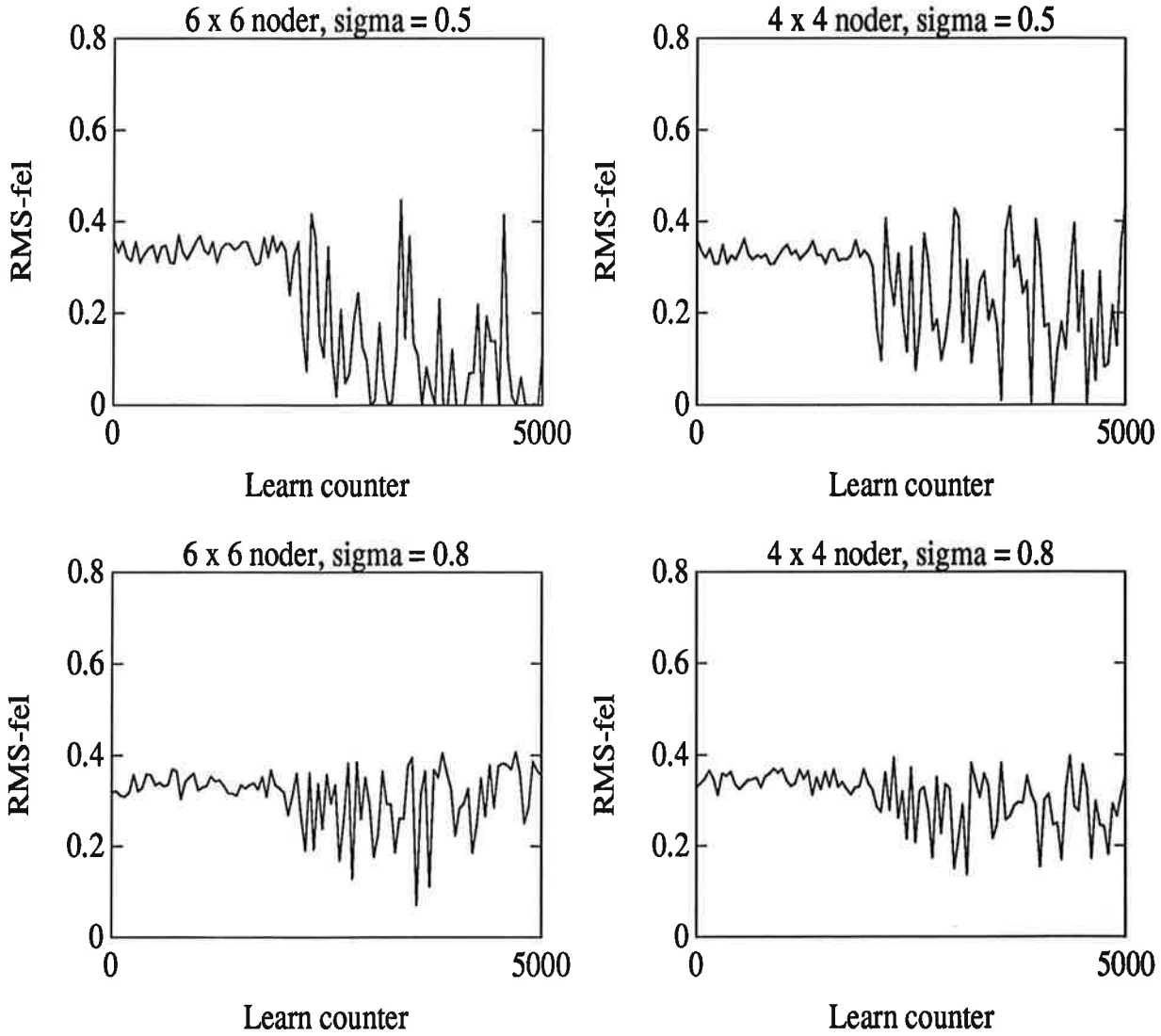


Diagram 30. RMS-felet som funktion antalet inlärda exempel, för olika antal noder i Kohonenlagret och olika brusnivåer. 4 set av indata, 13 gånger 9 pixels.

I 'linj1940' har en längre inlärning prövats, samma parametrar i övrigt som i 'linj1930'. Fas ett av inlärningen, sträcker sig här till exempel 200 000, därefter vidtar upplärningen av utlagret under fas två, upp till exempel 500 000. RMS-felet för denna körning, se diagram 31, går trots en lång inlärning, inte ner längre än till en nivå, strax under 0.3

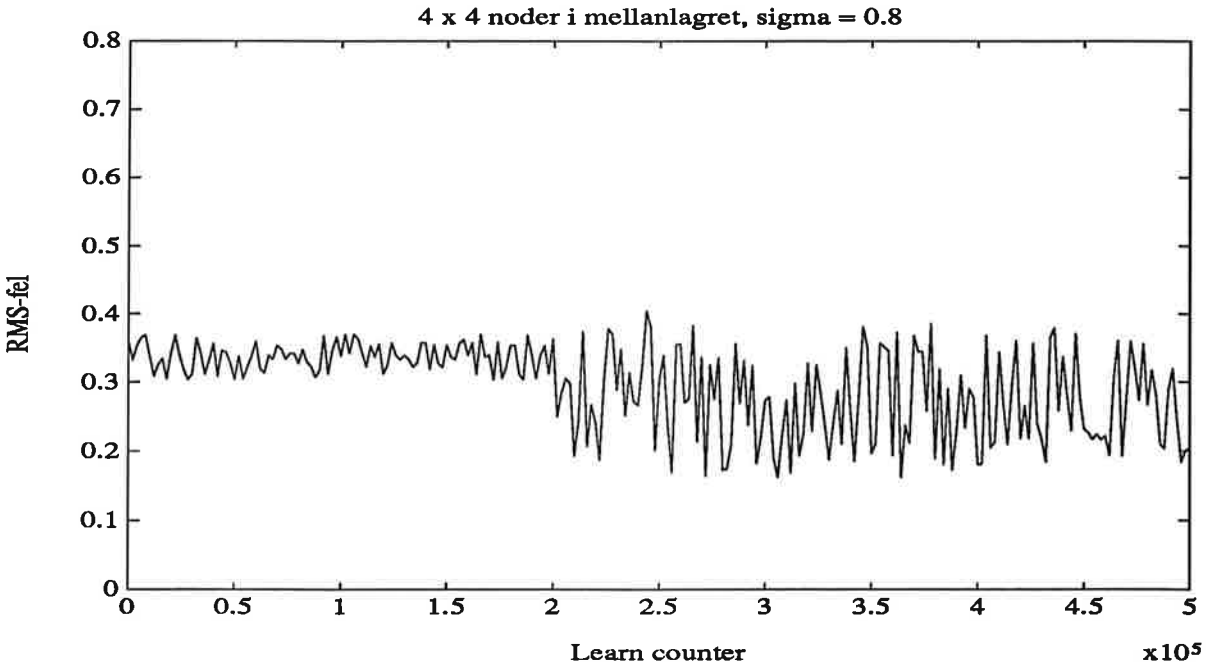


Diagram 31. RMS-felet som funktion antalet inlärda exempel, 4 gånger 4 noder i Kohonenlagret, sigma = 0.8, 4 set av indata, 13 gånger 9 pixels.

8.5.6 Organiseringen i Kohonenlagret

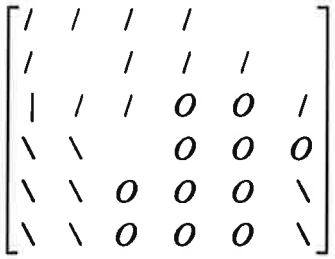
I NeuralWorks kan man under inlärning eller testning av nätverk, samtidigt iakttaga de värden, som nodernas utsignaler för tillfället har. Genom att fästa en overhead-film på skärmen och sedan markera på denna, vilka noder som aktiveras av respektive indata, kan man få en inblick i hur data organiseras i Kohonenlagret. Under olika intervall av inlärningen, har för varje nod ritats i, den kategori av indata som organiseras till just denna nod. För några noder var organiseringen inte entydig, utan mer än en kategori kopplades till samma nod i Kohonenlagret.

8.5.6.1 Nätverk, 7 gånger 5 pixels

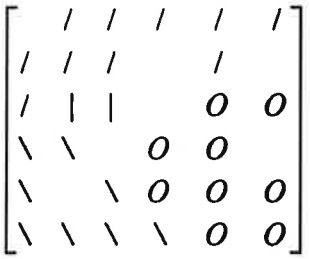
De positioner som lämnats blanka, motsvarar inaktiva noder, utan koppling till någon kategori, ett stort O står för indata utan linjemönster, medan strecken motsvarar respektive linjekategoris lutning.

'linj1770', 6 gånger 6 noder i Kohonenlagret. Aktuell mappning i respektive inlärningsintervall.

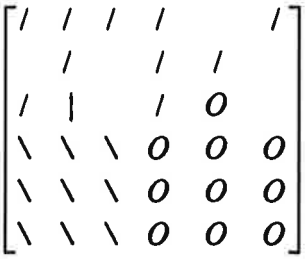
Exempel 1-600



Exempel 600-1000

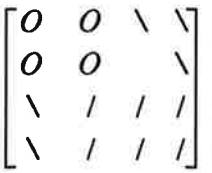


Exempel 1000-1500

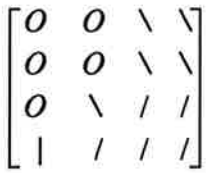


'linj1780', 4 gånger 4 noder i Kohonenlagret. Aktuell mappning i respektive inlärningsintervall.

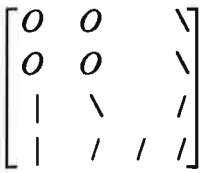
0-160



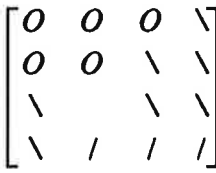
160-300



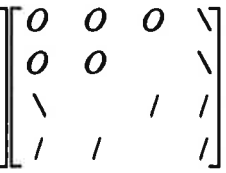
300-450



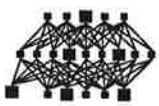
450-600



1900-2000



Notera att för 'linj1770', är områdena i Kohonenlagret för respektive kategori, inte sammanhängande från början. Efterhand som inlärningen fortsätter omorganiseras dock lagret så, att områdena blir sammanhängande. De noder som är inaktiva, återfinns främst i gränsområdet mellan olika kategorier. För det mindre lagret i 'linj1780', är antalet noder för litet. T ex den vertikala linjen, saknar representerande nod, i vissa inlärningsintervall.



8.5.6.2 Nätverk, 13 gånger 9 pixels

Kategoriseringen har i detta fall gjorts i 9 olika klasser se (6.5). I matriser nedan har nummer för respektive kategori satts ut. De blanka positionerna innebär att ingen linjekategori kopplats till denna nod, antingen kan noden vara inaktiv eller kopplad till bilder utan linjemönster.

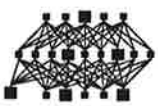
'linj1900', 6 gånger 6 noder i Kohonenlagret, inlärningsintervall exempel 4000 till 5000.

$$\begin{bmatrix} 9 & 8 & 7 & 7 & 6 & 6 \\ 9 & 8 & 7 & 6 & 5 & 5 \\ 9 & & 7 & 6 & 5 & 5 \\ & 2 & 1 & 2 & 4 & 4 \\ 3 & 2 & 1 & 1 & 4 & 4 \\ 3 & & 1 & 1 & 1 & 3 \end{bmatrix}$$

'linj1910', 4 gånger 4 noder i Kohonenlagret, inlärningsintervall exempel 4000 till 5000.

$$\begin{bmatrix} 98 & 7 & 76 & 6 \\ 9 & 87 & 76 & 5 \\ 1 & & 3 & 4 \\ 12 & 21 & 23 & 3 \end{bmatrix}$$

I 'linj1910' var kategoriseringen mindre entydig, varför två kategorier markerats för vissa noder. 'linj1900' gav däremot, p g a fler noder entydigare organisering. Notera att kategorierna, för linjer med närliggande lutningar, organiseras intill varandra i Kohonenlagret. Matrisbilderna ger stöd för tanken, att fler noder i Kohonenlagret, ger bättre klassificeringsegenskaper för nätet.



9 FÖRKLARINGAR TILL BEGREPP INOM NEURALA NÄTVERK

adaptive resonans theory networks, art

Nätverkstyp för kategorisering. Huvudsakligen består denna nättyp förutom inlager och utlager av två aktiva lager, F1 och F2. Dessa två lager har fullständiga dubbelriktade förbindelser mellan alla sina noder. F2 lagret är ett konkurrerande lager, där endast en nod är aktiv åt gången. F1 lagrets noder uppdateras ofta, så att detta lager bara kommer att innehålla närtidsinformation. Resonans mellan dessa två lager ger nätet dess önskade egenskaper. Ett problem med denna nättyp är att kategoriseringen inte är stabil, vilket delvis beror på att inlärningen inte är styrd mot några speciella utdata, utan sker helt autonomt. Vid ett visst stadium i inlärningen motsvarar en nod en viss kategori, medan den senare kan motsvara en helt annan kategori.

auto-associativt

Om indata har samma utseende som utdata, dvs om indatavektorn har samma längd och innehåll som utdatavektorn, kallas nätverket auto-associativt.

back-propagation

Den i särklass mest använda nätverkstypen. Lärs upp genom att felen i utlagret skickas bakåt till förgående lager, där nu även felet i detta lager kan beräknas och skickas vidare, osv. Olika varianter har tagits fram för minska den långa inlärningstiden. T ex back-propagation med pruning eller functional link, se detta ord.

