ISSN 0280–5316
ISRN LUTFD2/TFRT--5514--SE

Neurala nätverk i SattLine

Fredrik Bistedt

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Augusti 1994

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

Document name
MASTER THESIS

Date of issue
August 1994

Document Number
ISRN LUTFD2/TFRT--5514--SE

Author(s)
Fredrik Bistedt

Supervisor
Rolf Johansson, Lars Pernebo and Ulf Hagberg

Sponsoring organisation

Title and subtitle
Neurala nätverk i SattLine. (Neural Networks in SattLine).

Abstract

The purpose with this paper is to examine the possibilities to implement a support for neural networks in SattControl's graphical process control system SattLine. Though a literature study in the subject neural networks, with focus on applications with a control engineering, one may draw the conclusion that this is possible. A suitable algorithm for updating the network is the backpropagation algorithm. The tank process has been used in all simulations, identification and control experiment. It shows that the implemented support will act without remarks in identification of the process. However, during control of the level in the tank, the network will converge but after very long time.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
Swedish

Number of pages
46

Recipient's notes

Security classification

Innehåll

1 Introduktion

Syftet med detta examensarbete är att utröna möjligheterna att implementera stöd för neurala nätverk i SattControls grafiska processtyrsystem SattLine. Om så är fallet, dvs möjligheten finns, även implementera ett sådant stöd. Nätverken ska användas till största delen för reglering men även identifiering.

Anledningen till undersökningen är att teorier och praktiska användningar börjar ta fart och industrin ser en marknad för dylika tillämpningar. Poängen med att använda neurala nätverk i reglersammanhang är att man inte behöver ha så goda processkunskaper utan nätverket lär sig själv att uppträda på ett av användaren specificerat sätt. Tankarna går osökt över till adaptiv reglering. Teorin för neurala nätverk är å ena sidan en form av adaptiv reglering men å andra sidan behöver man inte den gedigna processkunskap som man måste ha för att få en bra reglering i det adaptiva fallet.

Ytterligare en fördel till de neurala näten är att de tar upp olinjäriteter på ett mycket effektivt sätt, näten i sig är olinjära vilket visas i kommande kapitel. Olinjäriteter finns i alla processer, och vad vore mer naturligt är att regulatorerna också är olinjära. Genom att använda neurala nätverk som regulatorer erhåller man regulatorer med olinjäriteter.

Arbetet i detta examensarbete är disponerat på följande sätt:

- Till att börja med har en studie gjorts över vad neurala nätverk egentligen innebär och vad de kan åstadkomma.
- Därefter studerades litteratur angående reglerproblem, där man använt neurala nätverk.
- En implementation för stöd till neurala nätverk har gjorts i SattLine
- Simuleringar i SattLine, både vad det gäller identifiering och reglering. Vad det avser reglering är det önskvärt att hitta en regulatorkonfiguration så att man inte behöver några processkunskaper, dvs regulatorn ska vara fullständigt självinställande.
- Arbetet avslutas med en utvärdering om tillämpningens förmåga och då speciellt SattLine men även i det stora hela.

De första kapitlen i denna rapport är av teoretisk karaktär och behandlar teorin för neurala nätverk och speciellt backpropagation behandlas i kapitel *2 Neurala nätverk*. Identifiering och reglering med hjälp av neurala nätverk behandlas i kapitel *3 Identifiering* respektive kapitel *4 Reglering* . Dessa kapitel kan ses som en sammanfattning av gjorda litteratursökning i ämnet.

Hur implementationen har gjorts redovisas i kapitel *5 Implementering*.

Resultaten och slutsatser från egna simuleringar återfinns i kapitel *6 Resultat* respektive kapitel *7 Slutsatser*..

Rapporten avslutas med en sammanfattning över examensarbetet, kapitel *8 Sammanfattning.*

2 Neurala Nätverk

I detta kapitel ges en introduktion till begreppet neurala nätverk. Den vanligaste förekommande nätverkstypen - Backpropagation - kommer ingående att beskrivas. Kapitlet avslutas med en diskussion om hur ett nätverk ska konfigureras, dvs hur många lager ska man ha, hur många noder per lager osv.

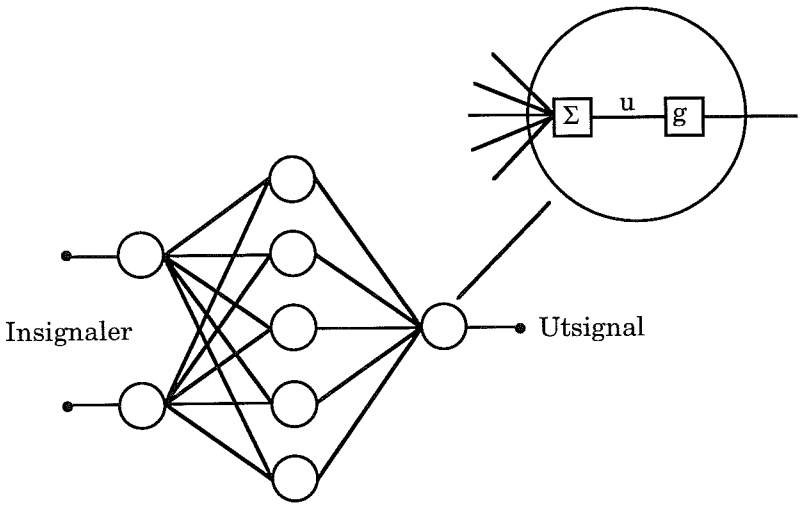
2.1 Inledning

Begreppet neurala nätverk börjar bli allt mer använt inom datavärlden men även i reglertekniska sammanhang. Finessen med att använda denna typ av nätverk är att de kan lära sig speciella beteendemönster. Man har försökt att kopiera den mänskliga hjärnan i dess allra enklaste form till ett verktyg i datorsammanhang.

Näten är som senare kommer att visas inte särdeles snabba i att lära sig saker och ting, samtidigt som näten oftast måste vara relativt stora för att de ska uppträda på önskat sett. Detta leder till att beräkningskapaciteten måste vara stor. Finns inte denna kapacitet i form av datorer, konstruerar man speciella högprestandakretsar som klarar uppgiften.

2.2 Det konventionella nätverket

Ett neuralt nätverk är uppbyggt av noder och förbindelser, se figur 2.1. Utsignalen från en nod är summan av insignalerna multiplicerat med en överföringsfunktion[1], *g(u)*. Anledningen till man har en överföringsfunktion är att man vill begränsa utsignalen från en viss nod. Vanliga funktionen är sigmoidfunktioner, och stegfunktioner, men man använder även linjära funktioner, oftast då på utnoder. De linjära noderna har ingen begränsande funktion



Figur 2.1 Ett trelagers nät samt förstoring av nod

[1]Benämningen överföringsfunktion ska tolkas som olinjärt funktionselement. Denna betydelse kommer att användas genomgående i den fortsatta texten.

Noderna i ett nätverk är ordnade i lager som i figur 2.1. Man talar om in- och utlager samt dolda lager. Nätets insignaler respektive utsignaler är kopplade till in- och utlager respektive .

Det finns delade meningar om hur man ska räkna antal lager i ett nätverk. Vissa förespråkar att ingångarnas terminallager ska räknas och vissa inte. Fortsättningsvis kommer den senare nomenklaturen att användas, dvs nätet i figur 2.1 har tre lager.

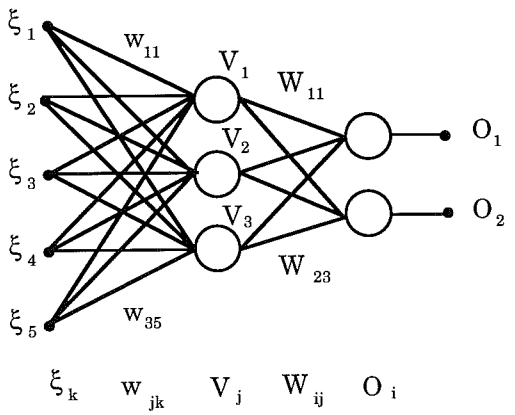
För att näten ska kunna lära sig någonting måste något kunna förändras i nätet. För detta ändamål introduceras härmed begreppet vikt. Varje nod-nod förbindelse har en associerad vikt som uppdateras efter vissa speciella regler. Dessa vikter benämns med w_{jk}. Indexet betecknar förbindelsen från nod k till nod j. Detta betyder att varje insignal till en nod, i_k multipliceras med respektive vikt, därefter adderas signalerna, och till slut via överföringsfunktionen får man utsignalen, O_j eller:

$$O_j = g(h_j) = g(\sum_k w_{jk} i_k) \tag{2.1}$$

2.3 Backpropagation

Backpropagation är den nätverkstyp som är mest använd. Mycket på grund av dess enkelhet samt att den är lätt att implementera. Namnet kommer av att vikterna uppdateras baklänges, dvs sista lagret uppdateras först, och sist uppdateras första lagret. Uppdateringen av vikterna är en vanlig minstakvadratfelsmetod.

Följande analys kommer att hänföra sig till ett nätverk med två lager., se figur 2.2. För att skilja på vilket mönster som läggs på ingångarna införes ett överindex μ. Ingång k antar värdet ξ_k^μ då mönstret μ lagts på terminalerna.



Figur 2.2. Två lagers nät med beteckningar enligt härledning

Givet mönstret μ, till dold nod j kommer följande signal

$$h_j^\mu = \sum_k w_{jk} \xi_k^\mu . \tag{2.2}$$

Denna nod producerar utsignalen

$$V_j^\mu = g(h_j^\mu) = g(\sum_k w_{jk}\xi_k^\mu). \tag{2.3}$$

Utnod i mottar utsignalerna från lager ovanför

$$h_i^\mu = \sum_j W_{ij}V_j^\mu = \sum_j W_{ij}g(\sum_k w_{jk}\xi_k^\mu) \tag{2.4}$$

och producerar den slutliga utsignalen

$$O_i^\mu = g(h_i^\mu) = g(\sum_j W_{ij}V_j^\mu) = g(\sum_j W_{ij}g(\sum_k w_{jk}\xi_k^\mu)). \tag{2.5}$$

Måttet på hur bra nätet definieras som kvadratiska differensen mellan önskad utsignal ζ och utsignalen O från nätet. Summa tas över alla tänkbara mönster vilket ger en förlustfunktion E.

$$E = \tfrac{1}{2}\sum_{\mu,i}(\zeta_i^\mu - O_i^\mu)^2 = \tfrac{1}{2}\sum_{\mu,i}[\zeta_i^\mu - g(\sum_j W_{ij}g(\sum_k w_{jk}\xi_k^\mu))]^2. \tag{2.6}$$

Denna funktion är kontinuerligt differentierbar, och därför kan en gradientalgoritm användas för att uppdatera vikterna, så att det kvadratiska felet når ett minimum.

Gradientmetodens algoritm ger följande uppdateringsregler för vikterna tillhörande förbindelserna mellan in- och utlager

$$\Delta W_{ij} = -\eta\frac{\partial E}{\partial W_{ij}} = \eta\sum_\mu[\zeta_i^\mu - O_i^\mu]g'(h_i^\mu)V_j^\mu = \eta\sum_\mu \delta_i^\mu V_i^\mu \tag{2.7}$$

där vi har definierat

$$\delta_i^\mu = g'(h_i^\mu)[\zeta_i^\mu - O_i^\mu]. \tag{2.8}$$

Parametern η betecknar steglängden.

För uppdaterings regeln av vikterna mellan signalingångar och första lagrets noder, Δw_{jk}, härleds på samma sätt som ovan. Härledningen är lite mer komplicerad men användning av kedjeregeln vid derivering kommer man fram till följande resultat:

$$\begin{aligned}\Delta w_{ij} &= -\eta\frac{\partial E}{\partial w_{ij}} = -\eta\sum_\mu \frac{\partial E}{\partial V_j^\mu}\frac{\partial V_j^\mu}{w_{ij}} \\ &= \eta\sum_{\mu,i}[\zeta_i^\mu - O_i^\mu]g'(h_i^\mu)W_{ij}g'(h_j^\mu)\xi_j^\mu \\ &= \eta\sum_{\mu,i}\delta_i^\mu W_{ij}g'(h_j^\mu)\xi_k^\mu = \eta\sum_\mu \delta_j^\mu\xi_k^\mu\end{aligned} \tag{2.9}$$

där

$$\delta_j^\mu = g'(h_j^\mu)\sum_i W_{ij}\delta_i^\mu \tag{2.10}$$

Notera att (2.9) har samma form som (2.7), men med olika definitioner av δ. Uppdateringsreglen för ett generellt nät med godtyckligt antal lager kan alltid skivas på formen

$$\Delta w_{pq} = \eta\sum_{mönster}\delta_{utsignal}\cdot V_{insignal} \tag{2.11}$$

där indexen *utsignal* och *insignal* refererar till det aktuella lagrets noders in- och utgångar. Det är nu lätt att definiera en uppdateringsalgoritm för alla vikter i ett nätverk med godtyckligt antal lager, se bilaga A.

Ett bra mått på hur bra nätverket har justerat sina vikter är naturligtvis förlustfunktionen E. I de fall man tränar nätverket i realtid har man inga förutbestämda mönster, dvs man kan inte använda formel (2.6). Istället kan man summera felen kvadratiskt över en tidsperiod T vilket ger

$$E[w(k)] = \frac{1}{T} \sum_{j,i=k-T+1}^{k} [\zeta_j(i) - O_j(i)]^2 \tag{2.12}$$

där k betecknar aktuell tid eller sampel.

2.4 Överföringsfunktioner

Valet av överföringsfunktion spelar en central roll i konstruerandet av ett nätverk. De överföringsfunktioner som används är av sigmoidtyp eller linjära. Sigmoidfunktionens uppgift är att begränsa signalen samt att normera densamma. Nätverket kan fortare uppdatera vikterna till korrekta värden om signalnivåerna ligger inom ett begränsat område 0/1 eller ±1. Oftast då man arbetar med kontinuerliga signaler använder man linjära överföringsfunktioner hos utnoderna.

Sigmoidfunktionerna som använd är logistisk sigmoidfunktion samt tangenshyperbolicus,

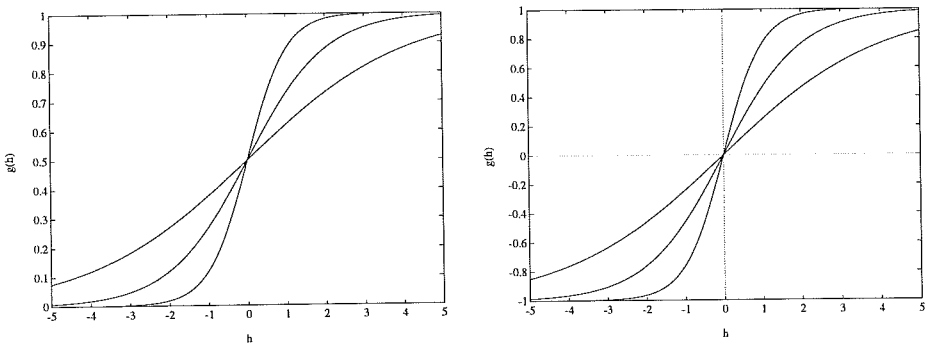
$$g(h) = \frac{1}{1 + e^{-2\beta h}} \tag{2.13}$$

och

$$g(h) = \tanh(\beta h) \tag{2.14}$$

Funktionerna (2.12) och (2.13) ligger i området 0/1 respektive ±1. Funktionerna är symmetriska och parametern β anger maximal lutning. Denna parameter sätts oftast till 1 eller 1/2. Figur 2.3 ger en grafisk presentation av sigmoidfunktionerna vid olika värden på parametern β.

