Elektronisk Institut
Danmarks Tekniske Universitet
Bygning 349
2800 Lyngby

Eksamensprojekt

Tidsserieprædiktion med rekursive neurale netværk

Udarbejdet af:

Morten With Pedersen
C888743

Abstrakt

Denne rapport omhandler rekursive neurale netværk, specielt fuldt forbundne feedback netværk. Der introduceres en lagdelt måde at opdatere disse fuldt rekursive netværk på, der kaldes *layered RTRL*. Det illustreres, at denne har klare fordele fremfor de traditionelle opdateringsmåder.

For de præsenterede arkitekturer udledes de anden afledede af den kvadratiske cost funktion, til anvendelse ved optræning med andenordens metoder samt beskæring. De anden afledede viser sig at være beregningsmæssigt meget dyre.

Layered RTRL anvendes til tidsserieprædiktion på solpletserien, og cost funktionen optimeres med både sædvanlige gradientmetoder samt andenordens metoderne Gauss-Newton samt Newtons metode. Andenordens metoderne resulterer i en dramatisk forøgelse af effektiviteten af optræningen.

Optimering med andenordens metoderne afslører store konditioneringsproblemer for Hessianmatricen med de anden afledede af cost funktionen. Der redegøres for, hvorledes disse problemer opstår, ligesom det vises, hvordan problemerne kan afhjælpes ved at udvide cost funktionen med et simpelt weight decay.

Der udføres beskæring af layered RTRL netværk med Optimal Brain Damage samt Optimal Brain Surgeon. De udførte forsøg viser, at der er stor forskel på de to metoders anvendelighed ved beskæring af rekursive netværk.

Forord

Denne rapport tjener som dokumentation for et eksamensprojekt i forbindelse med civilingeniørstudiet ved Danmarks Tekniske Universitet. Projektet blev udført på Elektronisk Institut med adjunkt Lars Kai Hansen som vejleder. Projektet blev påbegyndt i februar 1994 og afsluttet i august 1994.

Jeg vil her rette en stor tak til Henrik Madsen fra Instituttet for Matematisk Modellering for formidling af adgang til *G-baren*. Uden adgang til det store antal arbejdsstationer i databaren ville mange af de udførte forsøg ikke have kunnet lade sig gøre. Jeg vil også gerne takke Otto Mønsteds Fond for økonomisk støtte til min deltagelse i konferencen ICANN'94 i Sorrento, Italien, samt Lars Kai Hansen for godt rejseselskab. Endelig takker jeg Jan Larsen samt Claus Svarer fra neural netværks gruppen på EI for nyttige diskussioner i nødens stund.

Lyngby, den 31. august 1994

Morten With Pedersen

C888743

Indholdsfortegnelse

Indledning

Denne rapport omhandler tidsserieprædiktion med rekursive neurale netværk. Hvor der i *feedforward netværk* (se feks. [Hertz 91] for en generel introduktion til neurale netværk) som navnet antyder kun er forbindelse i een retning, fra enhederne i et lag til enhederne i det næste, kan rekursive netværk indeholde forbindelser i begge retninger imellem alle enheder, ligesom enhederne kan have forbindelse til sig selv, lokalt feedback ([Haykin 94]).

Outputtet fra et feedforward netværk vil ved præsentation af et givet eksempel udelukkende afhænge af de til eksemplet knyttede inputs. Outputtet fra et rekursivt netværk vil derimod, på grund af enhedernes indbyrdes forbindelser, yderligere afhænge af de foregående outputs, og dermed af de foregående eksempler. Rekursive netværk er således i stand til at opbygge en *hukommelse* om de tidligere præsenterede eksempler. Afhængigheden af tidligere præsenterede eksempler fordrer, at der er tidslig afhængighed imellem de præsenterede eksempler. Er dette ikke tilfældet, vil feedback ikke have mening, idet bidraget fra feedback vægtene så blot vil have karakter af støj.

I [Hertz 91] klassificeres nogle anvendelsesmuligheder for rekursive netværk som:

- **Sekvensgenkendelse.** Netværket skal producere et bestemt output, efter at en given inputsekvens har været præsenteret for netværket.
- **Sekvensreproduktion.** Netværket skal kunne generere resten af en sekvens, når det har set en del af den.
- **Temporal association.** Netværket skal generere en eller flere samtidige outputsekvenser på baggrund af en eller flere forskellige inputsekvenser.

En meget anvendt form for sekvensgenkendelse er genkendelse af sekvenser genereret af *finite state automata*[1], se feks. [Giles 92]. En anvendelse, der falder ind under kategorien temporal association, er *tidsserieprædiktion*, der er en vigtig del af den digitale signalbehandling ([Widrow 85]). Den her anvendte form for tidsserieprædiktion søger at prædiktere en værdi $\widehat{x}(t)$ fra en diskret tidsserie udfra en vektor med foregående *observerede* værdier $\mathbf{x}(t) = [x(t-1), x(t-2), \ldots, x(t-L)]$; L angiver dimensionen af *time-delay*

[1] Endelig tilstandsmaskine

rummet, der kaldes *lag space* ([Weigend 90]). For feedforward netværk er tidsserieprædiktion grundigt undersøgt i [Weigend 90, Svarer 93].

Et af de mange valg, der er forbundet med anvendelse af neurale netværk til tidsserieprædiktion, er valget af et 'optimalt' lag space L. Dette valg er langt fra altid indlysende, da L vil afhænge af det underliggende samplede dynamiske systems egenskaber, der søges modelleret. Een af de interessante egenskaber ved rekursive netværk er, at valget af L ikke er kritisk på samme måde som for feedforward netværk, da rekursive netværk som tidligere nævnt er i stand til at opbygge intern (i princippet uendelig) hukommelse om de tidligere præsenterede eksempler. Rekursive netværk kan således blot nøjes med et enkelt input $x(t-1)$ til at prædiktere $x(t)$, hvilket kan resultere i meget 'kompakte' netværk. For feedforward netværk vil det ikke give megen mening med blot et enkelt input $x(t-1)$, disse vil ikke kunne gøre det meget bedre end den bedste strategi ved random walk ([Weigend 94]), ifølge hvilken $\widehat{x}(t) = x(t-1)$.

Det teoretiske fundament for anvendelse af feedforward netværk er beviset for, at de er *universelle funktions-approksimatorer* ([Haykin 94]). For feedforward netværk er det bevist, at med en lineær output enhed kan et neuralt netværk approksimere en vilkårlig funktion med arbitrær nøjagtighed med maksimalt to lag skjulte enheder, blot der er tilstrækkelig med enheder. I beviset indgår nogle forholdsvis milde krav til de skjulte enheders aktiveringsfunktion, der alle er opfyldt af sigmoid-funktionerne, herunder den hyppigt anvendte tanh. Ydermere er det bevist, at et enkelt lag skjulte enheder er tilstrækkeligt til at approksimere en vilkårlig *kontinuert* funktion. Disse beviser er gennemført af Cybenko, se [Haykin 94] for referencer. Beviserne siger dog ikke noget om, *hvor mange* skjulte enheder, der skal til for en given type funktion.

For rekursive netværk er det tilsvarende vist ([Doya 93]), at et netværk med et lag skjulte enheder, lineære outputs og feedback fra outputtet til de skjulte enheder kan approksimere en vilkårlig *map*; en map er en beskrivelse af et dynamisk system ved et sæt af differensligninger ([Tsonis 94]). Beviset bygger netop på, at feedforward-delen i hvert tidsskridt kan approksimere en vilkårlig funktion.

I litteraturen er der efterhånden foreslået et væld af arkitekturer for neurale netværk, der indeholder feedback i en eller anden forstand. En gennemgang af nogle af disse samt referencer kan findes i [Hertz 91]. Mangfoldigheden af benyttede feedback arkitekturer blev tydeligt understreget ved professor Ah Chung Tsoi's besøg i april 1994 på Elektronisk Institut, DTU. Under hans foredrag omhandlende prædiktion af finansielle tidsserier med rekursive netværk beskrev han talrige varianter af netværk, der benyttede feedback på forskellig vis.

Den mest generelle form for rekursive netværk er de *fuldt forbundne feedback* netværk, eller *fuldt rekursive* netværk. For disse netværk gælder, at enhver enhed kan være forbundet til alle enheder, inklusive sig selv (lokalt fedback). Dette gør fuldt rekursive netværk specielt interessante, da de *i princippet* indeholder alle mindre netværk som specialtilfælde; mindre netværk kan opnås ved nulstilling af manglende vægte.

I [Hertz 91] beskrives den første foreslåede metode til at optræne et sådant netværk på, kaldet *backpropagation through time* (BPTT). For en inputsekvens af længde T består metoden i at transformere det fuldt forbundne netværk om til et feedforward netværk med T lag; hvert lag indeholder en kopi af samtlige enheder i det fuldt rekursive netværk. Vægtene imellem to lag i det 'udfoldede' netværk svarer til vægtene i det fuldt rekursive netværk til det tilsvarende tidsskridt. Opdatering af netværket fra det ene tidsskridt til det næste svarer nu til at føre outputtet fra det ene lag enheder frem til det næste, som det gøres ved opdatering af sædvanlige feedforward netværk. Dette indebærer, at det udfoldede netværk kan optrænes med almindelig backpropagation, heraf navnet på metoden.

Den store ulempe ved BPTT er, at det er nødvendigt at lagre en kopi af alle netværkets vægte for hvert af de T tidskridt, som inputsekvensen består af. Dette betyder, at metoden er særdeles upraktisk, hvis træningssættet er stort, da den så vil kræve meget store mængder af lagerplads. Metoden er derfor ikke særligt udbredt ([Hertz 91]).

En metode til optræning af fuldt rekursive netværk *uden* at duplikere hele netværket i hvert tidsskridt er beskrevet i [Williams 89]. Metoden indebærer *iterativ* beregning af vægtændringerne, således at 'udfoldning' af netværket undgås. Specielt fremhæves en *on line* metode kaldet *Real Time Recurrent Learning* (RTRL), hvor vægtene opdateres efter præsentation af hvert eksempel. RTRL åbner op for muligheden for at have uendelige inputsekvenser til optræning, hvilket ikke var muligt med BPTT.

Selvom Williams & Zipsers optræningsmetode har gjort det praktisk muligt at *anvende* fuldt rekursive netværk, er der i litteraturen meget få eksempler på anvendelser af disse netværk. Specielt er det vanskeligt at finde eksempler på anvendelse til tidsserieprædiktion, hvilket blev klart under litteraturstudiet til dette projektarbejde. I [Doya 93] bliver der også gjort opmærksom på litteraturmanglen. Endelig var der ved konferencen ICANN'94 ud af omkring 340 bidrag kun eet ([Guignot 94]), der omhandlede fuldt rekursive netværk.

Det generelle indtryk omkring fuldt rekursive netværk er, at mange taler om dem, men kun få anvender dem i praksis. Mulige årsager kan findes i indledningen til [Svarer 93]:

> *While the ultimate tool may be fully recurrent architectures, many problems arise during adaption of these. Even worse, the generalization properties of recurrent networks are not well understood, hence, model optimization is difficult.*

Lange optræningstider angives i [Guignot 94] som årsag til den manglende anvendelse af fuldt rekursive netværk. De generelle vanskeligheder med succesfuld optræning af fuldt rekursive netværk blev også bekræftet af A. C. Tsoi, der berettede om speciel langsommelig og vanskelig optræning for denne type netværk. Hans generelle råd omkring deres anvendelse var "*Don't touch them!*".

Et fælles træk for de fundne eksempler på anvendelse af fuldt rekursive netværk er, at de *alle* benytter de gradientbaserede optræningsmetoder, der er beskrevet i [Williams 89]. Dette er tilfældet i feks. [Guignot 94, Lindsey 92],[Catfolis 93, Schmidhuber 92] samt i [Giles 92, Giles 94]. A. C. Tsoi benyttede ligeledes disse *førsteordens* metoder til optræning af fuldt rekursive netværk, og havde ikke kendskab til, at der var benyttet andre optræningsmetoder.

For feedforward netværk er der opnået gode resultater med *beskæring* ([Haykin 94, Hertz 91]) af netværkene; beskæring indebærer eliminering af overflødige vægte i håbet om at tilpasse netværkets arkitektur til det forhåndenværende problem, med øget *generaliseringsevne* til følge ([Hertz 91]). Specielt opnåede Le Cun et al. øget generaliseringsevne med en brøkdel af det oprindelige antal vægte ved at benytte beskæring på deres netværk til genkendelse af håndskrevne *ZIP codes* ([LeCun 90a]). I [Svarer 93] er det ligeledes beskrevet, hvorledes generaliseringsevnen for et feedforward netværk anvendt til tidsserieprædiktion blev væsentligt forbedret ved kraftig beskæring.

Som for feedforward netværk er det for rekursive netværk vanskeligt på forhånd at bestemme, hvor mange enheder og hvilke vægte der er nødvendige for det givne problem, inden netværket optrænes. Derfor er det også relevant at udføre beskæring af rekursive netværk.

For rekursive netværk er beskæring kun afprøvet i meget begrænset omfang, og med forholdsvis primitive metoder. I [Giles 94] bortskæres efter endt optræning den *enhed*, til hvilken den mindste vægtvektor er knyttet. Med vægtvektor menes der de *indgående* vægte, mens der med 'mindste vektor' sandsynligvis menes mindst efter den Euklidiske 2-norm. Metoden er meget grov, da der fjernes en hel enhed ad gangen; den kan således kun anvendes til at give et groft overslag over det nødvendige antal enheder.

I [Guignot 94] elimineres vægtene individuelt. Deres betydning for netværket vurderes ved over et antal iterationer for hver vægt at bestemme, hvor stor en brøkdel af det samlede vægtede input til enheden, som vægten fører hen til, der stammer fra den pågældende vægts forbindelse. Er denne brøkdel under en fastsat grænse, fjernes vægten.

Dette projektarbejdes hovedformål er at indføre *andenordens* metoder i optræningen af de fuldt rekursive netværk, for herved at søge at effektivisere den vanskelige optræning af disse. Metoderne implementeres ved iterativ beregning af de afledede, på tilsvarende vis som det gøres i [Williams 89]. De andenordens metoder, som her vil blive anvendt, omfatter *Gauss-Newtons metode* ([Ljung 87, Dennis 83]), der er beregnet til at minimere summen af mindste kvadrater, netop svarende til den benyttede *cost funktion* (se [Hertz 91]):

$$\frac{1}{2}\sum_{t=1}^{T}[e(t)]^2 \quad , \quad e(t) = x(t) - \widehat{x}(t) \tag{0.1}$$

hvor $e(t)$ angiver netværkets begåede fejl for hvert af de T eksempler. Der vil også blive benyttet en approksimation til Gauss-Newtons metode, der for feedforward netværk er anvendt i feks. [Svarer 93]. Ydermere anvendes *Newtons metode* ([Hertz 91, Dennis 83]); Newtons metode kræver beregning af de anden afledede af cost funktionen (0.1), hvad der vil blive foretaget for de betragtede arkitekturer. I forbindelse med andenordens metoder kan der opstå numeriske problemer under optræningen. I dette arbejde vil der blive lagt stor vægt på at klarlægge, hvorledes de opstår, og hvordan de kan afhjælpes.

Projektarbejdet vil også introducere beskæring af rekursive netværk med metoderne *Optimal Brain Damage* ([LeCun 90b]) samt *Optimal Brain Surgeon* ([Hassibi 93]). Til forskel fra de for rekursive netværk hidtil anvendte metoder til beskæring søger OBD samt OBS at vurdere en vægts vigtighed på baggrund af dens betydning for *træningsfejlen*. Metoderne vurderer således vægtene efter deres betydning for det samlede netværks output, hvilket virker mere konstruktivt end at vurdere vægtene udfra interne forhold i netværket, uafhængigt af outputtet. OBD er for feedforward netværk med stor succes blevet anvendt i [LeCun 90a] samt [Svarer 93], mens der i [Hansen&With 94] var succes med anvendelse af OBS.

Oversigt over rapporten

Denne projektrapport består af otte kapitler samt ni appendiks. Indholdet af disse er som følger:

Kapitel 1 indeholder en gennemgang af de første- og andenordens metoder, der vil blive benyttet til optræning af de rekursive netværk. Det angives, hvilket stopkriterie der anvendes for optræningsmetoderne, ligesom det beskrives, hvordan cost funktionen udvides med simpelt weight decay for at assistere optræningen.

I **kapitel 2** beskrives nogle arkitekturer for rekursive netværk. Kapitlet indledes med en gennemgang af arkitekturen Recurrent Cascade Correlation. RCC behandles ikke yderligere i dette arbejde, men medtages af historiske årsager. For fuldt rekursive netværk beskrives Williams & Zipsers metode til opdatering samt træning. En ny lagdelt version af denne, kaldet *layered RTRL*, introduceres. For alle de beskrevne arkitekturer udledes de anden afledede af cost funktionen til anvendelse ved optræning samt beskæring. Kapitlet afsluttes med et overslag over beregningsomkostningerne ved optræning af fuldt rekursive netværk.

I **kapitel 3** sammenlignes W&Z RTRL med layered RTRL, og det illustreres, hvilke fordele layered RTRL har fremfor W&Z RTRL.

Kapitel 4 indeholder en beskrivelse af de metoder, der vil blive benyttet til beskæring af de rekursive netværk. Udtrykkene for Optimal Brain Damage samt Optimal Brain Surgeon udledes for en regulariseret cost funktion, specielt for en cost funktion udvidet med simpelt weight decay. Kapitlet afsluttes med en kort præsentation af Akaike's Final Prediction Error estimat.

Kapitel 5 indeholder en beskrivelse af solplet-serien, på hvilken de rekursive netværk afprøves. For at kunne sammenligne med feedforward netværk, angives nogle resultater, der i litteraturen er opnået for disse.

Kapitel 6 indeholder forsøg med optræning af layered RTRL på solpletserien. Kapitlet indledes med et lille eksempel, der illustrerer rekursive netværks evne til at opbygge hukommelse. Dernæst argumenteres der imod anvendelse af lokalt feedback i et lineært output. Effektiviteten af optræning med førsteordens samt de anvendte andenordens metoder sammenlignes ved kvantitative forsøg. I forsøget på at håndtere problemer, der opstår under optræningen med andenordensmetoder, deles vægtene op i to grupper med hvert sit weight decay. Det vises, hvordan visse af forbindelserne i et fuldt rekursivt netværk med lineært output medfører lineær afhængighed mellem nogle af vægtene, hvilket kan skabe problemer for andenordens metoderne. Afslutningsvis illustreres det, hvordan lineære afhængigheder imellem de partielle afledede af cost funktionen kan føre til singularitet i de anden afledede.

I **kapitel 7** introduceres beskæring af fuldt rekursive netværk med Optimal Brain Damage samt Optimal Brain Surgeon. Metoderne afprøves på et netværk, hvis optræning er beskrevet i kapitel 6.

I **kapitel 8** samles de overordnede konklusioner.

Appendiks A indeholder udledninger, der generaliserer udtrykkene for layered RTRL til at omfatte et arbitrært antal lag af skjulte, fuldt forbundne feedback enheder.

I **appendiks B** udledes der udtryk for OBD samt OBS, hvori der *ikke* gøres antagelser om, at cost funktionen er trænet til et minimum. De udledte udtryk tager således højde for en gradient, der er forskellig fra nul.

I **appendiks C** beskrives en metode til iterativ beregning af den inverse Hessian for cost funktionen. Beregningsomkostningerne for denne metode sammenlignes med de tilsvarende ved sædvanlig matrix-invertering, og det angives, hvilken metode der bør anvendes i en given situation.

Appendiks D beskriver konditionstal for matricer og den indflydelse på løsning af lineære ligningssystemer, dette kan have.

Appendiks E indeholder forsøg for fuldt rekursive netværk optrænet med Gauss-Newtons metode, der supplerer de i kapitel 6 beskrevne.

Appendiks F indeholder et forsøg, der illustrerer nogle af de problemer, som lineære afhængigheder imellem nogle af vægtene i fuldt rekursive netværk med lineært output kan betyde for optræning med andenordens metoder.

I **appendiks G** sammenlignes beregningstiden for en søgeretning for de forskellige optræningsmetoder, når de anvendes på layered RTRL. Beregningstiderne vises som funktion af netværkskompleksiteten, og væksten i denne sammenlignes med de overslag, der er foretaget i kapitel 2. Yderligere foretages der for et givet netværk en sammenligning mellem effektiviteten af førsteordens og andenordens metoderne, dels af det forbrugte antal iterationer, dels af den tid, som optræningen tog.

Appendiks H indeholder en summarisk gennemgang af det udviklede programmel, med kort beskrivelse af hver funktions opgave.

I **appendiks I** gengives en artikel, der i forbindelse med projektarbejdet er udarbejdet sammen med Lars Kai Hansen. Artiklen er optaget til præsentation ved konferencen NIPS'94, der afholdes i Denver, Colorado ultimo november 1994.

Kapitel 1

Anvendte optræningsmetoder

Når først cost funktionen er fastlagt, består *optræningen* af et neuralt netværk i at minimere denne. Denne minimering kan foretages med velkendte numeriske optimeringsmetoder, hvorfor betegnelserne optræning og optimering af et netværk vil blive benyttet i flæng. I næste kapitel beskrives en arkitektur kaldet Recurrent Cascade-Correlation (RCC). Ved optræning af denne arkitekturs skjulte enheder er der faktisk tale om *maksimering*, hvilket dog blot svarer til at bestemme minimum for funktionen med modsat fortegn. I det følgende vil optimeringen derfor betragtes som et minimeringsproblem.

1.1 Regularisering af cost funktionen

For at assistere optræningen, udvides cost funktionen med et regulariseringsled, således at der optimeres på udtrykket

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\mathbf{w}^T\mathbf{D}\mathbf{w} \tag{1.1}$$

hvor $E(\mathbf{w})$ er den sædvanlige kvadratiske cost funktion

$$E(\mathbf{w}) = \frac{1}{2}\sum_{t=1}^{T}[x(t) - V^o(t)]^2 = \frac{1}{2}\sum_{t=1}^{T}[e(t)]^2 \tag{1.2}$$

hvor $V^o(t)$ er outputtet fra netværket og $x(t)$ er den ønskede targetværdi. I det følgende vil regulariseringen have karakter af simpelt weight decay, hvor regulariseringsmatricen $\mathbf{D}$ er proportional med enhedsmatricen, $\mathbf{D} = \alpha\mathbf{I}$; simpelt weight decay beskrives yderligere i kapitel 4. Regulariseringsleddet vil således have formen

$$R(\mathbf{w}) = \frac{1}{2}\alpha\mathbf{w}^T\mathbf{w} = \frac{1}{2}\alpha\sum_{i=1}^{n_w} w_i^2 \tag{1.3}$$

hvor α er weight decay parameteren og n_w angiver antallet af vægte i netværket.

Hensigten med at indføre weight decay er primært at sikre numerisk stabilitet ved anvendelse af andenordens optimering, da det kan forekomme, at matricen af anden afledede for cost funktionen $E(\mathbf{w})$ er meget tæt på at være singulær, hvorved den inverse ikke kan beregnes. Weight decay'et indebærer, at der lægges en hyper-paraboloide ind over cost funktionen og 'trækker' vægtene mod nul, hvilket begrænser netværkets frihedsgrader. Denne begrænsning kan modvirke *overfitting* af træningseksemplerne med bedre generalisering til følge, ligesom det har en 'udglattende' effekt på cost funktionen, i og med at begrænsningen reducerer antallet af reelt opnåelige lokale minima; ved gentagne optræninger bliver de fundne løsninger så mere entydige.

Ved minimering af kvadratsummen skal α være *positiv* for at modvirke minima for numerisk store vægte. Ved maksimering af feks. korrelationen S for RCC er det maksima for numerisk store vægte, der skal undgås; α skal i dette tilfælde være *negativ*, som det påpeges i [Hansen&With 94].

1.2 Lokale minima – stopkriterie

Sædvanlige optimeringsmetoder udnytter, at gradienten i et minimumspunkt $\mathbf{w}^*$ er nul, $\nabla C(\mathbf{w}^*) = \mathbf{0}$, og søger efter disse. Generelle punkter, hvor dette er tilfældet, kaldes *stationære punkter*. Ikke alle stationære punkter er minima, disse punkter omfatter også maksimaer samt *sadelpunkter*. Bestemmende for punktets klassifikation er egenskaberne for Hessianmatricen med de anden afledede i punktet. Et stationært punkt er et minimum, hvis Hessianmatricen i dette er positivt definit. Er Hessianmatricen negativt definit, vil det stationære punkt være et maksimum. Er Hessianmatricen ikke-definit, vil det pågældende stationære punkt være et sadelpunkt (se [Tingleff 93]).

Mens det således er forholdsvis simpelt udfra Hessianmatricen at afgøre, om det fundne punkt $\mathbf{w}^*$ er et minimum, er det særdeles vanskeligt at afgøre, hvorvidt $\mathbf{w}^*$ er et *globalt* minimum eller om der findes et andet stationært punkt $\mathbf{w}'$, for hvilket der gælder $C(\mathbf{w}') < C(\mathbf{w}^*)$; er dette tilfældet, er $\mathbf{w}^*$ kun et*lokalt* minimum. Den eneste måde, hvorpå man kan sandsynliggøre, at $\mathbf{w}^*$ er global, er ved at gentage optimeringsproceduren for forskellige startpunkter; her skal der dog ydermere tages højde for, at $\mathbf{w}^*$ ikke vil være et *entydigt* globalt minimum, når optimeringen gælder optræning af neurale netværk. Dette skyldes blandt andet, at enhederne i et neuralt netværk pga. symmetri kan ombyttes, uden at netværkets funktion ændres. Denne permutation af enhederne medfører igen, at nogle af vægtene i vægtvektoren $\mathbf{w}^*$ ombyttes, således at $\mathbf{w}^*$ ikke er entydig.

I praksis er konvergens mod et lokalt maksimum ikke et problem, hvis man sikrer sig, at cost funktionen formindskes i hver iteration ([Tingleff 93]),

$$C(\mathbf{w}_{k+1}) < C(\mathbf{w}_k) \tag{1.4}$$

Dette hindrer også i vid udstrækning konvergens mod sadelpunkter, hvilket påpeges i [Tingleff 93]. Et muligt stopkriterie er $C(\mathbf{w}_{k-1}) - C(\mathbf{w}_k) < \varepsilon$, men man kan her ikke være sikker på, at det fundne $\mathbf{w}^*$ er stationært; man har ingen garanti for, at gradienten $\nabla C(\mathbf{w}^*) = \mathbf{0}$. Derfor vil der her (udover maksimum antal iterationer) blive benyttet stopkriteriet

$$\| \nabla C(\mathbf{w}_k) \|_2 < \varepsilon \tag{1.5}$$

hvor ε er en 'passende' lille konstant, og $\| \cdot \|_2$ angiver 2–normen, eller den euklidiske længde.

1.3 Gradient-baserede metoder

For at minimere cost funktionen, kan man benytte gradient-nedstigning, på engelsk *gradient descent*. Denne metode går ud på at søge efter minimum for cost funktionen i retning modsat gradienten. Metoden er meget ofte anvendt ved optræning af neurale netværk, hvor den kaldes *backpropagation* pga. måden de afledede beregnes på, se [Hertz 91]. Vægtene opdateres efter udtrykket

$$\Delta\mathbf{w} = -\eta \frac{\partial C}{\partial \mathbf{w}} = -\eta \left(\frac{\partial E}{\partial \mathbf{w}} + \alpha \mathbf{w} \right) \tag{1.6}$$

hvor $\mathbf{w}$ er en vektor indeholdende samtlige vægte i netværket, og η er en positiv konstant, kaldet *learning rate*, der angiver, hvor stort et 'skridt', der tages modsat gradientens retning. Opdatering af vægtene efter udtrykket 1.6 indebærer, at vægtene først opdateres, efter at alle træningseksempler er præsenteret for netværket, hvilket benævnes *batch mode* optræning. Alternativt kan opdateringen foretages *on line* efter præsentation af hvert eksempel,

$$\Delta\mathbf{w} = -\eta \left(\frac{1}{2} \frac{\partial e^2(t)}{\partial \mathbf{w}} + \alpha \mathbf{w} \right) \tag{1.7}$$

Der er generelt erfaring for, at on line opdateringen er en del hurtigere end batch mode, se [Hertz 91].

Konstanten η skal naturligvis vælges således, at der opnås konvergens mod et (lokalt) minimum for $C(\mathbf{w})$. Ved en for stor værdi opnås divergens, ved en for lille værdi går optræningen i stå. Når iterationspunktet $\mathbf{w}_k$ er så tæt på et (lokalt) minimum $\mathbf{w}^*$, at cost funktionen kan beskrives med en *kvadratisk approksimation* som (1.11), kan det nemt vises (se feks. [Widrow 85]), at η skal vælges således at

$$0 < \eta < \frac{2}{\lambda_{max}} \tag{1.8}$$

for at opnå konvergens mod minimumet; λ_{max} er den største egenværdi for Hessianmatricen.

Ved batch mode optræning bliver det praktisk muligt *adaptivt* at tilpasse η vha. *liniesøgning*, således at denne ikke på forhånd skal fastsættes 'manuelt'.

Ved liniesøgning forstås, at cost funktionen med udgangspunkt i et punkt $\mathbf{w}_0$ undersøges i en given retning $\Delta\mathbf{w}$ kaldet *søgeretningen*, og på denne baggrund bestemmes et η således, at udtrykket

$$C(\mathbf{w}_0 - \eta\Delta\mathbf{w}) \tag{1.9}$$

mindskes i denne retning. Hvis udtrykket 1.9 minimeres, kaldes liniesøgningen *exact linesearch*, og den resulterende gradientmetode kaldes *Steepest Descent*.