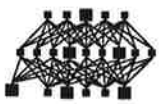
checkpoint

I programmet kan man med jämna intervaller sätta så kallade checkpoints. Vid inlärningen, när exempel efter exempel lärs in, passeras dessa checkpoints. När en checkpoint nås, utförs de alternativ som markerats i meny under checkpoint. T ex rita upp nätet, spara aktuellt nät eller utföra pruning.

counterpropagation

Klassificerande nätverkstyp vars gömda lager är ett Kohonenlager och utlager är ett Grossberg Outstarlager. Kohonenlagret grupperar likartade data intill varandra. Varje Grossberg Outstarnod ger utsignal, då ett visst inmönster skickas vidare från Kohonenlagret. För att vara funktionsdugligt måste denna typ av nätverk ha tillräckligt många noder, i Kohonenlagret, för att separeringen av indata skall bli tillräcklig. Vidare får indata ej vara överlappande och bör vara lika fördelat mellan de olika kategorierna.

directed random search, drs, se (8.3.1).



epoch	Då cumulative-delta-rule används som inlärningsregel, uppdateras inte vikternas värden efter varje exempel, utan felet från utnoderna ackumuleras och uppdateras efter ett visst antal genomgångna exempel. Det antal exempel som på detta sätt summeras benämns en epoch. Se även (8.2.8).
Flash code™	Då nätet är genererat och färdigtränat kan det kompileras till exekverbar C-kod. Denna kod kallas Flash code.
functional link	Vanligast förekommande i back-propagationnät. Efter inlagret har ett extra lager lagts till i nätstrukturen. I detta lager transformeras indata med någon enkel funktion, t ex sinusfunktionen, vilket kan ge kortare inlärningstid i vissa fall.
hetero-associativt	Om indata inte har samma utseende som utdata, dvs om indatavektorns längd och innehåll ej är detsamma som utdatavektorns, kallas nätverket hetero-associativt.
Kohonenlager	Klassificerande lager i nätverk. Noderna i detta lager konkurrerar, bara en nod är aktiv åt gången, den med starkast utsignal utses till vinnare. Används t ex i counterpropagation, lvq, pnn och s o m.
learn counter	Räknare som håller reda på hur många inlärningsexempel som hittills lärts in. I de flesta diagram över RMS-felet ritas detta som funktion av antalet inlärda exempel, dvs värdet på learn counter.
learning vector quantization, lvq	Nätverkstyp för klassificering, innehåller Kohonenlager. Indatavektorerna, med n stycken vektorelement, spänner upp en rymden R^n. I denna rymd tänkes noderna ligga. Lvq beräknar avståndet från varje indatavektor till samtliga noder. Den nod som ligger närmast utses till vinnare. Om den vinnande noden tillhör samma klass som indatavektorn, flyttas noden i riktning mot denna. Då den vinnande noden inte tillhör samma klass, flyttas den bort från vektorn, sk repulsion. Under inlärningen rör sig sålunda noderna till de områden i R^n där respektive klass är lokaliserad.
probabilistic neural network, pnn, se (8.4.1).	
pruning	Pruning bygger på antagandet att nätverk med mindre komplexitet har större förmåga att generalisera. Ett sätt att minimera komplexiteten är att ta bort små vikter. Detta görs med jämna mellanrum, genom att sätta ut checkpoints och vid dessa sortera bort valda vikter.

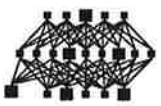
self organizing
map, s o m , se (8.5.1).

set, av indata

Varje set av indata innehåller alla olika linjemönster med brus superponerat. För att få en bättre inlärning används flera omgångar indata med olika brus, flera set av indata. Se även (6.7).

spatio temporal
pattern recognition,
spr

Nätverk lämpade för tidsberoende förlopp. Efter det att insignaler påverkat en nod, avtar dennas aktivitet långsamt. Under hela avklingningstiden påverkas utsignalerna från nätverket, vilket gör att nättypen kan känna igen signaler i tidsplanet. Spr är en självorganiserande nätverkstyp. Har bland annat använts för klassificering av fartyg utgående från hydrofonupptagningar av propellerljud.



10 LITTERATURREFERENSER

ALLMÄNT OM NEURALA NÄTVERK

1 Jonas Samuelsson, *Artificiella neurala nätverk*
Examensarbete utfört på Telub AB, våren 1991

2 Rumelhart D. & McClelland J., *Parallel Distributed Processing Vol 1 & 2*
Bradford Books / The MIT Press, 1986
Grundläggande bok, skriven på ett lättillgängligt sätt.

3 Simpson P. K., *Artificial Neural Systems*
Pergamon Press
Grundläggande bok.

4 DARPA, *DARPA Neural Network Study,*
Washington, AFCEA Press, 1988
En bok som diskuterar tillämpningar.

BACK-PROPAGATION

5 Gorman R. P.,Sejnowski T. J., *Analysis of hidden units in a layered network trained to classify sonar targets*
Neural Networks, 1, sid 75-89

6 Fahlmann Scott E., *An empirical study of learning speed in back-propagation networks*
CMU, Technical Report, CMU-CS-88-162, June 1988

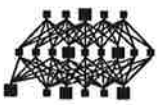
7 Jones William P., Hoskins Josiah, *Back-propagation, a generalized delta learning rule*
Byte, Oktober 1987

8 Lapedes A., Farber R., *Non-linear signal processing using neural networks: Prediction and system modelling*
Los Alamos National Laboratory report LAUR-87-2662

Counter-Propagation

9 Hecht-Nielsen Robert, *Counter-Propagation Networks*
IEEE First International Conference on Neural Networks, Volume II sid 19-32

10 Stork David G., *Counter-Propagation Networks: Adaptive hierachial networks for near-optimal Mappings*
Synapse Connection, Volume 1, Nr 2 sid 9-11,17



Direct-random-search

11 Baba Norio, Shoman T., Sawaragi Y., *A modified convergence theorem for a random optimization method*
Information Sciences, 1977, Volume 13 sid 159-166

12 Schrack G., Choit M, *Optimized relative step size random searches*
Mathematical Programming, 1977, Volume 10 sid 230-244

13 Solis F. J., Wets R. J-B., *Minimization by random search techniques*
Mathematics of Operations Research, Volume 6, No. 1 sid 19-30

PROBABILISTIC-NEURAL-NETWORKS

14 Specht D. F., *Probabilistic Neural Networks for Classification, Mapping or Associative Memory*
ICNN-88, Conference proceedings, 1988

SELF-ORGANIZING-MAPS

15 Kohonen T, *Self-Organizing and Associative Memory*
Second Edition, Springer-Verlag, New York

16 Willshaw D. J., von der Malsburg C., *How patterned neural connections can be set up by self-organization*
Proc. R. Soc. London. B. Volume 194 sid 431-445

NEURALWORKS PROFESSIONAL II

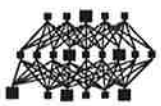
17 *Neural Computing*
Manual tillhörande programpaket NeuralWorks Professional II/PLUS, NeuralWare Inc, 1990

18 *Reference Guide*
Manual tillhörande programpaket NeuralWorks Professional II/PLUS, NeuralWare Inc, 1990

19 *Using NWorks*
Manual tillhörande programpaket NeuralWorks Professional II/PLUS, NeuralWare Inc, 1990

20 *Intressanta miljöer för läraktiga nät*
Datateknik, nr 4 1990, sid 63-66

21 Mark B. Fishman, Edmund L. Gallizzi, Eckerd College, *A pair of neural packages for research*
Computer, Januari 1991, sid 107-110



22 Peter G. Raeth, Wright Research and Development Center, Avionics Laboratory, *Applying technology with NeuralWorks Professional II* Computer, Januari 1991, sid 111-112

ANDRA NÄTVERKSPROGRAM

23 *Brainmaker: Krångligt neuronnät som lätt kraschar* Datateknik, nr 13 1990, sid 50-51

HÅRDVARA FÖR NEURALA NÄTVERK

24 Janet J. Barron, *Chips for the nineties and beyond* Byte, November 1990, sid 342-350

25 Michael W. Roth, Andreas G. Andreou, *Analog VLSI and Neural Systems* Proceedings of the IEEE, vol 78, no.11, november 1990

BILDBEHANDLING

26 David P. Wright, Christopher L. Scofield, *Divide and conquer* Byte, April 1990, sid 207-210

27 Rafel C. Gonzalez, Paul Wintz, *Digital Image Processing* Addison-Wesley Publishing Company Inc, 1987

SIGNALBEHANDLING

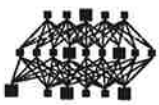
28 R. B. Randall, *Frequency analysis,* Brüel & Kjær, 3:e utgåvan, 1987

29 S. Lawrence Marple Jr., *Digital spectral analysis with applications* Prentice Hall Inc, 1987

30 Robert K. Otnes, Loren Enochson, *Applied time series analysis* John Wiley and Sons, 1978

AMERIKANSKA TIDSKRIFTER OCH ANDRA ÅTERKOMMANDE PUBLIKATIONER

31 *Neural Networks* P.O. Box 441166, Fort Washington, MD 20744, USA



32 *Neural Networks*
Maxwell Pergamon Macmillan Inc.
Maxwell House, Fairview Park, Elmsford, NY 10523, USA

33 *Journal of Neural Network Computing*
Warren, Gorham and Lamont
210 South St., Boston, MA 02111, USA

34 *International Journal of Neural Networks*
Learned Information Inc.
143 Old Marlton Pike, Medford, NJ 08055, USA

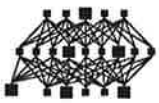
35 *Neural Network Review*
Lawrence Erlbaum Associates Inc.
365 Broadway, Hillsdale, NJ 07642, USA

36 *Neural Networks News*
AIWeek Inc.
2555 Cumberland Parkway, Suite 299, Atlanta, GA 30339, USA

37 *Neural Technology Update*
GRAEME Publishing Corporation
10 Northern Blvd, Amherst, NH 03031, USA

38 *Intelligence*
Edward Rosenfeld
P.O. Box 20008, New York, NY 10025, USA

39 *IEEE Transactions on Neural Networks*
The Institute of Electrical and Electronics Engineers Inc,
345 East 47th Street, New York, NY 10017, USA



11 FILFÖRTECKNING

Lista, över filer, med korta förklaringar. I de fall programlistor finns återfinns dessa i kapitel 12, PROGRAMLISTNINGAR.

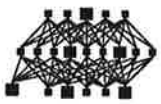
11.1 NÄTVERKSFILER, LÄSBARA AV NEURALWORKS

Varje filtyp slutar på det suffix som anges i respektive underrubrik. I körningarna har olika parametrar varierats. Nätverk med olika värden på samma parameter är i regel sparade i samma underbibliotek.

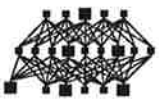
11.1.1 Nätverk, '.nnd'

Nätverken har sparats i filer med prefix 'linje' eller 'linj' följt av ett tre, respektive fyrsiffrigt tal. Fullständig förteckning över alla '.nnd'-filer, finns lagrad i Lotusfilen filtab.wk1.

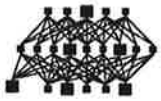
Nätverksfil '.nnd'	Nätverk typ	Parameter	Antal gömda noder	Indata '.nna' '.nnb'	Indata brusets standard-avvikelse	Linje matris	Testfil '.nna' '.nnb'	Testdata brusets standard-avvikelse	Bildfil '.met' '.ps'	Learn-counter
Linje050	backpropagation	brus stand. avvik.	10	monste22	1.1	7x5			bild15	1878
Linje051	backpropagation	brus stand. avvik.	10	monste22	1.1	7x5			bild15	2597
Linje052	backpropagation	brus stand. avvik.	10	monste22	1.1	7x5			bild15	3316
Linje060	backpropagation	brus stand. avvik.	10	monste24	1.0	7x5			bild15	1824
Linje061	backpropagation	brus stand. avvik.	10	monste24	1.0	7x5			bild15	3000
Linje062	backpropagation	brus stand. avvik.	10	monste24	1.0	7x5			bild15	6000
Linje063	backpropagation	brus stand. avvik.	10	monste24	1.0	7x5			bild15	9000
Linje064	backpropagation	brus stand. avvik.	10	monste24	1.0	7x5			bild15	12000
Linje065	backpropagation	brus stand. avvik.	10	monste23	0.9	7x5			bild15	66
Linje066	backpropagation	brus stand. avvik.	10	monste23	0.9	7x5			bild15	4704
Linje067	backpropagation	brus stand. avvik.	10	monste23	0.9	7x5			bild15	10069
Linje070	backpropagation	brus stand. avvik.	10	monste25	0.8	7x5			bild15	400
Linje071	backpropagation	brus stand. avvik.	10	monste25	0.8	7x5			bild15	1390
Linje072	backpropagation	brus stand. avvik.	10	monste25	0.8	7x5			bild15	2374
Linje075	backpropagation	brus stand. avvik.	10	monste26	0.7	7x5			bild15	1590
Linje080	backpropagation	brus stand. avvik.	10	monste27	0.6	7x5			bild15	1604
Linje085	backpropagation	brus stand. avvik.	10	monste28	0.5	7x5			bild15	1
Linje086	backpropagation	brus stand. avvik.	10	monste28	0.5	7x5			bild15	1707
Linje100	backpropagation	antal gömda noder	15	monste25	0.8	7x5			bild1,2,3	3000
Linje110	backpropagation	antal gömda noder	14	monste25	0.8	7x5			bild2,3	3000
Linje120	backpropagation	antal gömda noder	13	monste25	0.8	7x5			bild2,3	3000
Linje130	backpropagation	antal gömda noder	12	monste25	0.8	7x5			bild2,3	3000
Linje140	backpropagation	antal gömda noder	11	monste25	0.8	7x5			bild2,3	3000
Linje150	backpropagation	antal gömda noder	10	monste25	0.8	7x5			bild1,2,3	3000
Linje160	backpropagation	antal gömda noder	9	monste25	0.8	7x5			bild2,3	3000
Linje170	backpropagation	antal gömda noder	8	monste25	0.8	7x5			bild2,3	3000
Linje180	backpropagation	antal gömda noder	7	monste25	0.8	7x5			bild2,3	3000
Linje190	backpropagation	antal gömda noder	6	monste25	0.8	7x5			bild2,3	3000
Linje200	backpropagation	antal gömda noder	5	monste25	0.8	7x5			bild1,2,3	3000
Linje210	backpropagation	antal gömda noder	4	monste25	0.8	7x5			bild2,3	3000
Linje220	backpropagation	antal gömda noder	3	monste25	0.8	7x5			bild2,3	3000
Linje230	backpropagation	antal gömda noder	2	monste25	0.8	7x5			bild2,3	3000
Linje240	backpropagation	antal gömda noder	1	monste25	0.8	7x5			bild2,3	3000
Linje250	backpropagation	antal gömda noder	0	monste25	0.8	7x5			bild2,3	3000
Linje260	backpropagation	antal gömda noder	30	monste25	0.8	7x5			bild1,2,3	3000
Linje270	backpropagation	antal gömda noder	25	monste25	0.8	7x5			bild1,2,3	3000
Linje280	backpropagation	antal gömda noder	20	monste25	0.8	7x5			bild1,2,3	3000
Linje290	backpropagation	antal gömda noder	27	monste25	0.8	7x5			bild2,3	3000
Linje300	backpropagation	antal gömda noder	23	monste25	0.8	7x5			bild2,3	3000
Linje310	backpropagation	antal gömda noder	26	monste25	0.8	7x5			bild2,3	3000