Derivatan av funktionerna (2.12) och (2.13) kan uttryckas i termer av respektive funktion själv som $g'(h) = 2\beta g(1-g)$ respektive $g'(h) = \beta(1-g^2)$.



Figur 2.3 Grafisk representation av funktionerna (2.12), (vänstra delen) och (2.13), (högra delen). Graferna visar utseendet då β=1.0 , 0.5, 0.25.

2.5 Variationer av Backpropagation

Nedan följer en redovisning av variationer av backpropagation. Först introduceras begreppet *bias*, därefter *momentterm* och till sist *adaptiv steglängd.*

Bias

Då man arbetar med kontinuerliga signaler kan det hända att signalnivåerna har en förspänning, bias. Nätverket måste naturligtvis klara av även detta problem så därför införes en biasterm, B, i varje nod.

Utsignalen från nod j i lager m blir numera

$$V_j^m = g(h_j^m + B_j^m) = g([\sum_k w_{jk}^m V_k^{m-1}] + B_j^m), \tag{2.15}$$

där överindexet betecknar lagret i fråga. Beteckningarna i övrigt skiljer sig inte från tidigare härledning.

Naturligtvis måste även biastermen uppdateras. Detta görs på samma sätt som vid uppdateringen av vikterna, dvs man använder sig av en gradient metod enligt följande:

$$\Delta B_j^m = -\eta \frac{\partial E}{\partial B_j^m} = \eta \delta_j^m, \tag{2.16}$$

där δ_i^m betecknat det bakåtpropagerade felet enligt bilaga A.

Algoritmen i Bilaga A kan nu utökas till uppdatering av biastermen. Tillfoga (2.15) till formel (A.5) och beräkna den uppdaterade biastermen på samma sätt som (A.6)

Momentterm

I vissa fall då man vill minimera en funktion med flera minima med hjälp av gradientmetoden, kan det hända att man inte slutligen hamnar i det globala minimat utan i ett lokalt. Samma problem föreligger vid inlärningsprocessen av ett neuralt nätverk. Förlustfunktionen ligger på en konstant nivå som inte är globalt utan endast lokalt minimal. För att komma till rätta med detta problem inför man en momentterm, α, som har till uppgift att se till så att minimeringen inte hamnar i ett lokalt minimum.

$$\Delta w_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \alpha \Delta w_{pq}(t) \tag{2.17}$$

Idén bakom momenttermen är att ge en liten extra kick om det dyker upp ett lokalt minima. Minimeringen fortsätter förhoppningsvis om kicken varit tillräckligt stor.så man kommer förbi det lokala minimat.

Vikterna uppdateras enligt (2.17). Den första delen av formeln är traditionell, medan den andra delan gör så att vikterna uppdateras lite extra vid ett minimum, lokalt som globalt. Är momenttermen α tillräckligt stor kommer minimeringen att fortsätta föbi det lokala minimumet. Detsamma gäller vid ett globalt minimum, med kommer sedermera att konvergera i den globala minpunkten.

Momenttermen måste ligga i intervallet 0 till 1, och ofta sätts α till 0.9.

Adaption av steglängden

Steglängden är en mycket viktig parameter. Sätts inte den till ett lämpligt värde kan följande inträffa:

- Då steglängden är mycket liten kommer man att få mycket långa konvergeringstider
- Då steglängden är för stor kommer man inte att konvergera till ett globalt minimum utan hela tiden oscillera fram och tillbaka kring minpunkten.
- Är steglängden för stor kan systemet, nätverket, bli instabilt.

Ett sätt att förenkla inställandet av steglängdsparametern η, är att införa en adaptiv algoritm. Man studerar förlustfunktionen E, och om denna minskar till sitt värde har man tagit ett steg i rätt riktning, dvs mot minpunkten. I detta fall kan man öka steglängden något till nästa minimering. Om det förhåller sig så att förlustfunktionen ökar har man tagit ett steg i fel riktning, och η måste minskas om man vill konvergera till ett korrekt minimum. Ovanstående kan uttryckas matematiskt enligt formel (2.18).

$$\eta(t+1)=\begin{cases}\eta(t)a & \text{då} \quad \Delta E<0\\ \eta(t)b & \text{då} \quad \Delta E>0.\\ \eta(t) & \text{för} \quad \text{övrigt}\end{cases} \tag{2.18}$$

ΔE betecknar förändringen i förlustfunktionen. Adaptionsparametrarna a och b anger den procevtuella förändrinen av steglängden, där $a>1$ och $b<1$. Ofta anger b en kraftigare förändring av steglängden än a, eftersom det är väsentligt att snabbt komma tillbaka i en negativ gradientriktning.

För de tillämpningar och simuleringar som de neurala nätverken kommer att användas i, finns det ett annat sätt att uppdatera steglängden på. Resonemangen utgår från, som i varianten ovan, förändringen av kvadratsummafelet. Visar det sig att kvadratsummafelet ökar, ökar även skillnaden mellan nätverkets utsignal och referenssignal. Man bör i dessa läge ta ett större steg. I det omvända förhållandet då kvadratsummafelet minskar , minskar skillnaden mellan signalerna och därmed kan man även minska på steglängden. Följande kan utryckas matematiskt med samma beteckningar som ovan.

$$\eta(t+1)=\begin{cases}\eta(t)(1+a) & \text{då} \quad \Delta E>0\\ \eta(t)(1-b) & \text{då} \quad \Delta E<0\\ \eta(t) & \text{för} \quad \text{övrigt}\end{cases} \tag{2.19}$$

Det kan å andra sidan vara ganska farligt att justera steglängden efter denna princip. Om nätverket uppdateras i positiva gradientens riktning kommer kvadratsummafelet hela tiden att öka samtidigt som steglängden ökar. Detta kommer att leda till att nätverkets utsignal går mot oändligheten, positiva eller negativa. Nätverket kommer att bli instabilt.

Vid adaptering enligt den senare metoden bör det finnas säkerhetsrutiner så att nätverket uppdateras till ett instabilt beteende. Överhuvudtager behövs det fler säkerhetskontroller, då man uppdaterar någonting adaptivt. Ett litet steg i fel riktning kan resultera i instabila system.

2.6 Nätverkskonfigurering

För att konfigurera ett nätverk finns det inga generella metoder eller regler. Man får oftast förlita sig på *trial-and-error*-metoden, dvs genom en mängd försök komma fram till ett bra nätverk. Det existerar dock satser om hur ett nätverk ska konfigureras med avseende på antalet lager. Dessa teorier kommer att redovisas nedan.

Antag att man vill approximera en speciell mängd funktioner $F_i\{x_k\}$ till en given noggrannhet; hur många dolda lager och hur många neuroner per lager behövs? Svaret är att det behövs som mest ett dolt lager förutom in- och utlager om det finns tillräckligt med neuroner i varje lager. Det har också visats att det inte behövs något dolt lager för att approximera en continuerlig funktion, dvs ett nätverk med två lager är tillräckligt.

Som framgår av ovan är det fullt tillräckligt att konfigurera ett nätverk med tre lager. Vad avser antalet neuroner i varje lager finns det som sagt inga metoder, utan detta måste man experimentera sig fram till. Man kan dock säga att antalet neuroner växer exponentiellt med antalet ingångar.

3 Identifiering

Detta kapitel kommer att handla om identifiering av processer med hjälp av neurala nätverk. Identifieringen kan utformas efter två principer / metoder prediktionsfelsmetoden och utsignalfelsmetoden. Dessa metoder kan också benämnas serieparallell respektive parallell identifiering.

3.1 Inledning

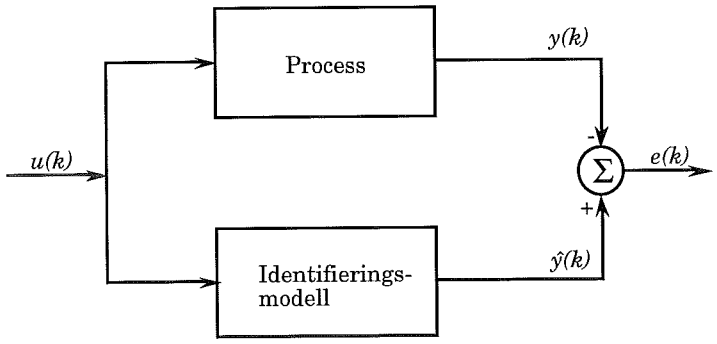
Vid en jämförelse mellan identifiering av linjära system och identifiering med hjälp av neurala nätverk finns det naturligtvis skillnader och likheter. Den stora skillnaden är ett man med neurala nätverk kan identifiera fram olinjäriteter hos en process. Några direkta verktyg som är heltäckande vid identifiering av olinjära system utan neurala nätverk finns idag inte. Däremot finns det många olika teorier hur man ska gå till väga.

Vid själva konfigureringen av nätverket kan man arbeta efter två olika principer. Antingen har man inga processkunskaper och använder de signaler som finnsw tillgängliga, ofta i långa fördröjningskedjor. Eller så känner man processen och använder de signaler som är nödvändiga. Det senare alternativet ger ett mycket mindre komplext nät än om man arbetar förutsättningslöst. I praktiken arbetar man efter en blandning av principerna, dvs man har processkunskaper och man anävnder fler signaler än nödvändigt.

Nackdelen med att använda neurala nätverk är att ju större näten blir desto långsammare blir konvergenstiden. Redan vid mycket små nät konvergerar en vanlig LMS-algoritm fortare.

Formellt kan identifiering formuleras enligt följande: In- och utsignaler från tidsinvariant, kausal diskret dynamisk process benämns $u(k)$ och $y(k)$, där $u(k)$ är en begränsad funktion i tiden. Processen antas vara stabil med känd parametrisering men med okända världen på parametrarna. Målet är att konstruera en lämplig identifieringsmodell, se figur 3.1, så att modellens utsignal, $\hat{y}(k)$ överensstämmer med processen utsignal, $y(k)$, dvs

$$e(k) = \left|\hat{y}(k) - y(k)\right| \to 0 \quad \text{då} \quad k \to \infty \tag{3.1}$$



Figur 3.1. Blockschema över identifiering

3.2 Identifieringsmetoder

Som tidigare nämnts kan man identifiera efter två olika metoder, nämligen prediktionsfelsmetoden och utsignalfelsmetoden. Dessa båda metoder är mycket snarlika med det finns en viktig skillnad. Den principiella skillnaden mellan identifieringsmetoderna är att utsignalfelsmetoden återkopplar nätets utgång till ingången.

Som insignal till de båda metoderna används i regel signaler i fördröjningskedjor (FK), dvs signalerna är fördröjda ett tidssteg mellan varje ingång. Om man ska använda signalen $x(k)$ i en fördröjningskedja om n steg, kommer signalerna $x(k)$, $x(k-1)$, ... , $x(k-n)$ att läggas som insignaler till nätverket.

Oavsett vilken metod man använder sig av måste systemet exciteras tillräckligt för att identifieringen ska lyckas. Det är därför önskvärt att känna till inom viket frekvensområde man ska identifiera och excitera process och nätverket med en signal som har ett önskat frekevensspektrum. En signal som innehåller alla frekvenser är ju vitt brus. Signaler med vitt brus-karraktär är naturligtvis att föredra framför andra typer såsom steg ramper etc. Använder man en stegsignal kommer man bara att identifiera processens uppförande för just detta steg. Om steget ändras till amplitud eller periodtid kommer nätverk och process längre att stämma överens. Man måste således ännu en gång adaptera vikterna i nätverket. Använder man sig däremot av en slumpartad stegsignal som insignal till processen, dvs slumpartat period och amplitud, kommer nätverket att ge samma utsignal som processen för många olika stegsignaler.

Prediktionsfelsmetoden

Prediktionsfelsmetoden är en metod som namnet antyder ett sätt att prediktera en signal y. Vid identifieringen använder sig metoden av gamla, fördröjda, signaler dock ej signaler från prediktorn, dvs det finns ingen direkt återkoppling från utgång till ingång.

Det tvistas ibland om att all modellpassning bygger på prediktionsfelsmetoder.i den meningen att man jämför modellens beteende och den modellbaserade prediktionen med experimentella data. Därefter justeras modellen mot bättre överensstämmelse med den verkliga processen.

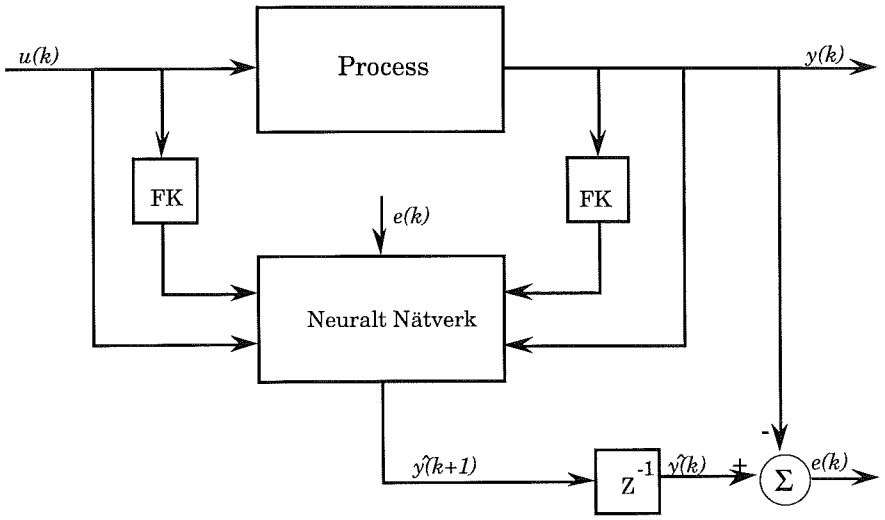
Ur matematiskt synvinkel önskas man minimera följande

$$\min_{\hat{\theta}} E\left\{\sum\left[\hat{y}(k+\tau|k)-y(k+\tau)\right]^2\right\} = \min_{\hat{\theta}} E\left\{\sum e^2(\hat{\theta})\right\} \qquad (3.2)$$

dvs man önska minimera väntevärdet av prediktionsfelet τ steg fram i tiden, med avseende på processparametrarna θ.

Figuren nedan, figur 3.2 visar ett blockschema över identifiering enligt prediktionsfelsmetoden. Av figuren framgår det att det rör sig om enstegsprediktor. Insignaler till det neurala nätverket är processens in- och utsignal i fördröjningskedja. Prediktionsfelet $e(k)$ propageras bakåt genom nätverket, och vikterna uppdateras, till dess att felet understiger en acceptabel nivå.