I litteraturen er der beskrevet et væld af liniesøgemetoder, se eksempelvis [Dennis 83]. I dette arbejde benyttes dog en simpel heuristisk metode, inspireret af [Svarer 93] samt [Dennis 83], der er let at implementere, og fungerer godt i praksis. Metoden er en *inexact linesearch*, da den ikke eksakt bestemmer det η, der minimerer udtrykket 1.9, men blot en approksimation til dette. Metoden er som følger:

η_{start} initialiseres til en given værdi, feks. een, og $C(\mathbf{w}_0 - \eta\Delta\mathbf{w})$ bestemmes. Hvis dette ikke er mindre end $C(\mathbf{w}_0)$, halveres η, og det resulterende nye punkt afprøves. Lykkes det ikke i et bestemt antal halveringer at opnå en nedgang i C afsluttes der; ellers undersøges det, om yderligere halvering af η giver større nedgang i C. Vægtvektoren $\mathbf{w}_0$ ændres endelig i overensstemmelse med den η_{slut}, der gav den største nedgang i C.

Gradientmetoder er *globalt konvergente* optimeringsmetoder, hvilket betyder, at de vil konvergere mod et (lokalt) minimum for cost funktionen fra stort set et hvilket som helst startpunkt ([Dennis 83]). Konvergensen er dog lineær (se [Dennis 83]), og kan være *meget* langsom 'tæt' på et optimalt punkt $\mathbf{w}^*$. Hvis den benyttede gradientmetode er steepest descent, vil iterationspunkterne $\{\mathbf{w}_k\}$ ved konvergens mod et (lokalt) minimum $\mathbf{w}^*$ opfylde

$$\begin{array}{c}\text{lim sup}\\ \text{k} \to \infty\end{array} \frac{\|\mathbf{w}_{k+1} - \mathbf{w}^*\|_2}{\|\mathbf{w}_k - \mathbf{w}^*\|_2} \leq c \;\;,\;\; c \triangleq \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right) \tag{1.10}$$

hvor λ_{max} samt λ_{min} er hhv. største og mindste egenværdi for Hessianmatricen i punktet $\mathbf{w}^*$ (se [Dennis 83]). Hvis der i dette punkt er stor forskel på største og mindste egenværdi, svarende til et stort konditionstal (se appendiks D), risikeres det derfor, at konvergensen praktisk taget går i stå.

1.4 Andenordens metoder

Andenordens metoder tager udgangspunkt i en Taylorudvikling af den udvidede cost funktion

$$C(\mathbf{w}) \approx C(\mathbf{w}_0) + \Delta\mathbf{w}^T \nabla C(\mathbf{w}_0) + \frac{1}{2}\Delta\mathbf{w}^T \mathbf{H} \Delta\mathbf{w} \tag{1.11}$$

hvor $\mathbf{w} = \mathbf{w}_0 + \Delta\mathbf{w}$, $\nabla C(\mathbf{w}_0)$ angiver gradienten af C i punktet $\mathbf{w}_0$, og $\mathbf{H} = \mathbf{A} + \alpha\mathbf{I}$ angiver Hessianmatricen for $C(\mathbf{w})$; $\mathbf{A}$ er her matricen med de anden afledede af $E(\mathbf{w})$. Heraf ses det, at weight decayet kan modvirke, at $\mathbf{H}$ bliver singulær ved at flytte egenværdierne, som det beskrives i appendiks D.

Motivationen for at benytte andenordens metoder er dels, at gradientmetoder i praksis kan have problemer med overhovedet at konvergere, dels at optræningstiden reduceres på trods af det øgede beregningsarbejde i hver iteration.

1.4.1 Newtons metode

Newtons metode består i at bestemme $\Delta\mathbf{w}$, således at der opnås minimum for udtrykket (1.11). Sættes de afledede af (1.11) mht. $\Delta\mathbf{w}$ til nul, haves

$$\begin{aligned} &\nabla C(\mathbf{w}_0) + \mathbf{H}\Delta\mathbf{w} = 0 \\ \Updownarrow& \\ &\Delta\mathbf{w} = -\mathbf{H}^{-1}\nabla C(\mathbf{w}_0) \end{aligned} \tag{1.12}$$

Den resulterende ændring af vægtvektoren $\Delta\mathbf{w}$ kaldes *Newton-skridtet*. Af ovenstående notation kan man få den opfattelse, at man for at bestemme $\Delta\mathbf{w}$ først skal invertere $\mathbf{H}$, og dernæst multiplicere denne med den negative gradient. Sædvanligvis er det dog hurtigere samt numerisk mere nøjagtigt at løse ligningssystemet $\mathbf{H}\Delta\mathbf{w} = -\nabla C(\mathbf{w}_0)$ ([Dennis 83]). Dette understøttes af [MATLAB], hvor det ligeledes anbefales at løse ligningssystemet direkte. Udover hastighed samt præcision er der dog også den fordel, at løsning af ligningssystemet er mere robust overfor en singulær Hessianmatrix. Ved singularitet kan $\mathbf{H}^{-1}$ ikke beregnes. Hvis blot $-\nabla C(\mathbf{w}_0)$ ligger i det (reducerede) billedrum for $\mathbf{H}$, vil ligningssystemet imidlertid have en løsning.

Hvis approksimationen (1.11) holder, ses Newtons metode at eliminere problemet med at bestemme learning rate η, men dette forudsætter, at approksimationen har minimum i et punkt $\mathbf{w}$, hvor der gælder $C(\mathbf{w}) < C(\mathbf{w}_0)$; hvis cost funktionen er meget ikke-kvadratisk og/eller $\mathbf{w}_0$ ligger 'langt' fra et lokalt minimum, behøver dette ingenlunde at være tilfældet. I praksis bestemmes vægtændringen derfor udfra udtrykket

$$\Delta\mathbf{w} = -\eta\mathbf{H}^{-1}\nabla C(\mathbf{w}_0) \tag{1.13}$$

hvor η bestemmes ved liniesøgningen beskrevet i afsnit 1.3.

Udtrykkene ovenfor fordrer imidlertid, at $\mathbf{H}$ er positivt definit, dvs. udgangspunktet $\mathbf{w}_0$ skal være 'tæt' på et (lokalt) minimum, ellers vil søgeretningen $\Delta\mathbf{w}$ være 'op ad bakke'. I praksis forekommer det under optræningen temmelig ofte, at $\mathbf{H}$ *ikke* er positivt definit, hvilket der må tages højde for ved implementering. Dette gøres ved at bestemme fortegnet for η således, at

$$\angle(-\nabla C(\mathbf{w}_0), \Delta\mathbf{w}) < 90^o \tag{1.14}$$

hvilket gøres udfra fortegnet af prikproduktet mellem gradienten og søgeretningen:

$$\text{sign}(\eta) = -\text{sign}(\nabla C(\mathbf{w}_0)\cdot\Delta\mathbf{w}) \tag{1.15}$$

I de tilfælde, hvor η for Newtons metode ovenfor bliver negativ, bliver den resulterende søgeretning ret tilfældig. Kravet (1.4) sørger dog for, at en sådan tilfældig søgeretning ikke får katastrofale konsekvenser for konvergensen mod et lokalt minimum. Alternativt kunne man tage et skridt med gradientmetoden beskrevet ovenfor, eller med Gauss-Newton metoden, der beskrives nedenfor.

Newtons metode er en *lokalt konvergent* optimeringsmetode, hvilket betyder, at den konvergerer mod et (lokalt) minimum, forudsat man er tilstrækkelig tæt på dette; er man for langt væk, risikeres det derimod, at metoden konvergerer mod et maksimum eller mod et sadelpunkt ([Dennis 83]), med mindre søgeretningen inverteres, som det beskrives ovenfor. Newtons metode bør derfor altid indledes med en globalt konvergent metode, som for eksempel gradient descent. Tilstrækkelig tæt på et minimum $\mathbf{w}^*$ har Newtons metode den egenskab, at den har *kvadratisk konvergens* mod $\mathbf{w}^*$ (se [Dennis 83] for definition), hvilket kan tolkes således, at antallet af korrekte cifre fordobles i hver iteration.

1.4.2 Gauss-Newtons metode

Hessianmatricen $\mathbf{H}$ for den udvidede cost funktion $C(\mathbf{w})$ indeholder de anden afledede for den kvadratiske cost funktion (1.2), hvor

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} = -\sum_{t=1}^{T}\left[e(t)\frac{\partial^2 V^o(t)}{\partial w_i \partial w_j} - \frac{\partial V^o(t)}{\partial w_i}\cdot\frac{\partial V^o(t)}{\partial w_j}\right] = A_{ij} \tag{1.16}$$

Leddet indeholdende de anden afledede af netværkets output kan være vanskeligt at bestemme, og desuden krævende at beregne. Hvis det negligeres, resulterer dette i approksimationen

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} \approx \sum_{t=1}^{T}\frac{\partial V^o(t)}{\partial w_i}\cdot\frac{\partial V^o(t)}{\partial w_j} \tag{1.17}$$

Anvendes denne approksimation til Hessianmatricen ved minimering af 1.11 som beskrevet for Newtons metode, kaldes metoden for *Gauss-Newtons metode*. Gauss-Newton approksimationen til Hessianmatricen bestemt ved 1.17 har den fordel, at den er positivt definit, således at søgeretningen for ikkesingulære approksimationer aldrig er 'op ad bakke' (se [Dennis 83]).

Gauss-Newtons metode er væsentlig mere robust end Newtons metode, idet den udover at være lokalt konvergent også besidder globale konvergensegenskaber, da søgeretningen altid er 'ned ad bakke'; metoden bør dog alligevel indledes med en metode med 'sikre' globale konvergensegenskaber, som for eksempel gradient descent. De lokale konvergensegenskaber afhænger af, hvor meget det udeladte andenordens led betyder. Er residualfejlene $e(t)$ nær et optimalt punkt $\mathbf{w}^*$ små, eller er krumningen nær dette ringe, vil Gauss-Newtons metode have kvadratisk slutkonvergens, som det er tilfældet for Newtons metode; i modsat fald vil slutkonvergensen normalt være lineær ([Madsen 91a]). En vurdering af andenordensleddets betydning er dog ikke simpel at foretage, idet der tilsyneladende ikke findes nogen generelt anerkendt måde at gøre dette på ([Nielsen 94]).

Anvendes Gauss-Newton skridtet som søgeretning for en liniesøgning som den beskrevet i afsnit 1.3, skal metoden egentlig betegnes *damped* Gauss-Newton ([Dennis 83]).

1.4.3 Pseudo Gauss-Newton

Selvom Gauss-Newton approksimationen 1.17 til Hessianmatricen er væsentlig simplere at beregne end den fulde Hessian, skal der stadig løses et ligningssystem i hver iteration for at bestemme søgeretningen. Udtrykket 1.17 simplificeres derfor ofte yderligere med en diagonal-approksimation, se feks. [LeCun 90b, Svarer 93], idet det antages, at diagonal-elementerne er de mest dominerende, således at de øvrige elementer kan negligeres, hvilket udtrykkes som $\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} = 0 \;\; , \;\; i \neq j$. Vægten w_i opdateres nu efter

$$\Delta w_i = -\eta \left(\frac{\partial E(\mathbf{w})}{\partial w_i} + \alpha w_i \right) \Big/ \left(\frac{\partial^2 E(\mathbf{w})}{\partial w_i^2} + \alpha \right) \tag{1.18}$$

hvilket er ækvivalent med at benytte Gauss-Newtons metode seperat på hver vægt for sig; $\frac{\partial^2 E(\mathbf{w})}{\partial w_i^2}$ beregnes udfra (1.17). Weight decayet ses her at være afgørende for den numeriske stabilitet, idet nævneren ellers ville være omtrent nul i nærheden af et stationært punkt. Metoden (1.18) vil i det følgende blive betegnet *pseudo Gauss-Newton*.

Konvergensegenskaberne for denne metode omtales ikke i den sædvanlige optimeringslitteratur. Den vil dog sandsynligvis være væsentligt langsommere end Gauss-Newton, grundet den grove approksimation.

Udover de ovenfor anførte eksisterer der et væld af optimeringsalgoritmer, feks. konjugerede gradienters metode, diverse quasi-Newton med flere; se feks. [Dennis 83]. Der har ikke været ønske om at afprøve disse metoder i dette arbejde, idet det primære formål her er at introducere andenordens metoder samt approksimationer til disse for rekursive neurale netværk.

Kapitel 2

Arkitekturer

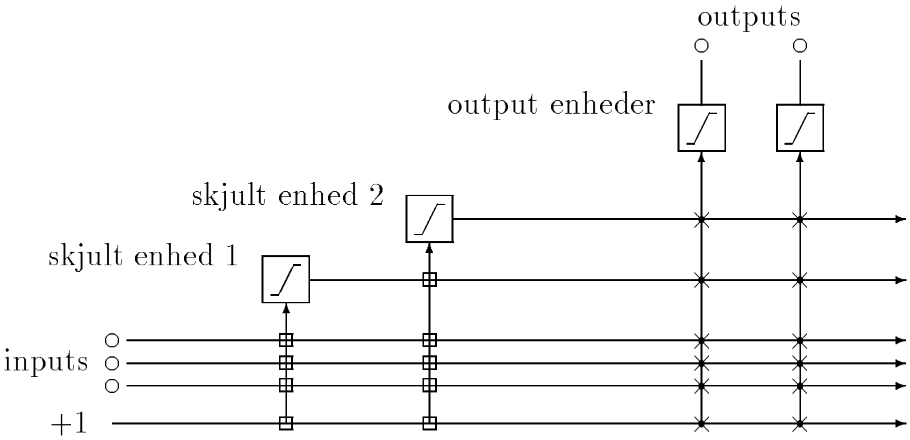
I dette kapitel gennemgås de grundlæggende arkitekturer for de benyttede netværk samt udtryk til anvendelse ved optræning. Ved arkitekur forstås dels, hvordan enhederne i netværket er indbyrdes forbundet samt forbundet til inputtet, dels hvordan opdateringen af enhedernes output foretages. Netværkstyperne omfatter Recurrent Cascade Correlation og fuldt forbundne feedback netværk opdateret med hhv. Williams & Zipsers RTRL samt en her foreslået ændring af denne kaldet layered RTRL.

Recurrent Cascade-Correlation behandles ikke yderligere i dette arbejde, men beskrivelsen af arkitekturen medtages af historiske årsager, idet det var den første feedback arkitektur, jeg beskæftigede mig med. En afprøvning af RCC på solplet-serien kan findes i [MWP 93]. Yderligere illustrerer arkitekturen den vel nok simpleste måde at indføre feedback i et netværk på.

2.1 Recurrent Cascade-Correlation

Cascade-Correlation arkitekturen er beskrevet i [Fahlman 90], og er en lidt utraditionel arkitektur, idet den starter med et lille netværk uden skjulte enheder. Enheder indsættes een ad gangen efter hinanden, heraf navnet på arkitekturen, og når en enhed først er indsat, ændres dens vægte ikke. Hver indsat enhed er forbundet ikke bare til de eksterne inputs til netværket, men også til alle de tidligere indsatte enheders output, som det er illustreret på figur 2.1. På denne figur er afbildet flere outputenheder, men da arkitekturen her skal anvendes til tidsserieprædiktion, antages det i den følgende udledning, at der kun er en enkelt outputenhed.

Den sidste del af arkitekturens navn refererer til den strategi, der benyttes ved optræning af en skjult enhed. Enhedens opgave i netværket er at reducere outputenhedens fejl mest muligt, og dette søges gjort ved at maksimere korrelationen mellem den skjulte enheds output og outputenhedens residualfejl, $e(t) = x(t) - V^o(t)$. Når den skjulte enhed indsættes i netværket, vægtes dens output på passende vis ved fornyet optræning af outputenheden, hvorved residualfejlen for det samlede netværks output reduceres.



Figur 2.1: Cascade arkitekturen efter at to enheder er indsat. Vægtene markeret med firkanter er konstante efter indsættelse, vægte markeret med X trænes løbende efter indsættelse af en ny skjult enhed.

Outputenhedens output $V^o(t)$ til tiden t defineres som

$$V^o(t) = f\left[\sum_{i=1}^{n_I} w_i x_i(t) + \sum_{j=1}^{n_S} w_j V_j^h(t)\right] \tag{2.1}$$

hvor n_I angiver antallet af eksterne inputs $x_i(t)$, og n_S angiver antallet af skjulte enheder med output $V_j^h(t)$. Sædvanligvis søges det for outputenheden at minimere den kvadratiske cost funktion

$$E(\mathbf{w}) = \frac{1}{2}\sum_{t=1}^{T}[e(t)]^2 \tag{2.2}$$

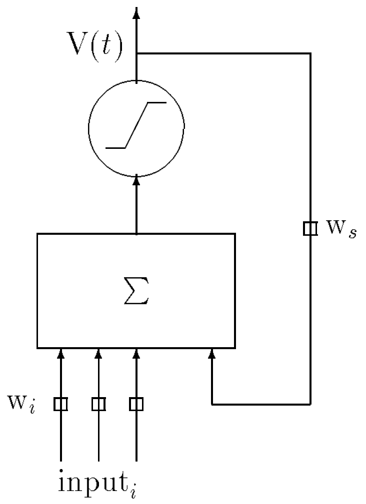
Optræning af outputenheden er forholdsvis simpel at foretage, da det ikke er nødvendigt af føre fejlen tilbage gennem de skjulte enheder, idet disses vægte holdes konstante. Der vil således blot være tale om at træne en enkelt enhed op; hvis denne enhed er en lineær feedforward enhed, kan dette feks. gøres i en enkelt iteration med matrixmetoden beskrevet i [Barton 90].

Når outputenheden er er færdigtrænet, og hvis man stadig ikke er tilfreds med netværkets præstation, kan det hjælpe at optræne en såkaldt kandidatenhed, og derefter indsætte denne som skjult enhed i netværket. Til dette har man som nævnt brug for outputenhedens residualfejl, så disse må bestemmes og gemmes.

Recurrent Cascade-Correlation er beskrevet i [Fahlman 91]. Ændringen i forhold til almindelig Cascade-Correlation består i, at de skjulte enheder har lokalt feedback, se figur 2.2; sættes w_s til nul, opnås standard CC.

Lad $\mathbf{x}(t)$ være en vektor indeholdende dels de eksterne inputs, dels outputs fra tidligere skjulte enheder til tiden t, og lad I angive sættet af indices for $\mathbf{x}$. Kandidatenhedens output til tiden t defineres som

$$V(t) = f[s(t)] = f\left[\sum_{i\in I} x_i(t) w_i + V(t-1) w_s\right] \tag{2.3}$$



Figur 2.2: Illustration af rekursiv enheds virkemåde.

hvor f er aktiveringsfunktionen, og w_s angiver feedbackvægten.

Ved optræning med gradientmetoder er der brug for de afledede af de skjulte enheder. De afledede af $V(t)$ er

$$\frac{\partial V(t)}{\partial \mathbf{w}} = \frac{\partial f[s(t)]}{\partial s(t)} \cdot \frac{\partial s(t)}{\partial \mathbf{w}}$$

hvor

$$\frac{\partial s(t)}{\partial w_i} = x_i(t) + w_s \frac{\partial V(t-1)}{\partial w_i} \quad , \quad i \in I$$

og

$$\frac{\partial s(t)}{\partial w_s} = V(t-1) + w_s \frac{\partial V(t-1)}{\partial w_s}$$

Ved optræning af en kandidatenhed søges det ved gennemløb af træningssættet at *maksimere* absolutværdien af korrelationen S mellem kandidatens output V samt outputenhedens residualfejl $e(t)$. S er defineret ved udtrykket

$$S = \left| \sum_{t=1}^{T} (V(t) - \overline{V})(e(t) - \overline{e}) \right|$$

hvor

T angiver antallet af træningseksempler,
$\overline{V}$ er middelværdien af $V(t)$ over alle tider t,
$e(t)$ er output-enhedens residualfejl til tiden t, og
$\overline{e}$ er middelværdien af output-enhedens residualfejl over
alle tidspunkter t i træningssættet.

Ved optræning med gradient *opstigning* opdateres vægtene for kandidatenheden efter udtrykket

$$\delta \mathbf{w} = \eta \frac{\partial S}{\partial \mathbf{w}}$$

hvor η er en positiv konstant. De afledede af S er

$$\frac{\partial S}{\partial \mathbf{w}} = \sigma \sum_{t=1}^{T} (e(t) - \overline{e}) \frac{\partial V(t)}{\partial \mathbf{w}} \quad , \quad S \neq 0$$

Her er σ fortegnet for udtrykket mellem numerisktegnene i S.

Ved præsentation af det første eksempel til tiden $t = 1$ sættes kandidatens tidligere tilstand til 0, dvs.

$$\begin{aligned} V(0) &= 0 \\ \frac{\partial V(0)}{\partial w_s} &= 0 \end{aligned}$$

Idet kandidaten ikke har nogen forbindelse til netværket under dens optræning, er det muligt at optræne flere enheder uafhængigt af hinanden. Når alle enheder i denne *pool* er optrænet, sammenlignes deres slutværdi for S, og enheden med den største værdi indsættes i netværket. Dette tiltag kan hjælpe med til at reducere sandsynligheden for, at en ubrugelig enhed indsættes permanent i netværket.

Det faktum, at en skjult enheds vægte ikke ændrer sig, når den først er indsat i netværket, kan udnyttes til at reducere antallet af nødvendige beregninger betydeligt. For hver skjult enhed kan man umiddelbart før dennes indsættelse i netværket lagre dens output for alle de benyttede input-mønstre i både trænings- samt testsæt. Dette betyder særdeles kraftig reducering af optræningstiden.

Hessian-matrix for Recurrent CC

I [MWP 93] blev der blot benyttet simple gradientmetoder til optræning af kandidatenhederne; her skal der imidlertid angives udtryk, der feks. muliggør anvendelse af andenordens metoder til optræning af disse.

Til brug feks. ved optræning med Newtons metode eller OBS-pruning har man brug for de anden afledede af korrelationsmålet S. I denne udledning antages det igen, at netværket kun har en enkelt outputenhed, som det er tilfældet for tidsserieprædiktion.

Lad $r(t)$ betegne $\sigma(e(t) - \overline{e})$. Idet σ er en trinfunktion, vil $r(t)$ være en konstant undtagen i specialtilfældet $S = 0$, hvor σ ikke er differentiabel. De anden afledede for $S \neq 0$ er

$$\frac{\partial^2 S}{\partial w_i \partial w_j} = \sum_{t=1}^{T} r(t) \frac{\partial^2 V(t)}{\partial w_i \partial w_j}$$

Herefter beregnes de anden afledede for outputtet $V(t)$. For $i \in I$, $j \in I$ haves

$$\frac{\partial^2 V(t)}{\partial w_i \partial w_j} = \frac{\partial^2 f[s(t)]}{\partial s(t)^2} \cdot \frac{\partial s(t)}{\partial w_i} \cdot \frac{\partial s(t)}{\partial w_j} + \frac{\partial f[s(t)]}{\partial s(t)} w_s \frac{\partial^2 V(t-1)}{\partial w_i \partial w_j}$$

For $i \in I$ haves

$$\begin{aligned} \frac{\partial^2 V(t)}{\partial w_i \partial w_s} &= \frac{\partial^2 f[s(t)]}{\partial s(t)^2} \cdot \frac{\partial s(t)}{\partial w_i} \cdot \frac{\partial s(t)}{\partial w_s} + \frac{\partial f[s(t)]}{\partial s(t)} \left(w_s \frac{\partial^2 V(t-1)}{\partial w_i \partial w_s} + \frac{\partial V(t-1)}{\partial w_i} \right) \\ &= \frac{\partial^2 V(t)}{\partial w_s \partial w_i} \end{aligned}$$

og endelig

$$\frac{\partial^2 V(t)}{\partial {w_s}^2} = \frac{\partial^2 f[s(t)]}{\partial s(t)^2} \cdot \left(\frac{\partial s(t)}{\partial w_s} \right)^2 + \frac{\partial f[s(t)]}{\partial s(t)} \left(2 \frac{\partial V(t-1)}{\partial w_s} + w_s \frac{\partial^2 V(t-1)}{\partial {w_s}^2} \right)$$

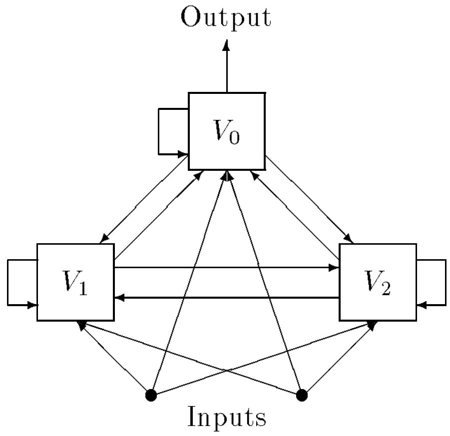
Det bemærkes, at der i udtrykkene indgår afledede samt anden afledede fra $t-1$, et tidsskridt tilbage, og at disse således afhænger rekursivt af samtlige forudgående udtryk. Ved implementering kan man dog udnytte, at de afledede beregnes inkrementelt, således at udtryk beregnet til tiden t indgår i opdateringen til tiden $t+1$.

2.2 Fuldt forbundne feedback netværk

Ved fuldt forbundne feedback netværk forstås, at alle enheder modtager samtlige inputdata, og samtidig er forbundet med outputtet fra samlige enheder. I figur 2.3 er illustreret et eksempel på et sådant netværk.

Fastlæggelse af sådanne netværks input-output relationer samt optræning kan foretages på flere måder. I dette afsnit gives et resumé af Williams & Zipsers metode, ligesom der gives en beskrivelse af en ny feedforward-inspireret måde at opdatere denne netværkstype på, der kaldes *layered RTRL*.

Som det vil fremgå af det følgende, dækker betegnelsen RTRL rettelig over en metode til optræning af fuldt forbundne feedback netværk. I daglig tale er de fuldt forbundne feedback netværk imidlertid omtalt som RTRL-netværk, hvorfor RTRL i det følgende vil blive benyttet i forbindelse med begge begreber; udfra sammenhængen vil der dog ikke være megen tvivl om, hvad der menes.



Figur 2.3: Fuldt forbundet feedback netværk med tre enheder.

2.2.1 Williams & Zipser – RTRL

Williams & Zipsers metode er beskrevet i [Williams 89]. Udgangspunktet er, at samtlige enheder i netværket skal kunne modtage eksterne inputs, ligesom der i hvert tidsskridt skal kunne defineres en target-værdi for en vilkårlig enhed. Dette opnår de ved at lade netværket fungere som en tilstandsmaskine, hvor hver enkelt enhed opdateres udfra en tilstandsvektor $\mathbf{z}$, der indeholder det samlede netværks tilstand til tidsskridtet før. Opdateringen af netværket defineredes oprindelig som følger:

Lad $\mathbf{x}(t)$ angive de m eksterne inputs til netværket til tiden t, og lad $\mathbf{V}(t)$ angive de n enheders output til tiden t. Ved netværkets tilstand til tiden t forstås vektoren $\mathbf{z}'(t)$,

$$z'_k(t) = \begin{cases} x_k(t) & , \quad k \in I \\ V_k(t) & , \quad k \in U \end{cases}$$

hvor I angiver indices tilhørende de m inputs, og U angiver indices tilhørende de n skjulte enheder. Outputtet fra den k'te enhed til tiden t defineres som

$$\begin{aligned} V_k(t) &= f_k\left[s'_k(t-1)\right] \\ &= f_k\left[\sum_{j \in I \cup U} w_{kj} z'_j(t-1)\right] \\ &= f_k\left[\sum_{j \in I} w_{kj} x_j(t-1) + \sum_{j \in U} w_{kj} V_j(t-1)\right] \end{aligned}$$

hvor f_k angiver aktiveringsfunktionen for den k'te enhed, og w_{kj} angiver vægten til enheden k fra enhed/input j.

Ovenstående fremgangsmåde kan beskrives på en fuldstændig ækvivalent form, der falder i tråd med den traditionelle måde at beskrive input-output relationerne for feedforward netværk på, samt letter overskueligheden, når netværkstypen anvendes til tidsserieprædiktion.

Vi lader nu $\mathbf{z}(t)$ beskrive inputtet til hver enkelt enhed til tiden t,

$$z_k(t) = \begin{cases} x_k(t) & , \quad k \in I \\ V_k(t-1) & , \quad k \in U \end{cases}$$

Ved tidsserieprædiktion er inputs ekskl. bias typisk $\mathbf{x}(t) = [x(t-1), x(t-2), \ldots, x(t-L)]$, hvor L er dimensionen af *lag space*. Outputtet fra den k'te enhed til tiden t defineres nu som

$$\begin{aligned} V_k(t) &= f_k\left[s_k(t)\right] \qquad (2.4) \\ &= f_k\left[\sum_{j \in I \cup U} w_{kj} z_j(t)\right] \\ &= f_k\left[\sum_{j \in I} w_{kj} x_j(t) + \sum_{j \in U} w_{kj} V_j(t-1)\right] \end{aligned}$$

Det bemærkes, at omskrivningen i forhold til den oprindelige definition blot består af en omordning af tidsindeks; de efterfølgende udledninger vil blive foretaget for denne omskrivning.

For en vilkårlig enhed k kan der specificeres en target-værdi (ønsket output) $\zeta_k(t)$ til tiden t. Hvis Ω angiver indices for enheder, hvor en target-værdi er specificeret, kan fejlmålet til tiden t for enhederne defineres som

$$e_k(t) = \begin{cases} \zeta_k(t) - V_k(t) & k \in \Omega \\ 0 & \text{ellers} \end{cases}$$

Cost-funktionen for netværket til tiden t er defineret til

$$E(t) = \frac{1}{2} \sum_{k \in U} \left[e_k(t)\right]^2$$

og den totale cost funktion er

$$E(\mathbf{w}) = \sum_{t=1}^{T} E(t)$$

hvor T angiver antallet af træningseksempler, og det første target specificeres til tiden $t = 1$.