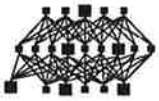
Linje320	backpropagation	antal gömda noder	24	monste25	0.8	7x5			bild2,3	3000
Linje330	backpropagation	antal gömda noder	22	monste25	0.8	7x5			bild2,3	3000
Linje340	backpropagation	antal gömda noder	21	monste25	0.8	7x5			bild2,3	3000
Linje350	backpropagation	antal gömda noder	28	monste25	0.8	7x5			bild2,3	3000
Linje360	backpropagation	antal gömda noder	29	monste25	0.8	7x5			bild2,3	3000
Linje370	backpropagation	antal gömda noder	34	monste25	0.8	7x5			bild2,3	3000
Linje380	backpropagation	antal gömda noder	33	monste25	0.8	7x5			bild2,3	3000
Linje390	backpropagation	antal gömda noder	32	monste25	0.8	7x5			bild2,3	3000
Linje400	backpropagation	antal gömda noder	31	monste25	0.8	7x5			bild2,3	3000
Linje410	backpropagation	antal gömda noder	40	monste25	0.8	7x5			bild1,2,3	3000
Linje420	backpropagation	antal gömda noder	39	monste25	0.8	7x5			bild2,3	3000
Linje430	backpropagation	antal gömda noder	38	monste25	0.8	7x5			bild2,3	3000
Linje440	backpropagation	antal gömda noder	37	monste25	0.8	7x5			bild2,3	3000
Linje450	backpropagation	antal gömda noder	36	monste25	0.8	7x5			bild2,3	3000
Linje460	backpropagation	antal gömda noder	35	monste25	0.8	7x5			bild1,2,3	3000
Linje470	backpropagation	antal gömda noder	16	monste25	0.8	7x5			bild2,3	3000
Linje480	backpropagation	antal gömda noder	17	monste25	0.8	7x5			bild2,3	3000
Linje490	backpropagation	antal gömda noder	18	monste25	0.8	7x5			bild2,3	3000
Linje500	backpropagation	antal gömda noder	19	monste25	0.8	7x5			bild2,3	3000
Linje510	backpropagation	antal gömda noder	50	monste25	0.8	7x5			bild1	3000
Linje520	backpropagation	antal gömda noder	45	monste25	0.8	7x5			bild1	3000
Linje530	backpropagation	antal gömda noder	55	monste25	0.8	7x5			bild1	3000
Linje540	backpropagation	antal gömda noder	60	monste25	0.8	7x5			bild1	3000
		Antal omgångar i indata								
Linje550	backpropagation	1	13	monste81	0.8	7x5			bild4,5	3000
Linje560	backpropagation	2	13	monste82	0.8	7x5			bild4	3000
Linje570	backpropagation	3	13	monste83	0.8	7x5			bild4	3000
Linje580	backpropagation	4	13	monste84	0.8	7x5			bild4	3000
Linje590	backpropagation	5	13	monste85	0.8	7x5			bild4,5	3000
Linje600	backpropagation	6	13	monste86	0.8	7x5			bild4	3000
Linje610	backpropagation	7	13	monste87	0.8	7x5			bild4	3000
Linje620	backpropagation	8	13	monste88	0.8	7x5			bild4,5	3000
Linje630	backpropagation	9	13	monste89	0.8	7x5			bild4	3000
Linje640	backpropagation	10	13	monste90	0.8	7x5			bild4,5	3000
Linje650	backpropagation	Epochsize 16	13	monste41	0.5	7x5	monste31	0.5	bild10,11	3000
Linje660	backpropagation	Epochsize 8	13	monste41	0.5	7x5	monste32	0.5	bild10,11	3000
Linje670	backpropagation	Epochsize 32	13	monste41	0.5	7x5	monste33	0.5	bild10,11	3000
Linje680	backpropagation	Epochsize 12	13	monste41	0.5	7x5	monste34	0.5	bild10,11	3000
Linje690	backpropagation	Epochsize 20	13	monste41	0.5	7x5	monste35	0.5	bild10,11	3000
Linje700	backpropagation	Epochsize 24	13	monste41	0.5	7x5	monste36	0.5	bild10,11	3000
Linje710	backpropagation	Epochsize 28	13	monste41	0.5	7x5	monste37	0.5	bild10,11	3000
		Transferfunction								
Linje720	backpropagation	tanh	13	monste41	0.5	7x5	monste31	0.5	bild8,9	3000
Linje730	backpropagation	sigmoid	13	monste41	0.5	7x5	monste32	0.5	bild8,9	3000
Linje740	backpropagation	linear	13	monste41	0.5	7x5	monste33	0.5	bild8,9	3000
Linje750	backpropagation	bsb	13	monste41	0.5	7x5	monste34	0.5	bild8,9	3000
Linje760	backpropagation	sine	13	monste41	0.5	7x5	monste35	0.5	bild8,9	3000
Linje770	backpropagation	signum	13	monste41	0.5	7x5	monste36	0.5	bild8,9	3000
Linje780	backpropagation	signum0	13	monste41	0.5	7x5	monste37	0.5	bild8,9	3000
Linje790	backpropagation	stepfunct	13	monste41	0.5	7x5	monste38	0.5	bild8,9	3000
Linje800	backpropagation	bam	13	monste41	0.5	7x5	monste39	0.5	bild8,9	3000
Linje810	backpropagation	perceptron	13	monste41	0.5	7x5	monste40	0.5	bild8,9	3000
		Summation								
Linje820	backpropagation	sum	13	monste41	0.5	7x5	monste31	0.5	bild6,7	3000
Linje830	backpropagation	cumsum	13	monste41	0.5	7x5	monste32	0.5	bild6,7	3000
Linje840	backpropagation	maximum	13	monste41	0.5	7x5	monste33	0.5	bild6,7	3000
Linje850	backpropagation	minimum	13	monste41	0.5	7x5	monste34	0.5	bild6,7	3000
Linje860	backpropagation	majority	13	monste41	0.5	7x5	monste35	0.5	bild6,7	3000
Linje870	backpropagation	norm1	13	monste41	0.5	7x5	monste36	0.5	bild6,7	3000
Linje880	backpropagation	normN	13	monste41	0.5	7x5	monste37	0.5	bild6,7	3000



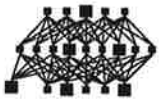
Linje890	backpropagation	normScale	13	monste41	0.5	7x5	monste38	0.5	bild6,7	3000
Linje900	backpropagation	normPolar	13	monste41	0.5	7x5	monste39	0.5	bild6,7	3000
Linje910	backpropagation	normMult	13	monste41	0.5	7x5	monste40	0.5	bild6,7	3000
Linje920	backpropagation	product	13	monste41	0.5	7x5	monste42	0.5	bild6,7	3000
Linje930	backpropagation	city_block	13	monste41	0.5	7x5	monste43	0.5	bild6,7	3000
Linje940	backpropagation	spr	13	monste41	0.5	7x5	monste44	0.5	bild6,7	3000
Linje950	backpropagation	dnna	13	monste41	0.5	7x5	monste45	0.5	bild6,7	3000
Linje960	backpropagation	som	13	monste41	0.5	7x5	monste46	0.5	bild6,7	3000
		Error								
Linje970	backpropagation	standard	13	monste41	0.5	7x5	monste31	0.5	bild16,17	3000
Linje980	backpropagation	quadratic	13	monste41	0.5	7x5	monste32	0.5	bild16,17	3000
Linje990	backpropagation	cubic	13	monste41	0.5	7x5	monste33	0.5	bild16,17	3000
Linj1000	backpropagation	tolerant	13	monste41	0.5	7x5	monste34	0.5	bild16,17	3000
		Noise								
Linje970	backpropagation	uniform	Se linje970 under Error för dessa data.						bild18,19	3000
Linj1010	backpropagation	gaussian	13	monste41	0.5	7x5	monste31	0.5	bild18,19	3000
Linj1020	backpropagation	none	13	monste41	0.5	7x5	monste32	0.5	bild18,19	3000
		Output								
Linje970	backpropagation	direct	Se linje970 under Error för dessa data.						bild22,23	3000
Linj1030	backpropagation	one_highest	13	monste41	0.5	7x5	monste31	0.5	bild22,23	3000
Linj1040	backpropagation	two_highest	13	monste41	0.5	7x5	monste32	0.5	bild22,23	3000
Linj1050	backpropagation	one_active	13	monste41	0.5	7x5	monste33	0.5	bild22,23	3000
Linj1060	backpropagation	two_active	13	monste41	0.5	7x5	monste34	0.5	bild22,23	3000
Linj1070	backpropagation	adaline	13	monste41	0.5	7x5	monste35	0.5	bild22,23	3000
Linj1080	backpropagation	spr	13	monste41	0.5	7x5	monste36	0.5	bild22,23	3000
Linj1090	backpropagation	lvq	13	monste41	0.5	7x5	monste37	0.5	bild22,23	3000
Linj1100	backpropagation	som	13	monste41	0.5	7x5	monste38	0.5	bild22,23	3000
Linj1110	backpropagation	pnn	13	monste41	0.5	7x5	monste39	0.5	bild22,23	3000
		Learn rule								
Linj1120	backpropagation	none	13	monste41	0.5	7x5	monste31	0.5	bild20,21	3000
Linj1130	backpropagation	Hebb	13	monste41	0.5	7x5	monste32	0.5	bild20,21	3000
Linj1140	backpropagation	Hebb/anti Hebb	13	monste41	0.5	7x5	monste33	0.5	bild20,21	3000
Linj1150	backpropagation	Hopfield	13	monste41	0.5	7x5	monste34	0.5	bild20,21	3000
Linj1160	backpropagation	bsb Widrow-Hoff	13	monste41	0.5	7x5	monste35	0.5	bild20,21	3000
Linj1170	backpropagation	bsb Hebb	13	monste41	0.5	7x5	monste36	0.5	bild20,21	3000
Linj1180	backpropagation	Widrow-Hoff	13	monste41	0.5	7x5	monste37	0.5	bild20,21	3000
Linj1190	backpropagation	Kohonen	13	monste41	0.5	7x5	monste38	0.5	bild20,21	3000
Linj1200	backpropagation	Kohonen1	13	monste41	0.5	7x5	monste39	0.5	bild20,21	3000
Linj1210	backpropagation	KohonenN	13	monste41	0.5	7x5	monste40	0.5	bild20,21	3000
Linj1220	backpropagation	delta-rule	13	monste41	0.5	7x5	monste42	0.5	bild20,21	3000
Linj1230	backpropagation	adaline	13	monste41	0.5	7x5	monste43	0.5	bild20,21	3000
Linj1240	backpropagation	perceptron	13	monste41	0.5	7x5	monste44	0.5	bild20,21	3000
Linj1250	backpropagation	cum-delta-rule	13	monste41	0.5	7x5	monste45	0.5	bild20,21	3000
Linj1260	backpropagation	spr	13	monste41	0.5	7x5	monste46	0.5	bild20,21	3000
Linj1270	backpropagation	art1	13	monste41	0.5	7x5	monste47	0.5	bild20,21	3000
Linj1280	backpropagation	norm-delta-rule	13	monste41	0.5	7x5	monste48	0.5	bild20,21	3000
Linj1290	backpropagation	som	13	monste41	0.5	7x5	monste49	0.5	bild20,21	3000
Linj1300	backpropagation	lvq	13	monste41	0.5	7x5	monste50	0.5	bild20,21	3000
Linj1310	backpropagation	user Kohonen	13	monste41	0.5	7x5	monste51	0.5	bild20,21	3000
Linj1320	backpropagation	Boltzmann	13	monste41	0.5	7x5	monste52	0.5	bild20,21	3000
Linj1330	backpropagation	pnn	13	monste41	0.5	7x5	monste53	0.5	bild20,21	3000
		500 förbindelser mellan hidden och och out. Andel existerande förbindelser till resp nod i %								
Linj1340	backpropagation	20.41	70	monste41	0.5	7x5	monste31	0.5	bild27	3000
Linj1350	backpropagation	23.81	60	monste41	0.5	7x5	monste32	0.5	bild27	3000
Linj1360	backpropagation	28.57	50	monste41	0.5	7x5	monste33	0.5	bild27,28	3000
Linj1370	backpropagation	35.71	40	monste41	0.5	7x5	monste34	0.5	bild27	3000