När identifieringen är avslutad och man vill arbeta med nätverket för t ex simulering måste man ju naturligtvis bibehålla insignalerna till nätverket. Nätverkets utgång återkopplas då till den fördröjningskedja som tidigare hade prcessens utsignalen som ingång.



Figur 3.2. Blockschema för prediktionsfelsmetoden, där FK betecknar fördröjningselement (se text) samt z^{-1} betecknar bakåtskiftoperator.

Utsignalfelsmetoden

Denna metod är egentligen ett sätt att skatta insignal- utsignal relationer eller överföringsfunktionen $H(z)$ till

$$y(k) = H(z)u(k) \tag{3.3}$$

Man önskar således minimera följande kostnadsfunktion

$$\min_{\hat{B},\hat{A}} \sum (y(k) - \frac{\hat{B}(z^{-1})}{\hat{A}(z^{-1})} u(k))^2 \tag{3.4}$$

Kriteriet (3.4) är inte detsamma som i prediktionsfelsfallet, jmf (3.2). Den stora skillnaden ligger i att (3.4) inte är en funktion av gamla y.

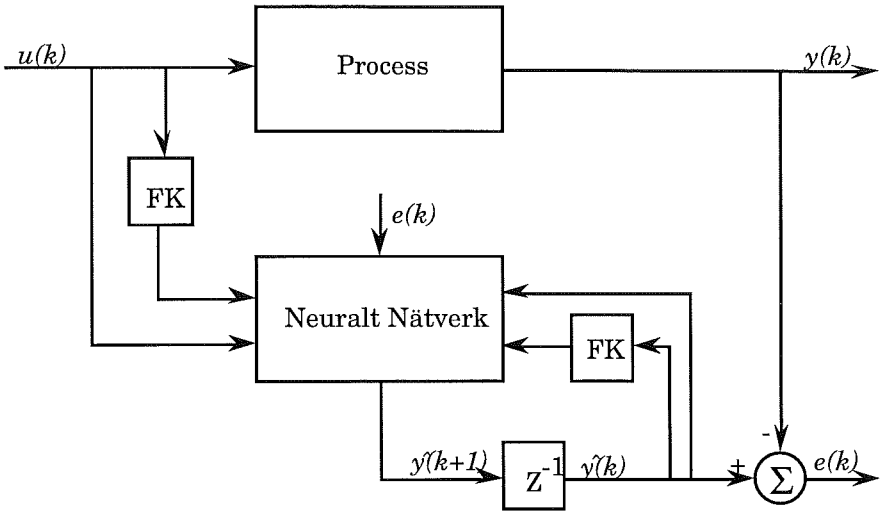
Denna skillnad åskådliggörs bäst med ett litet exempel. Nedan följer två enstegsprediktorer från de olika metoderna.

$$\hat{y}(k) = \hat{a}y(k-1) + \hat{b}u(k-1), \quad \text{Prediktionsfelsmetoden}$$

$$\hat{y}(k) = \hat{a}\hat{y}(k-1) + \hat{b}u(k-1), \quad \text{Utsignalfelsmetoden}$$

Figur 3.3 ger ett blockschema över metoden. Kopoplingen är mycket snarlik med prediktionsfelsmetoden men det finns en markant skillnad. Under själva identifieringen är det neurala nätverket återkopplat, dvs nätets utgång är återkopplad till ingången. Utsignalfelet propageras i sedvanlig ordning bakåt genom nätverket och vikterna uppdateras. '

Då man kopplar från processen för att använda nätverket vid simuleringar behöver man inte göra några ändringar vad gäller nätverkets insignaler som i föregående metod.



Figur 3.3. Blockschema över utsignalfelsmetoden, där FK betecknar fördröjningselement (se text) samt z^{-1} betecknar bakåtskiftoperator.

Metodjämförelse

Båda metoderna är mycket snarlika men de skiljer sig på ett par fundamentala punkter:

- Utsignalfelsmetoden viktar tidigare utsignaler från modellen
- Båda metoderna är egentligen minsta kvadrat problem med den skillnaden att (3.2) är linjär och (3.4) är olinjär.
- Vid identifiering enligt usignalfelsmetoden kan minimeringen avstanna i ett lokalt minimum. Detta sker inte med prediktionsfelsmetoden, eftersom det endast finns ett minumum vid denna typ av minimering, medan vid utsignalfelsmetoden kan det finnas flera.

Resultaten från egna simuleringar med de båda metoderna redovisas i kapitel 6. Utvärdering av resultaten återfinns i kapitel 7.

4 Reglering

I detta kapitel kommer teorin för reglering med hjälp av neurala nätverk att behandlas. Delavsnittet Adaptiv Reglering är inte få något vis fullständigt. Önskar läsaren en mer fullständig analys hänvisas denna till [1].

4.1 Regleringsmetoder

Problemet vid användning av neurala nätverk i reglersammanhang är att man på något sätt måste träna nätet till ett önskat beteende. För att kunna träna ett nät måste man ha tillgång till önskade signaler som ska jämföras med nätverkets utsignaler. Detta är inte alltid så lätt att åstadkomma, om man vill adaptera regulatorn on-line, dvs i realtid. Off-line träning är ju också möjlig. Man registrera en större mängd processdata, processignaler. Nätverket kan sedan tränas separat utan risk för att det totala systemet ska bli instabilt.

Vid träningen av nätverksregulatorn kan man arbeta efter tre olika principer, nämligen:

- Kopiera en fungerande regulator.
- Identifiera det inversa systemet för att använda detta som regulator
- Adaptiv regulator (indirekt och direkt)

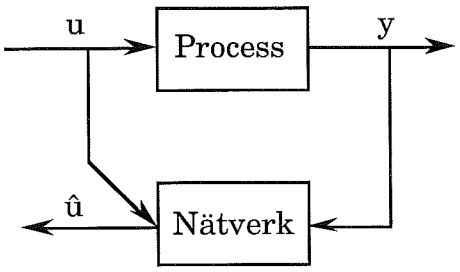
Nedan följer en kortfattad beskrivning av de olika metoderna.

Regulatorkopiering

Kopieringsmetoden kan man använda då den befintliga regulatorn beter sig på ett nästan önskvärt sätt. Man kanske önskar att olinjärt beteende hos regulatorn. Efter det att det neurala nätverket identifierat befintlig regulator, kopplas nätet in som ny regulator och man fortsätter att trimma regulatorn. Finessen med denna metod är att finjustering kommer att gå fortare när man väl kopplat in regulatorkopian än om man startar från början dvs med nollställd regulator.

Inversa systemet

En annan metod vid träningen av ett neuralt nätverk är att identifiera den inversa processmodellen, se figur 4.1.



Figur 4.1. Identifiering av det invers systemet

Insignal till processen är lämpligen en PRBS-signal, Pseudo Random Binary Sequence[1]. Denna signal ger ett slumpartat utseende hos processens utsignal eller nätverkets insignal. Detta är väldigt önskvärt eftersom man önskar identifiera processen för så stor bandbredd som möjligt. Via PRBS-signalen kan man identifiera för en speciell bandbredd.

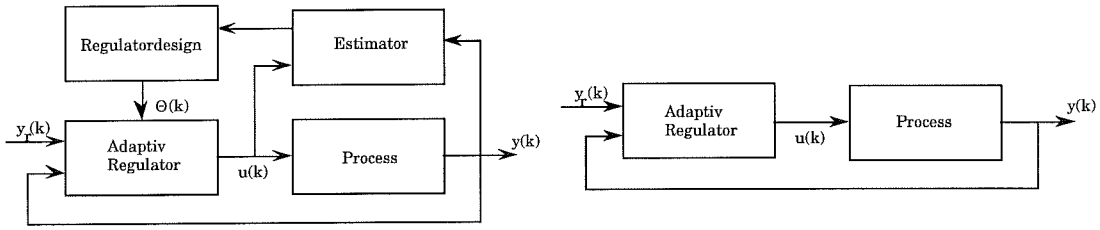
När väl identifieringen är klar, vänder man på nätverket, så att processens referenssignal får fungera som insignal till nätet. Nätets utsignal får i sin tur fungera som styrsignal till processen.

Adaptiv reglering

Då man arbetar med adaptiva regulatorer kan man konstruera dessa efter två olika principer:

- indirekt adaptiv reglering
- direkt adaptiv reglering.

Figur 4.2 visar den principiella skillnaden mellan de olika varianterna.



Figur 4.2. Indirekt (vänster) och direkt (höger) adaptiv regulator

Som framgår av figuren identifieras processen i det indirekta fallet. Därefter görs regulatordesignen baserad på estimerade processparametrar. I det direkta fallet sker ingen processidentifiering utan regulatorn adapterar sina parametrar direkt baserat på tillgängliga signaler; referens-, styr- och utsignal. Dessa båda fall kommer att analyseras nedan, även om det är den indirekta adaptiva regleringen som är av intresse, se kapitel 1.

Vidare kan man arbeta efter två olika design procedurer, antingen med MRAS (Model Reference Adaptive System) eller med STR (Self-Tuning Regulator) regulatorer. Skillnaden mellan metoderna är att i MRAS-fallet sker uppdatering av regulatorparametrarna baserat på felet mellan system och modell. I STR-fallet identifieras processen och regulatorn designas utifrån denna identifiering.

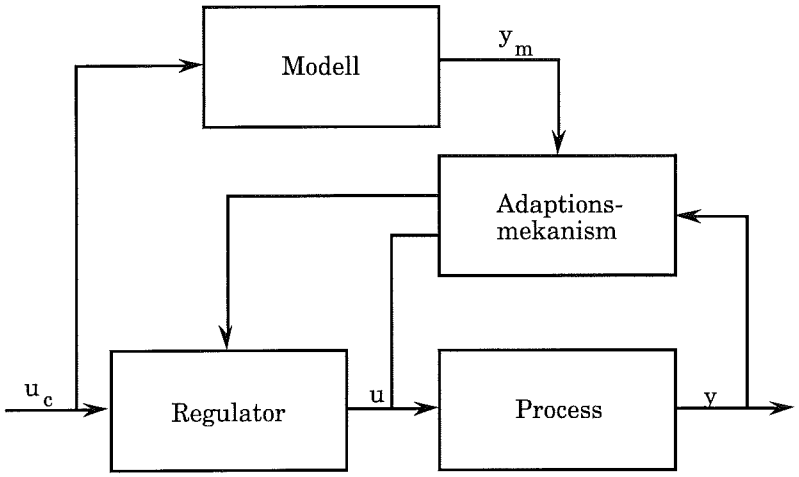
I den fortsatta analysen kommer endast genomgång av MRAS-fallet att göras, ty denna metod användas i simuleringarna, se kapitel 6.

[1]En PRBS-signal är en diskret form av vitt brus, vars signal genereras genom att man återkopplar ett skiftregister. Längden på skiftregistret bestämmer sekvensperioden. Se [3]för vidare förklaring.

4.2 Adaptiv modellreferens (MRAS)

MRAS bygger på att man specificerar en modell för det totala systemet, dvs process och regulator. Målet är det slutna systemet ska stämma överens med modellen i så stor utsträckning som möjligt. Figur 4.3 visar ett blockschema över ett MRAS-system.

Adaptionsmekanismen är konstruerad så att regulatorparametrarna justeras efter skillnaden mellan processignalen y och referensmodellens utsignal y_m.



Figur 4.3. Blockschema över ett MRAS-system.

Låt processen beskrivas av

$$Ay = Bu, \tag{4.1}$$

där u är styrsignalen och y är mätbar utsignal. Symbolerna A och B är polynom i differentialoperatorn p för kontinuerliga system eller i framåtskiftoperatorn q för tidsdiskreta system. Det förutsätts att polynomen A och B inte har några gemensamma faktorer samt att A är monisk. Polöverskottet $d = \deg A - \deg B$ beskriver för det tidsdiskreta fallet processens tidsfördröjning. Låt dessutom dynamiken från referenssignal y_{ref} till önskad utsignal beskrivas av

$$A_m y_m = B_m u_c \tag{4.2}$$

För att regulatorn ska uppföra sig på ett sätt så att processens utsignal överensstämmer med modellreferenssignalen måste en del villkor vara uppfyllda.

För det första måste det polöverskottet i modellen vara minst lika stort som för själva processen, eller

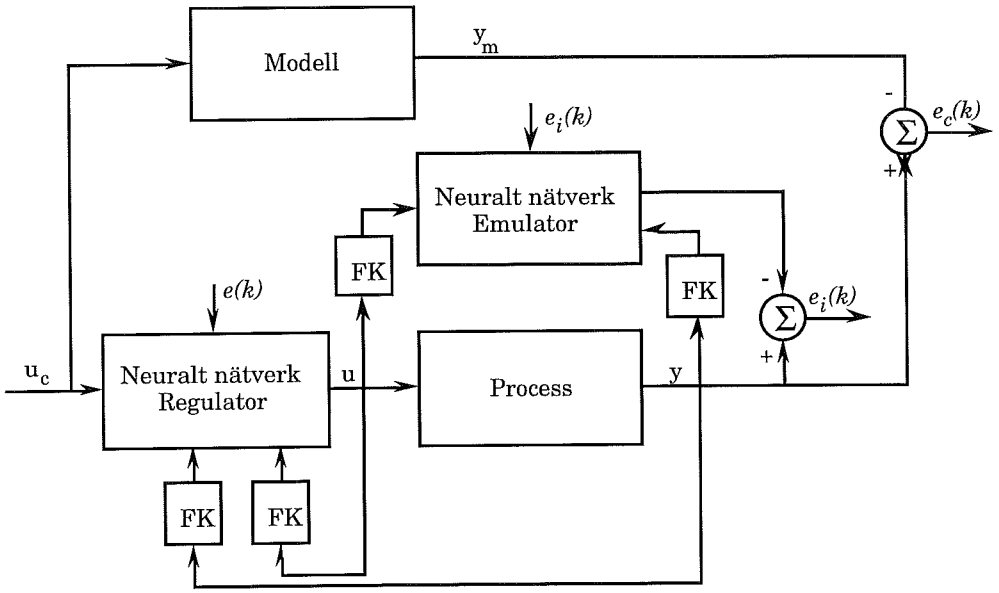
$$\deg A_m - \deg B_m \geq \deg A - \deg B \tag{4.3}$$

annars erhåller man ingen kausal regulator i det tidsdiskreta fallet.

För det andra kan inte modellen vara snabbare än själva processen. Detta leder till en mycket tydlig översläng i ett stegsvar. Man kommer med andra ord att begära för mycket av regulatorn.

Indirekt MRAS-reglering

Vid indirekt reglering använder man konfigurationen nedan, se figur 4.4. Som framgår av figuren innehåller konfigurationen två stycken neurala nätverk, ett som fungerar som regulator och ett som emulator av processen. Idén bakom denna metod är att träna regulatornätverket till den inversa processdynamiken medan emulatorn ska identifiera processen. Emulatorn används sedan för att generera jämförelsesignaler till regulatorn. Man kan säga att regulatorn tränas med ett fel som propageras först genom emulatorn och sedan själva regulatorn. Nedan följer en härledning om hur det egentligen går till.