Ved optræning kan hver enkelt vægt w_{ij} opdateres efter præsentation af et træningseksempel under anvendelse af gradient-nedstigning efter udtrykket

$$\Delta w_{ij}(t) = -\eta \frac{\partial E(t)}{\partial w_{ij}}$$

hvor η er en positiv konstant, idet gradienten for $E(\mathbf{w})$ kan separeres i de enkelte tidsskridt. Vægtene opdateres således ikke efter den negative gradient for den totale cost funktion, idet vægtene ændres gennem præsentationen af træningssættet, men snarere efter en for eksemplet (med fortid) 'lokal' gradient. Williams & Zipser kalder denne *on line* opdatering *Real Time Recurrent Learning*, RTRL.

Alternativt kan vægtene opdateres efter den negative gradient for den totale cost funktion $E(\mathbf{w})$, når alle træningseksempler er forevist (en epoke), efter udtrykket

$$\begin{aligned}\Delta w_{ij} &= -\eta \frac{\partial E(\mathbf{w})}{\partial w_{ij}} \\ &= -\eta \sum_{t=1}^{T} \frac{\partial E(t)}{\partial w_{ij}}\end{aligned}$$

hvilket betegnes batchmode optræning, som det også beskrives i kapitel 1.

Ved begge former for opdatering er der efter præsentation af hvert eksempel brug for at beregne

$$\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k \in \Omega} e_k(t) \frac{\partial V_k(t)}{\partial w_{ij}}$$

De afledede for de skjulte enheder findes ved at differentiere udtrykket for disse:

$$\frac{\partial V_k(t)}{\partial w_{ij}} = \frac{\partial f_k\left[s_k(t)\right]}{\partial s_k(t)} \cdot \frac{\partial s_k(t)}{\partial w_{ij}}$$

hvor

$$\begin{aligned}\frac{\partial s_k(t)}{\partial w_{ij}} &= \delta_{ki} z_j(t) + \sum_{j' \in I \cup U} w_{kj'} \frac{\partial z_{j'}(t)}{\partial w_{ij}} \\ &= \delta_{ki} z_j(t) + \sum_{j' \in U} w_{kj'} \frac{\partial V_{j'}(t-1)}{\partial w_{ij}}\end{aligned}$$

idet

$$\frac{\partial z_j(t)}{\partial w_{ij}} = \begin{cases} \frac{\partial V_j(t-1)}{\partial w_{ij}} & , \quad j \in U \\ 0 & , \quad j \in I \end{cases}$$

og hvor δ_{ki} angiver Kroneckers delta

$$\delta_{ki} = \begin{cases} 1 & k = i \\ 0 & \text{ellers} \end{cases}$$

Det bemærkes, at man for at beregne en enheds afledede til tiden t mht. en given vægt må have kendskab til *samtlige* enheders afledede til tiden $t-1$ mht. den samme vægt.

Når de første inputs præsenteres for netværket til tiden $t = 1$, det være sig ved optræning eller test, sættes netværkets interne tidligere tilstand til $\mathbf{0}$, dvs.

$$\begin{aligned}V_k(0) &= 0 \quad k \in U \\ \frac{\partial V_k(0)}{\partial w_{ij}} &= 0 \quad k, i \in U \ , \ j \in I \cup U\end{aligned}$$

ud fra den betragtning, at netværket ikke kan producere output, før det bliver påvirket af et input.

Hessian for W&Z RTRL

Den anden afledede af cost-funktionen $E(\mathbf{w})$ beregnes til:

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_{ij} \partial w_{pq}} = -\sum_{t=1}^{T} \sum_{k \in \Omega} \left[e_k(t) \frac{\partial^2 V_k(t)}{\partial w_{ij} \partial w_{pq}} - \frac{\partial V_k(t)}{\partial w_{ij}} \cdot \frac{\partial V_k(t)}{\partial w_{pq}} \right]$$

Her er

$$\frac{\partial^2 V_k(t)}{\partial w_{ij} \partial w_{pq}} = \frac{\partial^2 f_k\left[s_k(t)\right]}{\partial s_k(t)^2} \cdot \frac{\partial s_k(t)}{\partial w_{ij}} \cdot \frac{\partial s_k(t)}{\partial w_{pq}} + \frac{\partial f_k\left[s_k(t)\right]}{\partial s_k(t)} \cdot \frac{\partial^2 s_k(t)}{\partial w_{ij} \partial w_{pq}}$$

hvor

$$\frac{\partial^2 s_k(t)}{\partial w_{ij} \partial w_{pq}} = \left[\delta_{ki} \frac{\partial z_j(t)}{\partial w_{pq}} + \sum_{j' \in U} w_{kj'} \frac{\partial^2 V_{j'}(t-1)}{\partial w_{ij} \partial w_{pq}} + \delta_{kp} \frac{\partial z_q(t)}{\partial w_{ij}} \right] \tag{2.5}$$

Hessianmatricens symmetri bør udnyttes ved implementering, idet lagerforbruget samt beregningernes omfang er temmelig omfattende for selv relativt små netværk.

2.2.2 Layered RTRL

I dette afsnit beskrives et alternativ til Williams & Zipsers måde at opdatere et fuldt forbundet feedback netværk på. Hvor W&Z lader alle enheder tilhøre samme 'lag', dvs. opdaterer alle enheder udfra den samme tilstandsvektor, skelner den her beskrevne metode imellem deciderede skjulte enheder og en eller evt. flere output-enheder. Idet metoden her skal benyttes til prædiktion af en enkelt samtidig tidsserie, antages det i det følgende, at der kun eksisterer een outputenhed; udvidelse af udtrykkene til at omfatte flere, er dog triviel. Denne metode vil i det følgende blive betegnet *layered RTRL*.

Lad igen $\mathbf{x}(t)$ angive de m eksterne inputs til netværket til tiden t, og lad $\mathbf{V}^h(t)$ angive de n skjulte enheders output til tiden t, og lad endelig $V^o(t)$ angive outputenhedens output til tiden t. Hvor inputtet til enhederne med W&Z's metode kunne beskrives af en enkelt inputvektor, er det for layered RTRL nødvendigt med to, een for de skjulte enheder og een for outputenheden.

Inputtet til de skjulte enheder til tiden t defineres til

$$z_k^h(t) = \begin{cases} x_k(t) & , \quad k \in I \\ V_k^h(t-1) & , \quad k \in U \\ V^o(t-1) & , \quad k \in O \end{cases}$$

hvor I angiver indices tilhørende de m inputs, U angiver indices tilhørende de n skjulte enheder og O angiver outputenheden. Yderligere benyttes h

samt o som superscripts for at skelne mellem elementer vedrørende hhv. de skjulte enheder og outputenheden. Input til outputenheden er

$$z_k^o(t) = \begin{cases} x_k(t) & , \ k \in I \\ V_k^h(t) & , \ k \in U \\ V^o(t-1) & , \ k \in O \end{cases}$$

hvilket indebærer, at de skjulte enheders output beregnes først, og derefter føres frem til outputenheden, som det også er tilfældet for feedforward netværk. Ved tidsserieprædiktion er inputs ekskl. bias som for W&Z RTRL ligeledes $\mathbf{x}(t) = [x(t-1), x(t-2), \ldots, x(t-L)]$, hvor L er dimensionen af *lag space*. De skjulte enheders output til tiden t defineres som

$$\begin{aligned} V_k^h(t) &= f_k^h\left[s_k^h(t)\right] \\ &= f_k^h\left[\sum_{j \in I \cup U \cup O} w_{kj} z_j^h(t)\right] \\ &= f_k^h\left[\sum_{j \in I} w_{kj} x_j(t) + \sum_{j \in U} w_{kj} V_j^h(t-1) + w_{ko} V^o(t-1)\right] \end{aligned}$$

hvor f_k^h igen angiver aktiveringsfunktionen for den k'te skjulte enhed, og w_{ij} angiver vægten til enhed i fra enhed/input j. Outputtet fra outputenheden til tiden t defineres som

$$\begin{aligned} V^o(t) &= f^o\left[s^o(t)\right] \\ &= f^o\left[\sum_{j \in I \cup U \cup O} w_{oj} z_j^o(t)\right] \\ &= f^o\left[\sum_{j \in I} w_{oj} x_j(t) + \sum_{j \in U} w_{oj} V_j^h(t) + w_{oo} V^o(t-1)\right] \end{aligned} \tag{2.6}$$

Cost funktionen defineres som tidligere til

$$E(\mathbf{w}) = \frac{1}{2} \sum_{t=1}^{T} [e(t)]^2$$

hvor

$$e(t) = x(t) - V^o(t)$$

hvis der blot er en enkelt outputenhed. Gradienten af cost funktionen $E(\mathbf{w})$ er

$$\frac{\partial E(\mathbf{w})}{\partial w_{ij}} = -\sum_{t=1}^{T} e(t) \frac{\partial V^o(t)}{\partial w_{ij}}$$

Den afledede for outputenheden beregnes til

$$\frac{\partial V^o(t)}{\partial w_{ij}} = \frac{\partial f^o\left[s^o(t)\right]}{\partial s^o(t)} \cdot \frac{\partial s^o(t)}{\partial w_{ij}}$$

hvor

$$\begin{aligned}\frac{\partial s^o(t)}{\partial w_{ij}} &= \delta_{oi} z_j^o(t) + \sum_{j' \in U,O} w_{oj'} \frac{\partial z_{j'}^o(t)}{\partial w_{ij}} \\ &= \delta_{oi} z_j^o(t) + \sum_{j' \in U} w_{oj'} \frac{\partial V_{j'}^h(t)}{\partial w_{ij}} + w_{oo} \frac{\partial V^o(t-1)}{\partial w_{ij}}\end{aligned}$$

hvori indgår de afledede af de skjulte enheder

$$\frac{\partial V_k^h(t)}{\partial w_{ij}} = \frac{\partial f_k^h\left[s_k^h(t)\right]}{\partial s_k^h(t)} \cdot \frac{\partial s_k^h(t)}{\partial w_{ij}}$$

hvor

$$\begin{aligned}\frac{\partial s_k^h(t)}{\partial w_{ij}} &= \delta_{ki} z_j^h(t) + \sum_{j' \in U,O} w_{kj'} \frac{\partial z_{j'}^h(t)}{\partial w_{ij}} \\ &= \delta_{ki} z_j^h(t) + \sum_{j' \in U} w_{kj'} \frac{\partial V_{j'}^h(t-1)}{\partial w_{ij}} + w_{ko} \frac{\partial V^o(t-1)}{\partial w_{ij}}\end{aligned}$$

Som for W&Z RTRL sættes alle enheders output samt deres afledede til nul til tiden $t = 0$.

Hessian for layered RTRL

Som det blev gjort for W&Z RTRL beregnes Hessianmatricen tilsvarende her for layered RTRL. Den anden afledede af cost funktionen $E(\mathbf{w})$ er

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_{ij} \partial w_{pq}} = -\sum_{t=1}^{T} \left[e(t) \frac{\partial^2 V^o(t)}{\partial w_{ij} \partial w_{pq}} - \frac{\partial V^o(t)}{\partial w_{ij}} \cdot \frac{\partial V^o(t)}{\partial w_{pq}} \right]$$

Den anden afledede af outputtet $V^o(t)$ er

$$\frac{\partial^2 V^o(t)}{\partial w_{ij} \partial w_{pq}} = \frac{\partial^2 f^o\left[s^o(t)\right]}{\partial s^o(t)^2} \cdot \frac{\partial s^o(t)}{\partial w_{ij}} \cdot \frac{\partial s^o(t)}{\partial w_{pq}} + \frac{\partial f^o\left[s^o(t)\right]}{\partial s^o(t)} \cdot \frac{\partial^2 s^o(t)}{\partial w_{ij} \partial w_{pq}}$$

hvor

$$\frac{\partial^2 s^o(t)}{\partial w_{ij} \partial w_{pq}} = \delta_{oi} \frac{\partial z_j^o(t)}{\partial w_{pq}} + \sum_{j' \in U} w_{oj'} \frac{\partial^2 V_{j'}^h(t)}{\partial w_{ij} \partial w_{pq}} + w_{oo} \frac{\partial^2 V^o(t-1)}{\partial w_{ij} \partial w_{pq}} + \delta_{op} \frac{\partial z_q^o(t)}{\partial w_{ij}}$$

I dette udtryk indgår de anden afledede af de skjulte enheders output

$$\frac{\partial^2 V_k^h(t)}{\partial w_{ij} \partial w_{pq}} = \frac{\partial^2 f_k^h\left[s_k^h(t)\right]}{\partial s_k^h(t)^2} \cdot \frac{\partial s_k^h(t)}{\partial w_{ij}} \cdot \frac{\partial s_k^h(t)}{\partial w_{pq}} + \frac{\partial f_k^h\left[s_k^h(t)\right]}{\partial s_k^h(t)} \cdot \frac{\partial^2 s_k^h(t)}{\partial w_{ij} \partial w_{pq}}$$

hvor

$$\frac{\partial^2 s_k^h(t)}{\partial w_{ij} \partial w_{pq}} = \delta_{ki} \frac{\partial z_j^h(t)}{\partial w_{pq}} + \sum_{j' \in U} w_{kj'} \frac{\partial^2 V_{j'}^h(t-1)}{\partial w_{ij} \partial w_{pq}} + w_{ko} \frac{\partial^2 V^o(t-1)}{\partial w_{ij} \partial w_{pq}} + \delta_{kp} \frac{\partial z_q^h(t)}{\partial w_{ij}}$$

I forhold til udtrykkene for W&Z RTRL ses det, at udtrykkene for layered RTRL kompliceres yderligere af, at der nu også er tidslig afhængighed imellem de enkelte elementer i Hessianen.

2.3 Beregningsomkostninger for RTRL

I dette afsnit foretages der et overslag over, hvordan beregningsbyrden for fuldt forbundne feedback netværk afhænger af netværkskompleksiteten, når de ovenfor beskrevne udtryk for begge RTRL-opdateringer anvendes. I det følgende betragtes et netværk med ialt K enheder og I eksterne inputs inklusive bias.

Nedenfor foretages et overslag over beregningerne for gradienten efter præsentation af hvert eksempel. Udtrykkene angiver det *akkumulerede* antal beregninger:

Antallet af vægte	$K(K+I)$
Alle enheders output differentieret mht. disse	$K^2(K+I)$
Hver afledede involverer af størrelsesordenen $K+I$ operationer	$K^2(K+I)^2$

Heraf ses, at der til gradienterne kræves lagerplads af størrelsesordenen $O(K^3)$, og at antallet af beregninger er $O(K^4)$.

I [Schmidhuber 92] er beskrevet en algoritme til beregning af gradienten for W&Z RTRL, der angiveligt kan reducere beregningskompleksiteten til $O(K^3)$. Metoden går kort skitseret ud på, at træningssættet splittes op i sekvenser af længde h. For en sådan sekvens 'foldes' netværket ud i tid, hvilket svarer til *BackPropagation Through Time* (se [Hertz 91]) for sekvensen. Bidraget fra sekvensen til den totale gradient for RTRL-netværket beregnes herefter ved at benytte almindelig backpropagation på det udfoldede netværk. Det vises, at beregningerne for sekvensen 'koster' $(K+1)O(hK^2+K^3) \sim O(K^4)$ (se [Schmidhuber 92] for uddybelse af udtrykket), så hvis sekvenslængden h har størrelsesorden $O(K)$, vil hvert tidsskridt kræve $O(K^3)$. En lignende metode vil ikke blive benyttet her, da optræningen ofte vil involvere andenordens metoder, hvor beregning af gradienten ikke vil være den mest signifikante beregningsomkostning, hvilket fremgår nedenfor.

For den positivt definite del af Hessianmatricen haves:

Gradienter beregnes	$K^2(K+I)^2$
Ydre produkt af gradienter	$2K^2(K+I)^2$

hertil kommer leddet med de anden afledede:

Alle kombinationer af vægte	$K^2(K+I)^2$
Anden afledede for alle enheder	$K^3(K+I)^2$
Hver anden afledede involverer af størrelsesordenen $K+I$ operationer	$K^3(K+I)^3$

Heraf ses, at der til de anden afledede af alle enhedernes output kræves lagerplads af størrelsesordenen $O(K^5)$, og at antallet af beregninger er af orden $O(K^6)$. Ved anvendelse til feks. beskæring med Optimal Brain Surgeon (se kapitel 4) benyttes den inverterede af Hessianmatricen. Efterfølgende invertering af Hessianmatricen kræver med standardmetoder af størrelsesordenen $O\left[(K(K+I))^{\log_2 7}\right] \sim O(K^6)$ operationer (se [Press 92]). I [Hassibi 93] benyttes en iterativ metode til invertering af Hessianmatricen, men dels har udtrykkene for Hessianmatricen her ikke den krævede form, dels vises denne metode i appendiks C at være for dyr at benytte ved typiske anvendelser af neurale netværk.

Kapitel 3

W&Z RTRL vs. layered RTRL

I dette kapitel foretages en kvalitativ sammenligning imellem Williams & Zipsers RTRL samt layered RTRL. Først vises ulemperne ved W&Z RTRL for de generelle udtryk, dernæst illustreres forskellene mellem de to algoritmer med et simpelt eksempel.

3.1 Ulemper ved W&Z RTRL

Ved tidsserieprædiktion defineres inputvektoren til netværket eksklusive biasinput som tidligere nævnt som $\mathbf{x}(t) = [x(t-1), \ldots, x(t-L)]$, hvor L angiver dimensionen af lag space. For W&Z RTRL vælges en enhed som outputenheden V_0, for hvilken aktiveringsfunktionen f_0 ofte vælges til at være lineær. Prædiktionen af $x(t)$ er nu $\widehat{x}(t) = f_0\left[s_0(t)\right]$, defineret i (2.4). For at undgå at netværket degenererer til en lineær model, fjernes outputenhedens direkte forbindelser til de eksterne inputs. Outputtet bliver således

$$\widehat{x}(t) = V_0(t) = \sum_{j \in U} w_{0j} V_j(t-1) + w_{bias,0} \tag{3.1}$$

Idet begyndelsestilstanden for netværket er $\mathbf{V}(0) = \mathbf{0}$, bliver første prædiktion

$$\widehat{x}(1) = w_{bias,0} \tag{3.2}$$

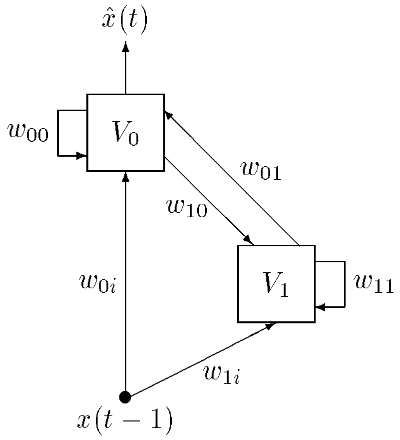
der meget vel kan være en dårlig prædiktion. Den fejlagtige prædiktion føres tilbage i netværket, og benyttes til fremtidige prædiktioner, hvilket introducerer en betydelig fejl i de umiddelbart efterfølgende prædiktioner.

I (3.1) vises udtrykket for outputtet, når outputenheden ikke modtager de eksterne inputs. Det bemærkes, at outputtet til tiden t udover bias samt tidligere outputs er baseret på de interne 'skjulte' enheders output til tiden $t-1$, hvilket igen er beregnet udfra $\mathbf{x}(t-1)$ og dermed udfra $x(t-2), \ldots$ W&Z RTRL vil således ved fjernelse af de eksterne inputs fra outputenheden søge at prædiktere to skridt frem i tiden, hvilket ikke er hensigtsmæssigt. Denne iagttagelse medfører yderligere, at W&Z RTRL ikke indeholder almindelige feedforward netværk som special tilfælde, som det hævdes i [Williams 89].

Ulemperne med W&Z RTRL beskrevet ovenfor kan undgås, hvis man benytter layered RTRL, hvilket fremgår af eksemplet nedenfor.

3.2 Simpelt eksempel til sammenligning

Forskellene på de to algoritmer belyses her gennem et eksempel for et simpelt fuldt forbundet feedback netværk, der er afbildet nedenfor på figur (3.1). Nettet består af to enheder, hvis output benævnes V_0 og V_1, samt et en-



Figur 3.1: Fuldt forbundet eksempel.

kelt input x. V_0 defineres til at være det lineære output, og V_1 er således skjult/intern enhed med en ikke-lineær aktiveringsfunktion f. Netværket tænkes anvendt til tidsserieprædiktion, og hvis elementerne i serien benævnes x, haves til tiden t:

input :	$x(t-1)$
prædiktion :	$\widehat{x}(t) = V_0(t)$
target :	$x(t)$

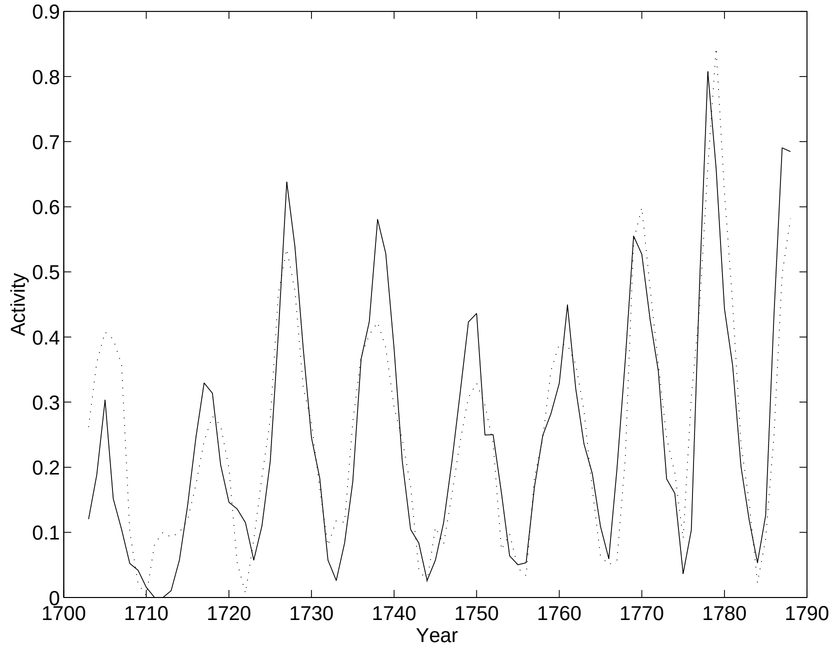
Netværkets output $V_0(t)$ opskrives nu for de to algoritmer udfra udtrykkene (2.4) hhv. (2.6).
W&Z RTRL :

$$\begin{aligned} V_0(t) &= w_{0i}x(t-1) + w_{00}V_0(t-1) + w_{01}V_1(t-1) + w_{bias,0} \\ V_1(t-1) &= f(w_{1i}x(t-2) + w_{11}V_1(t-2) + w_{10}V_0(t-2) + w_{bias,1}) \end{aligned}$$

Layered RTRL :

$$\begin{aligned} V_0(t) &= w_{0i}x(t-1) + w_{00}V_0(t-1) + w_{01}V_1(t) + w_{bias,0} \\ V_1(t) &= f(w_{1i}x(t-1) + w_{11}V_1(t-1) + w_{10}V_0(t-1) + w_{bias,1}) \end{aligned}$$

Forskellen på de to algoritmer ses umiddelbart at bestå i, at for W&Z RTRL er $V_0(t)$ en lineær funktion af $x(t-1)$ og en ulineær funktion af tidligere inputs, i modsætning til layered RTRL, hvor ulineariteten også omfatter



Figur 3.2: W&Z RTRL et-skridt prædiktion af solpletserien. 'Solid' angiver solpletserien, 'dotted' angiver prædiktionen med W&Z RTRL.

inputtet $x(t-1)$. Ved opstart af algoritmerne, hvor $V_0(0)$ og $V_1(0)$ er nul, vil den første prædiktion $V_0(1)$ for W&Z RTRL således udelukkende være en lineær funktion af inputtet $x(0)$, hvorimod $V_0(1)$ for layered RTRL vil være ikke-lineær grundet den lag-delte feedforward-inspirerede struktur.

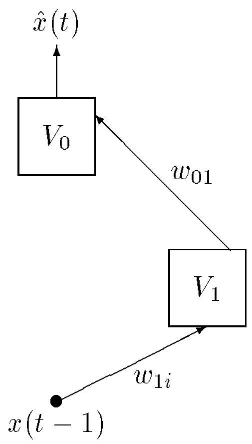
Hvis outputenheden ikke modtager det eksterne input, svarer dette til, at vægten w_{0i} er nulstillet, og netværket tvinges derved til at lave en ulineær model af de præsenterede data. Dette tiltag vil have nogle uheldige konsekvenser for W&Z RTRL, idet denne algoritme så udviser en vis træghed fra et input præsenteres for netværket, til det har effekt på outputtet. Prædiktionen $V_0(t)$ af værdien $x(t)$ ses i dette tilfælde at være baseret på $x(t-2)$ og tidligere inputs, hvilket svarer til at prædiktere to skridt ud i fremtiden.

Hvis nettet som beskrevet ovenfor tvinges til at lave en ikke-lineær model, vil W&Z RTRL lide under en degenereret initiel modelleringsevne, idet første prædiktion ses at blive

$$V_0(1) = w_{bias,0}$$

dvs. første prædiktion vil som vist i forrige afsnit være et gæt på outputenhedens bias, hvilket højst sandsynligt vil være et temmelig dårligt gæt. Netværket vil således konsekvent startes op med en stor prædiktionsfejl, hvilket er særdeles uheldigt, idet fremtidige prædiktioner pga. feedback'et til dels baseres på den fejlagtige initielle prædiktion.

På figur 3.2 er dette fænomen illustreret på solpletserien. Det pågældende netværk har fire enheder, der alle pånær outputenheden modtager 4 eksterne inputs. Nettets outputenhed har $w_{bias} = 0.262$, hvilket ses at svare til



Figur 3.3: Reducering til feedforward.

den første prædiktion, der medfører store unøjagtigheder i de efterfølgende prædiktioner.

Til sammenligning vil første prædiktion fra layered RTRL være

$$V_0(1) = w_{01}\,f(w_{1i}x(0) + w_{bias,1}) + w_{bias,0}$$

hvilket svarer til, at netværket initielt opfører sig som et almindeligt feedforward netværk.

Ved beskæring af de fuldt forbundne feedback netværk vil det være en fordel, hvis disse kan reduceres til et sædvanligt feedforward netværk, ifald en sådan arkitektur skulle vise sig optimal for den betragtede tidsserie. Det her betragtede netværk vil reduceres til et feedforward netværk, hvis feedbackvægtene w_{00}, w_{01} og w_{11} tillige med vægten w_{0i} nulstilles, som illustreret på figur (3.3).

Hvis det reducerede netværk opdateres med W&Z RTRL, bliver outputtet

$$\begin{aligned} V_0(t) &= w_{01}V_1(t-1) + w_{bias,0} \\ &= w_{01}f(w_{1i}x(t-2) + w_{bias,1}) + w_{bias,0} \end{aligned}$$

Det bemærkes, at dette netværk, som det var tilfældet ovenfor, prædikterer to tidsskridt ud i fremtiden og vil lide under opstartsproblemet med, at første prædiktion $V_0(1)$ vil være outputenhedens bias.

Til sammenligning reduceres outputtet med layered RTRL til

$$\begin{aligned} V_0(t) &= w_{01}V_1(t) + w_{bias,0} \\ &= w_{01}\,f(w_{1i}x(t-1) + w_{bias,1}) + w_{bias,0} \end{aligned}$$

hvilket ses at svare til en sædvanlig feedforward opdatering.

Ovenstående sammenligning imellem W&Z RTRL samt layered RTRL afslører en del uheldigheder ved W&Z RTRL, når denne anvendes til tidsserieprædiktion. Den reducerede modelleringsevne for de seneste eksterne inputs må dog også have negativ indflydelse for andre anvendelser såsom finite state automata, som af Williams & Zipser hovedsalig afprøves i [Williams 89]. Den eneste situation, hvor W&Z RTRL reelt vil være at foretrække fremfor layered RTRL, er ved anvendelser, hvor der er behov for at specificere targetværdier for forskellige enheder til forskellige tidspunkter.

På baggrund af ovenstående sammenligning konkluderes det, at layered RTRL er en mere hensigtsmæssig algoritme end W&Z RTRL ved langt de fleste anvendelser. Da dette arbejde omhandler tidsserieprædiktion, hvor layered RTRL er vist at have fordele fremfor W&Z RTRL, vil de efterfølgende undersøgelser og forsøg udelukkende blive udført for layered RTRL.

Kapitel 4

Beskæring af netværk

Bestemmelse af, hvilken type arkitektur for et neuralt netværk, der bedst egner sig til et givent formål, er ikke en simpel opgave. Når *typen* af arkitekturen er fastlagt, støder man ydermere på problemet med at bestemme den optimale arkitektur. Optimal defineres i denne henseende ofte i overensstemmelse med princippet kaldet *Ockhams Razor*, der er beskrevet i [Thodberg 91]. Princippet går ud på at minimere netværkets funktionelle kapacitet, herunder antallet af vægte i netværket, dog således, at netværket stadig kan uddrage de implicitte regler fra træningssættet. Ved at tilpasse netværkskompleksiteten til den givne opgave, kan man modvirke overfitting af træningseksemplerne med forhåbentlig bedre generaliseringsevne til følge.