Linj1380	backpropagation	47.62	30	monste41	0.5	7x5	monste35	0.5	bild27	3000
Linj1390	backpropagation	71.43	20	monste41	0.5	7x5	monste36	0.5	bild27	3000
Linj1391	backpropagation	14.29	100	monste41	0.5	7x5	monste37	0.5	bild27	3000
Linj1392	backpropagation	15.87	90	monste41	0.5	7x5	monste38	0.5	bild27	3000
Linj1393	backpropagation	17.86	80	monste41	0.5	7x5	monste39	0.5	bild27	3000
		250 förbindelser mellan hidden och och out. Andel existerande förbindelser till resp nod i %								
Linj1400	backpropagation	10.2	70	monste41	0.5	7x5	monste31	0.5	bild27	3000
Linj1410	backpropagation	11.9	60	monste41	0.5	7x5	monste32	0.5	bild27	3000
Linj1420	backpropagation	14.29	50	monste41	0.5	7x5	monste33	0.5	bild27,28	3000
Linj1430	backpropagation	17.86	40	monste41	0.5	7x5	monste34	0.5	bild27	3000
Linj1440	backpropagation	23.81	30	monste41	0.5	7x5	monste35	0.5	bild27	3000
Linj1450	backpropagation	35.71	20	monste41	0.5	7x5	monste36	0.5	bild27	3000
Linj1460	backpropagation	71.43	10	monste41	0.5	7x5	monste37	0.5	bild27	3000
Linj1461	backpropagation	7.14	100	monste41	0.5	7x5	monste38	0.5	bild27	3000
Linj1462	backpropagation	7.94	90	monste41	0.5	7x5	monste39	0.5	bild27	3000
Linj1463	backpropagation	8.93	80	monste41	0.5	7x5	monste40	0.5	bild27	3000
		750 förbindelser mellan hidden och och out. Andel existerande förbindelser till resp nod i %								
Linj1470	backpropagation	30.61	70	monste41	0.5	7x5	monste31	0.5	bild27	3000
Linj1480	backpropagation	35.71	60	monste41	0.5	7x5	monste32	0.5	bild27	3000
Linj1490	backpropagation	42.86	50	monste41	0.5	7x5	monste33	0.5	bild27,28	3000
Linj1500	backpropagation	53.57	40	monste41	0.5	7x5	monste34	0.5	bild27	3000
Linj1510	backpropagation	71.43	30	monste41	0.5	7x5	monste35	0.5	bild27	3000
Linj1511	backpropagation	21.43	100	monste41	0.5	7x5	monste36	0.5	bild27	3000
Linj1512	backpropagation	23.81	90	monste41	0.5	7x5	monste37	0.5	bild27	3000
Linj1513	backpropagation	26.79	80	monste41	0.5	7x5	monste38	0.5	bild27	3000
		Sannolikhet för varje förbindelse mellan hidden och out att existera.								
Linj1520	backpropagation	0.4286	50	monste41	0.5	7x5	monste31	0.5	bild28	3000
Linj1530	backpropagation	0.3571	50	monste41	0.5	7x5	monste32	0.5		3000
Linj1540	backpropagation	0.2857	50	monste41	0.5	7x5	monste33	0.5	bild28	3000
Linj1550	backpropagation	0.2143	50	monste41	0.5	7x5	monste34	0.5		3000
Linj1560	backpropagation	0.1429	50	monste41	0.5	7x5	monste35	0.5	bild28	3000
		Transferfunktion								
Linj1570	drs	sigmoid	13	monste41	0.5	7x5	monste31	0.5	bild25,26,24	115200
Linj1580	drs	tanh	13	monste41	0.5	7x5	monste32	0.5	bild24	115200
Linj1590	drs	antal gömda noder	10	monste41	0.5	7x5	monste33	0.5	bild25,26	115200
Linj1600	drs	antal gömda noder	12	monste41	0.5	7x5	monste34	0.5	bild25,26	115200
Linj1610	drs	antal gömda noder	11	monste41	0.5	7x5	monste35	0.5	bild25,26	115200
Linj1620	drs	antal gömda noder	9	monste41	0.5	7x5	monste36	0.5	bild25,26	115200
Linj1630	drs	antal gömda noder	8	monste41	0.5	7x5	monste37	0.5	bild25,26	115200
Linj1640	drs	antal gömda noder	7	monste41	0.5	7x5	monste38	0.5	bild25,26	115200
Linj1650	drs	antal gömda noder	6	monste41	0.5	7x5	monste39	0.5	bild25,26	115200
Linj1660	drs	antal gömda noder	5	monste41	0.5	7x5	monste40	0.5	bild25,26	115200
Linj1670	drs	antal gömda noder	4	monste41	0.5	7x5	monste42	0.5	bild25,26	115200
Linj1680	drs	antal gömda noder	3	monste41	0.5	7x5	monste43	0.5	bild25,26	115200
Linj1690	drs	antal gömda noder	2	monste41	0.5	7x5	monste44	0.5	bild25,26	115200
		Antal noder i								



		Kohonenlagret								
Linj1700	som-cat	20 gånger 20	400	monste41	0.5	7x5	monste35	0.5		22710
Linj1750	som-cat	12 gånger 12	144	monste41	0.5	7x5	monste31	0.5	bild29	5001
Linj1760	som-cat	8 gånger 8	64	monste41	0.5	7x5	monste32	0.5	bild29	5001
Linj1770	som-cat	6 gånger 6	36	monste41	0.5	7x5	monste33	0.5	bild29,30	5001
Linj1780	som-cat	4 gånger 4	16	monste41	0.5	7x5	monste34	0.5	bild29	5001
Linj1790	som-cat	6 gånger 6	36	monste70	0.8	7x5	monste35	0.5	bild30	5001
Linj1800	som-cat	3 gånger 3	9	monste41	0.5	7x5	monste36	0.5	bild29	5001
Linj1810	som-cat	2 gånger 2	4	monste41	0.5	7x5	monste37	0.5	bild29	5001
		Antal noder i Kohonenlagret								
Linj1900	som-cat	6 gånger 6	36	monst101	0.5	13x9	monst110	0.5	bild31	5000
Linj1900	som-cat	6 gånger 6	36	monst101	0.5	13x9	monst120	0.8		5000
Linj1910	som-cat	4 gånger 4	16	monst101	0.5	13x9	monst111	0.5	bild31	5000
Linj1910	som-cat	4 gånger 4	16	monst101	0.5	13x9	monst121	0.8		5000
Linj1920	som-cat	6 gånger 6	36	monst103	0.8	13x9	monst112	0.5		5000
Linj1920	som-cat	6 gånger 6	36	monst103	0.8	13x9	monst122	0.8	bild31	5000
Linj1930	som-cat	4 gånger 4	16	monst103	0.8	13x9	monst113	0.5		5000
Linj1930	som-cat	4 gånger 4	16	monst103	0.8	13x9	monst123	0.8	bild31,32	5000
Linj1940	som-cat	4 gånger 4	16	monst103	0.8	13x9	monst114	0.5		500000
		Error: tolerant Learnrule: cum-delta								
Linj2000	backpropagation	antal gömda noder	13	monste41	0.5	13x9	monste31	0.5		10000
Linj2010	backpropagation	antal gömda noder	10	monste41	0.5	13x9	monste32	0.5		10000
Linj2020	backpropagation	antal gömda noder	7	monste41	0.5	13x9	monste33	0.5		10000
Linj2030	backpropagation	antal gömda noder	30	monste41	0.5	13x9	monste34	0.5		10000
Linj2100	adaline	default	4	monste41	0.5	7x5		0.5		20000
Linj2200	madaline	default	13	monste41	0.5	7x5		0.5		20000
Linj2300	pnn	default	120	monst101	0.5	13x9		0.5		38000
Linj3000	backpropagation	Se linj1000	13	monste41	0.5	7x5	monste31	0.5		10000
Linj3010	backpropagation	Output: adaline Learnrule: adaline	13	monste41	0.5	7x5	monste32	0.5		100000
Linj3020	backpropagation	Se linj1250	13	monste41	0.5	7x5	monste33	0.5		100000
Linj3030	drs	Se linj1640	13	monste41	0.5	7x5	monste34	0.5		390094
Linj3031	drs	Se linj1640	13	monste41	0.5	7x5	monste35	0.5		9907200
linj3040	backpropagation		150	monst101	0.5	7x5	monst110	0.5		1000000



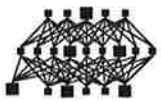
11.1.2 Indata, '.nna', '.nnb'

Filer med suffix .nna och .nnb innehåller samma information lagrat på olika sätt. Därför görs ingen åtskillnad mellan dessa typer i listan nedan.
Antal set syftar på hur många set av indata som lagrats (6.7). Med sigma menas brusets standardavvikelse och matris anger storleken på linjebilderna.

Filnamn	Antal set	Sigma	Matris
monste22	1	1.1	7x5
monste23	1	0.9	"
monste24	1	1.0	"
monste25	1	0.8	"
monste26	1	0.7	"
monste27	1	0.6	"
monste28	1	0.5	"
monste31	1	0.5	"
monste32	"	"	"
.			
.			
.			
monste40	"	"	"
monste41	8	0.5	7x5
monste42	1	0.5	7x5
.			
.			
.			
monste53	"	"	"
monste70	8	0.8	7x5
monste81	1	0.5	"
monste82	2	0.5	"
.			
.			
.			
monste89	9	0.5	"
monste90	10	0.5	"
monst100	1	0.5	13x9
monst101	8	0.5	"
monst102	1	0.8	"
monst103	8	0.8	"

11.1.3 Testresultat, '.nnr'

Resultatfilerna får samma namn som de indata från vilka de genererades. Samma indata har använts i flera körningar, varför flera resultatfiler finns med samma namn. Dessa ligger dock lagrade i olika underbibliotek.



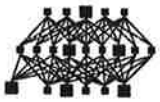
11.1.4 Instrument, '.nnp'

Instrumenten på skärmen kan loggas i filer. Dessa har i denna applikation samma namn, som de nätverksfiler där de genererades, men med suffixet '.nnp' i slutet. Filerna innehåller radvis information i tre kolumner. För att kunna laddas in i MATLAB har den inledande texten redigerats bort manuellt och därefter har kolumn två tagits bort, vilket kan göras med scriptfilen 'reduce.m' i MATLAB.

11.1.5 Skärmparametrar, display style, '.nns'

En enda skärmsättning har använts.

linstyle.nns	Svart bakgrund, vit text, nodernas utsignaler färgkodas med avtagande värden enligt: vit, beige, gul, ljusturkos, grön och mörkblå. Nodernas storlek 5 gånger 5 enheter.



11.1.6 Inlärningsscheman, Learn/Recall Schedule, '.nnt'

11.1.6.1 Back-propagation

backprp1.nnt

Learn count	1000	2000	3000	4000	5000
Coefficient1	0.9	0.4	0.1	0.05	0.02
Coefficient2	0.6	0.2	0.05	0.01	0.005

backprp2.nnt

Learn count	10000	20000	30000	40000	50000
Coefficient1	0.9	0.4	0.1	0.03	0.01
Coefficient2	0.6	0.2	0.05	0.01	0.002

backprp3.nnt

Learn count	100000	200000	300000	400000	500000
Coefficient1	0.9	0.4	0.1	0.03	0.01
Coefficient2	0.6	0.2	0.05	0.01	0.002

backprp4.nnt

Learn count	100000	200000	400000	600000	800000
Coefficient1	0.9	0.4	0.1	0.03	0.01
Coefficient2	0.6	0.2	0.05	0.01	0.002

11.1.6.2 Direct-random-search

drs1.nnt

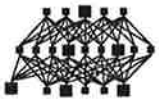
Learn count	10000	20000	50000	100000	500000
Temperature	1.0	0.5	0.25	0.031	0.0001
Coefficient1	657.7	0	0	0	0
Coefficient2	398.6	0	0	0	0

drs3.nnt

Learn count	100000	200000	500000	1000000	5000000
Temperature	0.125	0.125	0.031	0.016	0.0001
Coefficient1	191.4	125.5	77.08	59.59	0
Coefficient2	189.1	125.5	76.80	59.49	0

drs4.nnt

Learn count	2000000	4000000	6000000	8000000	9999999
Temperature	0.0078	0.078	0.0039	0.0039	0.0039
Coefficient1	21.99	13.68	9.161	6.729	5.462
Coefficient2	21.99	13.67	9.161	6.728	5.451



11.1.6.3 Probabilistic-neural-network

pnn1.nnt

Learn count	10000	20000	40000	60000	80000
Coefficient1	0.9	0.4	0.1	0.03	0.01
Coefficient2	0.6	0.2	0.05	0.01	0.002

11.1.6.4 Self-organizing-maps

som1.nnt

Learn count	500	1000	1500	2000	0
Coefficient1	0.2	0.15	0.10	0	0
Coefficient2	0.1	0.08	0.05	0	0
Coefficient3	2.0	1.50	1.00	0	0

som_cat1.nnt

Learn count	2000	4000	0	0	0
Coefficient1	0	0.15	0	0	0
Coefficient2	0	0	0	0	0

som_cat2.nnt

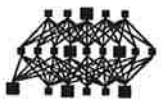
Learn count	2000	5000	0	0	0
Coefficient1	0	0.15	0	0	0
Coefficient2	0	0	0	0	0

11.1.7 Skalning av in- och utdata, '.nnm'

minma7_5.nnm	Innehåller minmaxtabell för nätverk med inmatris 7 gånger 5 element. Alla ut och insignaler skalas om till och från intervallet [0 , 1].
minm13_9.nnm	Samma som ovan men för matriser 13 gånger 9 element.

11.2 LOTUS 1-2-3 FILER

filtab	Filförteckning med vissa parametrar till körningarna.
neural	Datablad för utvärdering av resultat från körning med testdata i näten.



11.3 MATLABFILER

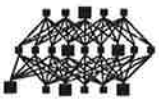
11.3.1 Scriptfiler

bildgen.m	Generering av bilder 24 gånger 24 pixels för testning i C-program.
felklass.m	Utvärdering av data från testkörningar i nätverken.
matco75.m	Generering av indatafiler till nätverk, bilder med upplösning 7 gånger 5 pixels.
matco913.m	Generering av indatafiler till nätverk, bilder med upplösning 7 gånger 5 pixels.
matexpan.m	Expanderar matriser så att varje matriselement blir en ny undermatris, där alla element har samma värde som det ursprungliga elementet.
matkonv.m	Konverterar rådata till vektorformat för att kunna användas i matco913.m, bilder med upplösning 13 gånger 9 pixels.
reduce.m	Laddar in instrumentlogfiler. Klipper bort kolumn 2 samt sparar åter filerna under sina gamla namn.