Figur 4.4 Blockschema över en indirekt MRAS-reglering, där FK betecknar fördröjningskedjor.

Algoritmen som ska härledas kallas *Backpropagation-through-time* eller *Bakåtpropagering genom tiden*. Algoritmen enligt bilaga A kan användas var för sig på nätverken med ett par tillägg.

- Framåtpropagering sker endast i ett nätverk per sampel.
- Emellertid kan man se bakåtpropagering som att båda näten propageras som ett enda, dvs felsignalen till regulatorn kommer från emulatorn.

I nedanstående härledning kommer följande beteckningar och indexering att användas. Överindex E och C härrör sig till emulator och regulator. δ betecknar felet som propageras baklänges i nätet medan I och O betecknat in respektive utsignal till ett lager i nätverket. För att hålla reda på vilket lager det är frågan om används underindex k och j för att beteckna utlager respektive dolda lager. Härledningen härrör sig till ett nätverk med tre lager, dvs ett in- och utlager samt ett dolt lager. Neuronernas överföringsfunktion antas vara logistiska sigmoidfunktioner förutom på utgångarna där man använder linjära neuroner.

Vi börjar härledningen med felekvationen vid emulatorutgången som skillnaden mellan aktuell utsignal från emulatorn och processignal.

$$E^E = \tfrac{1}{2}(y_m - y)^2 \tag{4.4}$$

Eftersom neuronen på utgången hos emulatorn har en linjär överföringsfunktion, kommer felet mellan dolt lager och utlager att vara

$$\delta_k^E = y_m - y. \tag{4.5}$$

Felsignalen mellan dolt lager och inlager för emulatorn blir enligt följande

$$\delta_j^E = -\frac{\partial E^E}{\partial I_j^E} = \frac{\partial E^E}{\partial O_j^E}\frac{\partial O_j^E}{\partial I_j^E}. \tag{4.6}$$

Genom att använda kedjeregeln kommer man fram till följande samband

$$\delta_j^E = \frac{\partial E^E}{\partial I_k^E}\frac{\partial I_k^E}{\partial O_j^E}\frac{\partial O_j^E}{\partial I_j^E} = \left(\sum_j \delta_k^E w_{kj}^E\right) O_j^E(1-O_j^E), \tag{4.7}$$

där w_{kj}^E är vikterna mellan emulatorns dolda lager och utlager. Regulatorn felsignal vid utlagret blir

$$\delta_k^C = -\frac{\partial E^E}{\partial I_k^c} = \frac{\partial E^E}{\partial O_k^c}\frac{\partial O_k^c}{\partial I_k^c}. \tag{4.8}$$

Emedan utlagrets neuroner hos regulatorn är har en linjär överföringsfunktion kommer utsignalens derivata med avseende på insignalen vara lika med ett eller

$$\frac{\partial O_k^C}{\partial I_k^C} = 1 \tag{4.9}$$

Detta leder till att (4.8) kan skrivas om enligt följande

$$\delta_k^C = \frac{\partial E^E}{\partial O_k^c} = \frac{\partial E^E}{\partial I_k^c}\frac{\partial I_k^c}{\partial O_k^c} = \sum \delta_j^C w_{kj}^C \tag{4.10}$$

,

och felsignalen mellan dolt lager och inlager blir

$$\delta_j^C = \left(\sum_j \delta_k^C w_{kj}^C\right) O_j^C(1-O_j^C) \tag{4.11}$$

Algoritmen för reglerstategin med emulator blir som följer

1. Mät signalen y_m
2. Generera kontrollsignal genom regulatornätverket. Propagera även kontrollsignalen genom emulatorn.
3. Beräkna reglerfelet $e_c = y_m - y$
4. Propagera detta fel baklänges genom emulatorn, dvs tills δ_j^E erhållits.
5. Propagera δ_j^E baklänges genom regulatornätverket genom att använda ekvationerna (4.10) och (4.11) ovan, och uppdatera vikterna.
6. Beräkna felet mellan emulator- och processutsignal, $e_i = y - \hat{y}$
7. Propagera identifieringsfelet baklänges genom emulatorn och uppdatera vikterna.

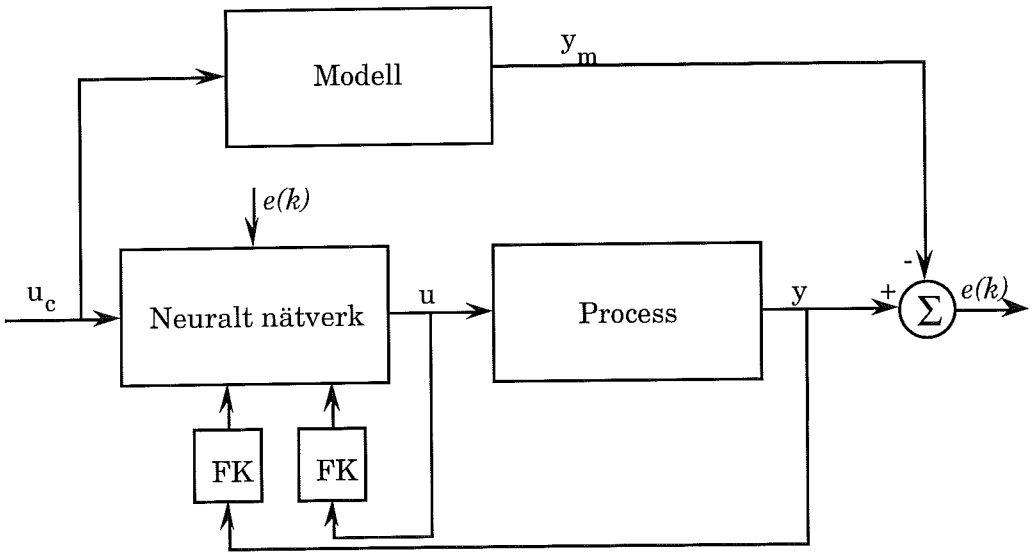
Direkt MRAS-reglering[1]

Det finns egentligen inga publicerade artiklar som tyder på att det går att reglera en process med neuralt nätverk efter principen med direkt MRAS. Det påpekas även i [5] att det i dag inte finns några metoder för att justera vikterna i nätverket direkt, baserat på felet mellan processignal och utsignal från referensmodell.

Oavsett dessa nedslående uppgifter kommer ett försöka att konfigurera en direkt regulator. Detta på grund av att arbetet kommer att te sig som mycket lättare för en operatör om han/hon inte behöver så gedigna kunskaper i reglerteknik. Ända sedan adaptiva regulatorer dök upp, har målet vara att koppla in en svart låda som själv ställer in sig till perfekt reglering. Användningen av neurala nätverk är ett steg i denna riktning, man behöver mindre och mindre processkunskaper för att åstadkomma någonting godkänt.

Vid direkt regulatoradaptering justeras regulatorns parametrar direkt utan att gå någon omväg vare sig via identifiering av processen eller via regulatordesign.

Försök kommer att göras med följande konfiguration, se figur 4.5. Insignaler till det neurala nätverket kommer att vara referenssignalen u_c, samt styrsignalen u och processignalen y, i fördröjningskedja. Felsignalen e får vara styrande vad gäller uppdatering av vikterna.



Figur 4.5 Blockschema över en direkt regulator

[1]Det finns egentligen ingenting som benämns direkt MRAS-reglering. I detta arbete används benämningen, eftersom man använder sig av ett modellreferenssystem.

5 Implementering

Detta kapitel behandlar implementeringen av neuralnätsstödet i SattLine. Två bibliotek med moduler har implementerats. Modulerna kommer att redovisas gruppvis med först en kort introduktion till SattLine.

5.1 SattLine

SattLine är ett grafiskt objektorienterat processtyrsystem utvecklat av SattControl i Malmö. Systemet innehåller en mängd fördefinierade moduler som kan kopplas samman till komplexa konfigurationer via intelligenta kopplingar. Om inte de fördefinierade modulerna räcker till, så kan man mycket lätt konstruera egna, så att de passar in i önskad konfiguration.

Kopplingarna mellan olika modulers in- respektive utgångar måste vara av samma datatyp. Någon annan begränsning vad gäller de grafiska kopplingarna finns inte. De grafiska kopplingarna kan skicka alla typer av data och dessutom i båda riktningarna.

Modulerna byggs bland annat upp av ekvations- eller sekvensblock samt ett operatörsgränssnitt med bland annat parameterlistor och grafiska presentationer. Flera ekvationsblock kan skrivas i samma modul och systemet sorterar själv i vilken ordning blocken ska exekveras. Detta lämpar sig väldigt väl till en neuralnätstillämpning med backpropagation-algoritmen, ty ett block beskriver framåtpropageringen medan ett annat beskriver bakåtpropageringen med uppdatering av vikterna

Den grafiska presentationen av värden i modulen kan göras med såväl stapel- , historikdiagram som värdetabeller.

Systemet är uppbyggt efter fönsterprincipen, dvs man kan plocka fram intressanta delar av konfigurationen och gömma andra. Utifrån de olika modulerna kan man sedan plocka fram ett antal olika fönster med diagram, värdetabeller, larmtabeller etc.

5.2 Neuralnätsbibliotek (Neurallib)

Översikt

Biblioteket *Neurallib* innehåller de moduler man behöver för att bygga upp ett neuralt nätverk. Det finns neuroner med de vanligast använda överföringsfunktionerna. Dessutom finns det en standardvariant (*Standard neurons*) och en utökad variant (*Extended neurons*). Den utökade varianten har tillägg för momentterm och adaptiv steglängd, se kapitel 2.

I biblioteket finns dessutom fördröjningselement för uppbyggnad av fördröjningskedjor samt konverteringsmoduler för konvertering mellan strukturerade och enkla datatyper och vice versa.

Datatyper

Följande datatyper har deklarerats i SattLine. Datatyperna används av de flesta modulerna i biblioteket.

```
NeuronConnection      RECORD
                            Forward     : REAL;
                            Delta       : REAL
                            Tr          : REAL;
                            Train       : BOOLEAN;
                      END;

NeuronConnection_ext        RECORD
                            Forward     :REAL;
                            Delta       :REAL;
                            Tr          :REAL;
                            Mom         :REAL;
                            Train       :BOOLEAN;
                            Momentum    :BOOLEAN;
                      END;
```

Anledningen till att det finns två datatyper, av likartat utseende är att det finns två klasser av neuroner. En standardvariant som använder datatypen `NeuronConnection` och en utökad variant som använder `NeuronConnection_ext`. Med två datatyper kan ingen blandning av neurontyperna göras, ty de grafiska kopplingarna mellan neuronerna måste ha samma datatyp i båda kopplingsändarna.

Posterna i datatyperna har följande funktion:

- Forward — Denna post används vid framåtpropageringen mellan neuronerna.
- Delta — Bakåtpropagering av felet mellan neuronerna.
- Tr — Steglängdsparameter.
- Mom — Momentterm för att undvika lokala minimum, specificerad av användaren, (se kapitel 2.5 Momentterm).
- Train — Logisk variabel för att stänga av eller sätta igång adaptionen, uppdatering av vikterna.
- Momentum — Logisk variabel för att stänga av eller sätta på momenttermen vid uppdateringen av vikterna mellan neuronerna.

Standardneuroner (Standard Neurons)

Neuroner med de tre vanligaste överföringsfunktionerna finns representerade, dvs linjär (*Linear*), tangens hyperbolikus (*Tanh*) och logistisk sigmoid (*Logsig*). Neuronerna har tio in- respektive utgångar. Egentligen vore det önskvärt att ha fler in- och utgångar men detta visar sig vara ohållbart, ty de grafiska kopplingarna mellan lagerna kommer att bli ofantligt många. Exempelvis mellan två lager på 20 neuroner behövs 400 grafiska kopplingar. En annan anledning till att in- och utgångar begränsats till tio är att det finns inget stöd för vektorer i SattLine. Detta gör att antalet variabler ökar med antalet ingångar.

In- och utgångarnas datatyp är *NeuronConnection.*

Uppdateringen sker enligt backpropagationprincipen, se kapitel 2 samt bilaga A. Enda skillnaden från resonemanget i kapitel 2 är beräkningen av kvadratsummafelet, (2.12). Som nämndes ovan finns inget vektorstöd, därför görs en filtrering av kvadratsummafelet enligt

$$E(k) = E(k-1) + [\varepsilon(k) - E(k-1)]/n \qquad n = \begin{cases} k & k < f_c \\ f_c & k \geq f_c \end{cases} \tag{5.1}$$

där f_c och ε betecknar filterkonstant respektive kvadratsummafelet mellan nätets utsignaler och önskade utsignaler.

Modulen *End* sköter jämförelsen mellan nätsignal och önskad signal. Modulen har tio stycken ingångar varav fem av dessa kopplas direkt till nätutgångarna medan resterande fem används till jämförelsesignalerna. Felet propageras bakåt till det sista lagret neuroner som i sin tur propagerar felet vidare enligt algoritmen. Denna modul handhar också operatörskommunikation, vad avser grafisk presentation och parametersättning.

Den grafiska presentationen består i att samtliga signaler plottas i ett historikdiagram. Operatören kan välja vilka signaler som man önskar studera. Dessutom presenteras det filtrerade kvadratsummafelet enligt (5.1) i logaritmisk skala.

Vid MRAS-reglering används modulen *End_MRAS*. Modulen ska användas vid direkt MRAS. Nätverket arbetar som regulator åt processen. Felet mellan önskad processignal, modellreferensen, och processignal propageras bakåt i nätverket. Modulen skiljer sig bara på hur ingångarna konfigureras. Tre ingångar kan kopplas till nätets utsignal - felet propageras via dess kopplingar, tre används till modellreferenser och tre ingångar används till processutsignaler.

Operatören kan ange följande parametrar

- Train — Logisk variabel som bestämmer om adaptionen ska vara påslagen
- Tr — Steglängden
- E2min — Målet för kvadratsummafelet. Då målet är nått avstannar adaptionen.
- FilterConst — Filterkonstanten f_c enligt (5.1)

Utökade neuroner (Extended Neurons)

Skillnaden mellan uppsättningen standardneuroner jämfört med den utökade varianten är att i den utökade varianten finns momentterm och adaptiv steglängd som tillägg. Neuronmodulerna har samma namn som för standardvarianten, dvs följer vilken typ av överföringsfunktion som finns implementerad med *_ext* som tillägg, dvs *logsig_ext, Tanh_ext, Linear_ext* etc.

Datatypen på in- respektive utgång är *NeuronConnection_ext.*

Modulen *End_ext* samt *End_ext_MRAS* handhar som tidigare jämförelsen mellan nätets utsignaler och önskade utsignaler respektive modellreferenssignal. Felet

propageras i sedvanlig ordning, bakåt via sista lagrets neuroner som i sin tur propagerar felet vidare bakåt. I dessa modul sker även adaptionen av steglängden.

Den grafiska presentationen är likadan som för standardvarianten med tillägg för plottning av steglängden i logaritmisk skala.

Vad avser parametrar finns det några fler än för standardvarianten. Nedan kommer en redovisning av de parametrar som operatören kan ange och som inte finns i standardvarianten..