Weight decay

Det således definerede optimale netværk kan kan søges opnået ved anvendelse af flere strategier. En simpel metode til at regulere netværkskompleksiteten er *weight decay*, der lægger en begrænsning på de værdier, som vægtene reelt kan antage. Denne begrænsning realiseres i praksis som oftest ved at regularisere cost funktionen. En simpel form for weight decay blev i forbindelse med optræningen af netværk introduceret i afsnit 1.1, hvor der til cost funktionen $E(\mathbf{w})$ adderes en konstant gange summen af vægtene kvadreret (se [Hertz 91]):

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\alpha \sum_i w_i^2$$

hvor α er weight decay parameteren, der regulerer styrken af decay'et; α er positiv ved minimering af cost funktionen, og negativ med maksimering, se [Hansen&With 94]. Addition af dette led har den effekt, at vægtene under optræning 'trækkes' imod nul. Dette opnås ved, at cost funktionen bliver 'stor' for numerisk store vægte, hvorved minima langt fra origo udviskes. Henfaldet indebærer, at der ved gradientnedstigning ved hver opdatering af en vægt fratrækkes en størrelse proportional med vægten:

$$w_i^{ny} = (1-\alpha) w_i^{gammel}$$

Hvis en vægt ikke er nødvendig for netværket, vil den langsomt blive trukket mod nul, hvorimod nødvendige vægte for rimelige værdier af α til en vis grad vil være i stand til at modstå dette træk, se [Hertz 91].

I sidstnævnte reference findes der flere andre og mere avancerede forslag til udformning af weight decay'et, men disse vil ikke blive anvendt her.

Arkitektur-tilpasning

Weight decayet beskrevet ovenfor begrænser vægtenes reelle dynamiske område, men overflødige vægte vil ikke *nødvendigvis* blive nul under optræningen. Alle vægte reguleres med samme α, hvilket påvirker store de vægte forholdsvis mere end de små. Simpelt weight decay er derfor ikke den bedste måde at tilpasse antallet af vægte i netværket til den aktuelle opgave, hvilket har beregningsmæssige omkostninger for den fremtidige anvendelse af netværket; desuden kan det ved hardware-implementering af et netværk være ønskeligt, at antallet af vægte er minimalt, således at ressourcebehovet begrænses.

Den optimale arkitektur kan søges opnået med i hvert fald to strategier. Den ene går ud på at anvende vækst-algoritmer, hvor der startes med et minimalt netværk (feks. en enkelt enhed), og adderes enheder samt vægte under optræningen, efterhånden som der er brug for dem. En sådan vækst-algoritme er Recurrent Cascade Correlation, der er beskrevet i afsnit 2.1.

Den anden strategi er at optræne et rigeligt stort netværk, og derefter fjerne vægte/enheder, indtil den ønskede optimale arkitektur opnås. Tilpasning af netværksarkitekturen til det forhåndenværende problem indebærer at fjerne de 'overflødige' vægte fra netværket, dvs. vægte der ikke er essentielle for at implementere den bagvedliggende 'regel' for data. Denne proces kaldes *pruning*[1], og forløber typisk efter følgende forskrift:

1. Optræn et rigeligt stort netværk.
2. Sortér vægtene efter vigtighed.
3. Eliminér den eller de mindst betydningsfulde vægte.
4. Genoptræn det reducerede netværk, og gentag 2), hvis yderligere pruning ønskes.

Flere algoritmer er foreslået til opdelingen af vægtene efter vigtighed. Den mest simple metode er *magnitude based* pruning, se [Hertz 91], hvor det antages, at vægtenes numeriske størrelse er et udtryk for deres vigtighed for netværket, således at de numerisk mindste vægte er de mindst betydende, og derfor dem, der skal elimineres. Dette er dog en meget grov antagelse, der kan føre til eliminering af de forkerte vægte.

I det følgende beskrives to metoder til pruning af et netværk, der for hver vægt estimerer vigtigheden af denne, kaldet *saliency*. Saliency defineres

[1] beskæring

som et estimat over, hvilken ændring der vil forekomme i cost funktionen, hvis en given vægt nulstilles/elimineres, og giver således et mere velfunderet beslutningsgrundlag at bedømme vægtene på fremfor udfra deres numeriske størrelse. Ifølge *Websters Encyclopedic Unabridged Dictionary of the English Language* er saliency synonymt med *vigtighed*, hvilket også er det, som vurderes: en given vægts vigtighed for netværket.

I udledningerne refereres der til ekstremum for cost funktionen. Dette kan substitueres med mimimum eller maksimum, alt efter hvilken type cost funktion man benytter.

4.1 Optimal Brain Damage

I dette afsnit udledes udtrykkene for Optimal Brain Damage, beskrevet i [LeCun 90b], for en regulariseret cost funktion, specielt for en cost funktion udvidet med simpelt weight decay. Resultatet af denne udledning kan også findes i [Svarer 93].

Som tidligere antages cost funktionen at have formen

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\mathbf{w}^T\mathbf{D}\mathbf{w} \tag{4.1}$$

hvor $\mathbf{D}$ er regulariseringsmatricen. Det erindres, at for simpelt weight decay er $\mathbf{D}$ proportional med enhedsmatricen, $\mathbf{D} = \alpha\mathbf{I}$.

Ideen bag OBD er at nulstille en given vægt for den regulariserede cost funktion, der er trænet til et lokalt ekstremum, og estimere den resulterende ændring i den oprindelige cost funktion $E(\mathbf{w})$, saliency. Dette gøres for alle vægte, og disse kan så ordnes efter størrelsen af deres respektive saliencies. *Om* vægtene skal elimineres, afgøres af et givet stopkriterie.

Lad det lokale ekstremum for $C(\mathbf{w})$ være $\mathbf{w}_0$. I dette punkt approksimeres den oprindelige cost funktion $E(\mathbf{w})$ med Taylorudviklingen

$$E(\mathbf{w}_0 + \delta\mathbf{w}) = E(\mathbf{w}_0) + \delta\mathbf{w}^T\nabla E(\mathbf{w}_0) + \frac{1}{2}\delta\mathbf{w}^T\mathbf{A}\delta\mathbf{w} \tag{4.2}$$

hvor $\delta\mathbf{w}$ angiver ændringen af vektoren $\mathbf{w}_0$, og $\mathbf{A}$ angiver Hessianmatricen for E i $\mathbf{w}_0$. Idet punktet $\mathbf{w}_0$ antages at være stationært for C, haves

$$\begin{aligned} & \nabla C(\mathbf{w}_0) = \nabla E(\mathbf{w}_0) + \mathbf{D}\mathbf{w}_0 = \mathbf{0} \\ \Updownarrow \quad & \\ & \nabla E(\mathbf{w}_0) = -\mathbf{D}\mathbf{w}_0 \end{aligned} \tag{4.3}$$

Dette udtryk indsættes i (4.2), hvilket giver

$$\begin{aligned} \delta E &= E(\mathbf{w}_0 + \delta\mathbf{w}) - E(\mathbf{w}_0) \\ &= -\delta\mathbf{w}^T\mathbf{D}\mathbf{w}_0 + \frac{1}{2}\delta\mathbf{w}^T\mathbf{A}\delta\mathbf{w} \end{aligned} \tag{4.4}$$

Hvis pertubationen $\delta\mathbf{w}$ har karakter af nulstilling af den j'te vægt i $\mathbf{w}_0$, $\delta\mathbf{w} = -(\mathbf{e}_j^T\mathbf{w}_0)\mathbf{e}_j$, hvor $\mathbf{e}_j$ er den j'te enhedsvektor, og regulariseringen har form af simpelt weight decay, kan den resulterende ændring i den oprindelige cost funktion E skrives som

$$\delta E_j = \left(\alpha + \frac{1}{2}\frac{\partial^2 E(\mathbf{w})}{\partial w_j^2}\right) w_{0,j}^2 \tag{4.5}$$

hvor $w_{0,j}$ er det j'te element i $\mathbf{w}_0$.

En væsentlig del af OBD-ideen er antagelsen om, at ændringen δE ved nulstilling af flere vægte i $\mathbf{w}_0$ samtidig kan approksimeres med summen af de estimerede ændringer, hvis vægtene blev nulstillet hver for sig:

$$\delta E_D = \sum_{j \in D} \delta E_j = \sum_{j \in D} \left(\alpha + \frac{1}{2}\frac{\partial^2 E(\mathbf{w})}{\partial w_j^2}\right) w_{0,j}^2 \tag{4.6}$$

hvor D angiver sættet af indices for de vægte, der nulstilles. Dette er ensbetydende med at antage, at alle ikke-diagonal elementer i $\mathbf{A}$ er negligible:

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} = 0 \quad , \quad i \neq j \tag{4.7}$$

Yderligere approksimeres diagonalelementerne i $\mathbf{A}$ oftest på samme måde som i (1.17):

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_i^2} \approx \sum_{t=1}^{T} \left(\frac{\partial V^\circ(t)}{\partial w_i}\right)^2 \tag{4.8}$$

Der opstilles nu en algoritme for pruning med OBD:

1. Cost funktionen $C(\mathbf{w})$ optrænes til et ekstremum, $\mathbf{w}_0$.
2. Netværket evalueres efter et givet stopkriterie. Er dette opfyldt, hoppes der til punkt 6, ellers fortsættes med punkt 3.
3. Beregn saliencies δE_j for alle vægte i punktet $\mathbf{w}_0$.
4. Vægtene ordnes efter størrelse af saliency.
5. Et antal af vægtene med lavest saliency elimineres, og der hoppes til 1.
6. Der er fundet en arkitektur, der opfylder det givne stopkriterie; pruningen er således tilendebragt.

Et muligt stopkriterie beskrives i afsnit 4.3.

Antallet af vægte, der elimineres i hver 'omgang', fastsættes ofte til een eller til en given procentdel af de tilbageværende vægte, se feks. [Svarer 93].

4.2 Optimal Brain Surgeon

I dette afsnit udledes udtrykkene for OBS for en *regulariseret* cost funktion, herunder en cost funktion udvidet med weight decay ([MWP 94] samt [Hansen&With 94]).

Lad den oprindelige cost funktion være $E(\mathbf{w})$, hvor $\mathbf{w}$ er vægtvektoren. Den udvidede cost funktion til regulariseret optræning er igen

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{D}\mathbf{w} \tag{4.9}$$

hvor $\mathbf{D}$ er regulariseringsmatricen.

Overordnet set består metoden i at nulstille en vægt for den regulariserede cost funktion, der er trænet til et lokalt ekstremum, og under disse forudsætninger bestemme den nødvendige ændring af vægtvektoren, således at den regulariserede cost funktion *igen* når et ekstremum. Ud fra denne optimering kan der gives et estimat af, hvor stor en ændring der vil forekomme i den *oprindelige* cost funktion. På denne baggrund afgøres det, om den pågældende vægt skal nulstilles/fjernes eller ej. Til forskel fra OBD baseres estimatet på den *fulde* Hessian, ikke blot på en diagonal-approksimation.

Lad det lokale ekstremum for $C(\mathbf{w})$ være $\mathbf{w}_0$. Cost funktionen approksimeres med Taylorudviklingen omkring $\mathbf{w}_0$

$$C(\mathbf{w}) \approx C(\mathbf{w}_0) + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}(\mathbf{A}+\mathbf{D})\delta\mathbf{w} \tag{4.10}$$

hvor $\mathbf{w} = \mathbf{w}_0 + \delta\mathbf{w}$. Førsteordensleddet forsvinder, idet $\mathbf{w}_0$ antages at være et ekstremum, og det antages yderligere, at afledede af orden større end to er uden betydning. $\mathbf{A}$ er matricen med de anden afledede af $E(\mathbf{w})$ i punktet $\mathbf{w}_0$; i det efterfølgende benyttes Hessian matricen for $C(\mathbf{w})$, $\mathbf{H} = \mathbf{A} + \mathbf{D}$.

Herefter nulstilles den j'te vægt, hvilket kan udtrykkes som

$$\delta\mathbf{w}_j^{\mathrm{T}}\mathbf{e}_j + \mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j = 0 \;\Leftrightarrow\; \delta\mathbf{w}_j^{\mathrm{T}}\mathbf{e}_j = -\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j \tag{4.11}$$

hvor $\mathbf{e}_j$ er den j'te enhedsvektor. Udtrykket (4.10) skal nu optimeres under lighedsbibetingelsen [Madsen 91b] (4.11), og til dette formål indføres Lagrange funktionen

$$\tilde{C}_j(\mathbf{w}) = C(\mathbf{w}) + \gamma_j(\delta\mathbf{w}_j + \mathbf{w}_0)^{\mathrm{T}}\mathbf{e}_j \tag{4.12}$$

hvor $C(\mathbf{w})$ er approksimationen (4.10). Koefficienten γ_j er en Lagrange multiplikator, og $\delta\mathbf{w}_j$ samt γ_j skal bestemmes således, at udtrykket (4.12) er optimalt, samtidig med at bibetingelsen (4.11) er opfyldt.

Først bestemmes $\delta\mathbf{w}_j$, således at der opnås et ekstremum for (4.12). Dette gøres ved at sætte den afledede af udtrykket lig nul:

$$\begin{aligned}
&\tfrac{\partial C(\mathbf{w})}{\partial(\delta\mathbf{w}_j)} + \gamma_j\mathbf{e}_j = \mathbf{0}\\
\Updownarrow\;&\\
&\mathbf{H}\delta\mathbf{w}_j + \gamma_j\mathbf{e}_j = \mathbf{0}\\
\Updownarrow\;&\\
&\delta\mathbf{w}_j = -\gamma_j\mathbf{H}^{-1}\mathbf{e}_j
\end{aligned} \tag{4.13}$$

Til at bestemme det tilhørende γ_j benyttes (4.13) indsat i (4.11):

$$\begin{aligned} &\gamma_j \mathbf{e}_j^{\mathrm{T}} \mathbf{H}^{-1} \mathbf{e}_j = \mathbf{w}_0^{\mathrm{T}} \mathbf{e}_j \\ \Updownarrow& \\ &\gamma_j = \tfrac{\mathbf{w}_0^{\mathrm{T}} \mathbf{e}_j}{\mathbf{e}_j^{\mathrm{T}} \mathbf{H}^{-1} \mathbf{e}_j} \end{aligned} \tag{4.14}$$

'Genoptræningen' efter nulstilling af den j'te vægt indebærer således en ændring af vægt-vektoren, der bestemmes ved indsættelse af (4.14) i (4.13):

$$\delta \mathbf{w}_j = -\frac{\mathbf{w}_0^{\mathrm{T}} \mathbf{e}_j}{\mathbf{e}_j^{\mathrm{T}} \mathbf{H}^{-1} \mathbf{e}_j} \mathbf{H}^{-1} \mathbf{e}_j \tag{4.15}$$

Med dette udtryk er det nu muligt at estimere ændringen i den *oprindelige* cost funktion, dvs. saliency for $E(\mathbf{w})$. Funktionen Taylorudvikles som $C(\mathbf{w})$ omkring $\mathbf{w}_0$, hvilket giver

$$E_j(\mathbf{w}) = E(\mathbf{w}_0) + \delta \mathbf{w}_j^T \nabla E(\mathbf{w}_0) + \frac{1}{2} \delta \mathbf{w}_j^{\mathrm{T}} \mathbf{A} \delta \mathbf{w}_j \tag{4.16}$$

Førsteordensleddet i udtrykket (4.16) kan bestemmes udfra den regulariserede cost funktion $C(\mathbf{w})$, idet der i punktet $\mathbf{w}_0$ antages at gælde

$$\nabla C(\mathbf{w}_0) = \nabla E(\mathbf{w}_0) + \mathbf{D}\mathbf{w}_0 = \mathbf{0} \;\Leftrightarrow\; \nabla E(\mathbf{w}_0) = -\mathbf{D}\mathbf{w}_0 \tag{4.17}$$

Dette udtryk indsættes i (4.16), hvilket giver

$$\delta E_j(\mathbf{w}) = -\delta \mathbf{w}_j^{\mathrm{T}} \mathbf{D} \mathbf{w}_0 + \frac{1}{2} \delta \mathbf{w}_j^{\mathrm{T}} \mathbf{A} \delta \mathbf{w}_j \tag{4.18}$$

hvor $\delta E_j(\mathbf{w}) = E_j(\mathbf{w}) - E(\mathbf{w}_0)$. Endelig indsættes ændringen for genoptræningen (4.13) i (4.18), og udtrykket, der approksimerer saliency ved nulstilling af den j'te vægt, bliver

$$\delta E_j(\mathbf{w}) = \gamma_j \mathbf{e}_j^{\mathrm{T}} \mathbf{H}^{-1} \mathbf{D} \mathbf{w}_0 + \frac{1}{2} \gamma_j^2 \mathbf{e}_j^{\mathrm{T}} \mathbf{H}^{-1} \mathbf{A} \mathbf{H}^{-1} \mathbf{e}_j \tag{4.19}$$

Det bemærkes, at hvis der ikke benyttes regularisering, svarende til at $\mathbf{D} = \mathbf{0}$, reducerer (4.19) til det, der blev udledt i [Hassibi 93]:

$$\begin{aligned} \delta E_{j,OBS}(\mathbf{w}) &= \frac{1}{2} \gamma_j^2 \mathbf{e}_j^{\mathrm{T}} \mathbf{A}^{-1} \mathbf{e}_j \\ &= \frac{1}{2} \frac{(\mathbf{w}_0^{\mathrm{T}} \mathbf{e}_j)^2}{\mathbf{e}_j^{\mathrm{T}} \mathbf{A}^{-1} \mathbf{e}_j} \\ &= \frac{1}{2} \frac{\mathbf{w}_{0,j}^2}{\mathbf{A}_{jj}^{-1}} \end{aligned} \tag{4.20}$$

hvor $\mathbf{w}_{0,j}$ er den j'te komponent af vægtvektoren i ekstremumet $\mathbf{w}_0$, og $\mathbf{A}_{jj}^{-1}$ er det j'te diagonalelement i den inverse Hessian matrix $\mathbf{A}^{-1}$ for $E(\mathbf{w})$ i $\mathbf{w}_0$.

Udtrykket (4.19) er beregningsmæssigt temmelig kompliceret, men kan simplificeres en smule, hvis regulariseringen har karakter af simpelt weight decay, således at $\mathbf{H} = \mathbf{A} + \alpha\mathbf{I}$. I dette tilfælde beregnes saliency til

$$\begin{aligned}
\delta E_j(\mathbf{w}) &= \alpha\gamma_j \mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0 + \frac{1}{2}\gamma_j^2 \mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}(\mathbf{H}-\mathbf{D})\mathbf{H}^{-1}\mathbf{e}_j \\
&= \alpha\frac{\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0)}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j} + \frac{1}{2}\frac{(\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j)^2}{(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j)^2}\left(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j - \alpha\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-2}\mathbf{e}_j\right) \\
&= \frac{1}{2}\frac{(\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j)^2}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j} + \alpha\left(\frac{\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0)}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j} - \frac{1}{2}\frac{(\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j)^2\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-2}\mathbf{e}_j}{(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j)^2}\right) \\
&= \frac{1}{2}\frac{\mathbf{w}_{0,j}^2}{\mathbf{H}_{jj}^{-1}} + \alpha\left(\frac{\mathbf{w}_{0,j}(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0)}{\mathbf{H}_{jj}^{-1}} - \frac{1}{2}\frac{\mathbf{w}_{0,j}^2\mathbf{H}_{jj}^{-2}}{(\mathbf{H}_{jj}^{-1})^2}\right) \qquad (4.21)
\end{aligned}$$

Dette udtryk afviger fra det tilsvarende udtryk i [Hansen&With 94] med en faktor $\mathbf{w}_{0,j}^2$ på sidste led, idet der i artiklen beklageligvis har indsneget sig en typografisk fejl; den være hermed rettet.

Der kan nu opstilles en algoritme for pruning med OBS:

1. Optræn cost funktionen $C(\mathbf{w})$ til et ekstremum, $\mathbf{w}_0$.

2. Netværket evalueres efter et givet stopkriterie. Er dette opfyldt hoppes der til punkt 6, ellers fortsættes der med punkt 3.

3. Beregn $\mathbf{H}^{-1}$ i punktet $\mathbf{w}_0$.

4. For alle vægte j i $\mathbf{w}_0$ bestemmes den tilhørende saliency δE_j. Hvis den mindste δE_j resulterer i en ændring af $E(\mathbf{w})$, der overholder stopkriteriet fra punkt 2, kan vægt j nulstilles/fjernes, og der fortsættes med punkt 5. I modsat fald hoppes der til punkt 6.

5. Alle vægte ændres i overensstemmelse med udtrykket (4.15) for det j, der blev bestemt i punkt 4, hvilket resulterer i et nyt ekstremum $\mathbf{w}_0$; der fortsættes med punkt 2.

6. Der kan ikke fjernes flere vægte, uden at ændringen af $E(\mathbf{w})$ bliver for stor; pruningen er således tilendebragt.

Udfra ovenstående algoritme bemærkes det, at der kun kan fjernes een vægt ad gangen, og $\mathbf{H}^{-1}$ skal beregnes påny hver gang dette forekommer, hvilket er temmelig beregningskrævende. I [Hassibi 93] lægges der dog op til at benytte *Gauss-Newton* approksimationen til Hessian matricen, dvs. udtrykket (1.17), hvilket letter beregningerne noget. Selve inverteringen foreslår de bliver udført iterativt under anvendelse af udtrykket

$$(\mathbf{A} + \mathbf{B}\cdot\mathbf{C}\cdot\mathbf{D})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\cdot\mathbf{B}\cdot(\mathbf{C}^{-1} + \mathbf{D}\cdot\mathbf{A}^{-1}\cdot\mathbf{B})^{-1}\cdot\mathbf{D}\cdot\mathbf{A}^{-1} \qquad (4.22)$$

men hvis optræningen kan foregå *off line* vil dette ikke kunne betale sig fremfor almindelig Hessian-beregning med efterfølgende matrix-invertering, i det mindste ikke for rimelige forhold imellem antal vægte og antal træningseksempler, som det vises i appendiks C.

OBS er oprindeligt formuleret således, at det ikke er nødvendigt med yderligere retraining af de tilbageblevne vægte. Denne formulering er dog helt afhængig af, at approksimationen (4.10) er god, og bliver ved med at være det gennem hele pruning-forløbet. Er dette ikke tilfældet, risikeres det, at gradienten for $C(\mathbf{w})$ i de nye punkter ikke er nul. Udover ringere saliency-estimater betyder dette, at genoptræningen (4.15) bliver fejlagtig, eller i hvert fald utilstrækkelig. Det skal derfor her anbefales, at en vægteliminering efterfølges af yderligere genoptræning, feks. med metoderne beskrevet i kapitel 1, inden den inverse Hessian beregnes påny. Gradienten vil dermed med væsentlig større sikkerhed være nul, således at usikkerheden på saliency-estimaterne vil formindskes.

Af udledningerne for OBD samt OBS ovenfor fremgår det, at en væsentlig forskel på de to metoder er, at OBS opererer med den *fulde* (Gauss-Newton) Hessian, hvorimod OBD negligerer alle elementer udenfor diagonalen; i teorien betyder dette, at saliency-estimaterne for OBS bør være mere pålidelige end for OBD. Hvis antagelserne for OBD imidlertid holder, er saliencies for de enkelte vægte additive, således at man kan vurdere ændringen i cost funktionen ved eliminering af flere vægte ved at addere vægtenes respektive saliencies, hvilket ikke umiddelbart er muligt for OBS.

En anden stor forskel på de to metoder består i, hvad saliency egentlig giver udtryk for. For OBD giver saliency δE_j et estimat for ændringen i cost funktionen, hvis en vægt j nulstilles. Estimatet udtaler sig således om ændringen i cost funktionen *før* retraining, men siger ikke noget om, hvordan ændringen vil være *efter* retraining. Efter retraining vil cost funktionen være enten formindsket eller uændret. For OBD vil saliency $\delta E_j^{før}$ før retraining derfor angive et estimat for den øvre grænse for ændringen i cost funktionen efter retraining δE_j^{efter},

$$\delta E_j^{før} \geq \delta E_j^{efter}$$

som det påpeges i [Gorodkin 93].

OBS-metoden søger ved eliminering af en vægt at bestemme, hvorledes de tilbageblevne vægte skal justeres, for at den reducerede cost funktion igen når et ekstremum; metoden har således 'indbygget' retraining, og saliency giver et estimat for ændringen i cost funktionen *efter* denne retraining, i modsætning til OBD.

Fælles for metoderne er antagelsen om, at cost funktionen kan approksimeres med en andenordens Taylorudvikling omkring udgangspunktet $\mathbf{w}_0$, selv i en 'stor' omegn af dette. Denne antagelse kan være meget grov, da cost funktionen oftest er kraftigt 'ukvadratisk'. Saliency-estimaterne kan derfor træffe at være så fejlbehæftede, at der elimineres vægte, der fører til en meget stor ændring i cost funktionen, der ikke kan udlignes med retraining. I dette tilfælde vil netværkets funktion sandsynligvis være spoleret så kraftigt, at stopkriteriet afbryder videre pruning.

En anden antagelse for OBD samt OBS er, at gradienten i udgangspunktet $\mathbf{w}_0$ er nul. Hvis netværket er optrænet med gradientmetoder (eller anden ordensmetode med utilstrækkelig regularisering, se kapitel 6), kan man ydermere komme ud for, at gradienterne i slutpunktet er så store, at de ikke kan negligeres. For at øge andenordens-approksimationernes pålidelighed, kan det derfor være ønskværdigt at benytte udtryk for OBD og OBS, hvor gradienten er medtaget i udledningen. I appendiks B er sådanne udledninger foretaget for både OBD samt OBS.

'Sand' saliency

Alt efter netværkets størrelse samt antallet af træningseksempler er det overkommeligt at beregne den '*sande*' saliency, dvs. den *rigtige* ændring af den kvadratiske cost funktion, fremfor at estimere denne. Vægtvektoren ændres simpelthen i overensstemmelse med specifikationerne for enten OBD eller OBS, og den oprindelige cost funktion $E(\mathbf{w})$ evalueres i det resulterende punkt, og sammenlignes med sin oprindelige værdi $E(\mathbf{w}_0)$. Med denne metode fjernes risikoen for at eliminere en forkert vægt grundet fejlagtige saliency-estimater; omkostningen er dog væsentligt øget beregningsarbejde.

4.3 Akaike's Final Prediction Error estimat

Som det fremgår af algoritmerne ovenfor, er der behov for et stopkriterie, der kan afgøre, hvornår den opnåede arkitektur er optimal; ved optimal forstås her optimal generaliseringsevne. For feedforward netværk er Akaike's Final Prediction Error estimat anvendt til dette formål med succes, se feks. [Svarer 93]. Estimatet søger at estimere den gennemsnitlige kvadratiske testfejl udfra fejlen på træningssættet med udtrykket

$$\widehat{E}_{test} = \frac{p+N}{p-N} E_{train} \tag{4.23}$$

hvor p er antallet af træningseksempler og N er antallet af frie parametre i netværket. En formel udledning af udtrykket kan feks. findes i [Rasmussen 93], og resultatet skitseres kort nedenfor. I udledningen gøres følgende antagelser:

- Uafhængighed mellem input og fejlen på outputtet.
- Tilstrækkelig kapacitet, dvs. netværket skal være i stand til at lære den bagvedliggende regel.
- Mange eksempler pr. vægt: $N/p \to 0$.

Akaike's FPE-estimat søger at estimere træningsfejlen givet ved den sædvanlige kvadratiske cost funktion (1.2) samt testfejlen i middel, når der midles over alle mulige træningssæt af en given størrelse fra den underliggende fordeling af træningseksempler, kaldet *læreren*. De optrænede *studenter* approksimeres med en førsteordens udvikling i vægtfluktuationer omkring lærerfunktionen. Herudfra udledes et udtryk for træningsfejlen samt testfejlen i middel over alle mulige træningssæt. De resulterende udtryk er

$$\langle E_{train} \rangle_p = \frac{p - N}{p} \cdot \sigma^2 + O\left[\left(\frac{N}{p}\right)^2\right] \tag{4.24}$$

samt

$$\langle E_{test} \rangle_p = \frac{p + N}{p} \cdot \sigma^2 + O\left[\left(\frac{N}{p}\right)^2\right] \tag{4.25}$$

hvor $\langle \cdots \rangle_p$ angiver middel over alle træningssæt af størrelse p. N angiver som nævnt ovenfor antallet af parametre i netværket, og σ^2 angiver variansen af støjen på data. Det bemærkes, at $\langle E_{train} \rangle_p$ ligger lavere end støjniveauet, mens $\langle E_{test} \rangle_p$ ligger over, svarende til at studenterne har 'lært' noget af støjen. Udfra disse udtryk kan man dels opnå Akaike's formel,

$$\langle E_{test} \rangle_p = \frac{p + N}{p - N} \langle E_{train} \rangle_p \tag{4.26}$$

dels et udtryk for støjniveauet

$$\sigma^2 = \frac{p}{p - N} \langle E_{train} \rangle_p \tag{4.27}$$

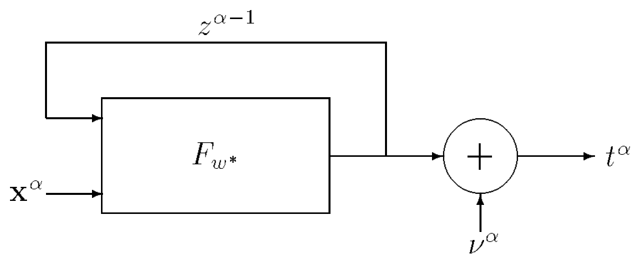
Ved optræning af et neuralt netværk har man ofte kun et enkelt, sparsomt træningssæt; dette er feks. tilfældet for solpletserien, der beskrives i kapitel 5. Derfor tilnærmes $\langle E_{train} \rangle_p$ blot med den faktisk opnåede træningsfejl E_{train}, hvilket resulterer i udtrykket (4.23) ovenfor.

Idet den kvadratiske cost funktion er udvidet med et weight decay, er der som nævnt i afsnit 1.1 lagt bånd på netværkets frihedsgrader. Dette burde der tages højde for ved angivelsen af N, som det også gøres i [Svarer 93]. Jeg har her valgt at antage, at forskellen imellem det effektive antal parametre og det faktiske antal parametre er ubetydelig, og N sættes derfor til det faktiske antal parametre.

Med Akaike's FPE-estimat er det nu muligt at estimere generaliseringsfejlen for hvert netværk i pruning-proceduren, og der vælges det netværk, der havde den laveste estimerede generaliseringsfejl.