11.3.2 Datafiler

Rådatafiler innehållande data för generering av indata till nätverken. Rena bilder utan brus, bara ettor och nollor i matriser och vektorer, samt utdata.

mons9_13.mat	Fil innehållande linjebilder 13 gånger 9 pixels. Varje bild lagrad i en matris. Vänsterlutande med namn 'l1' till 'l124', högerlutande med namn 'r1' till 'r124' och en vertikal med namn v. Ingen utdatainformation.
monster.nna	Linjebilder 7 gånger 5 pixels inklusive utdata. Bilder och utdata lagrade i radvektorform, med element 1 till 35 svarande mot bildelementen och element 36 till 39 svarande mot utdata för samma bild. Alla vektorer staplade på varandra i matrisen 'monster'.
vekt9_13.mat	Konvertering av 'mons9_13.mat' med 'matkonv.m' ger alla bilder i radvektorform där dessutom utdata lagts till sist i varje vektor. Av praktiska skäl, MATLAB klarar inte matriser med fler element än 8188, har vektorerna staplats på varandra och lagrats i 9 olika matriser kallade block1 till block9, i filen 'vekt9_13.mat'.



11.4 BILDFILER

11.4.1 Bilder ritade i FREELANCE

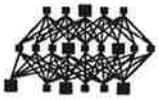
lth.cgm	Logotype för LTH.
footbild.cgm	Bild på nätverk, ingår i dokumentets footer.
figur1.cgm	Figur 1, skiss över neuralt nätverk.

11.4.2 Bilder från MATLAB

I MATLAB kan den aktuella skärmbilden sparas i en metafil med suffix '.met'. Denna görs sedan om med GPP, ett hjälpprogram till MATLAB, till en utskriftsfil i encapsulated-post-script format med suffix '.eps', vilken fritt kan användas i Lotus Manuscript. Till de flesta bilder finns även scriptfilen kvar, med vilken de ritades, samt datafilen för de aktuella körningarna. Dessa har suffixet '.m' respektive '.mat'.

Bildfiler hörande till motsvarande diagram

bild1	Diagram 5 a, b	bild2	Diagram 6 b
bild3	Diagram 6 a	bild4	Diagram 19
bild5	Diagram 20	bild6	Diagram 7 a, b
bild7	Diagram 8	bild8	Diagram 9
bild9	Diagram 10	bild10	Diagram 23
bild11	Diagram 24	bild12	Diagram 1
bild13	Diagram 2	bild14	Diagram 3
bild15	Diagram 4	bild16	Diagram 11
bild17	Diagram 12	bild18	Diagram 13
bild19	Diagram 14	bild20	Diagram 15 a, b, c
bild21	Diagram 16 a, b	bild22	Diagram 17 a, b
bild23	Diagram 18	bild24	Diagram 27
bild25	Diagram 25 a, b	bild26	Diagram 26
bild27	Diagram 21 a, b	bild28	Diagram 22
bild29	Diagram 28	bild30	Diagram 29
bild31	Diagram 30	bild32	Diagram 31



12 PROGRAMLISTNINGAR

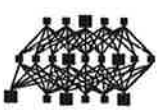
12.1 MATLAB, SCRIPTFILER

12.1.1 Bildgen.m

```
disp('')
disp('Generering av bild för demo av Flash code')
disp('=========================================')
disp('')
% Carl Henrik Petersson 910808
outfilename='bild24';
directory='c:\tc\calle\';
filetag='.dat';
% Ändra till aktuella filnamn.
load c:\nw2v40\work\monster\13_9\mons9_13.mat
rand('normal')
% Normalfördelning av slumptal.
disp('Bruset normalfördelat enligt : ')
sigma_effektiv = 0.5;
vantvarde = 0.0;
disp([' sigma_effektiv = ',num2str(sigma_effektiv)])
disp([' vantvarde = ',num2str(vantvarde)])
varians_effektiv=sigma_effektiv^2;
% Ändra gränsernas för att få andra fördelningar.
matris=[165 1117 v;147 r39 180];
matris=matris(1:24,1:24);
matris(7:19,8:16)=matris(7:19,8:16)+r43;
matris(11:23,16:24)=matris(11:23,16:24)+r56;
disp('Normering')
matris=matris./(matris+eps);
disp('Adderar bruset.')
matris=(matris+rand(matris)*varians_effektiv+vantvarde).^2;
disp(['Nya filen ',directory,outfilename,filetag,' sparas.'])
eval(['save ',directory,outfilename,filetag,' matris /ascii'])
```

12.1.2 Felklass.m

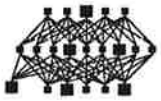
```
% Summering av resultat från NeuralWorks, .nnr-filer.
% Om variabeln ladda='ja' laddas den fil in som anges i variabeln filnamn,
% annars inte, default 'ja'. Suffixet .nnr läggs till automatiskt och skall
% alltså inte ingå i filnamnet.
% Ange path i variabel pathway, default c:\nw2v40\work\
% Nodernas beräknade värden jämförs med deras önskade värden.
% Bara fel större än ett viist tröskelvärde räknas, detta anges i
% variabeln troskel, default är 0.7
% Om variabeln count='alla' räknas alla fel, om count='ett' räknas högst ett
% fel per testexempel, default 'alla'
% Efter körning innehåller vektorn errsum antalet fel per testexempel och
% errsumsum inehåller totala antalet felklassificeringar.
disp('')
disp('')
disp('Summering av antalet felklassificering i utdata från NeuralWorks')
disp('')
if exist('filnamn')==1
        disp(['filnamn      ',filnamn])
        if exist('pathway')==0
                pathway='c:\nw2v40\work\';
        end
        disp(['path         ',pathway])
        if exist('ladda')==0,
                ladda='ja';
        end
        ladda=[ladda '      '];
        if ladda(1:2)=='ja',
                eval(['load ',pathway,filnamn,'.nnr'])
```

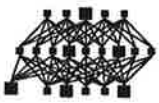
```
        eval(['matris=',filnamn,';'])
        eval(['clear ',filnamn])
    end
    [rad kol]=size(matris);
    if exist('troskel')==0
        troskel=0.7;
    end
    disp(['tröskelvärde  ',num2str(troskel)])
    if exist('count') == 0
        count='alla';
    end
    count=[count,'      '];
    disp(['count        ',count(1:4)])
    if kol == 8
        disp('7x5 pixels linjemönster')
        desired=matris(:,1:4);
        calculated=matris(:,5:8);
    elseif kol == 18
        disp('13x9 pixels linjemönster')
        desired=matris(:,1:9);
        calculated=matris(:,10:18);
    end
    abserror=(abs(desired-calculated))';
    error=max(abserror,troskel)-troskel;
    error=error./(error+eps);
    errsum=sum(error);
    % errsum innehåller antalet fel per testexempel
    if count(1:4)=='ett '
        errsum=min(errsum,1);
    end
    % errsumsum innehåller totala antalet fel
    errsumsum=sum(errsum);
    disp(['Totalt antal felklassificeringar ',int2str(errsumsum)])
    if kol == 8
        disp(['Antal fel i linjebilder        ',int2str(sum(errsum(1:rad/2)))])
        disp(['Antal fel i slumpbilder        ',int2str(sum(errsum(rad/2+1:rad)))])
    end
else
    disp('Inget filnamn och pathway')
end
disp('')
```

12.1.3 Matco75.m

```
disp('')
disp('Generering av indata till NeuralWorks Professional II/PLUS')
disp('============================================================')
disp('')
disp('Matriser 7x5')
disp('')
% Carl Henrik Petersson 910701
% ******************************************
monsterfile='monster';
outfilename='monste41';
directory='c:\nw2v40\work\';
filetag='.nna';
sigma_effektiv = 0.5;
vantvarde = 0;
%
% Ändra till aktuella filnamn och parametrar.
% ******************************************
antal_omg=8;
% Antalet olika uppsättningar med brus.
antal_raka=4;
% Mönsterfilen innehåller bara två vertikala linjemönster. För att bättre träna
% nätet på igenkänning av dessa så kan antalet vertikala linjemönster utökas
% med antal_raka stycken extra.
if exist(monsterfile) == 1,
```



```
        disp('Mönsterfilen redan inladdad.');
else
        disp(['Laddar in mönsterfilen ',directory,monsterfile,filetag])
        eval(['load ',directory,monsterfile,filetag])
        for index=1:antal_raka
                eval([monsterfile,'=[',monsterfile,'(1,:) ; ',monsterfile,'];'])
        end
end
for index=1:antal_omg
        eval(['monst_mat',int2str(index),'=',monsterfile,';'])
        eval(['rand_mat',int2str(index),'= zeros((68+antal_raka),39);'])
end
% Filen c:\nw2v40\monster.nna innehåller rena binaära indata utan brus. Varje
% bild av storlek 7x5 är lagrad som en radvektor med 35 element. Därefter kommer
% ytterligare fyra element som i tur och ordning står för linje/icke linje (1/0)
% lutning vänster/icke lutning vänster (1/0), lutning höger/ icke lutning
% höger (1/0), vertikal /icke vertikal linje (1/0). Varje bildrad består alltså
% av 39 element. De olika bilderna är sedan staplade på varandra. Hela mönster-
% filen har följaktligen storleken 68x39 element. Programmet skapar en ny
% indatafil där originalbilderna överlagras med brus med parametrar enligt
% nedan. För att få en mer varierad inlärning så upprepas hela bildsekvensen
% ett antal gånger med olika brus. Efter varje bildsekvens lagras lika många
% slumpmässiga bilder.
rand('normal')
% Normalfördelning av slumptal.
% Signalbrusförhållandet S/N=sqrt(m)/sigma, där m är antalet sample i fourier-
% spektrat och sigma är standardavvikelsen. Sätt sigma_effektiv=sigma/sqrt(m).
% Signalbrusförhållandet är nu omvänt proportionellt mot sigma_effektiv.
disp('Bruset normalfördelat enligt : ')
disp(['    sigma_effektiv = ',num2str(sigma_effektiv)])
disp(['    vantvarde      = ',num2str(vantvarde)])
varians_effektiv=sigma_effektiv^2;
% Ändra gränsernas för att få andra fördelningar.
disp('Adderar bruset.')
for index=1:antal_omg
        disp(['    omgång : ',int2str(index)])
        eval(['monst_mat',int2str(index),'(:,1:35)=(monst_mat',
                int2str(index),'(:,1:35)+rand((68+antal_raka),35)*
                varians_effektiv+vantvarde).^2;'])
        eval(['rand_mat',int2str(index),'(:,1:35)=
                (rand((68+antal_raka),35)*varians_effektiv
                +vantvarde).^2;'])
end
% % Histogram över indata
% antal_stap=50;
% disp(['Histogramindelar i ',int2str(antal_stap),'klasser.'])
% for index=1:antal_omg
%       disp(['    omgång : ',int2str(index)])
%       eval(['[nmonst',int2str(index),',xmonst',int2str(index),']=
                hist(reshape(monst_mat',int2str(index),'(:,1:35),
                (68+antal_raka)*35,1),antal_stap);'])
%       eval(['[nrand',int2str(index),',xrand',int2str(index),']=
                hist(reshape(rand_mat',int2str(index),'(:,1:35),
                (68+antal_raka)*35,1),antal_stap);'])
% end
% clg
% subplot(211), bar(xmonst1(1:antal_stap),nmonst1(1:antal_stap))
% title('Histogram över linjemönsterna med brus, sekvens 1.')
% subplot(212), bar(xrand1(1:antal_stap),nrand1(1:antal_stap))
% title('Histogram över slumpmönsterna, sekvens 1.')
alla_var=[];
for index=1:antal_omg
        alla_var = [ alla_var,' monst_mat',int2str(index),' rand_mat',int2str(index)];
end
disp(['Nya filen ',directory,outfilename,filetag,' sparas.'])
eval(['save ',directory,outfilename,filetag,alla_var,' /ascii'])
```

12.1.4 Matco913.m

```
disp('')
disp('Generering av indata till NeuralWorks Professional II/PLUS')
disp('============================================================')
disp('')
disp('Matriser 13x9')
disp('')
% Carl Henrik Petersson 910701
% ****************************************
monsterfile='vekt9_13.mat';
% Om vekt9_13.mat inte exister kan denna genereras med matkonv.m under
% förutsättning att omns9_13.mat existerar.
outfilename='monst101';
directory='c:\nw2v40\work\monster\';
filetag='.nna';
sigma_effektiv = 0.5;
vantvarde = 0;
antal_omg=1;
% Antalet olika uppsättningar med brus.
disp(['Laddar in mönsterfilen ',directory,monsterfile])
eval(['load ',directory,monsterfile])
% Filen vekt9_13.mat innehåller rena binaära indata utan brus. Varje bild av
% storleken 13x9 är lagrad som en radvektor med 117 element. Därefter kommer
% ytterligare 9 element som indikerar hur mycket linjen lutar. Endast ett
% element åt gången kan ha värdet ett. Övriga har värdet noll. Första
% elementet indikerar den starkaste lutningen åt vänster och därefter
% indikeras minskande lutningar in mot vertikala axels. Sedan följer ökande
% lutningar åt höger på samma sätt. Varje bildrad består alltså av 126
% element. De olika bilderna är sedan staplade på varandra i de olika
% variablerna block1-block9, en för vardera klassen av lutningar. Klasserna
% innehåller 29,27,30,24,29,24,30,27 respektive 29 mönster vardera.
% För att få en mer varierad inlärning så upprepas hela bildsekvensen
% ett antal gånger, varje gång med nytt brus. Före varje bildsekvens lagras
% i block0 30 slumpmässiga bilder utan linjer.
disp('Duplicerar de olika blocken i de olika seten')
for blocknr=1:9
        disp(['    block : ',int2str(blocknr)])
        for omg=1:antal_omg
                eval(['block',int2str(blocknr),int2str(omg),'=block',int2str(blocknr),';'])
                disp(['          omgång ',int2str(omg)])
        end
        eval(['clear block',int2str(blocknr)])
end
for omg=1:antal_omg
        eval(['block0',int2str(omg),'=zeros(30,(117+9));'])
end
% Mönster utan linjer.
blockstorlek=[30,29,27,30,24,29,24,30,27,29]
rand('normal')
% Normalfördelning av slumptal.
% Signalbrusförhållandet S/N=sqrt(m)/sigma, där m är antalet sample i fourier-
% spektrat och sigma är standardavvikelsen. Sätt sigma_effektiv=sigma/sqrt(m).
% Signalbrusförhållandet är nu omvänt proportionellt mot sigma_effektiv.
disp('Bruset normalfördelat enligt : ')
disp(['    sigma_effektiv = ',num2str(sigma_effektiv)])
disp(['    vantvarde      = ',num2str(vantvarde)])
varians_effektiv=sigma_effektiv^2;
% Ändra gränsernas för att få andra fördelningar.
disp('Adderar bruset.')
for blocknr=0:9
        disp(['    block : ',int2str(blocknr)])
        for omg=1:antal_omg
                disp(['          omgång : ',int2str(omg)])
                eval(['block',int2str(blocknr),int2str(omg),'(:,1:117)=
                        (block',int2str(blocknr),int2str(omg),'(:,1:117)+
                        rand(blockstorlek(blocknr+1),117)*varians_effektiv+
                        vantvarde).^2;'])
        end
end
```