- Momentum Logisk variabel som kontrollerar användandet av momentterm.
- Adapt Logisk variabel som styr adaptionen av steglängden.
- Tr_inc Ökning av steglängden då *Adapt* är sann(>1).
- Tr_dec Minskning av steglängden då *Adapt* är sann (<1).
- Mom Momenttermens värde.
- E2_ratio Hysteresområde i procent kvadratsummafelet. Steglängden ändras inte förrän hysteresområdet har passerats. E2_ratio=5 innebär att hysteresområdet är ±5%.
- AdaptConst Konstant som bestämmer hur många sampel det måste vara mellan två ändringar av steglängden.

Adaptionen av steglängden har implementerats på det sätt som finns beskrivit i kap 2.5, formel (2.18).

Skalmoduler (Scale Modules)

För att snabba upp konvergenstiden finns det skalmoduler (*Scale*) som skalar modulens insignal till intervallet [*MinOut,MaxOut*]. Den skalade utsignalen används sedan som insignal till det neurala nätverket. Detta gör att neuronerna arbetar i det område där överföringsfunktionerna är linjära, eller nästan linjära. Skalningen sker efter följande mönster

$$Out = \frac{MaxOut - MinOut}{MaxIn - MinIn}(In - MinIn) + MinOut \tag{5.2}$$

Parametrarna *MaxIn,* och *MinOut* anges av användaren. *Max* anger signalens maximala värde och *Min* signalens minsta värde. *In* och *Out* betecknar in- respektive utsignal.

Det finns naturligtvis även återskalningsmoduler (*ReScale*). Denna modul har i intervallet [*MaxIn,MaxIn*]. Modulens utsignal skalas till det ursprungliga intervallet [*MinOut, MaxOut*] enligt sambandet (5.2). Det är egentligen ingen skillnad mellan modulen *Scale* och *ReScale*. Användaren måste enge i vilket intervall insignalen ligger och till vilket intervall skalningen ska ske. Anledningen till att det finns två är mera av estetiska skäl. Man ska lätt kunna identifiera i SattLine-kopplingen hur signalerna skalas.

Modulerna använder således följande parameteruppsättning, som kan anges av operatören/användaren

- MaxIn Maximal signalnivå på insignal
- MinIn Minimal signalnivå på insignal
- MaxOut Maximal signalnivå på skalad utsignal

- MinOut Minimal signalnivå på skalad utsignal

Fördröjningselement (Delay Modules)

Fördröjningselementen finns för att man ska kunna bygga fördröjningskedjor. Modulen finn i två varianter en standard och en utökad. Modulen har en reell ingång samt elva utgångar varav en är reell och de andra följer datatyperna specificerade ovan, beroende på vilken variant som avses.

Datatypskonvertering (Converting Modules)

Denna modulgrupp gör det möjligt att koppla samman moduler från biblioteket *Neurallib* med andra moduler i andra bibliotek.

Konvertering kan ske mellan datatyperna *NeuronConnection* och den utökade varianten *NeuronConnection_ext* till den enkla datatypen *Real*. Vid konvertering är det bara posten `Forw` som konverteras.

5.4 Referensgeneratorbibliotek (Refgenlib)

Översikt

I biblioteket *Refgenlib* finns det moduler för generering av steg- eller sinussignaler. Dessutom finns det modellreferensmoduler som kan användas vid MRAS-reglering.

Datatyper

Modulerna använder sig av de vanliga enkla datatyperna förutom modellreferens modulerna som använder den strukturerade datatypen *FilterPar* enligt nedan.

```
FilterPar      RECORD
                     Time  :REAL
                     Zeta  :REAL
                     Omega :REAL
               END
```

Posterna i datatypen har följande funktion:

- Time Tidskonstant hos ett första ordningens filter.
- Zeta Dämpningen i ett andra ordningens system
- Omega Snabbheten hos det andra ordningens system.

Signalgeneratorer (Signal Gen)

Signalgeneratorerna som implementerats är en sinus- och en steggenerator. Sinussignalen är symmetrisk runt ett medelvärde. Stegsignalen kan göras asymmetrisk i både tid och amplitud.

Utgången är av reell typ. Detta gäller både för steg- och sinusgeneratorn.

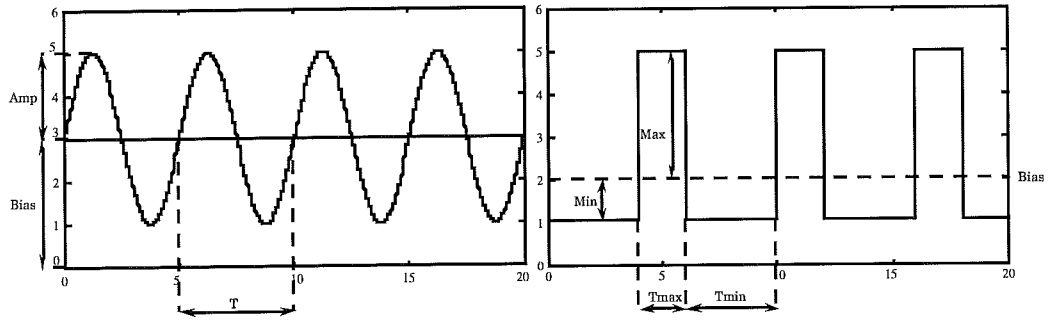
Användaren kan ange parametrar vilka ger önskad kurvform, se figur 5.1. Parametrarna för sinusgeneratorn är

- Amp Signalens amplitud
- Bias Signalens likspänningsnivå/medelvärde
- T Periodtid i sekunder

Parametrarna till steggeneratorn är följande

- Bias Medelnivå eller referensnivå.
- Max Maximal signalnivå. Antal enheter över referensnivån.
- Min Minimal signalnivå. Antal enheter under referensnivån
- Tmax Tid i sekunder som signalen ligger på maximalt värde.
- Tmin Tid i sekunder som signalen ligger på minimalt värde.

Via en parameter *Random* kommer amplituderna att slumpas. Angiven amplitud kommer att multipliceras med ett tal i intervallet noll till ett. Detta gäller för båda modulerna.



Figur 5.1. Parameterförklaring

Modellreferensmoduler (MRAS modules)

Modellreferensmodulerna består av ett första och ett andra ordningens system eller filter.

Modellreferenserna har följande matematiska utseende, kontinuerlig tid:

$$G_1(s) = \frac{1}{1+Ts} \qquad G_2(s) = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2} \tag{5.4}$$

där T, ζ, *och* ω anger tidskonstant, dämpning respektive snabbhet.

In och utgång är av reell typ.

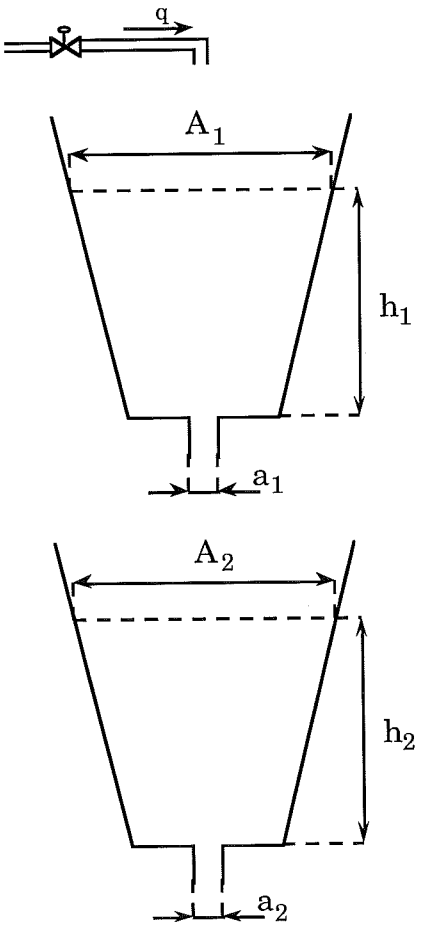
Användaren kan specificera filterkonstanterna enligt datatypen *FilterPar*, se ovan. För det första ordningens system anges systemets tidskonstant och för andra ordningens system, dämpning och snabbhet.

6 Resultat

Detta avsnitt kommer att behandla resultaten från körning i SattLine. Alla experiment är simulerade på den så kallade tankprocessen, därefter görs en kort introduktion till denna. Därefter presenteras resultaten från identifierings- respektive reglerings experimenten,

6.1 Tankprocessen

Den process som all simulering bedrivits mot är den så kallade tankprocessen. Processen består av två vattentankar (övre respektive undre). Den övre tanken har ett reglerbart vatteninflöde, medan den undre tankens inflöde är det samma som den övres utflöde. Systemet kan beskrivas enligt följande



Figur 6.1 Tankprocessen

I figuren ovan hänför sig beteckningarna enligt följande.
A och a betecknar vätskans ytarea respektive utloppsarea. Vätskans ytarea kan lätt beräknas om man känner tankens bottenarea samt hur mycket sidorna lutar i förhållande till ett tänkt vetikalplan. h betecknar vätskenivån. Inflödet till den övre tanken betecknas med q.

Följande härledning ligger till grund för systemekvationerna.

Utflödet q_{ut} från en tank ges av

$$q_{ut} = a\,v \tag{6.1}$$

där v betecknar utstömmningshastigheten. Denna ges av

$$v = \sqrt{2\,g\,h} \tag{6.2}$$

enligt Bernoullis lag om lamminär strömning. Eftersom ökningen per tidsenhet av vätskemängden i en tank är lika med inflöde minus utflöde får man

$$\frac{d}{dt}(Ah) = q_{in} - q_{ut}. \tag{6.3}$$

Insättning av (6.1) och (6.2) ger

$$\frac{dh}{dt} = -\frac{a\sqrt{2g}}{A}\sqrt{h} + \frac{1}{A}q_{in}. \tag{6.4}$$

Processen med de två tankarna kan nu beskrivas matematiskt med följande tillståndsekvationer

$$\begin{pmatrix} \frac{dh_1}{dt} \\ \frac{dh_2}{dt} \end{pmatrix} = \begin{pmatrix} -\frac{a_1\sqrt{2g}}{A_1} & 0 \\ \frac{a_1\sqrt{2g}}{A_1} & -\frac{a_2\sqrt{2g}}{A_2} \end{pmatrix} \begin{pmatrix} \sqrt{h_1} \\ \sqrt{h_2} \end{pmatrix} + \begin{pmatrix} \frac{1}{A_1} \\ 0 \end{pmatrix} q \tag{6.5}$$

Som framgår av ekvation (6.5) är systemet olinjärt.

Linjäriseras och samplas systemet (6.5), och resultatet skivas om till en överföringsfunktion erhåller man följande struktur

$$H(q) = \frac{b_1 q + b_0}{q^2 + a_1 q + a_0} \tag{6.6}$$

i framåtskiftoperatorn q.

Önskar man en differansekvation får man följande form

$$y(k) = -a_1 y(k-1) - a_0 y(k-2) + b_1 q(k-1) + b_0 q(u-2) \tag{6.7}$$

I ekvation (6.7) betecknar q inflödet till övre tanken och y betecknar vätskenivån i den undre tanken.

6.2 Identifiering

Som nämnts tidigare finns det två principiellt olika sätt att identifiera en process, nämligen via prediktionsfelsmetoden och utsignalfelsmetoden. Nedan följer en utvärdering av de båda varianterna. Före själva utvärderingen kommer en liten diskussion om insignaler till nätverket.

Nätverkssignaler

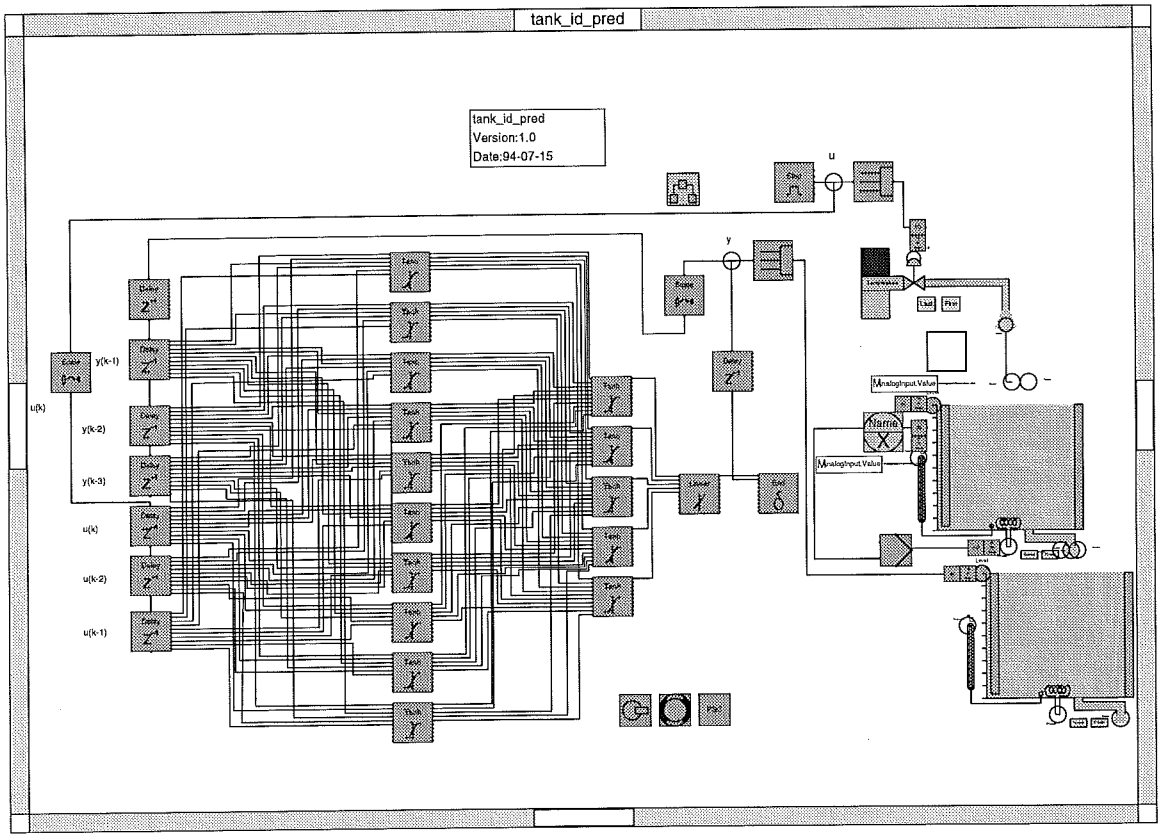
Som nämnts tidigare behöver man inte ha fullständig eller ringa processkännedom för att uppnå ett bra resultat. Man använder de signaler man har tillgång till och

lägger dessa till fördröjningskedjor, där antalet fördröjningssteg är stort (20 - 30). Med denna tankesätt kommer insignallagret att bli mycket stort, dvs många noder. Nätverket får mycket redundant information och kommer att konvergera långsamt.

Besitter man processkunskaper som i fallet med tankprocessen kan man ur (6.3) utläsa att det behövs 4 ingångar till nätet, dvs *q(k-1), q(k-2), y(k-1), y(k-2)*. Dessa fyra signaler ska tillsammans ge tillräcklig information för att identifiera processen.