Behandling af Akaike's FPE-estimat iøvrigt ligger *uden* for rammerne af dette projektarbejde. Det antages her uden videre, at FPE-estimatet også har berettigelse for rekursive netværk, selvom der hersker tvivl om, i hvilken udstrækning estimatet har gyldighed for tidsserieprædiktion med rekursive netværk.

I det følgende gøres der nogle 'uforpligtende' overvejelser omkring, hvad der skal gælde for et feedback system, for at udledningerne af FPE-estimatet



Figur 4.1: Model af lærerfunktionen for et feedback netværk.

i [Rasmussen 93] og de dér gjorte antagelser kan overføres på, og foretages tilsvarende for, et feedback system. Overvejelserne er baseret på ([Hansen 94b]).

I figur 4.1 vises en model af et feedback system med angivelse af lærerfunktionen F_{w^*}. $\mathbf{x}^\alpha$ er eksterne inputs til læreren til tidspunkt α. Lærerens output defineret ved

$$z^\alpha = F_{w^*}\left(z^{\alpha-1}, \mathbf{x}^\alpha\right) \tag{4.28}$$

antages at være et *støjfrit* feedback system. t^α er observationer af dette system, 'forurenet' med additiv, uafhængig støj:

$$t^\alpha = z^\alpha + \nu^\alpha \qquad\qquad \nu^\alpha \in \mathcal{N}\left(0, \sigma^2\right) \tag{4.29}$$

Studenterne søger at efterligne lærerfunktionen ved at minimere den kvadratiske cost funktion

$$E = \frac{1}{2}\sum_{\alpha=1}^{T}\left(t^\alpha - F_w\left(\hat{t}^{\alpha-1}, \mathbf{x}^\alpha\right)\right)^2 \tag{4.30}$$

hvor $F_w\left(\hat{t}^{\alpha-1}, \mathbf{x}^\alpha\right)$ er en students output til tidspunktet α. En students system antages at have samme opbygning som læreren i figur 4.1. Som for feedforward netværk antages det, at studenten har adgang til de samme inputs $\mathbf{x}^\alpha$ som læreren. For feedback systemer antages det ydermere, at de anvendte tidligere tilstande $\hat{t}^{\alpha-1}$ udelukkende afhænger af de tidligere værdier af $\mathbf{x}$, således at de er uafhængige af støjen ν. Dette er netop tilfældet, hvis studenten har samme opbygning som læreren, og modtager de samme inputs $\mathbf{x}^\alpha$.

Under disse antagelser, der ikke afviger fra de tilsvarende antagelser for feedforward netværk, approksimeres studenten som i [Rasmussen 93] med en førsteordens udvikling i vægtfluktuationer omkring lærerfunktionen

$$F_w\left(\hat{t}^{\alpha-1}, \mathbf{x}^\alpha\right) \approx F_{w^*}\left(z^{\alpha-1}, \mathbf{x}^\alpha\right) + \left.\frac{\partial F}{\partial \mathbf{w}}\right|_{w^*}\left(z^{\alpha-1}, \mathbf{x}^\alpha\right)\cdot\delta\mathbf{w} \tag{4.31}$$

hvorefter FPE-estimatet kan udledes som i [Rasmussen 93].

Ved sædvanlig et-skridts prædiktion af en tidsserie har studenten ikke adgang til $\mathbf{x}^\alpha$, men kun til et lag space af tidligere støjfyldte observationer af lærerens output, $t = z + \nu$. Hverken studentens input eller tidligere output

$\hat{t}^{\alpha-1}$ er i dette tilfælde uafhængige af støjen ν, hvorfor der i (4.31) yderligere vil forekomme afledede mht. støjen.

Hvis det imidlertid antages, at læreren og studenten kører *closed loop*, dvs. *ikke* modtager eksterne inputs $\mathbf{x}^{\alpha}$, vil studentens tidligere output $\hat{t}^{\alpha-1}$ være uafhængig af støjen ν, således at (4.31) er gyldig. Dette specielle tilfælde svarer til *itereret prædiktion* ([Weigend 90]), hvor netværket udelukkende baserer sine prædiktioner på egne tidligere outputs.

Kapitel 5

Datasæt

Solplet-serien

Solplet-serien er et datasæt, der har udviklet sig til en vigtig *benchmark* indenfor neurale netværk, se feks. [Weigend 90] og [Svarer 93]. Det skal også anvendes her, således at det er muligt at sammenligne resultaterne, der her opnås for de rekursive netværk med dem, der i litteraturen er opnået med feedforward netværk.

Datasættet består af gennemsnit for den årlige solplet-aktivitet fra 1700 til 1979, 280 punkter ialt; data er normeret til intervallet $[0;1]$. Pletterne på solen skyldes, at solens differentielle rotation forvrider, og dermed nedbryder, dens magnetfelt gennem dannelse af dipolpar, der lokalt afkøler overfladen. Det gennemsnitlige antal varierer fra år til år på overvejende periodisk vis med en frekvens på omring 11 år, som det fremgår af figur 5.1.

Datasættet deles normalt op i et træningssæt fra 1700 til og med 1920 samt i to testsæt fra hhv. 1921 til og med 1955 samt 1956 til og med 1979. Antallet af træningseksempler er således $p_{train} = 221 - L$, hvor L er antallet af forsinkede inputs, dimensionen af lag space.

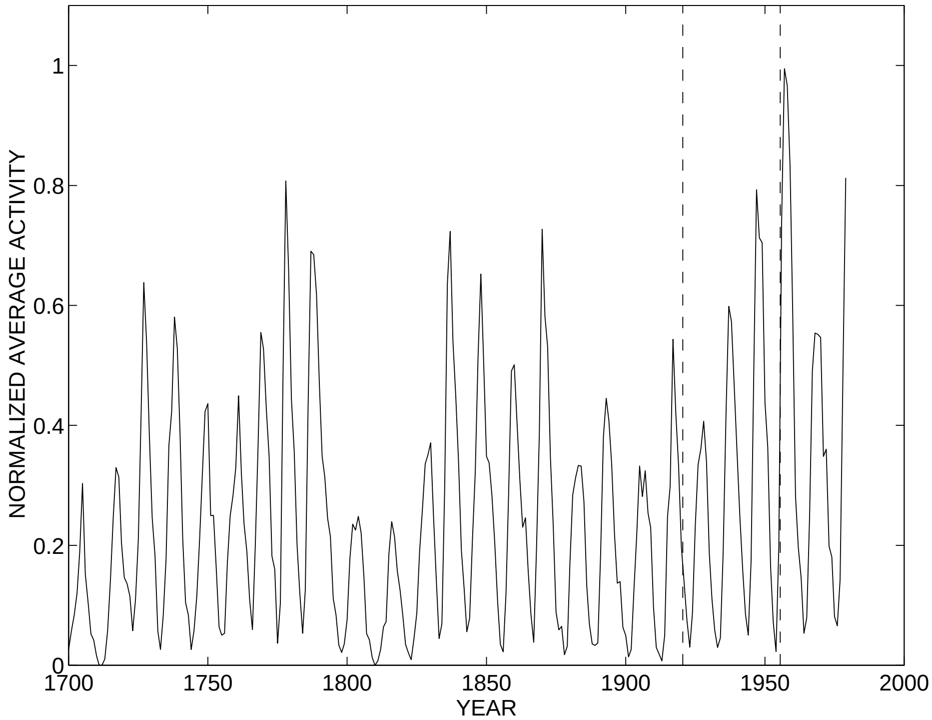
Årsagen, til at testsættet er delt op i to dele, er, at anden del er meget atypisk i forhold til det øvrige datasæt. Der vil derfor i de senere undersøgelser blive lagt størst vægt på generaliseringsevnen for testsæt 1 (1921–1955).

Performance for et netværk på et af ovenstående subset bestemmes udfra udtrykket

$$ARV_{set} = \frac{1}{\sigma^2}\frac{1}{p_{set}}\sum_{\alpha\in set}(x^\alpha - \widehat{x}^\alpha)^2 \qquad (5.1)$$

hvor ARV står for *Average Relative Variance.* σ^2 angiver variansen for hele datasættet, p_{set} angiver antallet af eksempler i det pågældende subset, og $x^\alpha - \widehat{x}^\alpha$ residualfejlen mellem det korrekte output for eksemplet og den prædikterede værdi. Ved at anvende dette udtryk til at evaluere et netværk, fjernes afhængigheden af de anvendte data's dynamiske område. Hvis prædiktionen svarer til middelværdien af datasættet, ses ARV at blive 1.0.

For at kunne sammenligne de rekursive netværk med feedforward netværk gengives her de resultater, der i [Weigend 90] blev opnået med et feedforward



Figur 5.1: Den normerede solplet-serie. 1700–1920 er træningssættet, 1921–1955 er testsæt 1 og 1956–1979 er testsæt 2.

netværk med 12 inputs, 8 skjulte enheder samt en lineær outputenhed på de tre subset:

$$ARV_{train} = 0.082 \qquad ARV_{test1} = 0.086 \qquad ARV_{test2} = 0.35$$

I [Svarer 93] blev der ved optræning af et antal netværk med samme opbygning som i [Weigend 90], hvor cost funktionen var forøget med weight decay, $\alpha^{in} = 0.02$ for inputvægtene samt $\alpha^{out} = 0.01$ for outputvægtene, opnået følgende middelværdier samt standardafvigelser:

$$ARV_{train} = 0.078 \pm 0.002 \qquad ARV_{test1} = 0.104 \pm 0.005$$

$$ARV_{test2} = 0.46 \pm 0.07$$

Disse netværk blev beskåret med Optimal Brain Damage, og for netværkene udpeget af Akaike's FPE-estimat opnåedes følgende:

$$ARV_{train} = 0.090 \pm 0.001 \qquad ARV_{test1} = 0.082 \pm 0.007$$

$$ARV_{test2} = 0.35 \pm 0.05$$

Kapitel 6

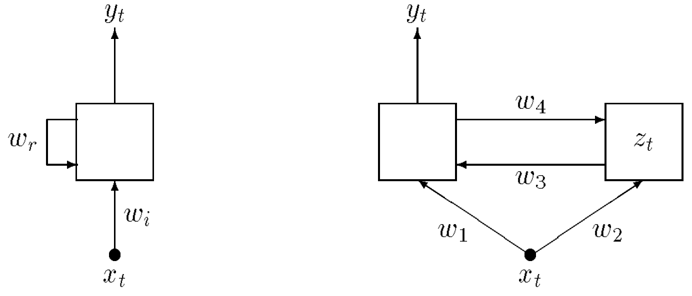
Optræning af layered RTRL

Dette kapitel omhandler optræning af layered RTRL netværk. Kapitlet indledes med et lille eksempel, der har til formål at illustrere de rekursive netværks muligheder for at opbygge hukommelse om tidligere inputs i modsætning til sædvanlige feedforward netværk. Herefter argumenteres der, inden de større tests påbegyndes, imod feedback i det lineære output, da dette kan resultere i ustabilitet, hvilket der gives et eksempel på. Herefter afprøves optræningsmetoderne beskrevet i kapitel 1 på et givent netværk. I forbindelse med optræning med andenordens metoder viser det sig, at der opstår problemer med dårligt konditionerede Hessian-matricer, og de resterende afsnit i kapitlet omhandler, hvorledes disse opstår og afhjælpes.

For at give mulighed for arbitrært dynamisk område, benyttes der lineær outputenhed. For de skjulte enheder benyttes aktiveringsfunktionen tanh.

6.1 Indledende forsøg

Rekursive neurale netværk er i stand til at opbygge en *intern* hukommelse tilpasset en given tidsserie (forudsat at netværket indeholder tilstrækkelige ressourcer i form af skjulte enheder) i modsætning til feedforward netværk, hvor hele hukommelsen bestemmes eksplicit af dimensionen af det anvendte lag space. Nedenfor antydes det, hvorledes denne interne hukommelse bygges



Figur 6.1: De to typer feedback forbindelser, som fuldt forbundne feedback netværk er opbygget af.

op, ved at opskrive simple udtryk for outputtet fra de to typer feedback forbindelser, som fuldt forbundne feedback netværk er opbygget af, se figur 6.1.

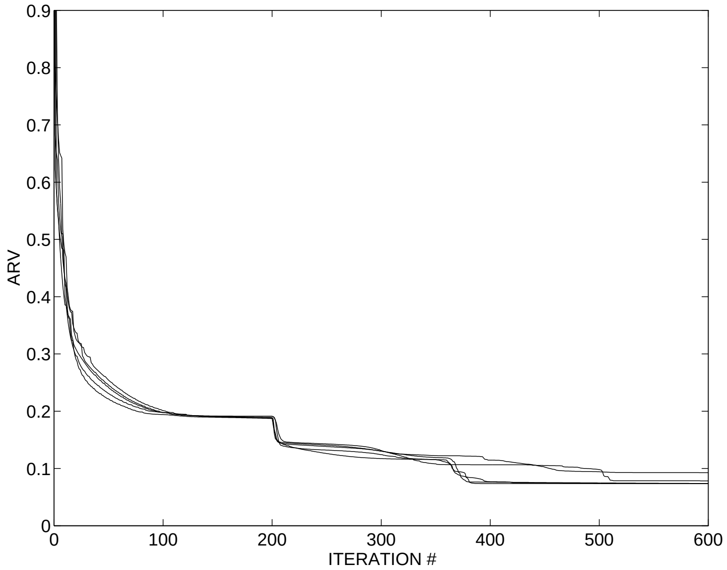
Hvis det for simpelhedens skyld antages, at enhederne har lineær aktiveringsfunktion, bliver outputtet y_t fra enheden til venstre med lokalt feedback

$$\begin{aligned} y_t &= w_i x_t + w_r y_{t-1} \\ &= w_i x_t + w_r(w_i x_{t-1} + w_r y_{t-2}) \\ &= w_i x_t + w_r(w_i x_{t-1} + w_r(w_i x_{t-2} + w_r y_{t-3})) \\ &= \ldots \end{aligned} \tag{6.1}$$

og for de ikke-lokale feedback til højre

$$\begin{aligned} y_t &= w_1 x_t + w_3 z_{t-1} \\ &= w_1 x_t + w_3(w_2 x_{t-1} + w_4 y_{t-2}) \\ &= w_1 x_t + w_3(w_2 x_{t-1} + w_4(w_1 x_{t-2} + w_3 z_{t-3})) \\ &= \ldots \end{aligned} \tag{6.2}$$

Begge former for feedback resulterer i outputs y_t, der afhænger af alle de foregående inputs x_k , $k \leq t$; de to former for feedback forbindelser ses her at være ækvivalente, hvis $w_1, w_2 = w_i$ og $w_3, w_4 = w_r$; feedback medfører således implicit et uendeligt lag-space. Enhedernes anvendelse af de tidligere inputs ses at være underlagt restriktioner i kraft af, at disse tidligere inputs i stigende grad modificeres af vægtene (samt aktiveringsfunktionen), jo længere tilbage i tiden de ligger, hvilket svarer til, at inputs i det implicitte lag space vægtes ned, jo længere tilbage i tiden, de ligger. Frihedsgraderne i



Figur 6.2: Træningsfejl for layered RTRL netværk med et enkelt input samt tre skjulte enheder.

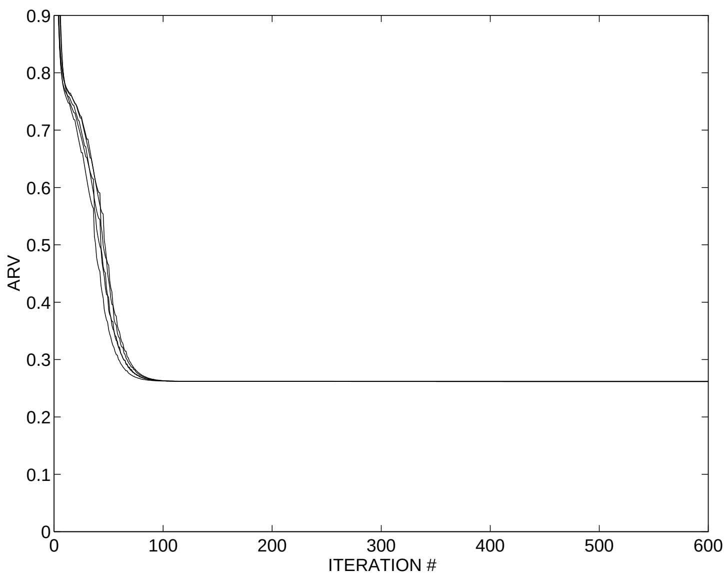
denne deformation øges med antallet af fuldt forbundne enheder i netværket, hvilket øger netværkets modelleringsevne svarende til, at dimensionen af det *effektive* lag-space forøges. Det effektive lag space kan beskrives ved netværkets *impulssvar*. I artiklen i appendiks I er impulssvaret vist for et netværk med et enkelt eksternt input, der er optrænet på solpletserien. Impulsen 'dør' først ud efter et tidsrum svarende til solpletseriens periode, hvilket kan tolkes som netværkets effektive lag space.

For at illustrere de rekursive netværks evne til at opbygge intern hukommelse, sammenlignes indlæringen for et sædvanligt feedforward netværk med otte skjulte enheder med et layered RTRL netværk analogt til figur 2.3 med tre skjulte enheder; netværkene modtog et enkelt input ad gangen, og havde hhv. 25 og 24 vægte. Optræningen foregik uden weight decay på solplet-serien med en gradient-metode efterfulgt af andenordens metoder for at træne 'helt i bund', og på figur 6.2 samt 6.3 vises forløbet af fem optræninger for hver netværk.

Middelværdi og standardafvigelse for fejlene på træningssættet blev

Feedforward	0.262 ± 0.000
Layered RTRL	0.079 ± 0.007

Da feedforward netværket ikke har nogen hukommelse, men blot implementerer en simpel funktion, skal det basere sin prædiktion på baggrund af blot et enkelt datapunkt. Den bedste strategi vil i dette tilfælde oftest være at lade prædiktionen være den foregående værdi, $\widehat{x}(t) = x(t-1)$. Hvis denne strategi forfølges for solplet-serien, resulterer dette i træningsfejlen $\mathrm{ARV}_{train} = 0.2903$, der ses at ligge meget tæt på feedforward netværkets fejl.

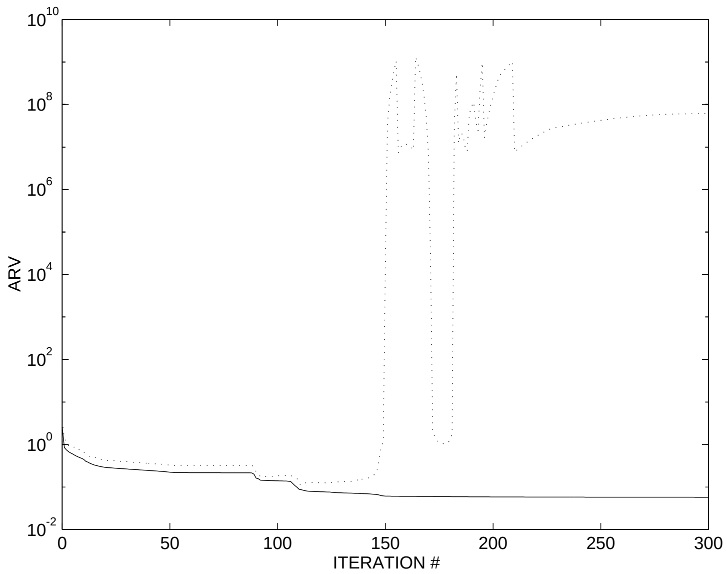


Figur 6.3: Træningsfejl for feedforward netværk med et enkelt input samt otte skjulte enheder.

Det rekursive netværks træningsfejl kan sammenlignes med fejlen fra de bedste feedforward rapporteret i [Svarer 93], hvilket indikerer, at selv et så relativt lille netværk er i stand til at opbygge en effektiv hukommelse.

6.2 Analyse af feedback i outputtet

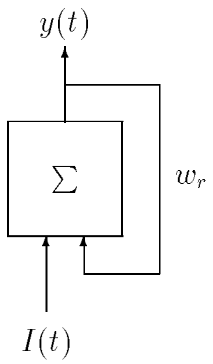
De i dette arbejde betragtede netværk har alle lineært output for at give mulighed for at dække et arbitrært dynamisk område. *Feedback* i den lineære outputenhed kan dog have katastrofale konsekvenser for netværkets stabilitet, hvilket illustreres i figur 6.4. I denne figur vises kurver for træningsfejl samt testfejl på solplet-serien, efterhånden som et fuldt forbundet feedback netværk med fire skjulte enheder samt et enkelt eksternt input optrænes.



Figur 6.4: Trænings- og testfejl for ustabilt feedback netværk. 'Solid' er fejlen på solpletseriens træningssæt, 'dotted' er fejlen på testsæt 1.

Mens fejlen på træningssættet falder monotont mod $ARV_{train} = 0.057$ i den sidste iteration, ses fejlen på testsættet at 'eksplodere' omkring iteration 150, den bliver pludselig meget stor. Årsagen til, at testfejlen ikke kommer over niveauet omkring 10^8 er, at der er indlagt en kunstig begrænsning for niveauet under beregningen af testfejlen. Var denne begrænsning ikke tilstede, ville den faktiske testfejl ligge på et niveau, der ikke kan repræsenteres i det anvendte 64–bit IEEE talformat, hvilket igen kan give besynderlige program-fejl. I den følgende analyse af den lineære outputenhed klarlægges det, hvad der forårsager den iagttagne ustabilitet.

Vi betragter nu en lineær outputenhed som den, der er afbildet i figur 6.5. Outputtet benævnes $y(t)$, feedback-vægten w_r, og det vægtede input fra de



Figur 6.5: Lineær outputenhed

skjulte enheder/eksterne inputs benævnes $I(t)$. Outputenheden er et lineært system, for hvilket der kan opskrives en differensligning:

$$y(t) = I(t) + w_r\, y(t-1) \tag{6.3}$$

Enhedens *impulssvar* bestemmes som svaret på inputtet $\delta(t) = (\ldots, 0, 1, 0, \ldots)$, der er nul for $t \neq 0$, se [Gabel 80]. Ved indsættelse af dette input i (6.3), findes impulssvaret til

$$h(t) = w_r^t \ , \quad t \geq 0 \tag{6.4}$$

Hvis input $I(t)$ til systemet er nul for $t < 0$ og $y(t) = 0$ for $t < 0$, kan output til tiden t findes som foldningen af inputsekvensen I med impulssvaret:

$$\begin{aligned} y(t) &= I(t) * h(t) \\ &= \sum_{n=0}^{t} I(n) \cdot h(t-n) \\ &= \sum_{n=0}^{t} I(n) \cdot w_r^{t-n} \end{aligned} \tag{6.5}$$

hvilket svarer til at addere svarene til tiden t på *impulserne* $\delta(n)\, I(n)$, hvor $n = 0 \ldots t$.

Et lineært system siges kort sagt at være stabilt, hvis en begrænset påvirkning af systemet fører til et begrænset svar; ved begrænset menes, at signalværdierne ikke vokser mod ∞ for $t \to \infty$. Dette er tilfældet, hvis rækken

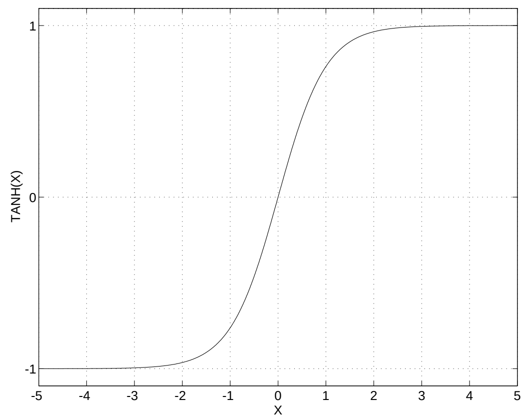
$$\sum_{t=0}^{\infty} |h(t)| \tag{6.6}$$

er konvergent ([Høholdt 89]). Rækken (6.6) for det her betragtede lineære system opskrives som

$$\sum_{t=0}^{\infty} \left| w_r^t \right| \tag{6.7}$$

der er konvergent for $|w_r| < 1$. For $|w_r| \geq 1$ vil det betragtede lineære system, her outputenheden for et neuralt netværk med feedback, være latent ustabilt.

En analyse som den foretaget ovenfor for en lineær enhed er ikke mulig for de ikke-lineære skjulte enheder, der her benytter aktiveringsfunktionen $y(x) = \tanh(x)$. Det er dog muligt at gøre sig nogle overordnede overvejelser over, hvordan den enkelte enhed vil opføre sig for forskellige inputs. For at hjælpe betragtningerne, er udseendet af aktiveringsfunktionen gengivet i figur 6.6.



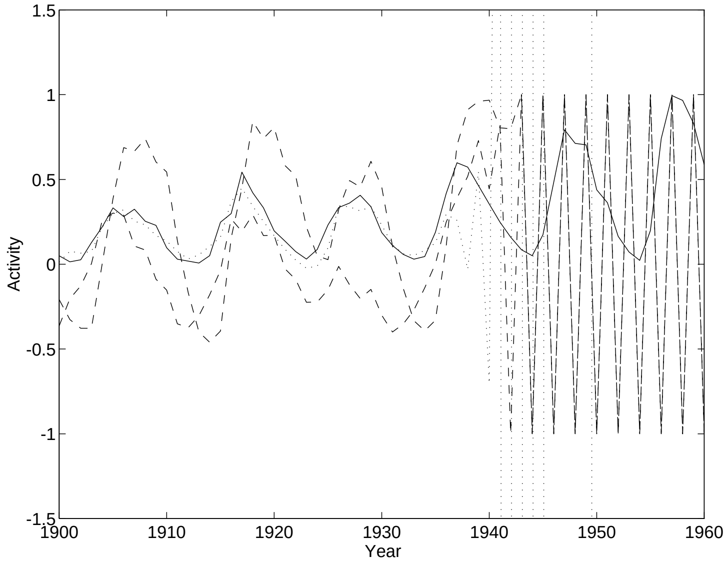
Figur 6.6: Aktiveringsfunktionen $\tanh(x)$.

Det bemærkes, at outputtet fra en skjult enhed er begrænset til intervallet $[-1\ ;\ 1]$, hvilket betyder, at de skjulte enheder *ikke* kan give anledning til ustabilitet, som den er defineret ovenfor.

Hvis feedback-vægten w_r for en skjult enhed numerisk er meget stor, samtidig med at de øvrige vægte i forhold til denne er numerisk små, vil den skjulte enhed opføre sig som en *flip-flop*. En flip-flop er kort fortalt et hukommelseselement anvendt i digitalteknikken, hvis output til tiden t antager een ud af to mulige tilstande, baseret på inputs til tiden $t-1$; se [Kohavi 78]. Hvis w_r er negativ, vil output *toggle* mellem $+1$ og -1, idet inputtet til enheden hele tiden skifter fortegn. Hvis w_r derimod er positiv, vil output være konstant, indtil de øvrige inputs til enheden evt. opnår en størrelse, der bevirker et fortegnsskifte.

Hvis alle vægte til den skjulte enhed imidlertid er af samme størrelse, er det dog vanskeligt om ikke umuligt at angive en kvalitativ analyse af den ikke-lineære enheds opførsel.