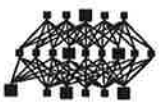
```
alla_var=[];
for blocknr=1:9
        for omg=1:antal_omg
                alla_var = [ alla_var,' block',int2str(blocknr),int2str(omg)];
        end
end
disp(['Nya filen ',directory,outfilename,filetag,' sparas.'])
eval(['save ',directory,outfilename,filetag,alla_var,' /ascii'])
```

12.1.5 Matexpan.m

```
% Givet matrisen a så fås en ny matris c där varje pixel har duplicerats
% expand_x gånger i x-led och expand_y gånger i yled. Deault : expand_x=5
% expand_y=5
% Duplicering är en fördel om matrisen skall studeras med mesh-kommandot.
[ y x ] =size(a);
if exist('expand_x')==0, expand_x=5; end
if exist('expand_y')==0, expand_y=5; end
b=[];
for kol=1:x
        disp(['Expanderar kolumn : ',int2str(kol)])
        for ant=1:expand_x
                b=[b a(:,kol)];
        end
end
c=[];
for rad=1:y
        disp(['Expanderar rad : ',int2str(rad)])
        for ant=1:expand_y
                c=[c ; b(rad,:)];
        end
end
```

12.1.6 Matkonv.m

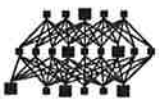
```
% Konvertering av indata.
% I filen mons9_13.mat finns matriser av storlek 13x9 sparade. Matriserna är
% lagrade i varsin variabel. Linjer med lutning åt vänster finns i variablerna
% l1-l124 och de med lutning åt höger i r1-r124. I matrisen v finns
% mönster med en vertikal linje.
% Detta program gör om matriserna till vektorform, lägger till utdata samt
% staplar vektorerna på varandra i variablerna block1-block9. Flera variabler
% måste användas p g a att största matrisstorlek annars överskrids.
% Utdata består av en vektor med 9 element där lutningarna har klassificerats
% i 9 olika klasser. Endast ett utelement åt gången kan ha värdet ett. Övriga
% har värdet noll. Första elementet indikerar den starkaste lutningen åt
% vänster. Därefter följer allt mindre lutningar i mot vertikala axeln och
% sedan ökande lutningar åt höger.
%
% Carl Henrik Petersson 910724
disp('Laddar in matriserna från mönsterfilen mons9_13.mat')
load c:\nw2v40\work\monster\mons9_13.mat
for t=1:9
        eval(['block',int2str(t),'=[];'])
end
% Innehåller de färdiga vektorerna.
disp('Gör om matriser till vektorer')
disp('')
disp('Vänsterlutande')
for t=124:-1:1
        disp(int2str(t))
        eval(['vekt=reshape(l',int2str(t),''',1,117);'])
        if t>95,
                utdata=[1 0 0 0 0 0 0 0 0];
                block1=[block1; vekt utdata];
```

```
        elseif t>68,
                utdata=[0 1 0 0 0 0 0 0 0];
                block2=[block2; vekt utdata];
        elseif t>38,
                utdata=[0 0 1 0 0 0 0 0 0];
                block3=[block3; vekt utdata];
        elseif t>14,
                utdata=[0 0 0 1 0 0 0 0 0];
                block4=[block4; vekt utdata];
        else,
                utdata=[0 0 0 0 1 0 0 0 0];
                block5=[block5; vekt utdata];
        end
        eval(['clear l',int2str(t)])
end
disp('Vertikal')
block5=[block5; reshape(v',1,117) utdata];
clear v
disp('Högerlutande')
for t=1:124
        disp(int2str(t))
        eval(['vekt=reshape(r',int2str(t),''',1,117);'])
        if t<15,
                utdata=[0 0 0 0 1 0 0 0 0];
                block5=[block5; vekt utdata];
        elseif t<39,
                utdata=[0 0 0 0 0 1 0 0 0];
                block6=[block6; vekt utdata];
        elseif t<69,
                utdata=[0 0 0 0 0 0 1 0 0];
                block7=[block7; vekt utdata];
        elseif t<96,
                utdata=[0 0 0 0 0 0 0 1 0];
                block8=[block8; vekt utdata];
        else,
                utdata=[0 0 0 0 0 0 0 0 1];
                block9=[block9; vekt utdata];
        end
        eval(['clear r',int2str(t)])
end
clear t
clear utdata
clear vekt
clear ans
disp('')
disp('Sparar block1-block9')
save c:\nw2v40\work\monster\vekt9_13.mat
```

12.1.7 Reduce.m

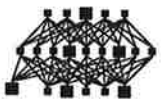
```
disp('Reduce, tar bort kolumn 2 i instrumentlogfilerna.')
disp('Bearbetar nu filen ')
% Ändra till aktuella filnummer,t , ev ändra linj till linje.
for t=1660:10:1660
        disp(['linj',int2str(t)])
        eval(['load c:\nw2v40\work\drs\linj',int2str(t),'.nnp'])
        eval(['linj',int2str(t),'=[linj',int2str(t),'(:,1) linj',int2str(t),'(:,3)];'])
        eval(['save c:\nw2v40\work\drs\linj',int2str(t),'.nnp linj',int2str(t),' /ascii'])
end
```



12.2 C-PROGRAM, KÄLLKODER

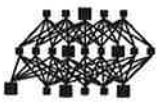
12.2.1 Huvudprogram för linjedetektering

```
// ***** LÄNKADE BIBLIOTEK ******

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <math.h>


// ***** VARIABELDEKLARATIONER ******

FILE *datafile;                 // Filpekare
char buf[80],filnamn[80];       // Teckenbuffert
int gdriver =DETECT, gmode, errorcode;//
int x,y,maxx, maxy;     // koordinater
int index;
int dec,sign;
int tal;
int indata,utdata;      // Räknare för vektorer
int entry,rad,kol;      // Övriga räknare
double dnum;
float num;
float skala[16];        // Skalvärden för gråskalning
float undre,ovre;       // Undre och övre gränser vid skalning
float diff,steg;        // Variabler i skalningen
float Inbild[24][24];   // Matrisbild från infilen
                        // 1:a index i y-led, 2:a index i x-led enligt MATLAB
float Utbild[24][24];   // Matrisbild, processad av nätverk
                        // 1:a index i y-led, 2:a index i x-led enligt MATLAB
float In[35];           // Invektor till nätverket
float Out[4];           // Utvektor från nätverket
float inmax,inmin;      // Största och minsta indatavärde
void *NetPntr;                  // Pekare vid anrop av nätverk, används ej.

char p0[8]={0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};// bit map 0
char p1[8]={0x88, 0x00, 0x00, 0x00, 0x88, 0x00, 0x00, 0x00};// bit map 1
char p2[8]={0x88, 0x00, 0x22, 0x00, 0x88, 0x00, 0x22, 0x00};// bit map 2
char p3[8]={0x88, 0x02, 0x50, 0x02, 0x88, 0x20, 0x05, 0x20};// bit map 3
char p4[8]={0xaa, 0x00, 0x55, 0x00, 0xaa, 0x00, 0x55, 0x00};// bit map 4
char p5[8]={0x20, 0xaa, 0x02, 0xd8, 0x02, 0xaa, 0x20, 0x8d};// bit map 5
char p6[8]={0x20, 0xaa, 0x02, 0xfa, 0x02, 0xaa, 0x20, 0xaf};// bit map 6
char p7[8]={0xbb, 0x44, 0x55, 0x44, 0xbb, 0x44, 0x55, 0x44};// bit map 7
char p8[8]={0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55};// bit map 8
char p9[8]={0x44, 0xbb, 0xaa, 0xbb, 0x44, 0xbb, 0xaa, 0xbb};// bit map 9
char p10[8]={0xdf, 0x55, 0xfd, 0x05, 0xfd, 0x55, 0xdf, 0x50};// bit map 10
char p11[8]={0xdf, 0x55, 0xfd, 0x27, 0xfd, 0x55, 0xdf, 0x72};// bit map 11
char p12[8]={0x55, 0xff, 0xaa, 0xff, 0x55, 0xff, 0xaa, 0xff};// bit map 12
char p13[8]={0x77, 0xfd, 0xaf, 0xfd, 0x77, 0xdf, 0xfa, 0xdf};// bit map 13
char p14[8]={0x77, 0xff, 0xdd, 0xff, 0x77, 0xff, 0xdd, 0xff};// bit map 14
char p15[8]={0x77, 0xff, 0xff, 0xff, 0x77, 0xff, 0xff, 0xff};// bit map 15
char p16[8]={0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};// bit map 16


int main(void)
{
        printf("Linjedetektering med neurala nätverk\n");
        printf("==================================\n\n\n");
        printf("Initierar ...\n\n");
        // ****** INITIERINGAR *******

        randomize();            // Initiera slumptalsgeneratorn

        // ****** INLÄSNING ******

        printf("I vilken fil är bildmatrisen lagrad ? ");
        gets(filnamn);
        printf("\n\n");
        // Öppnar datafil, felkontroll
        if ((datafile = fopen(filnamn, "r")) == NULL)
        {
```

```
        printf("Error opening text file for reading\n");
        exit(0);
    }

    indata=0;                   // Nollställ räknare, indata
    inmax=0;
    inmin=9999;
    // Läser in matrisbilden, 24 gånger 24 pixels från indatafilen
    for (y=1; y<25; y++)
    {
            for (x=1; x<25; x++)
            {       fscanf(datafile, "%s", buf);
                    Inbild[y][x]=atof(buf);
                    if (inmin>atof(buf)) inmin=atof(buf);
                    if (inmax<atof(buf)) inmax=atof(buf);
                    indata++;               // Öka med ett i räknare, indata
            }
    }

// Stänger datafilen
    fclose(datafile);

    printf("Minsta  indatavärde : %5.2f\n",inmin);
    printf("Största indatavärde : %5.2f\n",inmax);

    printf("\nSkalning i 17 intervall av gråskala för indatamatris.\n");
//  printf("Ange gränser.\n");
    printf("   Undre gräns 0");
//  gets(buf);
//  undre=atof(buf);
    undre=0;
    printf("\n   Övre gräns  1.0");
//  gets(buf);
//  ovre=atof(buf);
    ovre=1.0;
    diff=ovre-undre;
    steg=diff/15;
    for (index=0; index<16; index++)
    {
            skala[index]=undre+index*steg ;
    }


    initgraph(&gdriver, &gmode, "");        // Initiera grafiken
    errorcode=graphresult();
    if (errorcode !=grOk)                   // Felkontroll
    {
            printf("Graphics error: %s\n", grapherrormsg(errorcode));
            printf("Tryck valfri tangent för att avbryta!\n\n");
            getch();
            exit(1);                        // Avbryt
    }
    maxx=getmaxx();                         // Största x-koordinat
    maxy=getmaxy();         // Största y-oordinat
    setcolor(getmaxcolor());


    if ((datafile = fopen(filnamn, "r")) == NULL)
    {
        printf("Error opening text file for reading\n");
        exit(0);
    }
            for (x=50; x<290; x=x+10)
    {       for (y=200; y<440; y=y+10)
            {       fscanf(datafile, "%s", buf);
                    tal=(int)(17*atof(buf));
                    if (tal>16) tal=16;
                    if (tal==0) setfillpattern(p0, getmaxcolor());
                    else if (tal==1) setfillpattern(p1, getmaxcolor());
                    else if (tal==2) setfillpattern(p2, getmaxcolor());
                    else if (tal==3) setfillpattern(p3, getmaxcolor());
                    else if (tal==4) setfillpattern(p4, getmaxcolor());
                    else if (tal==5) setfillpattern(p5, getmaxcolor());
                    else if (tal==6) setfillpattern(p6, getmaxcolor());
                    else if (tal==7) setfillpattern(p7, getmaxcolor());
                    else if (tal==8) setfillpattern(p8, getmaxcolor());
                    else if (tal==9) setfillpattern(p9, getmaxcolor());
                    else if (tal==10) setfillpattern(p10, getmaxcolor());
                    else if (tal==11) setfillpattern(p11, getmaxcolor());
```