I praktiska tillämpningar har man sällan en fullständig matematisk modell över processen och i dessa fall förlänger man fördröjningskedjan något.

Prediktionsfelsmetoden



Figur 6.2 Sattline-konfiguration; prediktionsfelsmetoden

Nätverket i konfigurationen ovan, figur 6.2, har följande uppbyggnad:

- tre lager mer 10, 5 respektive 1 neuron.
- inlager och dolt lager har neuroner med överföringsfunktioner av tangenshyperbolikus, utlager har linjär neuron
- *y(k-1), y(k-2), y(k-3), u(k), u(k-1)* och *u(k-2)* används som insignaler till nätverket

Insignalerna till nätverket skalas till intervallet [-5, 5] för att snabba upp konvergensen.

Under simuleringen genereras en stegsignal till ventilen för vattenflödet till den övre tanken. Steggeneratorns parametrar är satta enligt följande:

- Max = 15
- Min = 15
- Bias = 25
- Tmax = Tmin = 10 s
- Random = True

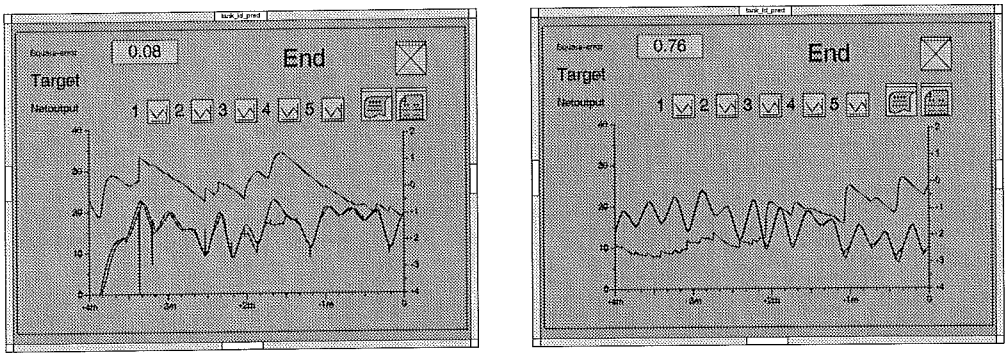
Parametern *Random* anger att amplituden ska vara slumpmässig inom intervallet [*Min, Max*]. Periodtiden är *Tmin* +*Tmax* =20 s.

Nedan följer en rad experiment med att ändra värden på steglängd och införandet av momentterm samt adaptiv steglängd. Sampeltiden kommer också att varieras för att dess inverkan ska kunna observeras.

Grundinställningen om inget annat anges vad gäller parametrar är följande

- Sampelintervall; 100 ms
- Steglängd; 0.01
- filterkonstant vid filtrering av kvadratsummafelet; 100

Konvergeringsresultat

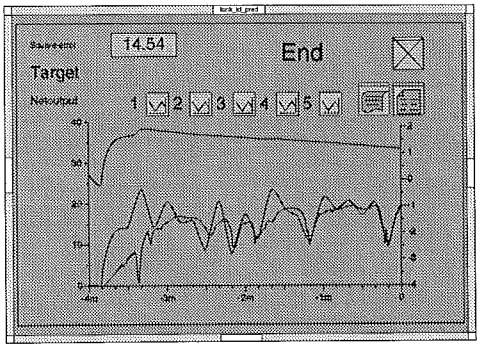


Figur 6.3 Studie om nätverket konvergerat

I samtliga experiment kan man få den uppfattningen att nätverket redan efter några få sampel, med parametervärde enligt grunduppsättningen, identifierat processen. Så är dock ej fallet som även illustreras i figur 6.3 ovan. vänstra delen, vid tiden -2m slås adaptionen av strax därefter kommer det ett steg till processen. Resultatet blir att nätverkets utsignal inte längre följer processen. Nätet har således inte lärt sig att följa processen fullt ut. Sätts adaptionen på igen, vilket görs vid tiden -1m börjar nätverket följa igen.

Den högra figuren visar resultatet efter cirka 30 minuter när adaptionen är frånslagen. Nätverket följer nu processen oavsett vilket steg som än kommer till ventilen.

Förändring av samplingsperiod

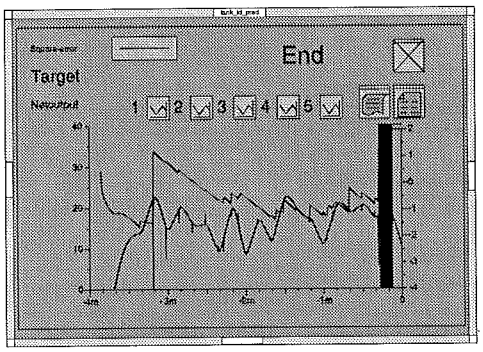


Figur 6.4 Samplingsperiodens inverkan på identifieringen;

Förändringen av har den inverkan att vid väldigt kort samplingsperiod följer processen mycket bra (se figur 6.3, vänstra delen). Förlängs tiden mellan två sampel kommer nätverket inte längre att följa lika bra i uppstarten av identifieringen, se figur 6.4. Detta får som resultat att konvergenstiden kommer att förlängas drastiskt med ökat samplingsperiod.

Anledningen till detta fenomen är att processen inte förändrar sig så väldigt mycket mellan två sampelpunkter när tidsintervallet mellan punkterna är kort. Detta resulterar i att en ändring av slutneuronens vikter räcker för att nätverket ska följa processen. De andra neuronernas vikter förblir nästintill oförändrade. Man kan se att då det händer något drastiskt med processen, vid ett steg, kommer nätets utsignal att vara spikartad.

Steglängdens inverkan



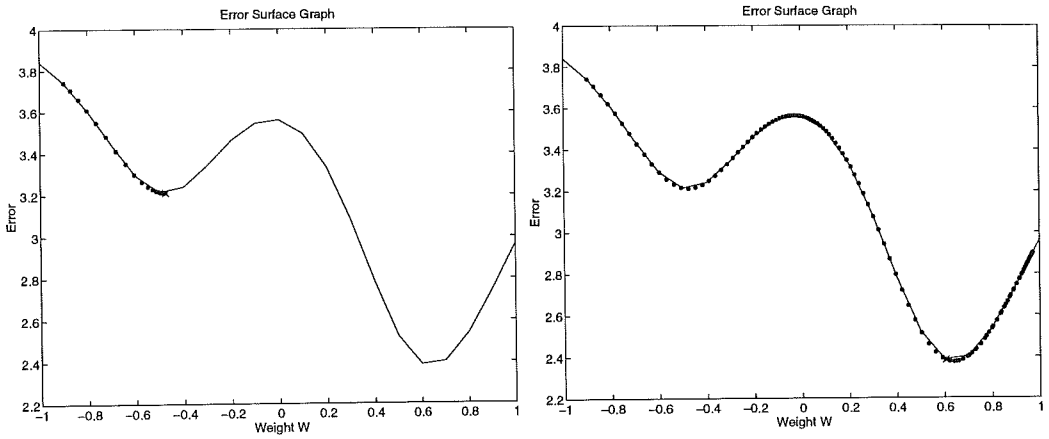
Figur 6.5 Steglängdens inverkan på minimeringen

Steglängden är en viktig parameter. I figur 6.5 ovan visas inverkan av storleken på seglängden. Tre fall går att urskilja. Till att börja med är steglängden satt till 0.01. Nätverket följer processen nästan. Vid tiden -2.5 min ändras steglängden till 0.1 och resultatet låter inte vänta på sig, nästan perfekt följning. Efter ytterligare en stund, vid -1m, ändras steglängden till 0.5 och nätverket blir instabilt.

Vid de tre valen av steglängden inträffar följande. Vid kort steglängd kommer nätverket nästan att justera sina vikter så att nätverket följer processen. Ökas steglängden ändras vikterna lite mer och en perfekt följning blir resultatet. I det sista fallet, då nätverket blir instabilt, kommer justeringen av vikterna att vara för stor och man minimerar inte längre i negativa gradientens riktning utan en ändring

av vikterna görs i positiv gradient. Resultatet blir att minimeringen kommer att alternera med att ta något steg i fel riktning och något steg i rätt riktning. Har man riktig otur kan nätverket bli totalt instabilt, dvs utsignalen blir obegränsad.

Momenttermens inverkan



Figur 6.6 Momenttermens inverkan; vänstra figuren utan momentterm, högra figuren med momentterm

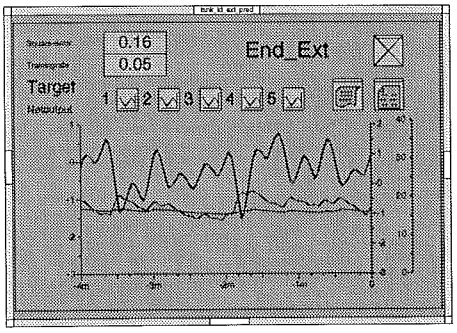
Momenttermens inverkan är ganska svår att åskådliggöra genom simulering av tankprocessen. Istället visas ett exempel simulerat i Matlab.

Nätverket i simuleringen innehåller ett lager med en neuron. I figuren ovan , figur 6.6, finns kurvan för kvadratiska felet med avseende på viktens värde. Det framgår att kurvan innehåller ett lokalt minimum, då viktens värde är -0.5, samt ett global minimum, då vikten har värdet 0.6. Punkterna i figuren motsvarar ett steg i uppdateringen av vikten.

Initialt sätts vikten till -0.90 i båda simuleringarna. I simuleringen utan momentterm, vänstra delen av figur 6.6, avstannar uppdateringen i det lokala minimumet. Då en momentterm införes, se högra delen av figur 6.6, fortsätter minimeringen förbi det lokala minimumet, för att till slut avstanna i det den globala minpunkten. Momenttermen sätts i denna simulering till 0.98.

Man kan likna momenttermens inverkan med att minimeringen ges en extra knuff för att komma förbi lokala minpunkter. Detta resulterar å andra sidan i att minimeringen kommer att gå förbi det globala minimumet en bit, för att sedan vända tillbaka och konvergera till önskat värde. Detta fenomen illustreras i figur 6.6, högra delen. Vikten uppdateras nästan ända till värdet 1.0. Därefter görs minimeringen i andra ritningen för till slut konvergera med viktens värde till 0.6323

Adaption av steglängd



Figur 6.7 Simulering med adaptiv steglängd

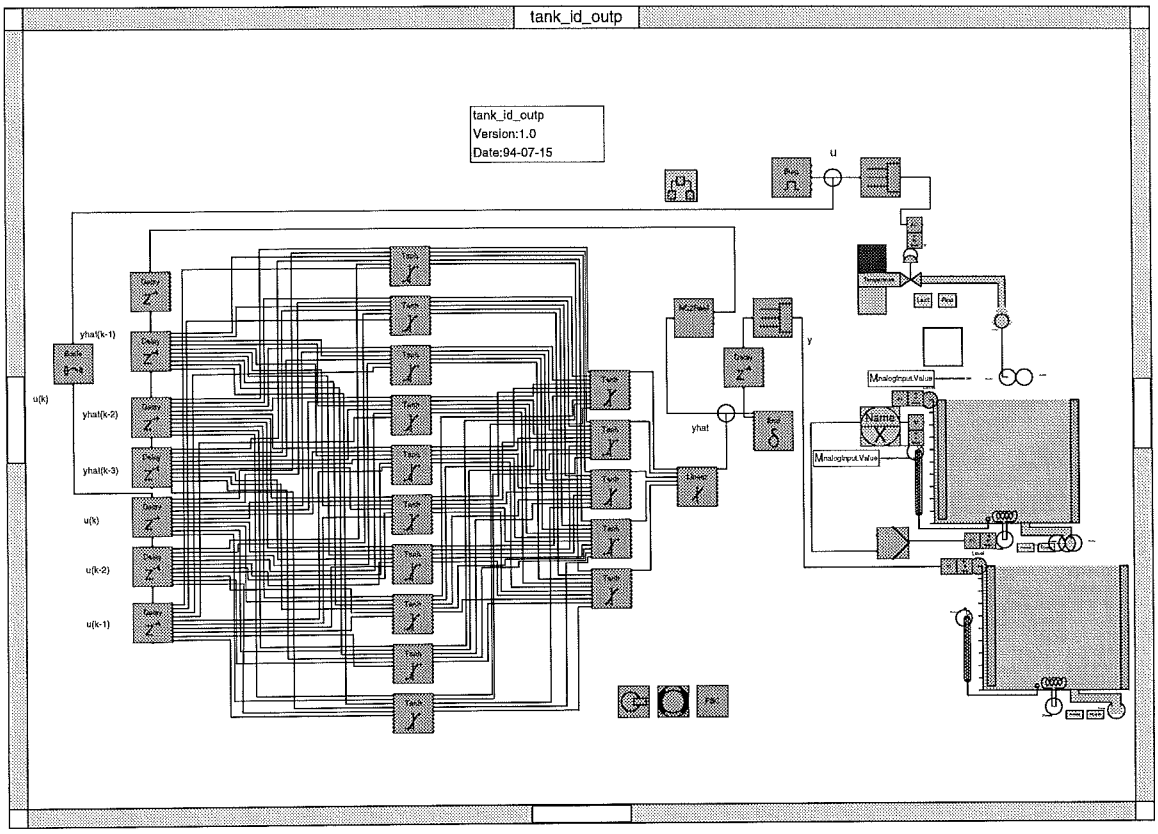
Vid adaption av steglängd måste man ange ytterligare ett antal parametrar. I experimentet med tankidentifiering har följande parametervärden använts:

- Tr_inc - ökning av steglängd: 1.02
- Tr_dec - minskning av steglängd: 0.95
- E2_ratio - hysteres hos kvadratsummafelet: 5
- AdaptConst - adaptionskonstant: 20

När träningen av nätverket startas sätts steglängden till ett initialt värde, 0.1. Som framgår av figuren ovan, ändras steglängden om kvadratsummafelet ändras tillräckligt mycket, ändring stötte än hysteresområdet, i detta fall 5%. Vid justering av de andra parametrarna kan man säga att en ökning av steglängden ska göras försiktigt medan en minskning kan göras mer drastiskt. Det framgår också av ovanstående parameterval, att ökningen sker med 2 % medan en minskning sker med 5 %. Man får laborera sig fram till ett bra parameterval. En för stor minskning av steglängden kan leda till att steglängden blir försvinnande liten och vikterna i nätverket uppdateras inte. Ökas seglängden för mycket kan nätverket bli instabilt, i detta exempel inträffar detta då steglängden är större än 0.35.

Ytterligare en parameter som är besvärlig att justera är adaptionskonstanten. Det finns inga tumregler utan man måste genom simuleringar komma fram till ett lämpligt värde. Anledningen till att den finns är för att förhindra en alltför ofta återkommande ändring av steglängd. Vid t ex uppstart kommer kvadratsummafelet att öka markant. Ändrar man steglängden vid varje sampel kommer till slut steglängden vara så liten att vikterna inte uppdateras. Dock bör adaptionskonstanten sättas till ett ganska långt värde i förhållande till filterkonstanten som används vid beräkning av kvadratsummafelet. I detta exempel skiljer det en faktor tio mellan konstanterna.