Vi vender nu tilbage til eksemplet, der indledte dette afsnit. Figur 6.7 viser det udsnit af solpletserien, hvor netværkets output begynder at vokse eksponentielt, og netværket bliver ustabilt. På figuren er udover solpletserien gengivet outputtet fra netværket samt outputtet fra skjult enhed nummer et og to. Outputtet fra netværket ses pludselig at begynde at divergere kraftigt fra det ønskede, hvorefter det begynder at oscillere; at outputtet tilsyneladende pludseligt holder op med at skifte fortegn, skyldes begrænsninger i MATLAB's plotfunktion, der bevirker, at ikke alle nulgennemgange for de meget store amplitudeforskelle gengives i det betragtede, forholdsvis snævre, interval. De store amplitudeforskelle understreges af, at for det her betrag-



Figur 6.7: Udsnittet af solpletserien, hvor ustabiliteten indtræffer. 'Solid' angiver solpletserien, 'dotted' netværkets output og stiplet angiver output fra de to første skjulte enheder.

tede netværk er $w_r \approx -11$! Nedenfor gengives samtlige vægte for netværket. Søjleindeks angiver, hvor forbindelsen kommer fra, rækkeindeks angiver, hvor forbindelsen går til:

Vægte for det betragtede netværk:

$\substack{fra\rightarrow \\ \downarrow til}$	$h1$	$h2$	$h3$	$h4$	$output$	$input$	$bias$
$h1$	1.0582	-0.4868	-0.1222	0.3346	-0.9162	1.1406	0.0921
$h2$	-2.3322	0.1933	3.4319	0.2691	-0.1343	2.6057	-0.1127
$h3$	1.3264	-0.3066	-0.5439	0.3075	-1.1250	0.7979	0.1307
$h4$	2.9897	0.2793	-2.7995	0.2243	-0.3509	-0.2343	-0.0734
out	24.4856	1.0150	-31.6309	6.2257	-10.7771	-2.0628	2.4116

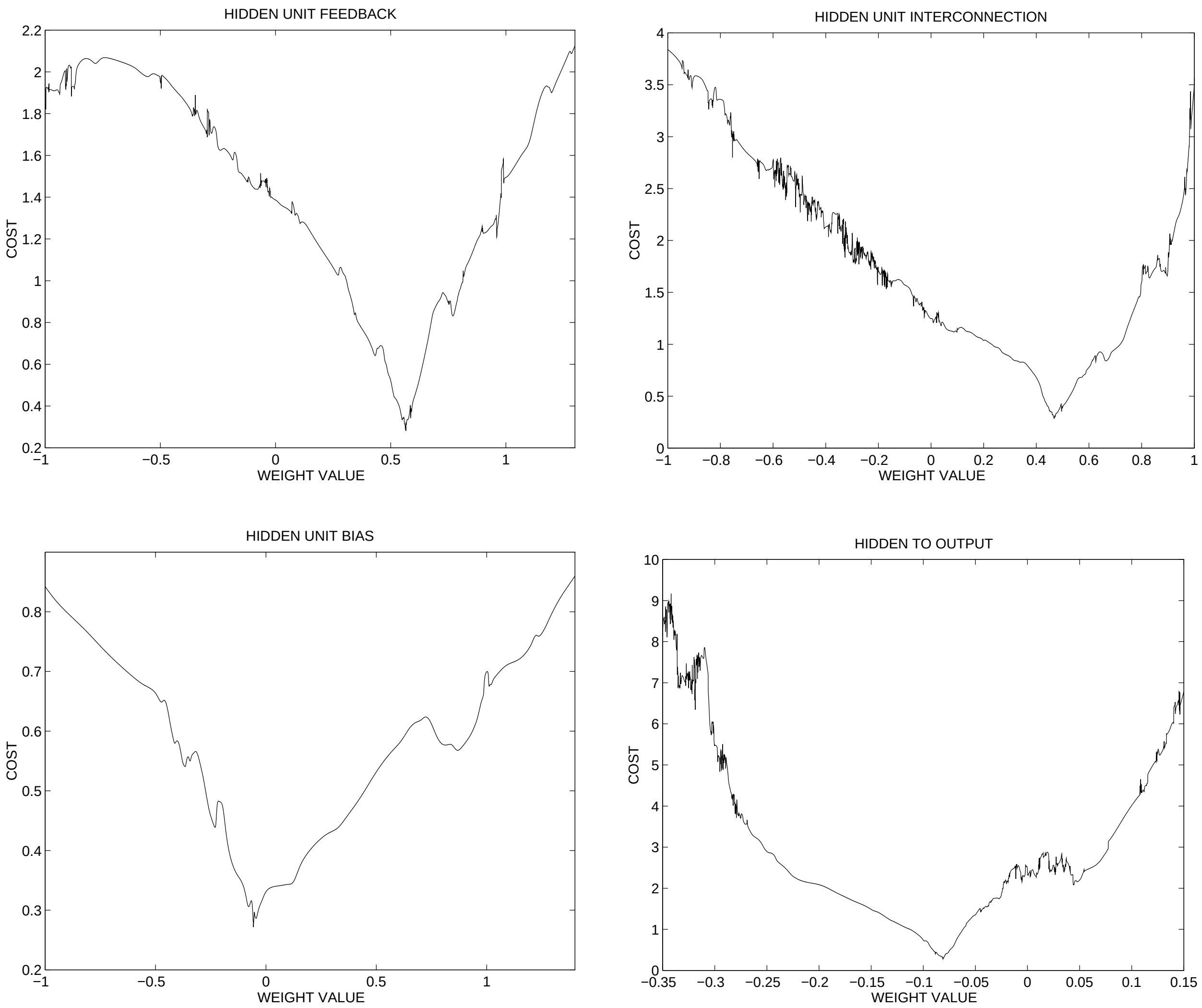
Når outputtet fra netværket vokser, ses de skjulte enheder at opføre sig som flip-flops, de toggler mellem $+1$ og -1. Dette sker, fordi de tidligere outputs fra outputenheden, der føres tilbage til de skjulte enheder, dominerer alle øvrige inputs. Da vægtene fra outputtet til de skjulte enheder alle ses at være positive, vil de skjulte enheders output til tiden t således have samme fortegn, som outputenheden havde til $t-1$.

Det bemærkes, at vægtene til outputenheden er meget store i forhold til de øvrige vægte i netværket; at netværket ikke bliver ustabilt før et stykke inde i testsættet må betyde, at vægtene 'udligner' hinanden gennem træningssættet; ellers ville der ikke være opnået så lav en træningsfejl, som tilfældet er. I testsættet mødes der en række 'atypiske' inputs, der *ikke* kan udlignes, og ustabiliteten er en realitet.

På grund af de stabilitetsproblemer, der kan opstå med feedback i outputenheden, vil dette *ikke* blive benyttet i det følgende.

6.3 Illustration af cost funktion

I dette afsnit søges det at give et billede af, hvordan cost funktionen for et layered RTRL net egentlig ser ud. I figur 6.8 er vist et snit gennem cost funktionen for forskellige vægte, dvs. een vægt varieres, mens de øvrige holdes konstante. Vægtene stammer fra et optrænet netværk med fire skjulte enheder der modtager et enkelt eksternt input. Cost funktionen er udvidet med et *lille* weight decay, $\alpha = 10^{-6}$.



Figur 6.8: Snit gennem den forøgede cost funktion for 4-1 layered RTRL netværk *uden* feedback i outputtet.

Cost funktionen ses at være kraftigt ikke-kvadratisk, og er i visse områder endog særdeles følsom overfor små ændringer i vægtene, hvilket vanskeliggør optimering. Tilsvarende snit for feedforward netværk afslører sædvanligvis en mere 'glat' flade (se feks. [Gorodkin 93]), så de kraftige variationer for layered RTRL må tilskrives feedback'et.

6.4 Optræning med førsteordens metoder

Dette afsnit indleder en mere systematisk test af optræningsmetoder på layered RTRL arkitekturen. Metoderne vil blive afprøvet på solpletserien, da der for denne serie findes et rigt sammenligningsgrundlag, se kapitel 5. Solpletseriens testdata er delt op i to separate testsæt, men i det følgende vil der af hensyn til overskuelighed kun rapporteres testresultater på testsæt 1, den del af testsættet, der har størst lighed med træningssættet.

Det testede netværk har fire skjulte enheder, og modtager et enkelt eksternt input, der også er forbundet til outputenheden. Den eneste vægt i dette fuldt forbundne netværk, der ikke benyttes, er outputenhedens feedbackforbindelse til sig selv; netværket indeholder således 34 vægte. Disse initialiseres med tilfældige værdier i intervallet $[-0.1\,;\,0.1]$, der antages at være så tilpas små, at de skjulte enheder ikke fra start går i mætning, hvilket vil hæmme optræningen.

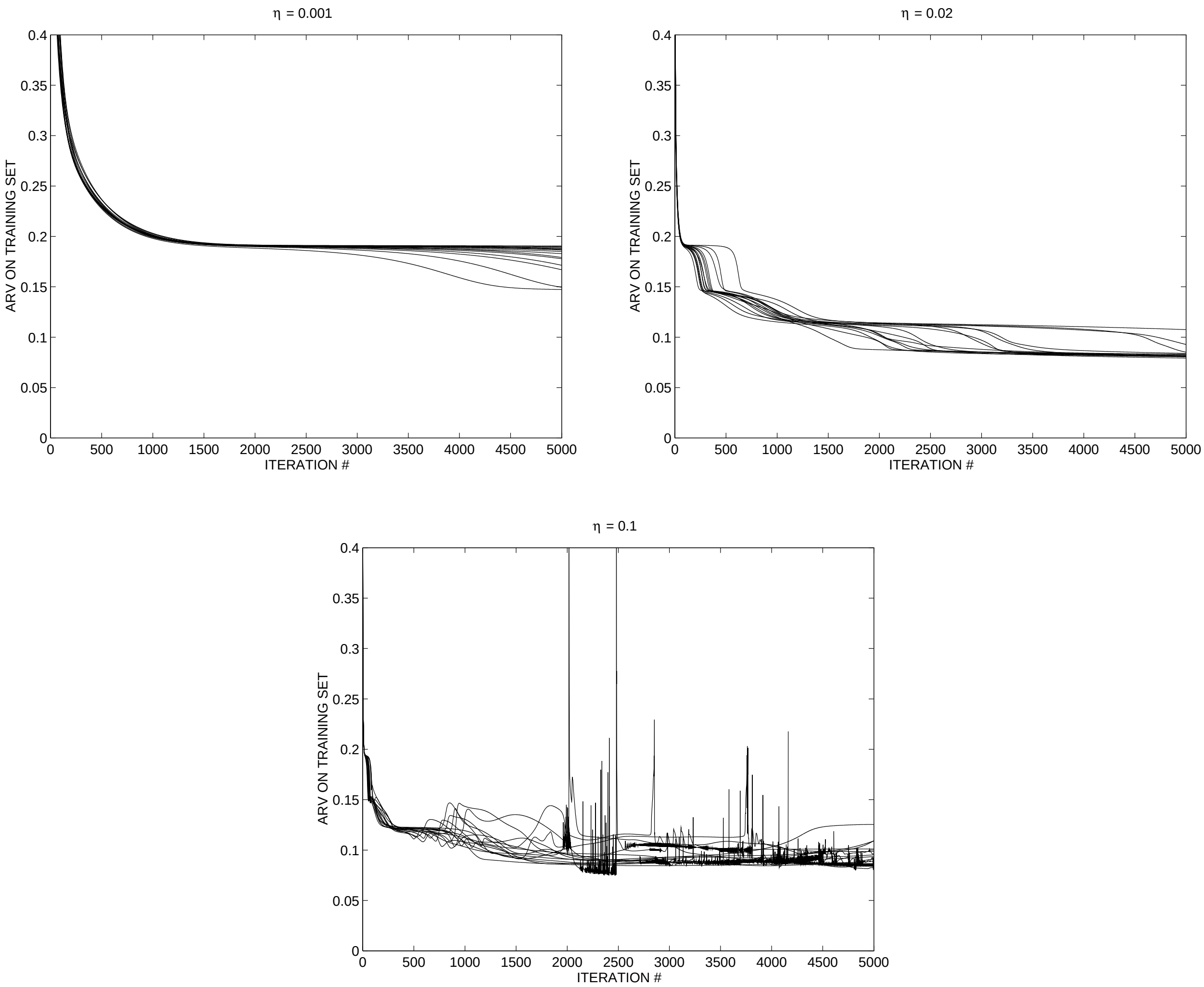
De anvendte førsteordens metoder er Real Time Recurrent Learning algoritmen samt gradient descent med liniesøgning. Descent. Fælles for begge metoder er, at de blot benytter gradienten som søgeretning efter minimum.

6.4.1 Real Time Recurrent Learning

Der indledes med den oprindelige optræningsmetode foreslået af Williams & Zipser, *Real Time Recurrent Learning*, der er beskrevet i afsnit 2.2.1. Som navnet antyder, er RTRL en *Online*-metode, dvs. vægtene opdateres efter præsentation af hvert eksempel i retning modsat gradienten for det enkelte eksempel. Som det ofte er tilfældet for gradient-baserede optræningsmetoder for neurale netværk, fastsættes learning rate parameteren η forud for optræningen, se feks. [Hertz 91]. Valget af η er dog temmelig afgørende for metodens effektivitet, hvilket fremgår af figur 6.9. Her er afbildet størrelsen af træningsfejlen under optræningen for tre forskellige værdier af η; for at give et statistisk mere velfunderet indtryk af metoden, er der for hver værdi af η foretaget 15 forsøg. For de viste optræningsforløb var cost funktionen udvidet med et *lille* weight decay, $\alpha = 10^{-6}$. Hver iteration svarer til en præsentation af hele datasættet for netværket, en såkaldt *epoke* ([Weigend 90, Hertz 91]).

For $\eta = 0.001$ ses niveauet for træningsfejlen for de fleste netværk at ligge omkring 0.18 selv efter 5000 iterationer, hvilket er *mange*. Sammenlignet med netværket fra afsnit 6.1, der med en enhed mindre kunne trænes ned til $ARV_{train} \approx 0.08$, er dette temmelig ringe. For de fleste netværk ses træningsfejlen allerede efter 1500 iterationer at stabilisere sig på det høje niveau. Kun to netværk skiller sig ud og når ned omkring $ARV_{train} = 0.15$. Ændringer i træningsfejlen ses generelt at foregå meget langsomt, hvilket indikerer, at den anvendte η er for lille.

Med $\eta = 0.02$ er forløbet af træningsfejlen helt anderledes. Dels falder fejlen i begyndelsen langt hurtigere end før, dels når alle netværk ned på et væsentligt lavere niveau for fejlen, de fleste omkring $ARV_{train} = 0.08$;



Figur 6.9: Forløb af 15 forsøg med *Online* RTRL-optræning med hhv. for lille, 'tilpas' og for stor learning rate η

$\eta = 0.02$ er således et væsentlig bedre valg end $\eta = 0.001$.

Den sidste graf i figur 6.9 er for $\eta = 0.1$. I begyndelsen ses fejlen at falde hurtigere, end det var tilfældet for de to andre η-værdier, men herefter begynder forløbet af fejlen at divergere voldsomt, træningsfejlen skiftevis stiger og falder. Dette er billedet af en for stor η, 'skridtene' bliver for lange, og fører ofte træningsfejlen op på et højere niveau. Det bemærkes dog, at næsten alle træningsfejl befinder sig på et forholdsvis lavt niveau ved afbrydelse af optræningen, men dette beror på en tilfældighed.

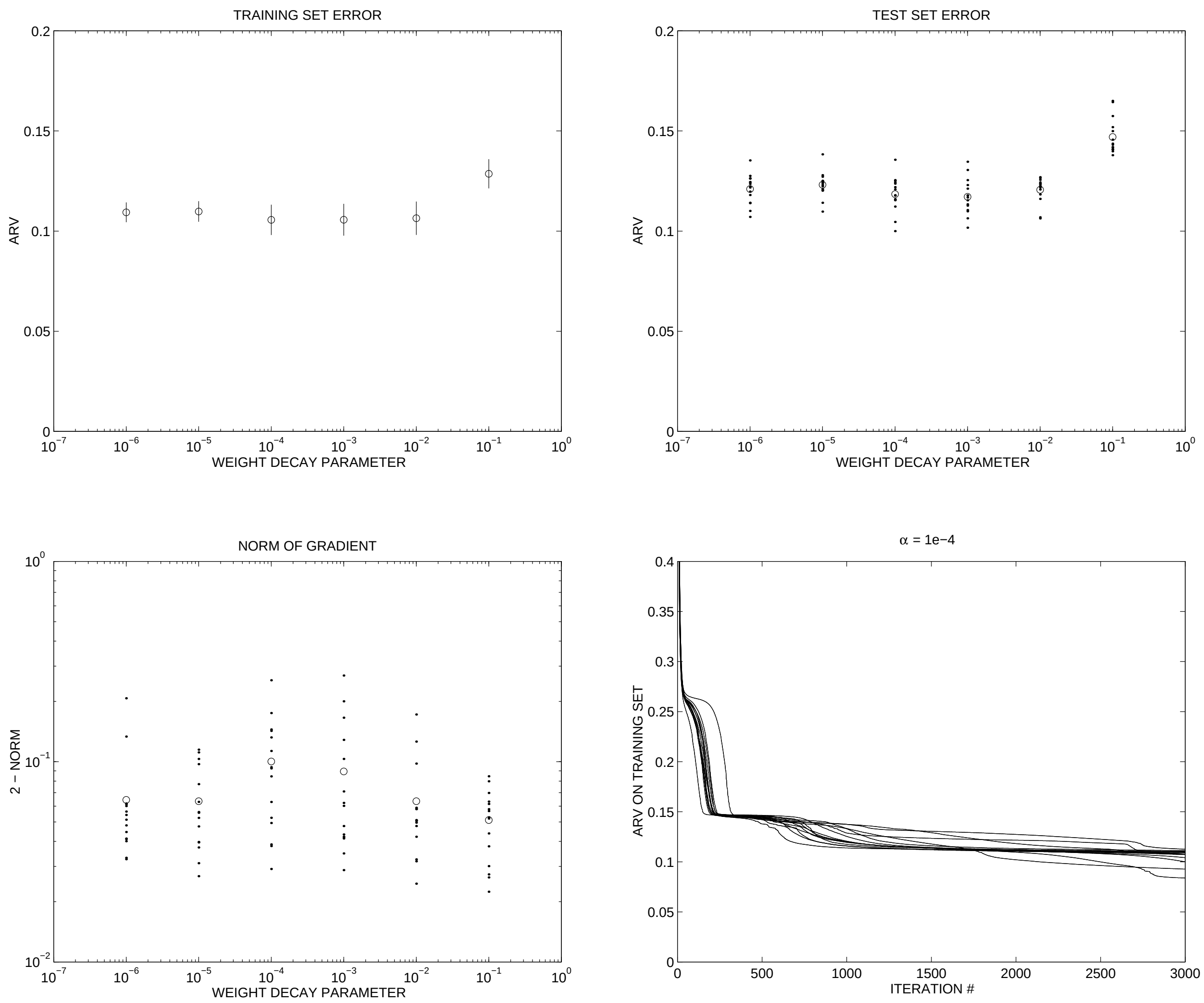
I tabellen nedenfor gengives middelværdi samt standardafvigelse for træningsfejlen efter 5000 iterationer for de tre η-værdier:

η	ARV_{train}
0.001	0.1786 ± 0.0136
0.02	0.0842 ± 0.0069
0.1	0.0965 ± 0.0113

De tre grafer i figur 6.9 illustrerer, hvor vigtigt et korrekt valg af η er for forløbet af optræningen af netværket. Eftersom der ikke eksisterer nogen måde, hvorpå man med sikkerhed kan udpege den 'korrekte' η-værdi at benytte, kan anvendelse af RTRL, og for så vidt også andre metoder med fast forhåndsbestemt learning rate, byde på store vanskeligheder i forsøget på at opnå tilfredsstillende optræning af netværket. Ydermere er konvergensen mod et (lokalt) minimum temmelig langsom, selv for et relativt fornuftigt valgt η. RTRL kan derfor kun anbefales til problemer, hvor online optræning er strengt nødvendig.

6.4.2 Gradient Descent

Gradient descent er en *batch mode* metode, dvs. vægtene opdateres i modsætning til RTRL først efter præsentation af samtlige træningseksempler i



Figur 6.10: Gradient descent optræning for forskellige weight decay parametre α.

retning modsat gradienten for den totale cost funktion. Metoden indeholder en adaptiv bestemmelse af learning rate η, således at problemet med valg af optimalt η elimineres. Begyndelsesværdien for η er sat til een, og den optimale værdi bestemmes i overensstemmelse med liniesøgningen beskrevet i afsnit 1.3.

Figur 6.10 indeholder grafer, der illustrerer resultatet af de udførte forsøg. For forskellige værdier af weight decay parameteren α er der for hver værdi udført 15 forsøg, hvert med maksimalt 3000 iterationer, hvor en iteration igen svarer til en epoke. Graferne angiver for de trænede netværk dels middelværdi (cirkel) og standardafvigelse for træningssættet, dels middelværdi med datapunkter for testfejl og 2-normen af gradienten i slutpunktet, samt træningsfejlene som funktion af iterationsantallet for den værdi af weight decayet, der gav den mindste middel-træningsfejl.

Det observeres, at weight decayet stort set ingen indflydelse har på den opnåede middeltræningsfejl, den ligger omtrent konstant lige over $ARV_{train} = 0.1$; den mindste middelfejl opnås med $\alpha = 10^{-4}$, hvor $ARV_{train} = 0.1056 \pm 0.0076$. For denne værdi af α er gengivet forløbet af træningsfejlene som funktion af iterationerne. For $\alpha = 0.1$ stiger træningsfejlen en del, svarende til, at weight decayet nu lægger betydelige begrænsninger på netværkets frihedsgrader.

Middelfejlene på træningssættet ligger en del over det bedste resultat med RTRL. Dette skal dog ses i lyset af, at der ved RTRL-forsøgene trænes 5000 iterationer, mens der med gradient descent, grundet det store antal forsøg, 'kun' trænes 3000 iterationer. Hvis man ser på forløbet af træningsfejlene, tyder det på, at fejlene ville falde yderligere, hvis der udførtes flere iterationer. Den benyttede gradient descent med adaptiv η-bestemmelse anses derfor for at være lige så god som RTRL med et optimalt valgt η.

Middeltestfejlene på det benyttede testsæt 1, se kapitel 5, påvirkes heller ikke af de varierende α-værdier, og ses ydermere at ligge en del over niveauet, der blev opnået i [Weigend 90], hvor $ARV_{test} = 0.086$. Dette skyldes ikke *overtræning*, da forløbet af testfejlene er meget lig forløbet af træningsfejlene, dvs. monotont faldende; testfejlene er dog ikke afbildet. Det høje niveau for testfejlene kan således tilskrives, at netværkene ikke er trænet tilstrækkeligt godt op.

Normen af gradienterne ligger på et relativt[1] højt niveau, ingen når ned på det tilsyneladende optimistiske stopkriterie sat til $\|\nabla C(\mathbf{w})\|_2 < 10^{-3}$; alle optræninger benytter de maksimale 3000 iterationer.

Dette faktum, sammenholdt med det høje niveau for træningsfejlen, peger på, at netværkene er langt fra at have nået et lokalt minimum for den udvidede cost funktion, hvorfor konvergensen mod et minimum for den her anvendte gradient-metode må betegnes som værende særdeles langsommelig.

[1]Niveauet af gradientnormen sammenlignes her med det niveau, der er opnået med andenordens metoder. Med Newtons metode opnås ofte $\|\nabla C(\mathbf{w})\|_2 = 10^{-10} - 10^{-12}$. Cost funktionens værdi ligger i et område omkring een, så de opnåede gradienter med Newtons metode er *meget* små.

6.5 Optræning med andenordens metoder

De i dette afsnit benyttede andenordens metoder omfatter pseudo Gauss-Newton samt *damped* Gauss-Newton; sidstnævnte benævnes dog ofte blot Gauss-Newtons metode. Sluttelig vises et eksempel på optræning med Newtons metode. Til forskel fra gradient-metoderne benytter metoderne anvendt her i større eller mindre grad information fra de anden afledede af cost funktionen til at bestemme den aktuelle søgeretning. Metoderne er således batch-metoder.

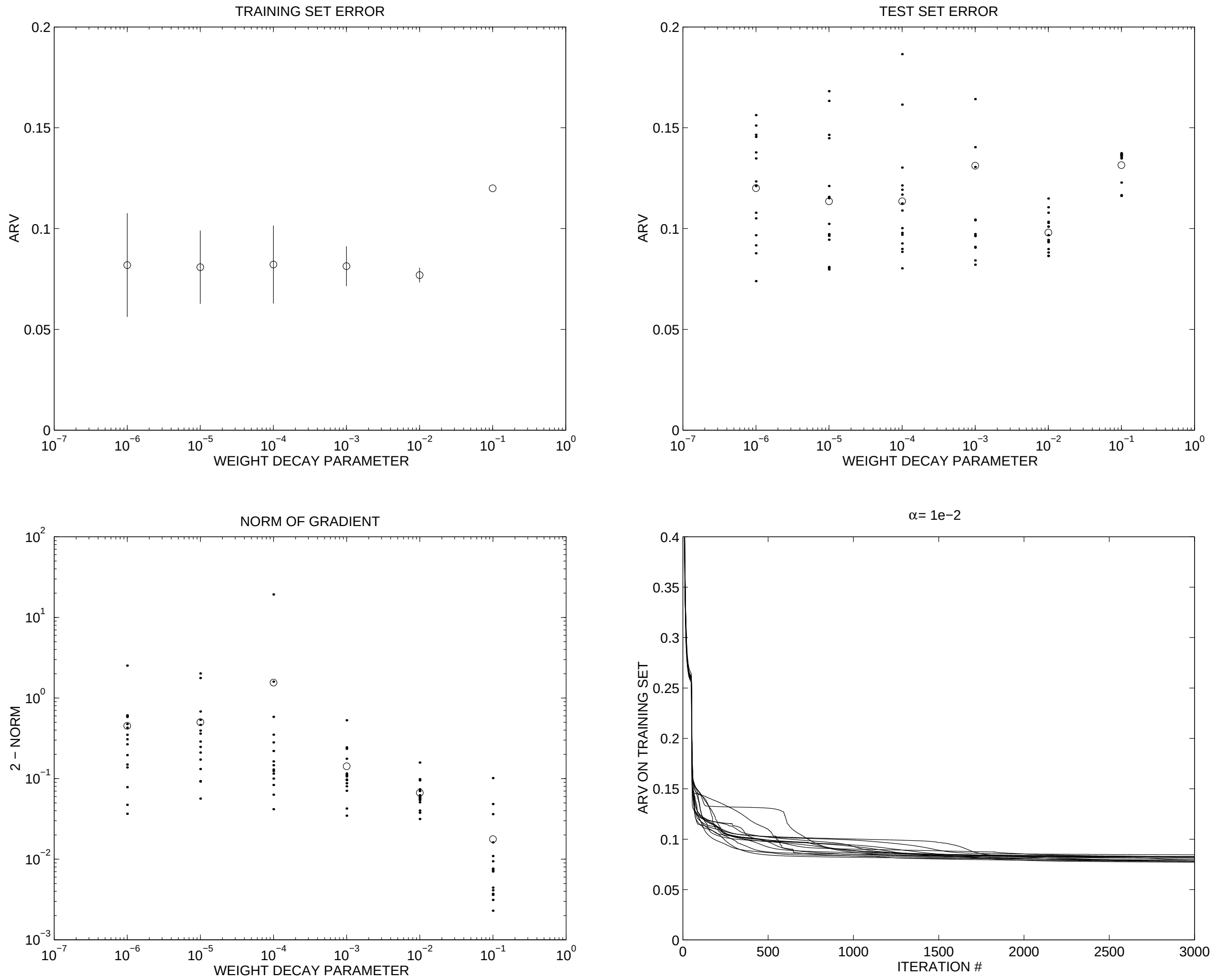
6.5.1 Pseudo Gauss-Newton

Pseudo Gauss-Newton benytter en grov diagonal-approksimation til Hessian-matricen med de anden afledede af cost funktionen, hvilket dog letter beregningsarbejdet ved bestemmelse af den aktuelle søgeretning. Når søgeretningen er bestemt, bestemmes 'skridtlængden' i form af learning rate η adaptivt på samme måde, som det blev gjort for gradient descent ovenfor; dog sættes begyndelsesværdien for η nu til den rigeligt store værdi $\eta_0 = 8$ for at give metoden mulighed for at tage store skridt, og slippe forbi eventuelle lokale minima. Pseudo Gauss-Newton er beskrevet i afsnit 1.4.3, hvor det også anbefales at indlede metoden med nogle iterationer af en ren gradientmetode, feks. gradient descent. Således indledes hvert af forsøgene i figur 6.11 med 50 iterationer af gradient descent-metoden, hvorefter der fortsættes med maksimalt 2950 iterationer af pseudo Gauss-Newton.

Graferne i figur 6.11 illustrerer de opnåede resultater for forskellige værdier af weight decay parameteren α, hvor der for hver værdi igen er udført 15 forsøg. Som tidligere angiver graferne for de trænede netværk dels middelværdi (cirkel) og standardafvigelse for træningssættet, dels middelværdi med datapunkter for testfejl og 2-normen af gradienten, samt træningsfejlene som funktion af iterationsantallet for den værdi af weight decayet, der gav den mindste middel-træningsfejl.

Weight decayet ses at have stor indflydelse på standardafvigelsen af træningsfejlen, jo større α desto mindre afvigelse. Dette skyldes sandsynligvis, at efterhånden som det stigende weight decay lægger større og større bånd på vægtene, reduceres antallet af lokale minima for cost funktionen. Ekstremet ses for $\alpha = 0.1$, hvor standardafvigelsen er så lille som $3.9 \cdot 10^{-5}$; samtlige optræninger resulterer i netværk med næsten ens træningsfejl.

Middelværdien af træningsfejlen ligger så godt som konstant omkring $ARV_{train} = 0.082$ indtil $\alpha = 10^{-2}$, hvor den falder lidt til $ARV_{train} = 0.0769$. Dette niveau ligger 20 – 25 % under de tilsvarende værdier for gradient descent-metoden, opnået med samme antal iterationer. Sammenlignes grafen for middel-træningsfejlene i figur 6.11 med den tilsvarende i figur 6.10 ses det, at niveauet for træningsfejlen for de dårligste netværk trænet med pseudo Gauss-Newton ligger på højde med niveauet for de bedste netværk trænet med gradient descent. Yderligere sammenlignes graferne på de to figurer,

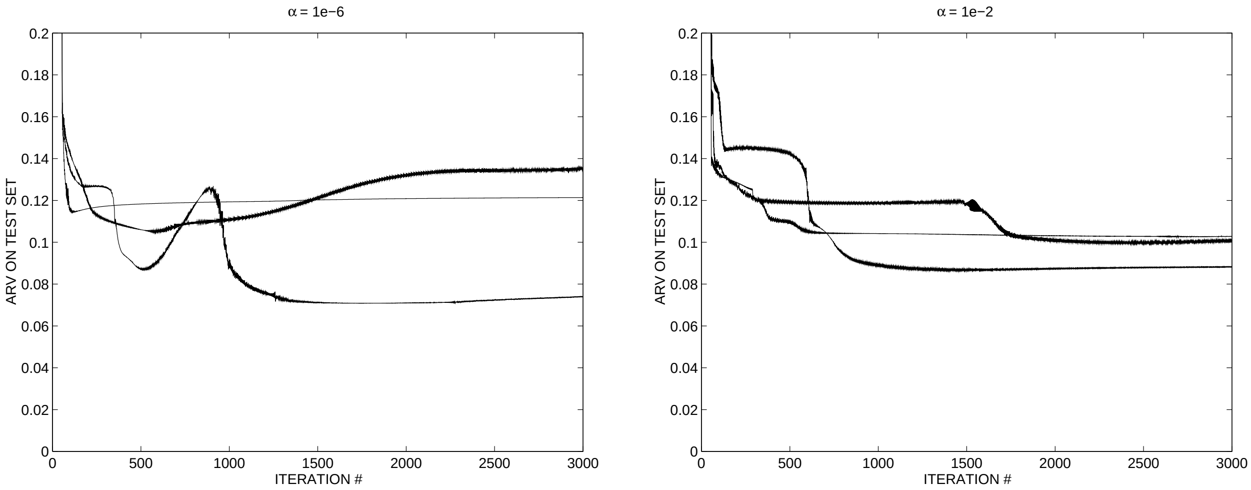


Figur 6.11: Pseudo Gauss-Newton optræning for forskellige weight decay parametre α.

der viser forløbet af træningsfejlene som funktion af iterationsantallet for de mindste middeltræningsfejl. Det ses her, at fejlen med pseudo Gauss-Newton falder langt hurtigere og til et lavere niveau, end det er tilfældet for gradient descent. Forløbet af træningsfejlene på figur 6.11 afslører ydermere, at disse allerede efter godt 1500 iterationer forbliver så godt som uændrede, fejlen falder her kun *meget* langsomt. Pseudo Gauss-Newton må således betegnes som værende væsentlig mere effektiv til optræning end gradient descent.