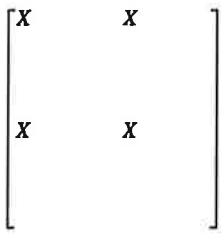
```
                        else if (tal==12) setfillpattern(p12, getmaxcolor());
                        else if (tal==13) setfillpattern(p13, getmaxcolor());
                        else if (tal==14) setfillpattern(p14, getmaxcolor());
                        else if (tal==15) setfillpattern(p15, getmaxcolor());
                        else if (tal==16) setfillpattern(p16, getmaxcolor());
                        bar(x,y,(x+8),(y+8));
                        setfillpattern(p16,getmaxcolor());
                        bar(x+310,y,x+318,y+8);
                }
        }
    // Stänger datafilen
        fclose(datafile);

        rectangle(46,196,293,443);
        rectangle(356,196,603,443);

        // sveper över första raden
        for (y=21; y>3; y--)
        {       for (x=3; x<23; x++)
                {       setcolor(getmaxcolor());
                        rectangle((x-2)*10+39,(y-3)*10+189,(x+2)*10+50,(y+3)*10+200);
                        for (rad=-3; rad<4; rad++)
                        {       for (kol=-2; kol<3; kol++)
                                {index=(rad+3)*6+kol+2;
                                In[index]=Inbild[y+rad][x+kol];
                                }
                        }
                        drs_net(NetPntr,In,Out);// Anrop av flash code rutin
                        setcolor(0);
                        setfillpattern(p0,getmaxcolor());
                        bar(300,0,600,190);
                        setcolor(getmaxcolor());
                        outtextxy(310,20,"Linje ");
                        itoa((int)(Out[0]*100),buf,10);
                        outtextxy(400,20,buf);
                        outtextxy(460,20,"/100");
                        outtextxy(310,50,"Vänster ");
                        itoa((int)(Out[1]*100),buf,10);
                        outtextxy(400,50,buf);
                        outtextxy(460,50,"/100");
                        outtextxy(310,80,"Höger ");
                        itoa((int)(Out[2]*100),buf,10);
                        outtextxy(400,80,buf);
                        outtextxy(460,80,"/100");
                        outtextxy(310,110,"Vertikal ");
                        itoa((int)(Out[3]*100),buf,10);
                        outtextxy(400,110,buf);
                        outtextxy(460,110,"/100");
                        setcolor(0);
                        if (Out[0]>0.6)
                                if (Out[1]>Out[2])
                                if (Out[1]>Out[3])
                                if (Out[1]>0.3) line(x*10+351,y*10+191,x*10+357,y*10+197);
                                else if (Out[3]>0.3) line(x*10+354,y*10+191,x*10+354,y*10+197);
                                else if (Out[2]>Out[3])
                                if (Out[2]>0.3) line(x*10+357,y*10+191,x*10+351,y*10+197);
                                else if (Out[3]>0.3) line(x*10+354,y*10+191,x*10+354,y*10+197);
                        /*
                        else if (Out[0]>0.9) rectangle(x*10+352,y*10+192,x*10+356,y*10+196);
                        else if (Out[0]>0.8) rectangle(x*10+353,y*10+193,x*10+355,y*10+195);
                        else if (Out[0]>0.7) rectangle(x*10+354,y*10+194,x*10+354,y*10+194);*/

                    //  getch();
                        rectangle((x-2)*10+39,(y-3)*10+189,(x+2)*10+50,(y+3)*10+200);
                }
        }


        setcolor(15);
        outtextxy(200,470,"Tryck valfri tangent för att avsluta.");
        getch();

        closegraph();
        return 0;
}
```

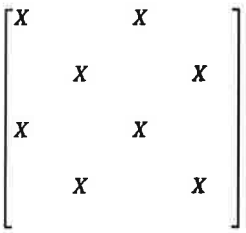
12.2.2 Bitmappade gråskalor

Olika gråskalenivåer uppbyggda av 8 gånger 8 pixels. Nedan anges hur många pixels som är ettställda och hur stor andel dessa motsvarar.

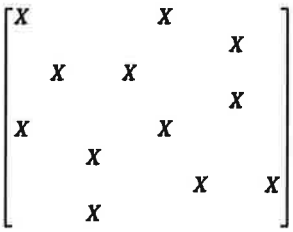
4 pixels, 6.2 %



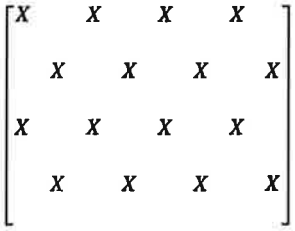
8 pixels, 12.5 %



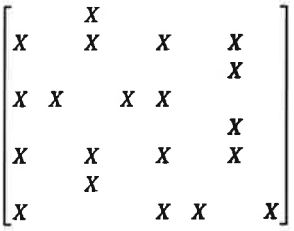
12 pixels, 18.8 %



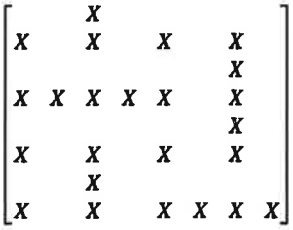
16 pixels, 25.0 %



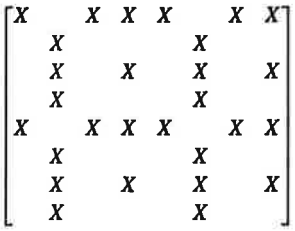
20 pixels, 31.2 %



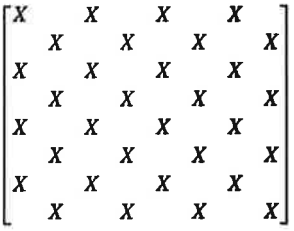
24 pixels, 37.5 %



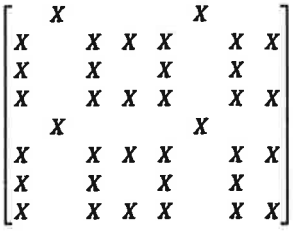
28 pixels, 43.8 %



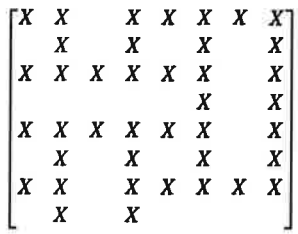
32 pixels, 50.0 %



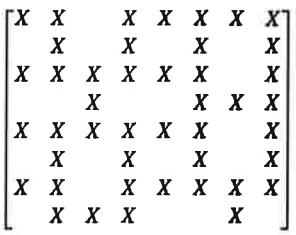
36 pixels, 56.2 %



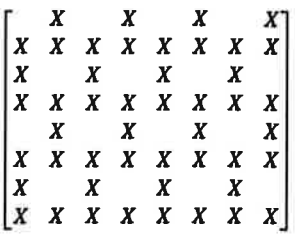
40 pixels, 62.5 %



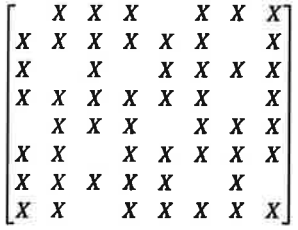
44 pixels, 68.8 %



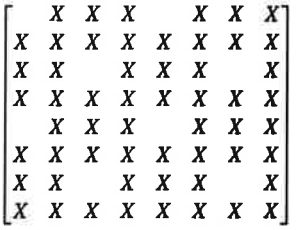
48 pixels, 75 .0 %



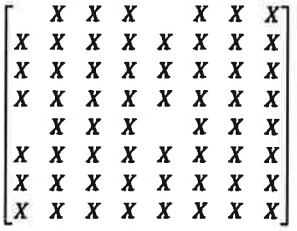
52 pixels 81.2 %



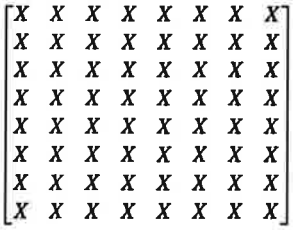
56 pixels, 87.5 %



60 pixels, 93.8 %



64 pixels, 100.0 %



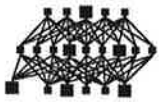
12.2.3 Flash code, kompilerade nätverk

12.2.3.1 Back-propagation, 'linj3000'

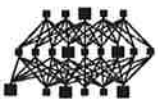
Samma utseende som för drs nätverk, andra koefficienter.

12.2.3.2 Direct-random-search, 'linj3031'

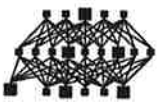
```
/* Thu Aug 01 10:19:33 1991 (drsnet.c) Recall-Only Run-time for <linj3030> */
/*    Control Strategy is: <drs> */

#if __STDC__
#define  ARGS(x) x
#else
#define  ARGS(x) ()
#endif /* __STDC__ */

/* --- External Routines --- */
extern double exp   ARGS((double));
/* *** MAKE SURE TO LINK IN YOUR COMPILER's MATH LIBRARIES *** */
```

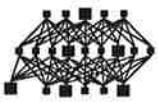
```
#if __STDC__
int drs_net( void *NetPtr, float Yin[35], float Yout[4] )
#else
int drs_net( NetPtr, Yin, Yout )
void  *NetPtr; /* Network Pointer (not used) */
float Yin[35], Yout[4];  /* Data */
#endif /* __STDC__ */
{
   float  Xout[54];  /* work arrays */
   long   ICmpT;  /* temp for comparisons */

   /* *** WARNING: Code generated assuming Recall = 0 *** */


   /* Read and scale input into network */
   Xout[2] = Yin[0] * (2) + (-1);
   Xout[3] = Yin[1] * (2) + (-1);
   Xout[4] = Yin[2] * (2) + (-1);
   Xout[5] = Yin[3] * (2) + (-1);
   Xout[6] = Yin[4] * (2) + (-1);
   Xout[7] = Yin[5] * (2) + (-1);
   Xout[8] = Yin[6] * (2) + (-1);
   Xout[9] = Yin[7] * (2) + (-1);
   Xout[10] = Yin[8] * (2) + (-1);
   Xout[11] = Yin[9] * (2) + (-1);
   Xout[12] = Yin[10] * (2) + (-1);
   Xout[13] = Yin[11] * (2) + (-1);
   Xout[14] = Yin[12] * (2) + (-1);
   Xout[15] = Yin[13] * (2) + (-1);
   Xout[16] = Yin[14] * (2) + (-1);
   Xout[17] = Yin[15] * (2) + (-1);
   Xout[18] = Yin[16] * (2) + (-1);
   Xout[19] = Yin[17] * (2) + (-1);
   Xout[20] = Yin[18] * (2) + (-1);
   Xout[21] = Yin[19] * (2) + (-1);
   Xout[22] = Yin[20] * (2) + (-1);
   Xout[23] = Yin[21] * (2) + (-1);
   Xout[24] = Yin[22] * (2) + (-1);
   Xout[25] = Yin[23] * (2) + (-1);
   Xout[26] = Yin[24] * (2) + (-1);
   Xout[27] = Yin[25] * (2) + (-1);
   Xout[28] = Yin[26] * (2) + (-1);
   Xout[29] = Yin[27] * (2) + (-1);
   Xout[30] = Yin[28] * (2) + (-1);
   Xout[31] = Yin[29] * (2) + (-1);
   Xout[32] = Yin[30] * (2) + (-1);
   Xout[33] = Yin[31] * (2) + (-1);
   Xout[34] = Yin[32] * (2) + (-1);
   Xout[35] = Yin[33] * (2) + (-1);
   Xout[36] = Yin[34] * (2) + (-1);
LAB110:

   /* Generating code for PE 0 in layer 3 */
   Xout[43]  = (float)(0.27451852) + (float)(-1.0325774) * Xout[2] +
      (float)(-0.19697522) * Xout[3] + (float)(0.50900102) * Xout[4] +
      (float)(-1.1370068) * Xout[5] + (float)(-0.031973533) * Xout[6] +
      (float)(-0.76503706) * Xout[7] + (float)(0.13116775) * Xout[8] +
      (float)(0.093453735) * Xout[9] + (float)(-1.1723057) * Xout[10] +
      (float)(0.19732265) * Xout[11] + (float)(-1.7304682) * Xout[12] +
      (float)(0.20811363) * Xout[13] + (float)(-0.33511117) * Xout[14] +
      (float)(-0.48535776) * Xout[15] + (float)(-1.6463003) * Xout[16] +
      (float)(-2.2775099) * Xout[17] + (float)(-1.9656895) * Xout[18] +
      (float)(-0.15626201) * Xout[19] + (float)(-0.64434201) * Xout[20] +
      (float)(-0.6527617) * Xout[21] + (float)(-0.11859096) * Xout[22] +
      (float)(0.81472898) * Xout[23] + (float)(2.0626132) * Xout[24] +
      (float)(0.084977001) * Xout[25] + (float)(1.7676989) * Xout[26] +
      (float)(0.79036343) * Xout[27] + (float)(0.49010882) * Xout[28] +
      (float)(-1.8093277) * Xout[29] + (float)(-1.9063441) * Xout[30] +
      (float)(-0.11449566) * Xout[31] + (float)(-0.85238123) * Xout[32] +
```