Utsignalfelsmetoden



Figur 6.8 Sattline-konfiguration; prediktionsfelsmetoden

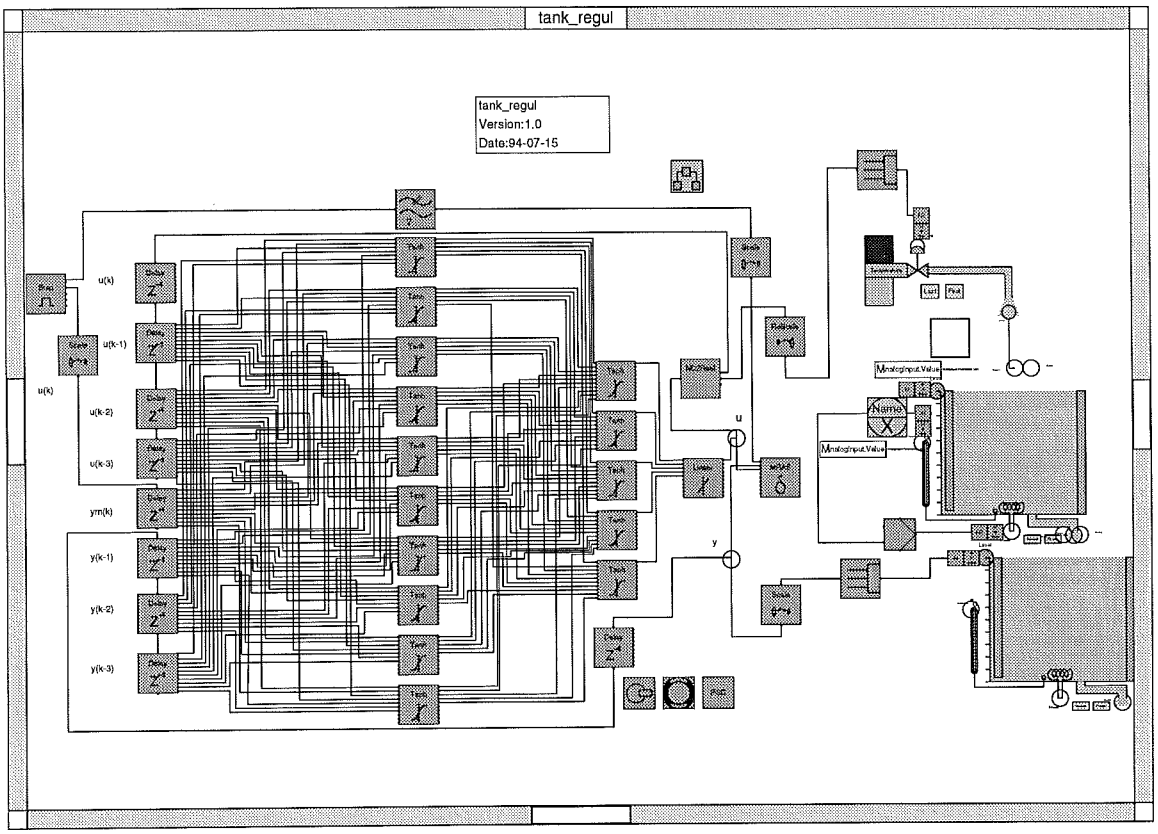
Nätverket i konfigurationen ovan, figur 6.8, har följande uppbyggnad:

- tre lager mer 10, 5 respektive 1 neuron.
- inlager och dolt lager har neuroner med överföringsfunktioner av tangenshyperbolikus, utlager har linjär neuron
- *y(k-1), y(k-2), y(k-3), u(k), u(k-1)* och *u(k-2)* används som insignaler till nätverket

Insignalerna till nätverket skalas till intervallet [-5, 5] för att snabba upp konvergensen.

Samma parameterexperiment utfördes för denna identifieringsmetod. Den enda skillnaden som kunde observeras var att konvergenstiden var något kortare. Anledningen till detta antas vara att minimeringen med hjälp av prediktionsfelsmetoden ovan, fastnade temporärt i lokala minima. Detta kunde förmodligen kompenseras med införandet av momentterm i uppdateringen av vikterna i nätverket.

6.3 Reglering



Figur 6.9 Sattline-konfiguration; direkt MRAS-reglering

Nätverket i konfigurationen ovan, figur 6.9, har följande uppbyggnad:

- tre lager mer 10, 5 respektive 1 neuron.
- samtliga neuroner är av tangenshyperbolikus-typ förutom utgångsneuronen som är av det linjära slaget.
- *y(k-1), y(k-2), y(k-3), u(k), u(k-1)* och *u(k-2)* samt *ym(k)* används som insignaler till nätverket

Insignalerna till nätverket skalas till intervallet [-1, 1] för att förbättra konvergenshastigheten samt att begränsa signalen till ventilen, som reglerar inflödet till övre tanken. Skalning gör att biastermen inte uppdateras nämnvärt. Detta leder till att styrsignalen till ventilen , innan återskalning, kommer att ligga i intervallet [-1, 1]. Signal återskalas till ventilens ursprungliga intervall [0, 100]. Detta förfaringssätt ger en bra kontroll på strysignalen.

Som modellreferenssignal används en filtrerad stegsignal. Steget skickas via ett filter till jämförelseneuronen. Filtret som används är ett andra ordningens filter. Detta filter uppfyller kraven enligt kapitel 4.2. Filtret har följande parametrar:

- $\omega = 0.2$ rad/s
- $\zeta = 0.7$.

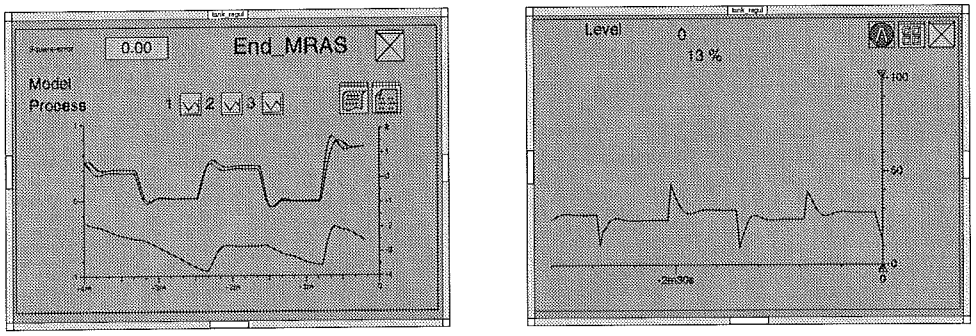
Följande parametervärden används för steggeneratorn.

- Max = 10
- Min = 10
- Bias = 25
- Tmax = Tmin = 50 s
- Random = True

Parametern *Random* anger att amplituden ska vara slumpmässig inom intervallet [*Bias-Min, Bias+Max*]. Periodtiden är *Tmin +Tmax* =100 s.

Experimenten som görs är de samma som för identifieringsmetoderna ovan. I stort sett kommer man fram till samma resultat. Den största skillnaden vad avser resultatet kan observeras i experimentet, då man varierar steglängden samt då adaptiv steglängd används. Därför kommer endast dess simuleringar att redovisas.

Konvergensstudie



Figur 6.10 Träning av regulatornätverk. Vänstra delen -

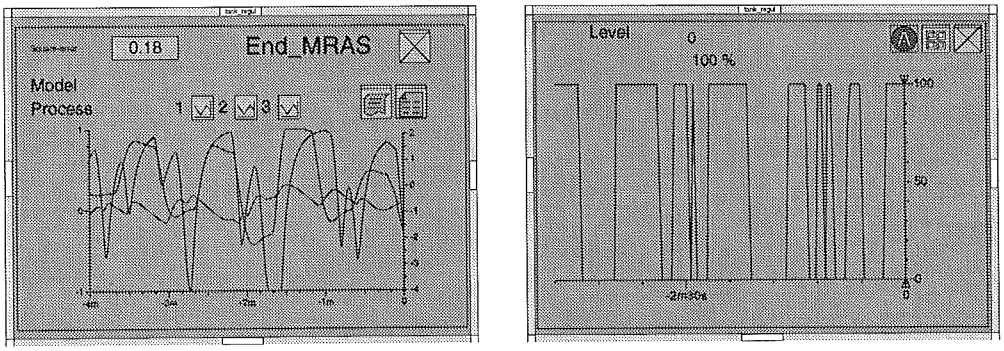
Jämför man konvergenstiderna för identifierings och regulatornätverk kan man klart konstatera att regulatornätverket konvergerar långsammare. Någon följning sker inte initialt som i fallet då nätverket ska identifiera processen.

Figur 6.10, ovan, visas nätverkets status efter ca en timmas uppdatering. Som framgår av den vänstra delen av figuren ovan har nätverket inte lärt sig tillräckligt för att följa referensmodellen fullt ut. Ytterligare uppdatering behövs.

Styrsignalen från nätverket till ventilen visas i den högra delen av figuren.

Nätverket har uppdaterats med en steglängd på 0.0001. Resultatet om man använder större steglängd utreds nedan.

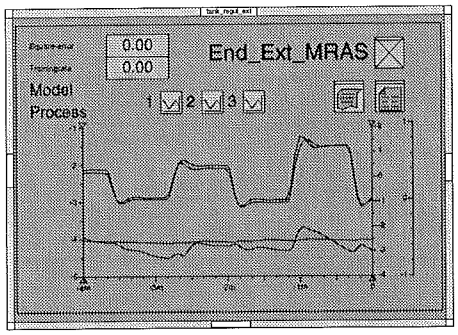
Steglängdens inverkan.



Figur 6.11 Uppdatering av regulatornätverket mer steglängden satt till 0.01.

Steglängden är den viktigaste parametern att justera även i reglerexperimenten. Sätts steglängden till ett för stort värde får man som resultat en instabil regulator. I figuren ovan, figur 6.11, har seglängden satts till 0.01. Efter en kort stund av uppdatering av vikterna erhålls detta resultat. Utsignalen från regulatorn eller styrsignalen till ventilen bottnar. Detta resulterar i att processen inte kan följa referensmodellen. Enda sättet att undvika detta beteende är att minska på steglängden.

Adaption av steglängd



Figur 6.12 Resultat då steglängdens adaptionsalgoritm

Då man använder sig av steglängdsadaption under uppdateringen av nätverkets vikter förkortas konvergenstiden. Den förkortas inte drastiskt med dock. Figuren ovan, figur 6.12, redovisar en simulering med adaptiv steglängd. Efter ca 45 minuter följer processen någorlunda referensmodellen. Fortsätter man uppdateringen kommer man att erhålla ett bättre resultat.

Initialt sattes steglängden till 0.001. Adaptionskonstanten sattes till 20 samt ökning respektive minskning av steglängden sattes till 1.1 respektive 0.95.

Svårigheten med att använda detta uppdateringssätt är att regulatorn kan lätt bli instabil om steglängden sätts initialt till ett för stort värde. Dessutom måste man ange lämpliga värden på öknings- respektive minskningsparametern. Detta leder till att måste prova sig fram till lämpliga värden.

Ofta måste man justera parametrarna under tiden som nätverket tränas. Det kan annars inträffa att steglängden blir för stor, vilket resulterar i en instabil regulator. Blir steglängden för liten är inte så stor skada skedd. Uppdateringen avstannar i

stort sätt och det finns ingen fara för instabilitet. Användaren/operatören kan själv justera steglängden till ett acceptabelt värde. Skulle regulatorn ha kommit in i en instabil mod, kan det ta mycket lång tid för densamma att kom ur den instabila moden. Oftast är det lika bra att börja om från början med inlärningsprocessen.

7 Slutsatser

Detta kapitel ger en sammanställning över slutsatserna man kan dra av gjorda simuleringar.

7.1 Neurala nätverk och implementation

Neurala nätverk kommer förmodligen bli en storsäljare inom många områden. Framför allt inom mönsterigenkänning. Redan idag finns det relativt många kommersiella produkter för teckenigenkänning etc. Vad gäller användandet i reglersammanhang är jag inte lika övertygad. Anledningen till min skepsis kan man läsa nedan.

Frågan om SattLine och neurala nätverk hör ihop är lätt att svara på. Inte i den form som finns implementerad idag, eftersom nätverken är begränsade till tio neuroner per lager. För att få generella nätverk bör man utöka antalet noder i ett lager till åtminstone det dubbla. Detta kräver å andra sidan att man implementerar ett vektorstöd i SattLine. Utan vektorstöd lämpar sig inte implementationer av detta slag. Dels behövs en ringbuffert när man beräknar kvadratsummafelet, och dels förenklas kodningen om vikterna kan deklareras som vektorer eller kanske ännu hellre som matriser. Antalet variabler ökar snabbt om det maximala antalet noder utökas. Det är dessutom ohållbart att sitta och koppla varje nod-nod-förbindelse för hand med grafiska kopplingar. En vidare utveckling är att göra standardnät i modulform med ett antal in- och utgångar. Det räcker förmodligen med tre lager, ty de simuleringar som utförts samt artiklar i ämnet tyder inte på att fler lager är nödvändigt. Antalet noder i det dolda lager bör nog vara relativt stort, 20-40 noder, för att få en så generella nätverk som möjligt

Vad gäller algoritmen *Backpropagation* kan man säga att den är mycket långsam. Optimeringar enligt negativa gradienten är kända för att vara mycket långsamma. Då det inte finns några tidskrav vad gäller konvergenstider kan man mycket väl använda algoritmen, ty den är enkel att förstå och implementera. För tillämpningar i den reglertekniska sektorn bör man utveckla en snabbare algoritm för uppdatering av vikterna. Vid en jämförelse mellan självinställande regulatorer och neurala nätverk är det stora skillnader i inställningstider. Självinställande regulatorer är mycket snabbare än de neurala nätverken.

Dessutom är det inte trivialt att justera steglängdsparametern. I många fall kan ett felaktigt val av just denna parameter leda till förödande konsekvenser. Adaptionsalgoritmen i kapitel 2 (2.18) lämpar sig bra om kvadratsummafelet implementeras enligt (2.12). Implementationen av kvadratsummafelet i SattLine är egentligen en filtrering på grund av att vektorstödet saknas. Detta leder till att algoritm (2.12) inte fungerar på avsett sätt. Istället används en algoritm, där användaren anger en adaptionskonstant som anger hur ofta steglängden får ändras, dvs vart tionde eller vart hundrade sampel etc

Det stora problemet med att göra backpropagation mer generell, dvs av det mer självskötande slaget, är att det tillkommer en mängd parametrar som användaren måste ha kännedom om. Bara adaptionen av steglängden kräver fyra extra parametrar. Till detta kommer momentterm som förhindrar att minimeringen fastnar i

lokala minima. Detta leder till att vid konfigureringen av nätverk och därtill lämpliga parametrar måste göras via *trial-and-error-metoden*. Någon on-line körning kan inte göras av säkerhetsskäl. Optimeringen kan lätt spåra ur.