Middelværdierne for testfejlene ligger for de laveste værdier af α mellem 0.11 – 0.13, omtrent samme niveau, som det var tilfældet for gradient descent. Spredningen omkring middel er dog væsentlig større for pseudo Gauss-Newton, der er både særdeles gode og ret dårlige testfejl imellem. Den mindste testfejl findes for $\alpha = 10^{-6}$, hvor $ARV_{test} = 0.0739$; dette enkeltstående tilfælde kan dog ikke benyttes til ret meget andet end som en indikation af, at der rent faktisk *eksisterer* netværk med den i dette afsnit benyttede arkitektur, der kan måle sig med feks. de bedste feedforward netværk fra [Svarer 93], se kapitel 5.

Det bemærkes, at middelfejlen på testsættet for $\alpha = 10^{-2}$ er væsentlig mindre, end det er tilfældet for de lavere værdier af α; for $\alpha = 10^{-2}$ haves middelværdien for testfejlen over de 15 forsøg $\langle ARV_{test} \rangle = 0.0980$; samtidig ses spredningen af middelfejlene at være mindre. Dette skyldes, at netværkene med denne værdi af α er mindre tilbøjelige til at overtræne, hvilket nu er et problem med små værdier for weight decayet. For $\alpha = 0.1$ begrænses netværkets frihedsgrader dog så meget, at det går ud over generaliseringsevnen.



Figur 6.12: Forløbet af testfejl ved optræning med pseudo Gauss-Newton for to værdier af weight decay parameteren α.

I figur 6.12 gengives forløbet af testfejlen for nogle af de optrænede netværk for to forskellige værdier af weight decay parameteren α. For $\alpha = 10^{-6}$ ses overtræningen helt tydeligt, *niveauet* for testfejlen falder i begyndelsen, hvorefter det begynder at stige. Et enkelt af netværkene 'ombestemmer' sig dog efter ca. 1000 iterationer, hvorefter træningsfejlen falder brat til det lave niveau, der er beskrevet ovenfor. For $\alpha = 10^{-2}$ ses niveauet for testfejlen at være monotont faldende, weight decayet har modvirket overtræningen.

I beskrivelsen ovenfor af testfejlene i figur 6.12 skal 'niveau' forstås som middelværdien over et (mindre) antal iterationer, da det tydeligt ses, at testfejlen oscillerer temmelig kraftigt imellem de enkelte iterationer; dette er årsagen til det 'fluebensagtige' udseende af testfejlsforløbene. Så hurtige fluktuationer i testfejlene observeres sædvanligvis ikke for feedforward netværk, når der anvendes pseudo Gauss-Newton. Fænomenet tilskrives de rekursive netværks store følsomhed overfor ændringer i parametrene; en lille ændring i vægtene kan have store konsekvenser, når der benyttes feedback.

Niveauet for normen af gradienterne i slutpunktet varierer kraftigere, end det var tilfældet for gradient descent. Middelværdien falder dog med større weight decay, men ingen netværk når ned under stopkriteriet, der som for gradient descent er sat til $\|\nabla C(\mathbf{w})\|_2 < 10^{-3}$. Gradienternes niveau indikerer, at netværkene ikke helt har nået et lokalt minimum $\mathbf{w}_0$ for cost funktionen, i hvilket $\|\nabla C(\mathbf{w}_0)\|_2 = 0$; de efterfølgende andenordens metoder kan komme meget tættere på. I betragtning af det konstante niveau for træningsfejlene på grafen i figur 6.11 i de 1000 sidste iterationer er det tvivlsomt, om pseudo Gauss-Newton med et endeligt antal iterationer for rimelige værdier af weight decayet er i stand til at komme så tæt på et lokalt minimum, at stopkriteriet opfyldes.

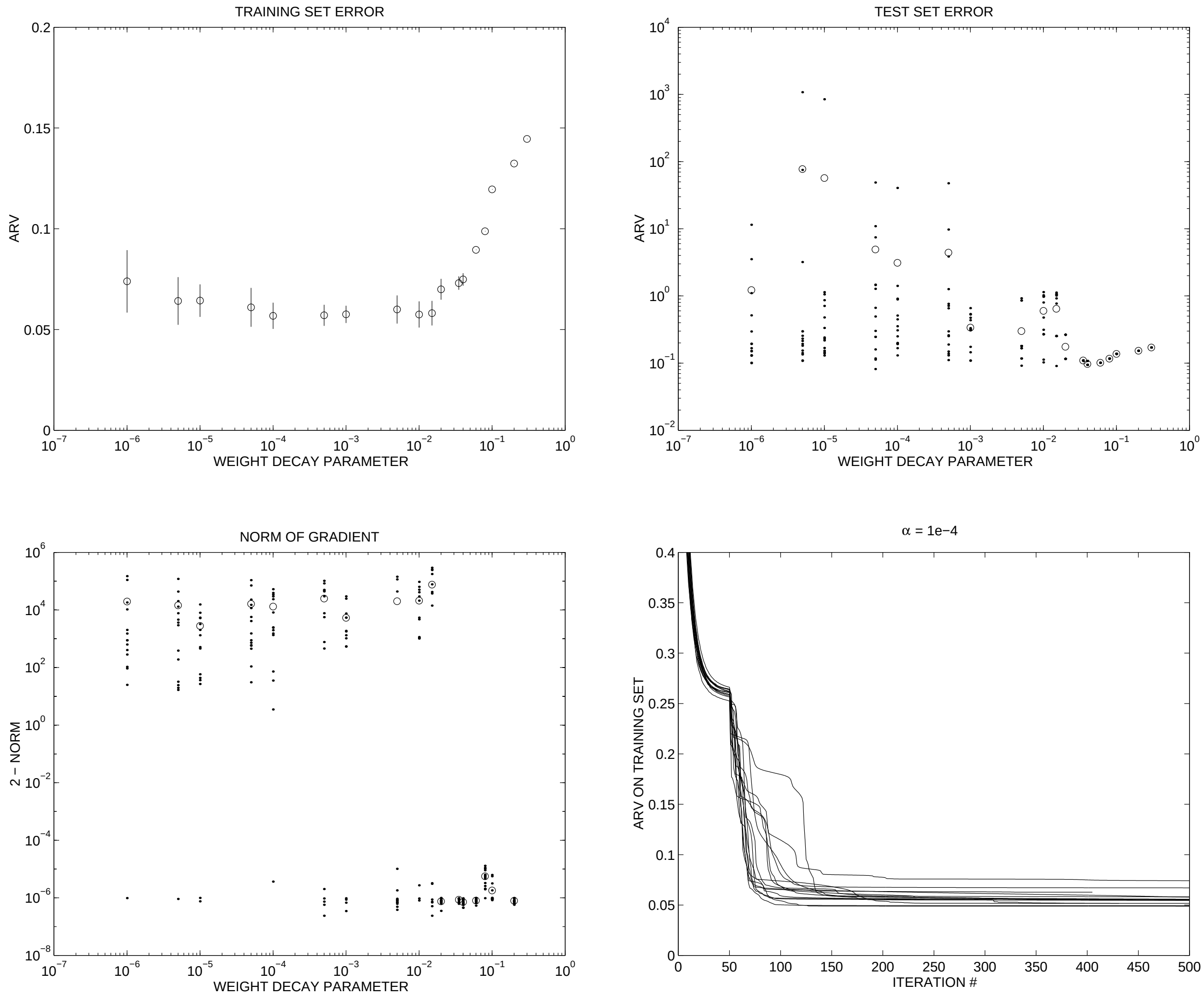
6.5.2 Gauss-Newton

Gauss-Newton's metode benytter en approksimation til den *fulde* Hessian-matrix, og det er således nødvendigt at løse et ligningssystem i hver iteration for at bestemme søgeretningen, som det beskrives i afsnit 1.4.2; de øgede anstrengelser for at bestemme søgeretningen skulle gerne resultere i hurtigere konvergens mod et (lokalt) minimum. Som for pseudo Gauss-Newton bestemmes skridtlængden η med den samme simple liniesøgning, af hvilken årsag metoden så rettelig burde kaldes *damped* Gauss-Newton, som det påpeges i [Dennis 83]; brug af det foranstillede tillægsord vil dog alligevel undlades i det følgende, da dets udeladelse ikke vurderes at kunne give anledning til alvorlige meningsforstyrrelser. Som det blev gjort for pseudo Gauss-Newton, indledes Gauss-Newtons metode konsekvent med 50 iterationer gradient descent for at bringe cost funktionen hen *i nærheden* af et (lokalt) minimum, inden metoden påbegyndes.

Graferne i figur 6.13 viser som for de tidligere beskrevne metoder de opnåede resultater for forskellige værdier af weight decay parameteren α. For hver værdi af α er der udført 15 forsøg, hvert med et maksimum på 450 Gauss-Newton iterationer efter de 50 indledende gradient descent iterationer.

Middelværdien for træningsfejlen ses at falde lidt med øget weight decay, hvorefter den stiger temmelig brat. Standardafvigelsen mindskes ligeledes med øget weight decay, for til sidst at blive meget tæt på nul. Det stigende weight decay lægger større og større begrænsning på netværkets frihedsgrader, således at Gauss-Newton optimeringerne til sidst finder frem til nøjagtigt det samme netværk, pånær en permutation af de skjulte enheder, i alle 15 forsøg for det pågældende α; påstanden om, at netværkene *er* de samme, understøttes af testfejlene for disse netværk, der alle ses at være ens.

Middelværdien af træningsfejlen ses i intervallet $\alpha \in [10^{-4}; 10^{-2}]$ at ligge nogenlunde konstant. Den mindste fejl opnås for $\alpha = 10^{-4}$, hvor $ARV_{train} = 0.0568 \pm 0.0065$; denne middelfejl er næsten fuldstændig identisk med fejlen for $\alpha = 10^{-2}$, og den ligger omkring 25 % under den tilsvarende laveste middelfejl for pseudo Gauss-Newton. Med Gauss-Newtons metode er det således muligt at bringe træningsfejlen væsentligt længere ned, end det viste sig praktisk muligt med pseudo Gauss-Newton endsige gradient descent. Forløbet af iterationerne for $\alpha = 10^{-4}$ i figur 6.13 illustrerer, at konvergensen mod den resulterende træningsfejl er temmelig meget hurtige-

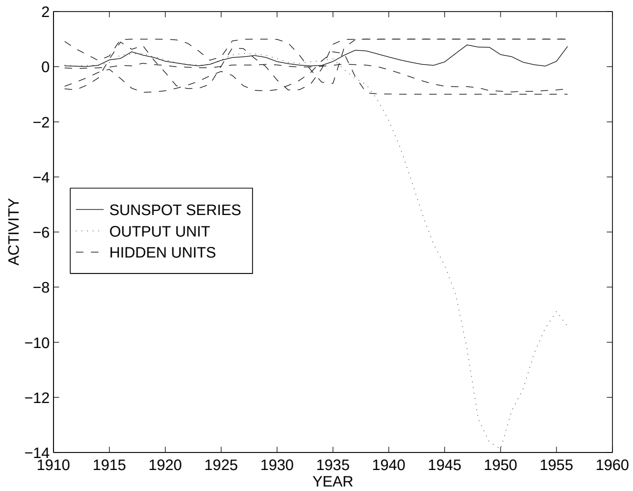


Figur 6.13: Gauss-Newton optræning for forskellige weight decay parametre α. Bemærk den logaritmiske skala for testfejlene.

re end nogen af de hidtil betragede metoder, målt i antal iterationer. Efter ca. 200 iterationer (inklusive de indledende 50 iterationer gradient descent) har træningsfejlen for langt de fleste netværk nået et niveau, hvor den ikke falder yderligere. Senere beskrives det, hvorfor den videre optræning så ikke bare afbrydes af stopkriteriet, men fortsætter for de fleste netværk, som det observeres i figur 6.13.

For de mindre værdier af weight decay parameteren α ses testfejlen i mange tilfælde at blive voldsomt stor, på trods af de lave træningsfejl. Disse store testfejl er den direkte årsag til, at testfejlene konsekvent illustreres ved de faktiske værdier, og ikke som en standardafvigelse, som det gøres for træningsfejlene; denne ville i givet fald nå ned *under* nul i nogle tilfælde.

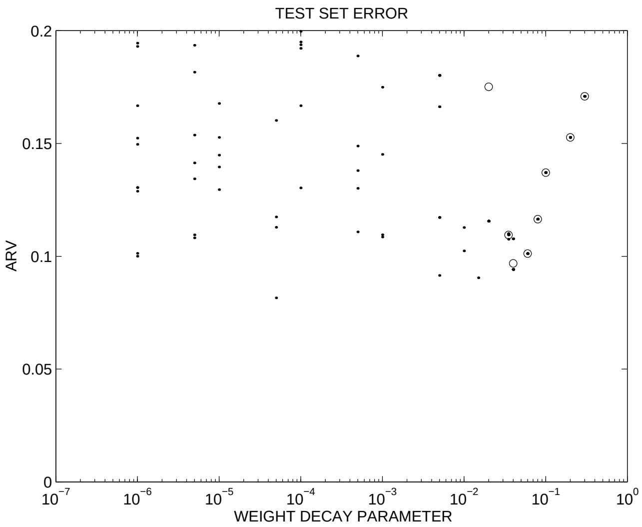
De store testfejl skyldes en kraftig overtræning, de enkelte netværk er ekstremt tilpasset træningssættet samt den støj, dette er behæftet med. De allerstørste testfejl må forklares med, at det pågældene netværk er ustabilt, som det var tilfældet i afsnit 6.2. Her kan årsagen blot ikke være feedback i outputenheden, men må tilskrives de skjulte enheder. Disse har begrænset output, således at det er vægtene til outputenheden, der sætter grænsen for outputtets størrelse. Som det blev nævnt i afsnit 6.2 er det dog ikke til at lave stabilitetsanalyse for de ikke-lineære skjulte enheder på samme måde, som det blev gjort for den lineære outputenhed. I figur 6.14 illustreres outputtet fra det netværk for $\alpha = 5 \cdot 10^{-6}$, der har størst testfejl. For dette netværk gælder, at $ARV_{train} = 0.05678$ samt $ARV_{test} = 1080$! Det ses, at de skjulte enheder går i mætning et stykke inde i testsættet, og derefter afhænger outputtet fra netværket udelukkende af det eksterne input samt af vægtene til outputenheden.



Figur 6.14: Output fra netværk med stor testfejl.

Der er stor spredning på de enkelte testfejl, og for at illustrere niveauet for de mindste, vises testfejlene i figur 6.15 med lineær skala på ordinat-aksen. Niveauet for disse mindste testfejl ses at ligge en del over de tilsvarende fejl for pseudo Gauss-Newton i figur 6.11, hvilket skyldes en kraftigere overtræning. Yderligere ses et helt klart minimum for middeltestfejlen for $\alpha = 0.04$, hvor $\langle ARV_{test} \rangle = 0.0969$. Weight decayet ses her at være så kraftigt, at resultatet af alle optræninger fordeler sig på blot to forskellige netværk pånær en permutation af de skjulte enheder.

Normen af gradienterne i slutpunktet ses for de mindre værdier af weight decay parameteren α for de fleste netværk at blive adskillige størrelsesordner større, end det har været tilfældet for de øvrige betragtede metoder. Kun ganske få når ned under stopkriteriet, der for Gauss-Newton metoden er sat til $\|C(\mathbf{w})\|_2 < 10^{-6}$, der vurderes at være *passende* lavt. Nogle få netværk med lille weight decay *når* dog ned under stopkriteriet, hvilket tages som indikation af, at stopkriteriet ikke er sat urimeligt lavt. Efterhånden som weight decayet øges, ses flere og flere netværk at ligge på niveau med det



Figur 6.15: Middeltestfejl samt datapunkter for forskellige weight decay parametre α ved optræning med Gauss-Newtons metode.

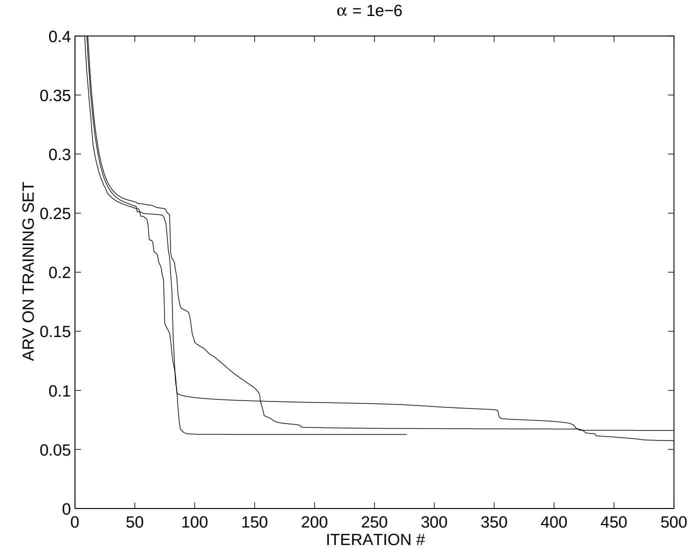
fastsatte stopkriterie; med et større maksimalt antal iterationer ville disse netværk alle kunne bringes til at opfylde stopkriteriet.

Det bemærkes, at der for de mindre α-værdier er en skarp opdeling mellem normen af gradienten i slutpunkterne, enten er den meget høj, eller også ligger den på niveau med stopkriteriet; i modsætning hertil er de tidligere undersøgte metoder, hvor normen af gradienterne i slutpunkterne var jævnt fordelt over et interval. Dette fænomen peger i retning af, at der opstår situationer under optræningen med Gauss-Newtons metode, der har afgørende indflydelse på, om metoden konvergerer mod et (lokalt) minimum, hvor $\|C(\mathbf{w})\|_2 = 0$, eller om metoden divergerer, hvad de store gradient-normer kunne tyde på.

Diagnose for gradient-problemet

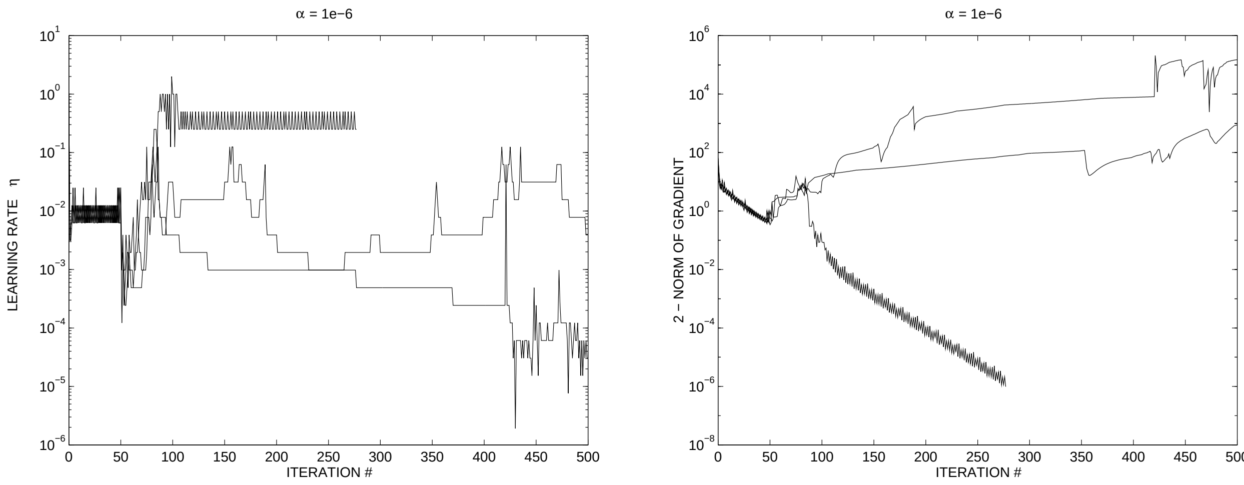
I figur 6.16 vises forløbet af træningsfejlen for tre af de netværk, der blev trænet op med weight decay parameteren $\alpha = 10^{-6}$. Det bemærkes, at træningsfejlen for alle tre er monotont faldende, den vokser på intet tidspunkt. Nu er træningsfejlen ikke proportional med den udvidede cost funktion $C(\mathbf{w})$, som der optimeres på, se kapitel 1, men den lille værdi af weight decayet betyder, at forløbet af træningsfejlen med en vis rimelighed afspejler forløbet af den udvidede cost funktion. Der mindes i denne forbindelse om, at den adaptive bestemmelse af learning rate η *altid* giver en værdi, der *formindsker* den udvidede cost funktion, som det beskrives i kapitel 1; kan dette ikke lade sig gøre med et fastsat antal halveringer af η, afbrydes optræningen, der så betragtes som mislykket.

I figur 6.17 er vist det tilsvarende forløb af skridtlængden η samt normen af gradienten for de tre netværk. For alle tre netværk gælder det, at normen



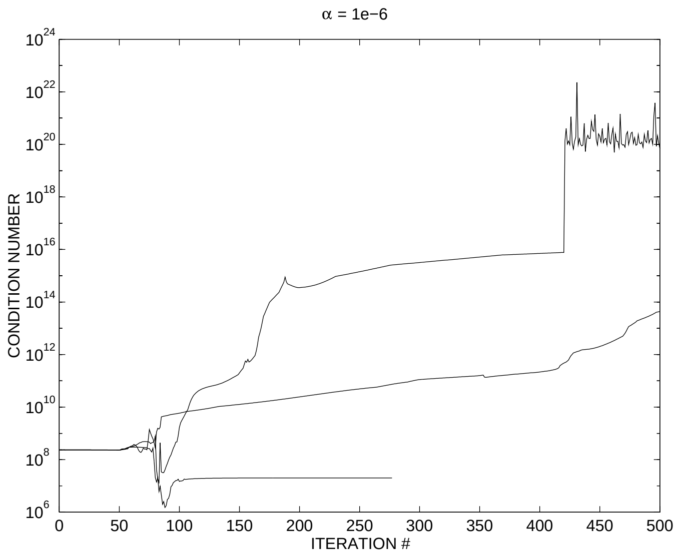
Figur 6.16: Forløb af træningsfejl for tre netværk med $\alpha = 10^{-6}$.

af gradienten stiger lidt i de første iterationer, hvor Gauss-Newton metoden anvendes. For det ene netværk begynder gradient-normen herefter at bevæge sig ned mod det fastsatte stopkriterium, samtidig med at η bevæger sig i området lige under een; der konvergeres mod et (lokalt) minimum. For de to andre netværk begynder gradient-normen derimod at vokse kraftigt, samtidig med at de tilhørende η'er bliver temmelig små. η-værdierne bliver dog ikke så små, at iterationerne afbrydes. Det maksimale antal tilladte neddelinger er sat til 24, og da den initielle værdi af η er sat til 8, svarer de 24 neddelinger til $\eta_{min} \approx 4.8 \cdot 10^{-7}$.



Figur 6.17: Forløbet af henholdsvis learning rate η og normen af gradienten for tre netværk med $\alpha = 10^{-6}$.

De små værdier af skridtlængden η betyder, at søgeretningen, der bestemmes ved at løse et lineært ligningssystem (se kapitel 1), er dårligt bestemt. Hvad der kan have indflydelse på dette, er den anvendte Hessian-approksimations *konditionstal*. I figur 6.18 er afbildet forløbet af konditionstallet som funktion af iterationsantallet for de tre betragtede netværk.



Figur 6.18: Forløb af konditionstallet for Gauss-Newton Hessianmatricerne for tre netværk med $\alpha = 10^{-6}$.

I appendiks D defineres konditionstallet for en matrix, ligesom konditionstallets betydning ved løsning af lineære ligningssystemer analyseres. Det oplyses, at løsningen til et lineært ligningssystem *kan* være upålidelig, hvis konditionstallet for den indgående matrix er større end $\epsilon^{-\frac{1}{2}}$, hvor ϵ er maskinnøjagtigheden for det benyttede tal-format. For det her benyttede IEEE 64-bit floating point format er $\epsilon \approx 2.22 \cdot 10^{-16}$, hvilket betyder, at søgeretningen *kan* være behæftet med signifikante fejl, hvis Hessian-approksimationen har et konditionstal, der er større end $6.7 \cdot 10^7$. I figur 6.18 ses dette i høj grad at være tilfældet for de to netværk, hvor det 'går galt'.

Årsagen til de store gradienter og heraf dårligt bestemte (lokale) minima for cost funktionen synes nu at være klarlagt, den består i dårligt konditionerede Hessian-approksimationer. I senere afsnit analyseres nogle af de mulige *årsager* til, at konditionstallene bliver så store.

De dårlige konditionstal kan give en forklaring på, hvorfor problemet med de store gradienter og heraf dårligt bestemte minima bliver mindre for stigende weight decay. I appendiks D defineres det her anvendte konditionstal for en matrix som værende den numeriske værdi af forholdet mellem største og mindste egenværdi for denne, og er således baseret på 2-normen for en matrix. Hvis matricen kaldes $\mathbf{H}$, har denne egenværdierne λ_j. Nu gælder

der ([Dennis 83]], at egenværdierne for $\mathbf{H} + \alpha\mathbf{I}$, svarende til Hessianmatricen for en cost funktion udvidet med simpelt weight decay, er $\lambda_j + \alpha$. Dette betyder, at hvis de mindste egenværdier er meget små, kan man forbedre Hessianmatricen $\mathbf{H}$'s konditionstal ved at regularisere cost funktionen med et simpelt weight decay:

$$\kappa(\mathbf{H} + \alpha\mathbf{I}) = \left|\frac{\lambda_{max} + \alpha}{\lambda_{min} + \alpha}\right| \approx \left|\frac{\lambda_{max}}{\alpha}\right| \tag{6.8}$$

Konditionstallet for $\mathbf{H}$ vil ved udvidelse med et simpelt weight decay α således omtrent ændres med

$$\kappa^{ny} \approx \kappa^{gammel} \cdot \left|\frac{\lambda_{min}}{\alpha}\right| \tag{6.9}$$

Effekten af et stigende weight decay konstateres i figur 6.13, hvor problemet er elimineret for de største værdier af α.

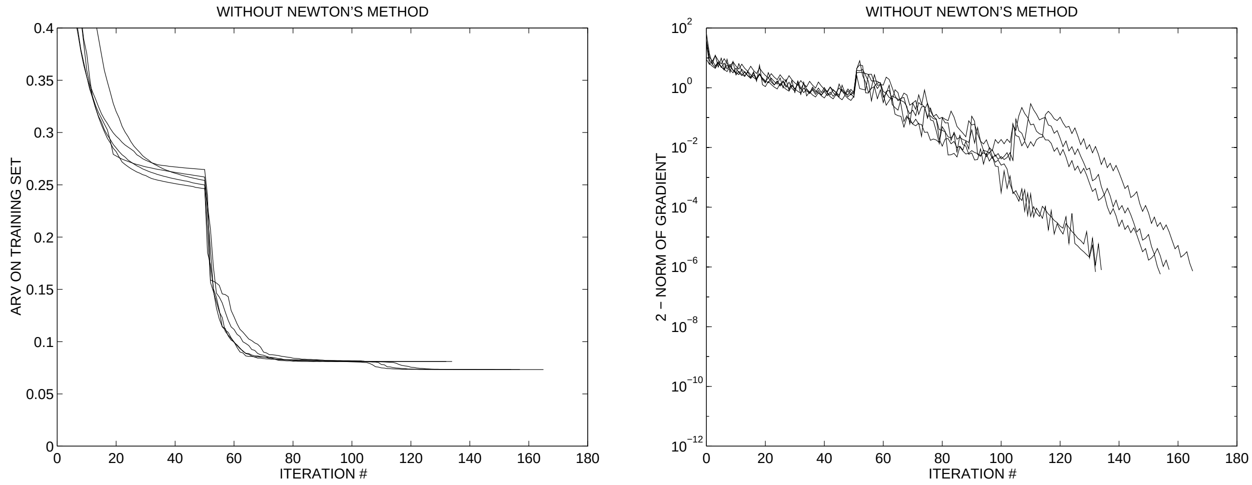
I appendiks E.1 gengives resultatet fra en forsøgsrække, hvor de 450 Gauss-Newton iterationer indledes med 500 pseudo Gauss-Newton iterationer for at bringe cost funktionen tættere på et minimum. Forsøget blev udført udfra den betragtning, at den dårlige konditionering kunne skyldes, at Gauss-Newtons metode blev påbegyndt 'for langt' fra et minimum, hvor andenordens approksimationen kan være meget ringe ([Dennis 83]). Som det kan ses, havde dette tiltag overhovedet ingen effekt, resultaterne er stort set identiske med de tilsvarende i figur 6.13.

6.5.3 Newtons metode

Newtons metode benytter til forskel fra de tidligere beskrevne andenordens metoder den *fulde* Hessian-matrix ved bestemmelse af søgeretningen. Som beskrevet i afsnit 1.4.1 kan det dog indtræffe, at den fulde Hessian i modsætning til Gauss-Newton approksimationen i et givent iterationspunkt ikke er positivt definit. Dette kan igen medføre, at søgeretningen bliver 'op ad bakke'; hvis denne inverteres ved at skifte fortegn for learning rate η, som det beskrives i afsnit 1.4.1, bliver den resulterende søgeretning tilfældig. For ikke at benytte en tilfældig søgeretning, kan man i disse tilfælde foretage en iteration med en anden optimeringsmetode, feks. Gauss-Newton, hvilket dog er beregningsmæssigt dyrt, da det kræver bestemmelse af endnu en søgeretning.