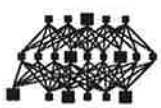
```
    (float)(0.056318238) * Xout[33] + (float)(-0.43593833) * Xout[34] +
    (float)(-1.1465225) * Xout[35] + (float)(0.21133459) * Xout[36];
Xout[43] = 1.0 / (1.0 + exp( -Xout[43] ));

/* Generating code for PE 1 in layer 3 */
Xout[44]  = (float)(-0.21153066) + (float)(0.68942827) * Xout[2] +
    (float)(0.96629548) * Xout[3] + (float)(-0.92265004) * Xout[4] +
    (float)(-0.8896026) * Xout[5] + (float)(0.9021529) * Xout[6] +
    (float)(1.406589) * Xout[7] + (float)(-0.5737831) * Xout[8] +
    (float)(-1.0168679) * Xout[9] + (float)(0.15878655) * Xout[10] +
    (float)(-1.1437176) * Xout[11] + (float)(-0.20704725) * Xout[12] +
    (float)(-0.52376866) * Xout[13] + (float)(0.59143507) * Xout[14] +
    (float)(-0.36792266) * Xout[15] + (float)(0.58934808) * Xout[16] +
    (float)(0.72440213) * Xout[17] + (float)(-0.15898579) * Xout[18] +
    (float)(1.9154999) * Xout[19] + (float)(-0.21889806) * Xout[20] +
    (float)(-2.0938334) * Xout[21] + (float)(1.9306302) * Xout[22] +
    (float)(-1.7939385) * Xout[23] + (float)(1.4104791) * Xout[24] +
    (float)(1.4985405) * Xout[25] + (float)(-1.8783424) * Xout[26] +
    (float)(-0.39218578) * Xout[27] + (float)(1.9983474) * Xout[28] +
    (float)(-1.7601295) * Xout[29] + (float)(-0.24250022) * Xout[30] +
    (float)(0.72352612) * Xout[31] + (float)(0.069152713) * Xout[32] +
    (float)(0.14830044) * Xout[33] + (float)(1.6755154) * Xout[34] +
    (float)(-0.1290445) * Xout[35] + (float)(2.2518167) * Xout[36];
Xout[44] = 1.0 / (1.0 + exp( -Xout[44] ));

/* Generating code for PE 2 in layer 3 */
Xout[45]  = (float)(-0.040369727) + (float)(0.59013522) * Xout[2] +
    (float)(0.70014048) * Xout[3] + (float)(0.38872167) * Xout[4] +
    (float)(0.20016283) * Xout[5] + (float)(-1.1530634) * Xout[6] +
    (float)(2.0681551) * Xout[7] + (float)(1.9123344) * Xout[8] +
    (float)(-0.63609278) * Xout[9] + (float)(-1.1595538) * Xout[10] +
    (float)(2.2415104) * Xout[11] + (float)(0.89492702) * Xout[12] +
    (float)(-0.55381483) * Xout[13] + (float)(-0.10281128) * Xout[14] +
    (float)(0.20503578) * Xout[15] + (float)(1.2043284) * Xout[16] +
    (float)(-1.5343574) * Xout[17] + (float)(1.663774) * Xout[18] +
    (float)(-0.89651787) * Xout[19] + (float)(1.4197764) * Xout[20] +
    (float)(-1.2588884) * Xout[21] + (float)(-1.2355982) * Xout[22] +
    (float)(-0.55635881) * Xout[23] + (float)(-0.21985552) * Xout[24] +
    (float)(-0.58891797) * Xout[25] + (float)(0.82604796) * Xout[26] +
    (float)(0.29342911) * Xout[27] + (float)(0.048363112) * Xout[28] +
    (float)(-0.18655829) * Xout[29] + (float)(-0.9350884) * Xout[30] +
    (float)(1.7539024) * Xout[31] + (float)(-0.28717339) * Xout[32] +
    (float)(-0.20624517) * Xout[33] + (float)(0.17664143) * Xout[34] +
    (float)(-0.69197637) * Xout[35] + (float)(0.67858797) * Xout[36];
Xout[45] = 1.0 / (1.0 + exp( -Xout[45] ));

/* Generating code for PE 3 in layer 3 */
Xout[46]  = (float)(-1.4891977) + (float)(-1.1506571) * Xout[2] +
    (float)(0.045416266) * Xout[3] + (float)(0.30334508) * Xout[4] +
    (float)(-0.91020894) * Xout[5] + (float)(-0.22334906) * Xout[6] +
    (float)(2.7069979) * Xout[7] + (float)(0.046636924) * Xout[8] +
    (float)(-0.39211845) * Xout[9] + (float)(-1.4231176) * Xout[10] +
    (float)(-0.88025045) * Xout[11] + (float)(0.37868476) * Xout[12] +
    (float)(-0.57890046) * Xout[13] + (float)(-0.029082265) * Xout[14] +
    (float)(-0.30225104) * Xout[15] + (float)(1.4195441) * Xout[16] +
    (float)(0.27470285) * Xout[17] + (float)(-0.023501217) * Xout[18] +
    (float)(-1.0635197) * Xout[19] + (float)(0.86326355) * Xout[20] +
    (float)(-1.1244451) * Xout[21] + (float)(1.0667434) * Xout[22] +
    (float)(-1.0771892) * Xout[23] + (float)(-1.1272597) * Xout[24] +
    (float)(1.6872387) * Xout[25] + (float)(-1.4148906) * Xout[26] +
    (float)(0.35637692) * Xout[27] + (float)(-0.35444906) * Xout[28] +
    (float)(-0.17417821) * Xout[29] + (float)(-1.2337673) * Xout[30] +
    (float)(-1.4781713) * Xout[31] + (float)(1.9974136) * Xout[32] +
    (float)(-0.84048587) * Xout[33] + (float)(-0.98318458) * Xout[34] +
    (float)(-1.5309391) * Xout[35] + (float)(0.51366878) * Xout[36];
Xout[46] = 1.0 / (1.0 + exp( -Xout[46] ));
```

```
/* Generating code for PE 4 in layer 3 */
Xout[47]  = (float)(0.46677145) + (float)(-0.58192241) * Xout[2] +
   (float)(0.47492024) * Xout[3] + (float)(0.56743908) * Xout[4] +
   (float)(0.84225667) * Xout[5] + (float)(2.2559001) * Xout[6] +
   (float)(-0.11626125) * Xout[7] + (float)(-1.0208485) * Xout[8] +
   (float)(-1.6344739) * Xout[9] + (float)(1.3225497) * Xout[10] +
   (float)(-1.1989088) * Xout[11] + (float)(1.1882415) * Xout[12] +
   (float)(-1.0353546) * Xout[13] + (float)(0.89946944) * Xout[14] +
   (float)(0.14920375) * Xout[15] + (float)(-0.8010993) * Xout[16] +
   (float)(-0.26320872) * Xout[17] + (float)(1.5577346) * Xout[18] +
   (float)(-0.11456562) * Xout[19] + (float)(-0.51345372) * Xout[20] +
   (float)(0.80046159) * Xout[21] + (float)(-0.29427886) * Xout[22] +
   (float)(0.36821768) * Xout[23] + (float)(0.130408) * Xout[24] +
   (float)(-1.1756277) * Xout[25] + (float)(-2.1521854) * Xout[26] +
   (float)(0.96327341) * Xout[27] + (float)(1.4800159) * Xout[28] +
   (float)(-0.62535906) * Xout[29] + (float)(-0.98009551) * Xout[30] +
   (float)(-0.39463434) * Xout[31] + (float)(1.1706688) * Xout[32] +
   (float)(-0.15659018) * Xout[33] + (float)(0.17125906) * Xout[34] +
   (float)(-1.5516598) * Xout[35] + (float)(-0.055400595) * Xout[36];
Xout[47] = 1.0 / (1.0 + exp( -Xout[47] ));

/* Generating code for PE 5 in layer 3 */
Xout[48]  = (float)(-0.8981325) + (float)(-0.80971014) * Xout[2] +
   (float)(-0.025689228) * Xout[3] + (float)(1.2135017) * Xout[4] +
   (float)(-0.094259143) * Xout[5] + (float)(-0.091819838) * Xout[6] +
   (float)(2.1717095) * Xout[7] + (float)(0.76560009) * Xout[8] +
   (float)(-1.1653053) * Xout[9] + (float)(-2.1544285) * Xout[10] +
   (float)(2.0479932) * Xout[11] + (float)(-0.24130802) * Xout[12] +
   (float)(0.7441594) * Xout[13] + (float)(1.4997684) * Xout[14] +
   (float)(-2.204608) * Xout[15] + (float)(0.36481982) * Xout[16] +
   (float)(-0.48955458) * Xout[17] + (float)(-0.90831828) * Xout[18] +
   (float)(1.37491) * Xout[19] + (float)(-0.11539096) * Xout[20] +
   (float)(-0.80938101) * Xout[21] + (float)(-0.91504842) * Xout[22] +
   (float)(1.1707996) * Xout[23] + (float)(0.25000137) * Xout[24] +
   (float)(-0.37640205) * Xout[25] + (float)(0.050036196) * Xout[26] +
   (float)(1.1150255) * Xout[27] + (float)(0.75421321) * Xout[28] +
   (float)(-0.42594266) * Xout[29] + (float)(-0.025824867) * Xout[30] +
   (float)(-0.19917992) * Xout[31] + (float)(1.1789471) * Xout[32] +
   (float)(-1.4571849) * Xout[33] + (float)(0.4950479) * Xout[34] +
   (float)(-0.8092916) * Xout[35] + (float)(0.44383147) * Xout[36];
Xout[48] = 1.0 / (1.0 + exp( -Xout[48] ));

/* Generating code for PE 6 in layer 3 */
Xout[49]  = (float)(-0.93634671) + (float)(2.4207404) * Xout[2] +
   (float)(2.585629) * Xout[3] + (float)(-0.2054292) * Xout[4] +
   (float)(-0.88111091) * Xout[5] + (float)(-0.4049044) * Xout[6] +
   (float)(0.3244774) * Xout[7] + (float)(-0.21303822) * Xout[8] +
   (float)(0.10355018) * Xout[9] + (float)(-0.58534533) * Xout[10] +
   (float)(-2.0782988) * Xout[11] + (float)(-1.3681841) * Xout[12] +
   (float)(-0.71155488) * Xout[13] + (float)(-1.0835922) * Xout[14] +
   (float)(-0.54116344) * Xout[15] + (float)(-1.2031281) * Xout[16] +
   (float)(1.2164038) * Xout[17] + (float)(0.077069335) * Xout[18] +
   (float)(-0.58618122) * Xout[19] + (float)(-0.55768722) * Xout[20] +
   (float)(-0.35150003) * Xout[21] + (float)(1.5010203) * Xout[22] +
   (float)(-0.42301014) * Xout[23] + (float)(0.5924269) * Xout[24] +
   (float)(-1.4159428) * Xout[25] + (float)(1.7623982) * Xout[26] +
   (float)(-0.8206197) * Xout[27] + (float)(0.82396966) * Xout[28] +
   (float)(-0.35673562) * Xout[29] + (float)(0.37944558) * Xout[30] +
   (float)(-0.15961653) * Xout[31] + (float)(0.97323024) * Xout[32] +
   (float)(1.9218152) * Xout[33] + (float)(0.98502499) * Xout[34] +
   (float)(0.95550334) * Xout[35] + (float)(0.32358116) * Xout[36];
Xout[49] = 1.0 / (1.0 + exp( -Xout[49] ));

/* Generating code for PE 0 in layer 4 */
Xout[50]  = (float)(-1.8025522) + (float)(2.1436057) * Xout[43] +
   (float)(0.11472957) * Xout[44] + (float)(0.11265014) * Xout[45] +
   (float)(-1.7393559) * Xout[46] + (float)(-0.013900596) * Xout[47] +
   (float)(2.3177147) * Xout[48] + (float)(1.0026133) * Xout[49];
Xout[50] = 1.0 / (1.0 + exp( -Xout[50] ));
```



```
  /* Generating code for PE 1 in layer 4 */
  Xout[51]  = (float)(0.70503575) + (float)(-0.72114944) * Xout[43] +
     (float)(0.61798757) * Xout[44] + (float)(-0.55950952) * Xout[45] +
     (float)(-0.11997398) * Xout[46] + (float)(-1.9910197) * Xout[47] +
     (float)(0.52046871) * Xout[48] + (float)(0.56334198) * Xout[49];
  Xout[51] = 1.0 / (1.0 + exp( -Xout[51] ));

  /* Generating code for PE 2 in layer 4 */
  Xout[52]  = (float)(-0.81822854) + (float)(-0.027860815) * Xout[43] +
     (float)(0.43857783) * Xout[44] + (float)(-1.1541589) * Xout[45] +
     (float)(-0.89110529) * Xout[46] + (float)(1.326779) * Xout[47] +
     (float)(0.13184731) * Xout[48] + (float)(-0.1802253) * Xout[49];
  Xout[52] = 1.0 / (1.0 + exp( -Xout[52] ));

  /* Generating code for PE 3 in layer 4 */
  Xout[53]  = (float)(0.27345273) + (float)(-0.66505527) * Xout[43] +
     (float)(0.22771275) * Xout[44] + (float)(0.64313936) * Xout[45] +
     (float)(-0.44254383) * Xout[46] + (float)(-0.5173713) * Xout[47] +
     (float)(-0.59487462) * Xout[48] + (float)(-0.75232923) * Xout[49];
  Xout[53] = 1.0 / (1.0 + exp( -Xout[53] ));

  /* De-scale and write output from network */
  Yout[0] = Xout[50] * (1.6666666) + (-0.33333333);
  Yout[1] = Xout[51] * (1.6666666) + (-0.33333333);
  Yout[2] = Xout[52] * (1.6666666) + (-0.33333333);
  Yout[3] = Xout[53] * (1.6666666) + (-0.33333333);
  return( 0 );
}
```

13 KOMPILERADE NÄTVERK, PROGRAM FÖR LINJEDETEKTERING

Då de olika nätverken är färdigtränade, kan de kompileras till källkod i C. Nätverken blir där funktioner, vilka kan anropas från omgivande program eller från andra funktioner. I och med att de kompilerade nätverken har samma gränssnitt, kan de fritt bytas ut mot varandra. Ett kort program har skrivits för att praktiskt demonstrera hur neurala nätverk kan användas för linjedetektering. Då man vill testa ett nytt nätverk, behöver bara include-direktivet för den aktuella nätverksfilen ändras och därefter programmet åter kompileras, till exekverbar kod. Eftersom C är ett för mig nytt språk, har jag inte studerat manualer mer än nödvändigt, för att kunna skriva ett fungerande program. Programmet har inga menyval eller liknande, utan efter varje körning måste det startas om, med eventuellt nya parametervärden.

I detta program laddas först data från en fil, innehållande en matrisbild bestående av 24 gånger 24 element. Därefter ritar programmet upp en gråskalebild av denna matris på vänstra halvan av skärmen. Bilderna, som är tänkta att innehålla vertikala och svagt lutande linjer, scannas horisontellt, genom att ett mindre fönster, av storlek 7 gånger 5 pixels, förs över bilden. För varje position anropas nätverksfunktionen. Om denna känner igen någon linje fås en utsignal på nod 1 i utlagret. Denna nod anses vara aktiverad då utsignalen överskrider ett visst tröskelvärde. Då det indikerats att linje påträffats, analyseras signalerna på de tre övriga utnoderna. Eftersom signalerna inte är lika lättolkade som de, på vilka nätet tränats, måste en viss analys utföras. Om klassificeringen är riktig, skall bara en av dessa tre noder ge utsignal. Nätet som är tränat för att ge utsignalerna noll och ett, ger i själva verket signaler inom ett mindre intervall. Genom att statistiskt undersöka fördelningen av utdata, kan lämpliga tröskelvärden för aktiverade noder bestämmas.

Då det är klart att en linje har påträffats, och riktningen på denna är känd, antingen vertikal eller lutande åt vänster eller höger, ritas motsvarande linje i högra skärmhalvan. Scanningen börjar vid tredje rutan räknat från vänster, i fjärde raden underifrån, och fortsätter radvis uppåt. Närmare kanten uppstår problemet att bildelement saknas. Dessa kan extrapoleras, men i denna tillämpning har inte denna utvidgning genomförts.

Allt eftersom scanningen pågår, ritas i högra skärmhalvan, tecken för vertikala eller lutande linjer, på de ställen i bilden där linjer upptäckts. Från början är höger skärmhalva tom, men fylls på med information under tiden som bilden scannas av.

Källkoderna till såväl program som kompilerade nätverk återfinns i (12.2).