Fördelarna med att använda neurala nätverk är att olinjäriteter kan approximeras mycket bra. Det kan dock krävas att nätverket innehåller relativt många noder, 40-50 st per lager, för att approximera gravt olinjära funktioner. En annan stor fördel är att man inte behöver känna till vilka signaler som kan tänkas behövas. Ofta tar man de signaler man har tillgång till. Haken med detta synsätt är att antalet dolda neuroner ofta ökar exponentiellt med ingångarna, vilket i sin tur leder till längre konvergenstider.

Ett sätt att minska konvergenstiden är att skala nätverkets insignalerna och jämförelsesignalerna till lämpligt intervall. Lämpligt intervall för tangenshyperbolicusneuronerna är [-5, 5] och neuronerna med logistisk sigmoidfunktion [0, 5]. Anledningen till att detta snabbar upp konvergensen är att om signalnivåerna är större kommer överföringsfunktionerna att bottna. I bottningslägena är derivatan av funktionen nära noll och ringa uppdatering sker. Skalas däremot signalerna till intervallen ovan kommer neuronerna hela tiden att kunna uppdatera vikterna och konvergenstiden minskar.

7.2 Identifiering

Vid identifieringar av tankprocessen användes de omtalade identifieringsmetoderna. Oavsett vilken metod som användes erhålls ungefär samma resultat. Konvergenstiden var i båda fallen lång, flera timmar, kanske något kortare för utsignalfelsmetoden.

Det hände dock att prediktionsfelsmetoden fastnade i ett lokalt minimum, dvs uppdateringen av vikterna avstannade fastän att det globala minimat inte nåtts. Detta kunde man emellertid eliminera om momentterm infördes i uppdateringsalgoritmen, 0.8-0.9 är ett vanligt värde på denna term och lämpade sig väl i denna tillämpning.

Något som förvånade många iakttagare var att redan efter några få sampel hade nätverket identifierat processen. Det är en sanning med stor modifikation, ty avstannades uppdateringen kunde man tydligt se att nätverket i själva verket var långt ifrån färdig med identifieringen. Detta beteende kunde minskas om steglängden minskades. Steglängden hade med andra ord den inverkan att sista lagrets neurons vikter justerades till nästan total överensstämmelse mellan nätverkets utsignal och jämförelsesignalen. Ovanliggande lagers neuronvikter uppdaterades i mycket små steg på grund av den stora ändringen i slutneuronen. Effekten av total följning redan efter några få sampel kunde i stort elimineras om sampeltiden ökades. Vid total följning var sampeltiden ofta satt till 100 ms dvs mycket kortare än processens tidskonstant som är ungefär 5 s. Ökades sampeltiden till några sekunder tog det mycket längre tid för nätverket att följa processen. Effekten av sampeltiden är egentligen att nätverket identifierar en mängd integratorer. En annan förklaring på fenomenet är att skillnaden mellan två processvärde, samplade med väldigt litet samplingsintervall, är liten. Detta leder till att nätverket kan i stort behålla sina vikter för att följa processen. Endast en liten uppdatering i slutneuronen räcker för att följningen ska erhållas vid nästa sampel.

Vid identifieringen av en process kommer nätverket att justera sina vikter efter den/de signaler som presenteras på ingångarna. Detta innebär att ett nätverk som har identifierat en process för med en viss insignal, kan inte följa processen om signalen ändras i tid eller amplitud. Ska nätverket klara av även detta måste alla tänkbara insignaler läggas på processen, dvs insignalen måste vara slumpmässig. En PRBS-signal är nog att föredra. Tyvärr har en sådan generator inte implementerats, men man borde kanske gjort det. Istället kan amplituden hos både sinus- och stegsignalen genereras slumpmässigt.

7.3 Reglering

Av de beskrivna reglerstategierna i kapitel 4 är det endast direkt MRAS-reglering som simulerats. Målet med detta examensarbete är ju att försöka finna en enkel regulator som är så långt som möjligt adaptiv.

Vid specifikationen av modellreferensen måste villkoret (4.3) vara uppfyllt, annars har nätverket ingen möjlighet att justera vikterna till ett korrekt värde. Vidare kan inte modellreferensens tidskonstant vara mindre än själva processen, dvs modellen är snabbare än processen. I dessa fall kommer visserligen nätverket att justera vikterna till ett nästa korrekt värde, men eftersom att reglerfelet, skillnaden mellan processutsignal och referenssignal, kommer vara skild ifrån noll, kommer uppdatering att fortgå i all oändlighet. Detta kan leda till att regulatorn efter en tid börjar reglera sämre och sämre. När väl regulatorn är ur balans kan vikternas värden komma längre och längre ifrån de optimala värdena, men oftast justeras vikterna tillbaka till det optimala läget. Med anledning av detta resonemang måste man åtminstone veta processordning och tidskonstant för att erhålla en bra regulator.

Steglängden är även här en viktig parameter. Är steglängden stor, >0.01, kommer det totala systemet, regulator och process, att bli instabilt. För att erhålla ett bra resultat måste man vara mycket försiktig med steglängdens parameterval.

Vidare finns det en koppling mellan steglängd och samlingsperiod. Då samplingsperioden är lång kan man inte ta ta ett litet steg under minimeringen. Processens tillstånd har förändrats relativt mycket mellan två sampel, detta ger att en större ändring av nätverkets vikter är befogad. I annat fall kommer det att ta mycket längre tid för nätverket att konvergera. I det omvända fallet, när sampelperioden är liten, kan man inte ta allt för stora steg. Procesens tillstånd har inte ändrats så mycket och det kan vara ödesdigert att ta ett för stort steg i minimeringen, det totala systemet kan bli instabilt.

Försök med adaptiv steglängd resulterade i en bra fungerande regulator, naturligtvis med vissa reservationer. Steglängden för inte vara för stor. Detta resulterar i att regulatorn blir instabil. Valet av de andra parametrarna är också ganska kritiskt. Ett felaktigt val leder till att regulatorn blir instabil, på grund av för stor steglängd. Ofta måste inlärningsproceduren övervakas med jämna mellanrum. Man får justera parametrarna efterhand som träningen av nätverket fortgår.

Att reglera bort störningar är inget problem om nätverket tränats då störningar funnits i insignalerna till nätverket. I annat fall kommer regulatorn att få ett sämre beteende då störningar uppträder. Å andra sidan kan regulatorn kompensera för detta om adaptionen är påslagen, då störningen uppträder.

Liksom tidigare är vare sig storleken eller antalet insignaler avgörande för konvergensen. Ett trelagers nätverk med tio, fem och en neuron/-er i respektive lager är tillräckligt. Däremot kan antalet insignaler vara avgörande för konvergens eller ej. I det fall fördröjningskedjorna är för små konvergerar nätverket inte till ett optimalt läge. Regulatorn kommer dock att reglera men med något försämrat resultat. Om däremot fördröjningskedjorna är för långa, kommer nätverket att konvergera och man får en optimal reglering. Konvergenstiden förlängs dock något på grund av att fler vikter måste justeras. Slutsatsen av detta är att större nät konvergerar säkrare än ett mindre.

8 Sammanfattning

Neurala nätverk är ett ganska nytt verktyg för approximering av funktioner. Teorierna kom redan på 50-talet med det är först nu som de neurala nätverken har en framtid. Detta beroende på att beräkningskapaciteten måste vara stor. Ju större nätverk man konfigurerar desto fler beräkningar måste utföras. Sambandet mellan antalet beräkningar tidsenhet/steg förhåller sig som antalet neuroner i kubik.

De neurala nätverken approximerar olinjära funktioner på ett mycket effektivt sätt. Denna egenskap gör nätverken intressanta i reglertekniska sammanhang. Alla processer är i praktiken olinjära och genom att använda olinjära regulatorer kan man uppnå en bättre reglering.

Nätverken som studerats i detta examensarbete har uppdaterat vikterna efter backpropagation-metoden. Felet mellan nätutsignal och önskad utsignal propageras baklänges i nätet och vikterna uppdateras enligt en gradientalgoritm. Denna algoritm är mycket enkel och lättbegriplig. Algoritmen är dessutom lätt att implementera. Nackdelen är att konvergenstiderna är ganska långa. Redan vid ganska små nätverk konvergerar en vanlig LMS-algoritm fortare än ett neuralt nätverk, dock utan att approximera olinjäriteter på ett lika effektivt sett som de neurala nätverken.

Nätverk med varianter av backpropagation har också studerats. Lägger man till en momentterm vid uppdatering av vikterna, kan man undvika att konvergera i lokala minima. Detta förutsätter dock att momenttermen är satt till ett lämpligt värde. Steglängden är en viktig parameter. Sätts steglängden till ett för lågt värde konvergerar algoritmen långsammare samtidigt som man lättare kan konvergera i lokala minima. Om däremot steglängden sätts till ett för stort värde kommer uppdateringen att få ett oscillativt beteende. Några bra adaptiva algoritmer finns inte vid inställandet av steglängden. Det presenteras dock en adaptiv algoritm i kapitel två men den är inte helt robust.

Vid identifiering av processer kan man arbeta efter två olika principer; prediktionsfelsmetoden (PE) och utsignalfelsmetoden (OE). Skillnaden ligger i att PE inte använder några utsignaler från nätverket som insignaler utan endast in- och utsignaler till respektive från processen. Däremot återkopplas utsignalen från det neurala nätverket till nätingången vid OE. Nackdelen med att använda OE är att en minimering enligt denna metod kan avstanna i lokala minimum. Vid användandet av PE uppträder det aldrig några lokala minimum, utan endast globala sådana.

Fördelen med att använda neurala nätverk vid reglering är att man inte behöver full processkunskap som i de flesta andra adaptiva reglermetoder. Man kan arbeta efter två olika principer; direkt och indirekt adaptiv reglering. Används den direkta metoden uppdateras vikterna i regulatorn direkt med avseende på reglerfelet. Den indirekta metoden identifieras först processen. Den identifierade processmodellen används sedan som emulator för att generera jämförelsesignaler till regulatornätverket. För tillämpningar som SattLine är den direkta metoden att föredra, ty målet är att koppla in en svart låda som sköter allting själv. Nackdelen med denna variant av adaption är att konvergensen är mycket lång. Detta beror på att steglängdsparametern måste sättas till ett mycket litet värde. I annat fall kommer regulatorn att ge ett instabilt system.

Det bör dock påpekas att om neuronnätstillämningar ska få en kommersiell framgång måste man forska vidare på algorilmer vad gäller uppdatering av vikter och adaption av steglängd. Algoritmerna som presenteras i detta arbete är allför långsamma om man jämför med självinställande PID-regulatorer.

Avslutningsvis vill jag tacka min handledare, Rolf Johansson på Institutionen för Reglerteknik vid Tekniska Högskolan i Lund, för värdefull hjälp och intressanta synpunkter. Dessutom vill jag tacka Ulf Hagberg på SattControl i Malmö för all behövlig hjälp för att implementera stödet för neurala nätverk i SattLine .

Referenser

[1] Karl Johan Åström, Björn Wittenmark. *Adaptive Control,* Reading Mass, Addison-Wesley Publ Comp 1989.

[2] Rolf Johansson. *System modeling and identification.* Engelwood Cliffs NJ, Prenctice-Hall Inc 1993

[3] John Hertz, Anders Krough, Rickard G Palmer. *Introduction to the theory of Neural computation.* Redwood City CA, Addison-Wesley Publ Comp 1991

[4] W Thomas Miller III, Richard S Sutton, Paul J Werbos. *Neural Networks for Control.* Cambridge Mass, MIT Press 1990

[5] Kumpati s Narendra, Kannan Parthasarathy. "Identification and Control of Dymanical Systems using Neural Network". *IEEE Trans on Neural Networks Vol 1*, No 1, 1990, s 4-27

[6] Snehasis Mukhopadhyay, Kumpati S Narendra. "Disturbance rejection in Nonlinear Systems Using Neural Netwoks". *IEEE Trans on Neural Networks Vol 4*, No 1, 1993, s 63-72

[7] Asriel U Levin, Kumpati S Narendra. "Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization". *IEEE Trans on Neural Networks Vol 4*, No 2, 1993, s 192-206

[8] Yoshiaki Ichikawa, Toshiyuki Sawa. "Neural Network Application for Direct Feedback Controllers". *IEEE Trans on Neural Networks Vol 3*, No 2, 1992, s 224-231.

[9] Marzuki Kahlid, Sigeru Omatu. "A Neural Network Based Controller Scheme with an Adaptive Neural Model Reference structure". *Proceedings – 1991 IEEE International Joint conference on Neural Networks*, s 2128-2133.

[10] Marzuki Kahlid, Rubiyah Yosof, Sigeru Omatu. "A Comparison Among Neuro-Contrl, Self-Tuning Adaptive Control and Conventional Feedback Control Approach on a Process Control System". *Proceedings – Japan / USA symposium on Flexible Automation ASME -1992, vol 2*, s 189 - 196.

[11] Peter M Mills, Moses O Tadé, Albert Y Zomaya. "Adaptive Non-Linear Control Using Neural Networks". *Proceedings – Control -92 - Enhancing Australias Productivity Trough Automation, Control and Instrumentation*, 1992 s 7 - 13.

[12] Dennis Nilsson. "Neurala Nätverk". Institutonen för Datalogi och Numerisk Analys, Lunds Tekniska Högskola, 1992, LU-CS-EX:92-15.

A Algoritm - BP

Nedan följer en algoritm för uppdatering av vikterna i ett godtyckligt backpropagationnät. Vi betraktar ett nätverk med M lager, $m=1,2,...,M$, där V_i^m betecknar utsignal från nod i i lager m. V_i^0 är det samma som ξ_i, den i:te insignalen. Observera att överindex m beteckna lager och inte mönster. Vi låter w_{ij}^m beteckna vikterna mellan V_j^{m-1} och V_i^m. Algoritmen blir som följer:

1. Initiera vikterna till små slumpmässiga värden
2. Välj ett mönster ξ_k^μ och lägg det till inlagret (m=0), så att

$$V_k^0 = \xi_k^\mu \qquad \text{för alla } k \tag{A.1}$$

3. Propagera signalerna framåt genom nätet genom att använda för varje i tills dess att den slutliga utsignalen V_i^m har beräknats.

$$V_i^m = g(h_i^m) = g(\sum_j w_{ij}^m V_j^{m-1}) \tag{A.2}$$

4. Beräkna felet vid utlagret genom att jämföra aktuell utsignal V_i^m med önskad utsignal ζ_i^μ för det givna mönstret μ

$$\delta_i^m = g'(h_i^M)[\zeta_i^\mu - V_i^M] \tag{A.3}$$

5. Beräkna deltan till det förgående lagret genom att propagera felet baklänges för alla $m = M,M-1,...,2$ tills delta har beräknats för varje nod i nätverket.

$$\delta_i^{m-1} = g'(h_i^{m-1})\sum_j w_{ji}^m \delta_j^m \tag{A.4}$$

6. Uppdatera vikterna genom att använd följande uttryck

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1} \tag{A.5}$$

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}^m \tag{A.6}$$

7. Gå till punkt 2 och fortsätt men nästa mönster.