For så vidt muligt at undgå dette, bør Newtons metode først påbegyndes, når iterationspunktet er så 'tæt' på et (lokalt) minimum, at Hessian-matricen med stor sandsynlighed er positivt definit; dette vil være tilfældet, hvis normen af gradienten $\|C(\mathbf{w})\|_2$ er tilstrækkelig lille. Når først denne betingelse er opfyldt, vil Newtons metode konvergere væsentligt hurtigere mod det lokale minimum, end det er tilfældet for Gauss-Newtons metode, hvilket illustreres med et lille eksempel.

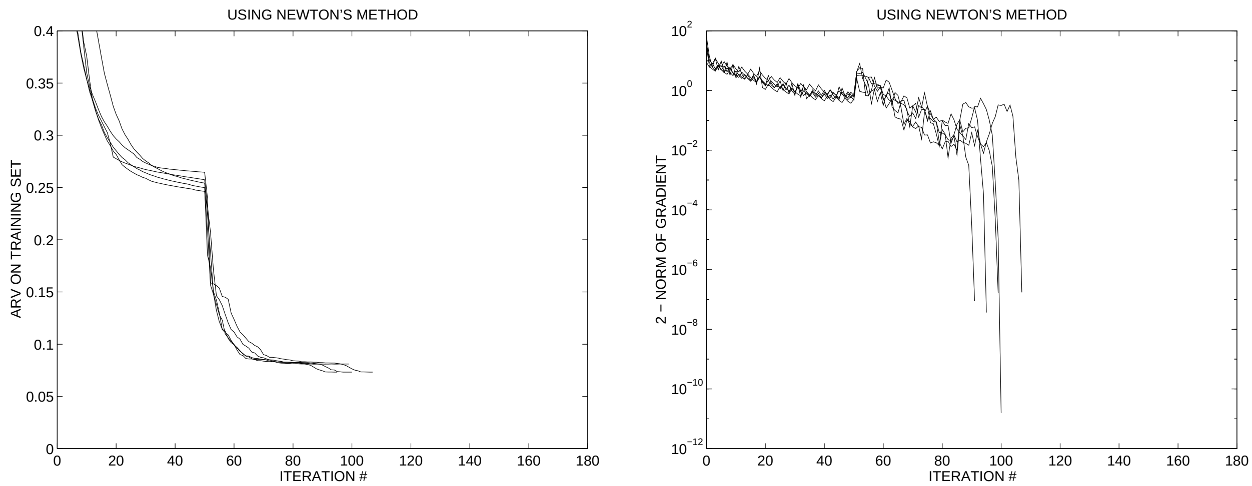
I figur 6.19 vises henholdsvis træningsfejl samt 2-normen af gradienten som funktion af iterationsantallet for fem netværk med fire skjulte enheder



Figur 6.19: Optræning af fem netværk med Gauss-Newtons metode indtil stopkriteriet $\|C(\mathbf{w})\|_2 < 10^{-6}$ opfyldes.

samt et enkelt eksternt input, som benyttet i de foregående afsnit. Cost funktionen er udvidet med et weight decay, $\alpha = 0.04$ for at undgå konditioneringsproblemer for Hessianmatricen. Optræningen af netværkene indledes med 50 iterationer gradient descent, hvorefter der fortsættes med Gauss-Newtons metode, indtil stopkriteriet $\|C(\mathbf{w})\|_2 < 10^{-6}$ opfyldes.

Herefter gentages optræningerne *fra samme startpunkter*, blot med den forskel, at der nu yderligere benyttes Newtons metode, når $\|C(\mathbf{w})\|_2 < 10^{-2}$, der i dette tilfælde har vist sig at være tilstrækkeligt tæt på et minimum til, at Newtons metode kan anvendes. For Newtons metode benyttes der som før



Figur 6.20: Optræning af fem netværk med Gauss-Newtons metode indtil stopkriteriet $\|C(\mathbf{w})\|_2 < 10^{-2}$ opfyldes; herefter fortsættes med Newtons metode.

stopkriteriet $\|C(\mathbf{w})\|_2 < 10^{-6}$. I figur 6.20 vises forløbet af den ændrede optræning. Det bemærkes, at hvor 2-normen af gradienten med Gauss-Newtons metode kun faldt langsomt, behøver Newtons metode blot nogle få iterationer til at opfylde stopkriteriet til rigt mål. Træningsfejlene i slutpunkterne er de samme som i figur 6.19; for begge metoder opnås middelværdi & standardafvigelse $ARV_{train} = 0.0764 \pm 0.0038$. Newtons metode konvergerer således formentlig mod de samme (lokale) minima som Gauss-Newton, konvergensen er blot væsentlig hurtigere.

Forskellen imellem de to metoders konvergens er dog ikke lige så udpræget, som det er tilfældet imellem Gauss-Newton og hhv. gradient descent samt pseudo Gauss-Newton. Hvor de to sidstnævnte metoder ikke var i stand til indenfor et stort antal iterationer at få træningsfejlen ned på samme niveau som Gauss-Newton pga. ekstremt langsom konvergens, ses Gauss-Newtons metode blot at være lidt flere iterationer om at nå ned på et givent niveau for gradientnormen end Newtons metode.

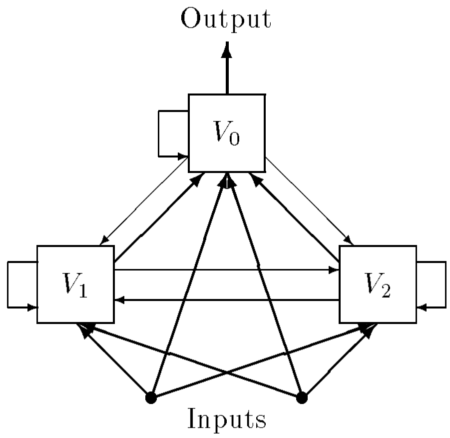
Når først iterationspunktet er tilstrækkeligt tæt på det lokale minimum (hvilket er vanskeligt at afgøre), vil der være en hastighedsgevinst i form af færre iterationer ved at anvende Newtons metode fremfor Gauss-Newton. I form af beregningstid er gevinsten dog stærkt afhængig af netværkets størrelse, hvilket fremgår af appendiks G. Her illustreres det, at forskellen imellem beregningstiden for søgeretningerne for de to metoder vokser kraftigt for stigende netværkskompleksitet. Det skal ligeledes tages med i betragtningerne, at Newtons metode kræver beregning af den *fulde* Hessianmatrix, hvilket kan være særdeles vanskeligt at implementere, jvf. afsnit 2.2.2. Eftersom der yderligere ikke er nogen kvalitetsmæssig forskel på de bestemte minima i form af opnået træningsfejl, vil der alt ialt generelt ikke vindes meget ved at implementere Newtons metode.

6.6 Opdeling af vægte

I afsnit 6.5.2 blev der beskrevet en forsøgsrække for et layered RTRL netværk med varierende weight decay parameter α, der optrænedes med Gauss-Newtons metode. Det blev konstateret, at for den værdi af α, hvor der blev opnået den laveste middeltestfejl, var regulariseringen så kraftig, at optræningerne resulterede i meget få *forskellige* netværk; de resulterende netværk var omtrent ens pånær en permutation af enhedernes indbyrdes placering. Det samme gør sig gældende for de forsøgsrækker, der beskrives i appendiks E.

Det anvendte weight decay havde yderligere indflydelse på, om Hessian-approksimationen blev dårligt konditioneret, med store gradienter i slutpunktet til følge. Ligeledes galdt det her, at weight decayet virkede meget restriktivt for de værdier af α, hvor konditioneringsproblemerne var løst.

Disse forhold kan skyldes, at der for de hidtil betragtede forsøgsrækker er benyttet samme weight decay parameter α for *alle* vægte, i overensstemmelse med definitionen (1.1). Det optimale er muligvis at have individuelt weight decay for hver enkelt vægt, som det foreslås i feks. [Rasmussen 93]. Her skal vægtene dog blot deles op i to grupper, der kan have hver sit weight decay. Den ene gruppe består af de vægte, der normalt også indgår i et feedforward netværk (eksterne inputs til outputenheden regnes dog også som hørende til denne gruppe), den anden gruppe består af feedback vægtene, der gør netværket rekursivt.



Figur 6.21: Fuldt forbundet feedback netværk med tre enheder. Tyk streg angiver feedforward forbindelser, tynd streg feedback forbindelser.

På figur 6.21 illustreres det, hvilke vægte, der hører til hvilken gruppe. Forbindelserne med tyk streg hører til gruppen af feedforward vægte. Denne gruppe inkluderer også bias til de skjulte enheder, der implementeres som en implicit del af inputtet. De *rekursive* feedback vægte er tegnet med tynd streg. Pilen i outputtet er ikke en forbindelse med tilknyttet vægt, og tilhører således ingen af de to grupper.

Hvis der indføres separat weight decay parameter for de to grupper, bliver den udvidede cost funktion analogt til (1.1) nu

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\alpha_{ff}\sum_{i\in F} w_i^2 + \frac{1}{2}\alpha_{rec}\sum_{i\in R} w_i^2 \tag{6.10}$$

hvor $E(\mathbf{w})$ angiver den kvadratiske cost funktion, F angiver indeks for vægtene tilhørende feedforward gruppen, der har weight decay α_{ff}, og R angiver indeks for vægtene tilhørende *den rekursive* gruppe, der har weight decay α_{rec}.

Ovenstående opdeling af vægtene giver nu mulighed for at regulere *graden* af rekursion for et givent netværk, da et feedback netværk vil reduceres til et feedforward netværk, hvis $\alpha_{rec} \to \infty$!

6.7 Forsøg med opdeling af vægte

I det følgende beskrives en større forsøgsrække på solplet-serien, hvor der varieres på de to weight decays α_{ff} og α_{rec}. For at kunne sammenligne resultaterne med *kendte* feedforward netværk ([Weigend 90, Svarer 93]), benyttes der 12 eksterne inputs til de fire skjulte enheder. Outputenheden modtager som feedforward netværk ikke disse inputs, hvilket udfra forsøgene beskrevet i appendiks E.2 ikke har nogen signifikant betydning for netværkets opførsel. Yderligere benyttes der heller ikke lokalt feedback for outputenheden, i overensstemmelse med afsnit 6.2. De afprøvede netværk indeholder således ialt 77 vægte.

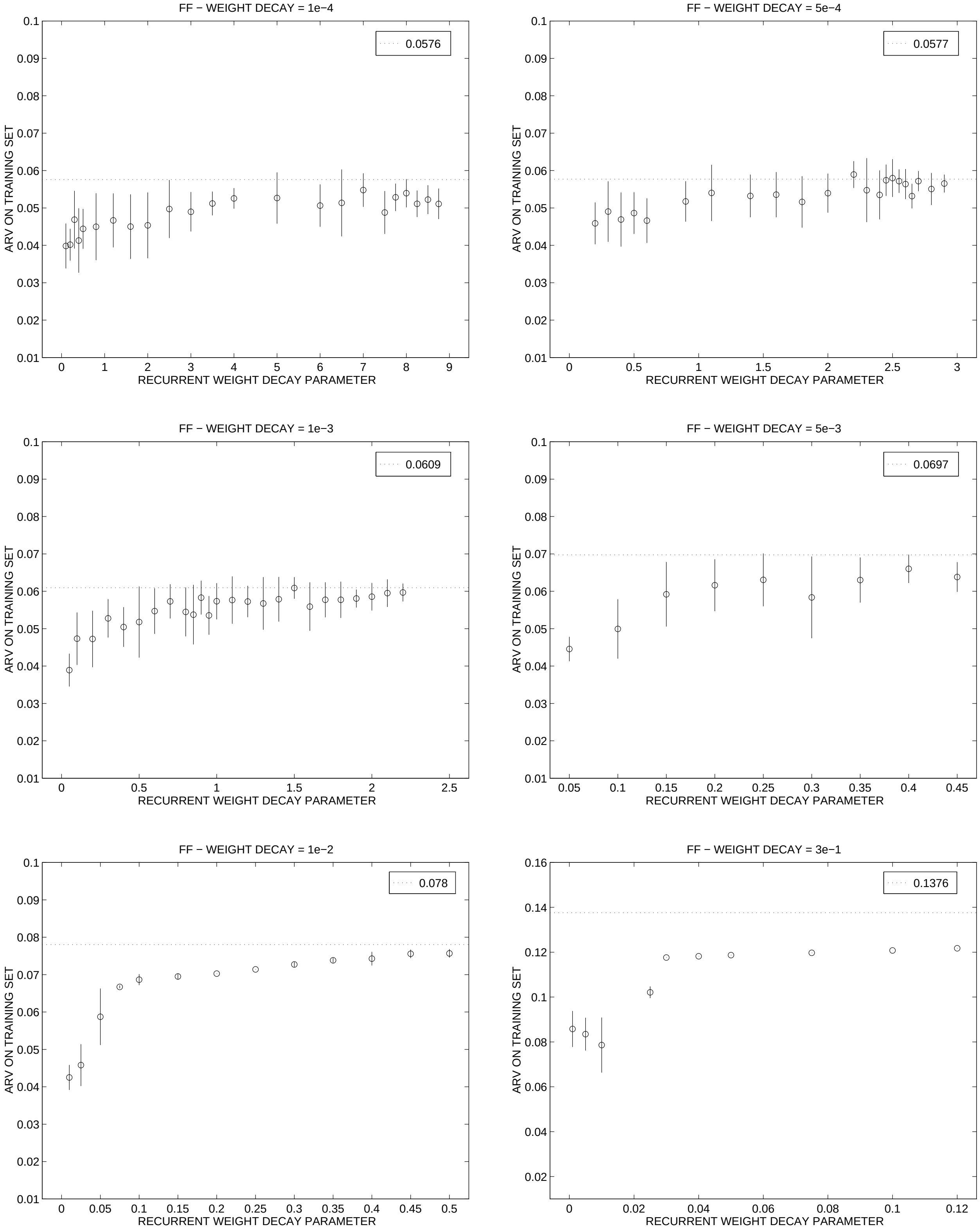
På figurerne 6.22 – 6.24 gengives resultaterne fra forsøgsrækken. For hver graf i figurerne gælder det, at α_{ff} holdes konstant på en angivet værdi, og α_{rec} varieres. For hver afprøvet kombination af de to weight decays er der optrænet 10 forskellige netværk. Optræningen indledtes med 50 iterationer gradient descent, hvorefter der fortsattes med maksimalt 550 iterationer med Gauss-Newtons metode; vægtene blev initialiseret med tilfældige værdier i intervallet $[-0.2\,;\,0.2]$. Intervallet er gjort lidt bredere end i afsnit 6.4 og afsnit 6.5 i forsøget på at skabe større forskellighed mellem de optrænede netværk for de store værdier af weight decayet. Tilsyneladende har udvidelsen dog ingen betydning.

Figur 6.22 indeholder grafer med middelværdien for træningsfejlen med angivelse af standardafvigelsen for de afprøvede kombinationer af α_{ff} samt α_{rec}. Figur 6.23 indeholder grafer med de tilsvarende middeltestfejl samt faktiske testfejl, og figur 6.24 indeholder middelværdi samt faktiske værdier for 2-normen af gradienten i slutpunkterne. Hver graf indeholder en horisontal stiplet linie. Denne linie angiver middelværdi for 10 optrænede 12–4–1 *feedforward* netværk. Linierne svarer således til de undersøgte netværks opførsel i middel i grænsen $\alpha_{rec} \to \infty$ for fastholdt α_{ff}, og giver samtidig mulighed for at sammenligne med tilsvarende feedforward netværk (der dog indeholder færre parametre, ialt 57 vægte).

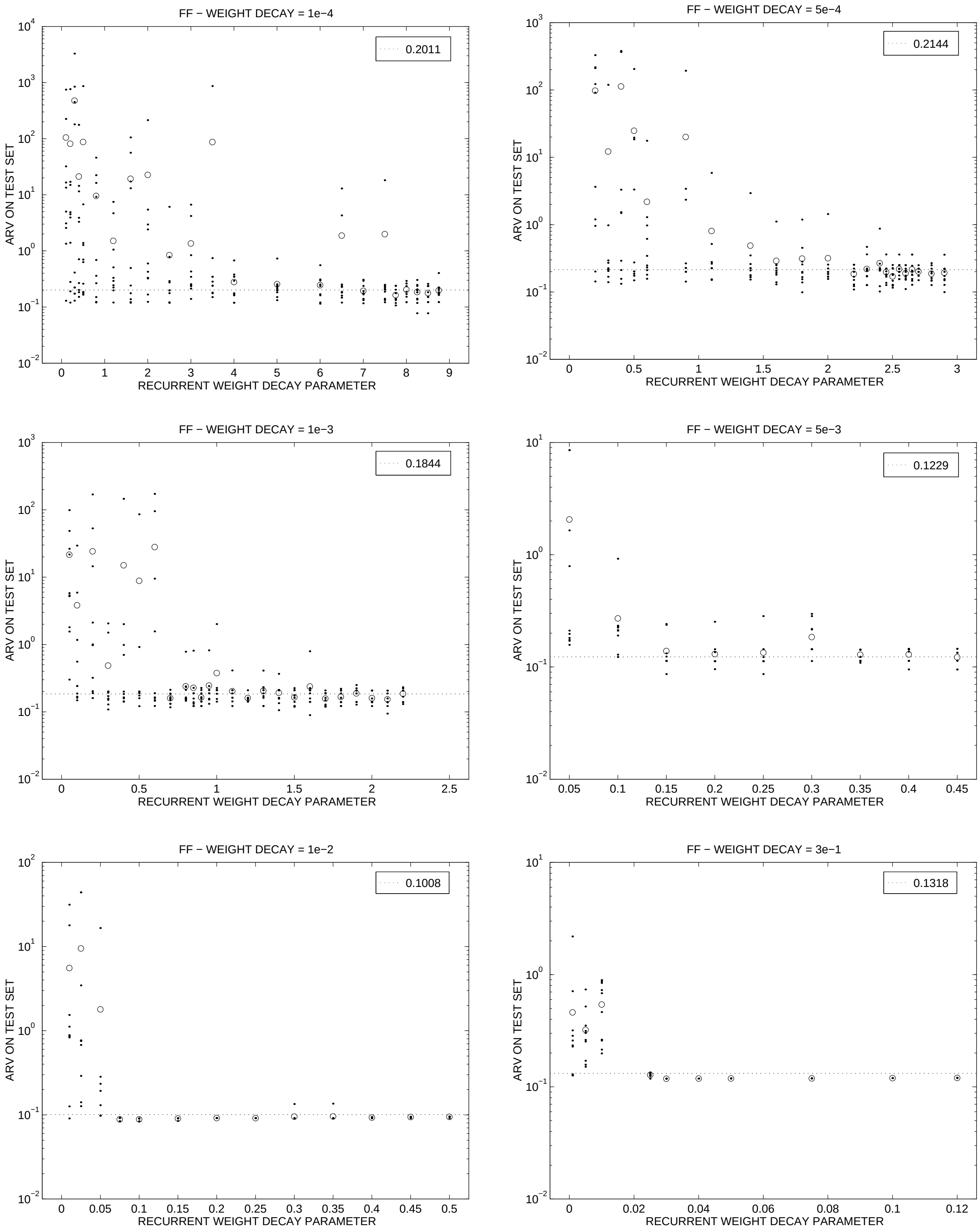
I figur 6.22 ses middeltræningsfejlene for lave værdier af α_{rec} at ligge et godt stykke under middelfejlen for feedforward netværket, for de laveste værdier omkring 30 % under. Efterhånden som α_{rec} øges, begrænses frihedsgraderne for de rekursive vægte; det *effektive* antal (se feks. [Svarer 93]) rekursive vægte reduceres. Det aktuelle netværk får større lighed med feedforward netværket, og middeltræningsfejlene bevæger sig op imod feedforward netværkets middelfejl. Det bemærkes, at feedforward netværkets middeltræningsfejl stiger støt for stigende værdier af α_{ff}, som det sædvanligvis ses (feks. i [Rasmussen 93]), svarende til at feedforward netværkets frihedsgrader begrænses mere og mere.

Standardafvigelsen fra middeltræningsfejlen ses at falde i takt med de stigende begrænsninger, som netværket underlægges. Dette kan tolkes i retning af, at antallet af (lokale) minima for den udvidede cost funktion (6.10) mindskes for stigende regularisering, svarende til den 'udglattende' effekt omtalt i afsnit 1.1.

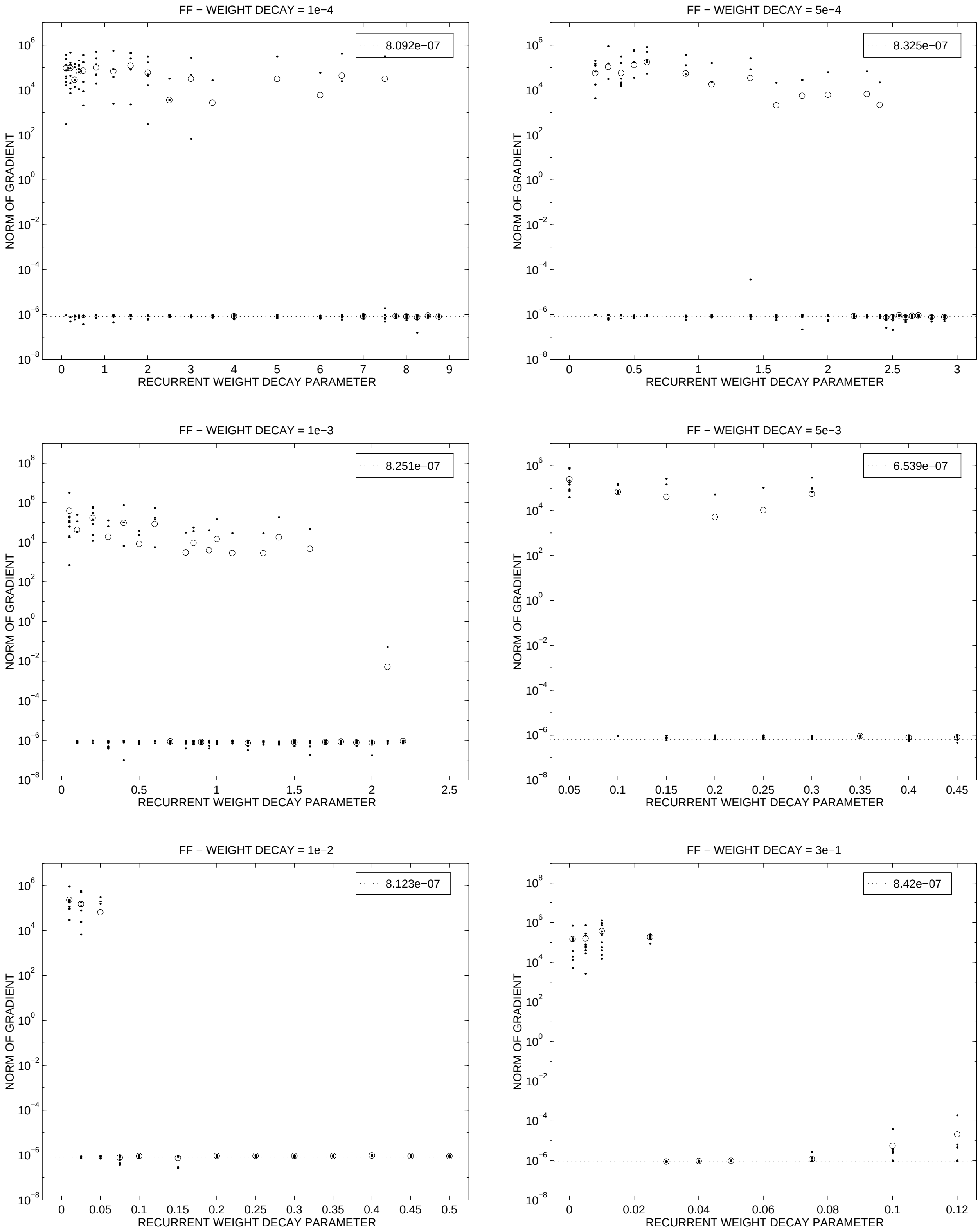
Testfejlene i figur 6.23 ses at variere mindre og mindre for stigende værdier af α_{rec}. Dette svarer til, at der overtrænes mindre og mindre, ligesom muligheden for ustabilitet for de skjulte enheder mindskes, jvf. figur 6.14.



Figur 6.22: Træningsfejl for et 12-4-1 feedback netværk. Den horisontale stiplede linie angiver middel for et tilsvarende feedforward netværk.



Figur 6.23: Testfejl for et 12-4-1 feedback netværk. Den horisontale stiplede linie angiver middel for et tilsvarende feedforward netværk. Bemærk den logaritmiske skala på ordinat-aksen.



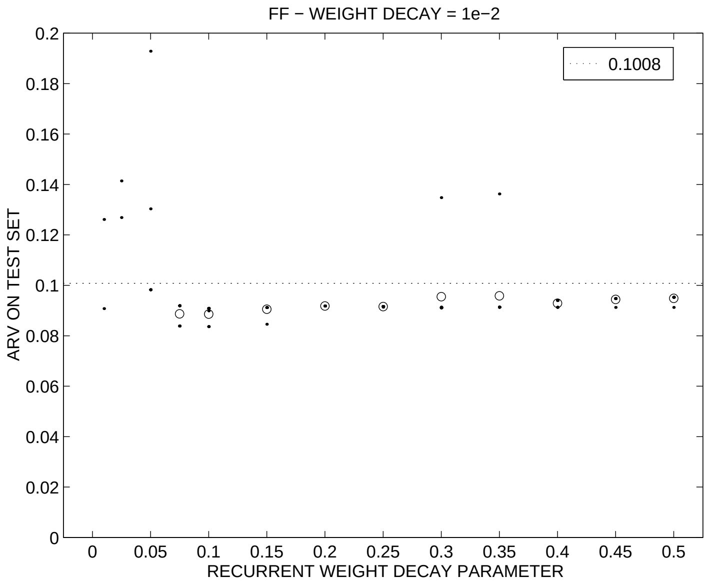
Figur 6.24: Gradientnorm for et 12-4-1 feedback netværk. Den horisontale stiplede linie angiver middel for et tilsvarende feedforward netværk.

For de større værdier af α_{rec} nærmer middeltestfejlene sig niveauet for feedforward netværket svarende til stigende begrænsning af frihedsgrader, som nævnt ovenfor. Feedforward netværkets middeltestfejl ses at falde for stigende værdier af α_{ff}, og har minimum $\langle ARV_{test} \rangle = 0.1008$ for $\alpha_{ff} = 10^{-2}$, hvorefter den atter begynder at stige. Det forholdsvis store weight decay $\alpha_{ff} = 0.3$ i sidste graf benyttes for at illustrere, at hvis weight decayet skrues tilstrækkeligt op, reducerer netværket reelt til en lineær model. For en lineær model med 12 inputs haves:

Lineær model:	
ARV_{train}	0.1356
ARV_{test}	0.1304

Den fundne 'optimale' værdi $\alpha_{ff} = 10^{-2}$ svarer omtrent til de benyttede værdier i [Svarer 93], hvor der benyttes $\alpha_{ff}^{in} = 0.02$ for inputvægtene og $\alpha_{ff}^{out} = 0.01$ for outputvægtene, ligesom testfejlene ligger på samme niveau; se iøvrigt kapitel 5.

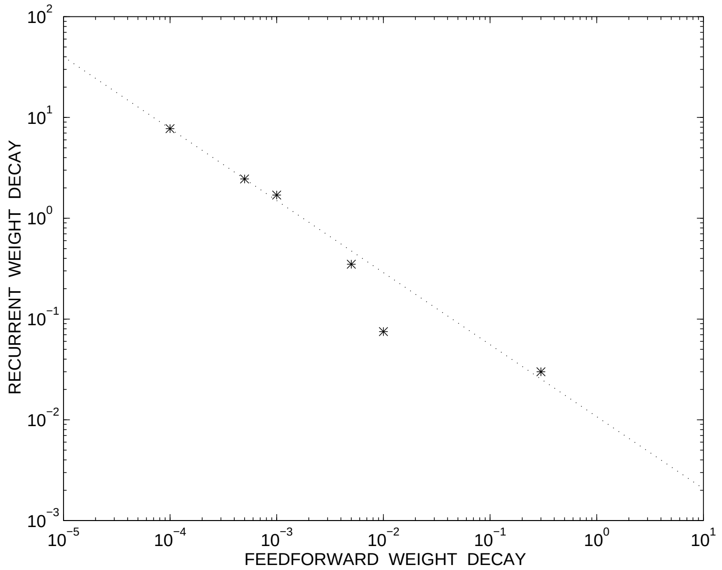
Det bemærkes, at især for den optimale værdi $\alpha_{ff} = 10^{-2}$ ligger middeltestfejlen for layered RTRL netværket for de større værdier af α_{rec} lidt *under* feedforward netværkets middelfejl. For at illustrere dette tydeligere, vises i figur 6.25 et udsnit af grafen med lineær skala på ordinat-aksen. På denne figur ses det tydeligt, hvordan middeltestfejlene dykker under niveauet for feedforward netværket, og langsomt nærmer sig dette, efterhånden som α_{rec} øges. Den mindste middeltestfejl opnås for $\alpha_{rec} = 0.1$, hvor $\langle ARV_{test} \rangle = 0.0886$.



Figur 6.25: Middeltestfejl samt datapunkter for optimalt feedforward α.

Denne værdi er ikke helt så lille som for de bedste netværk rapporteret i [Weigend 90, Svarer 93], men kan være et godt udgangspunkt for beskæringsalgoritmerne beskrevet i kapitel 4.

I figur 6.24 ses der at være meget stor forskel på, hvor stor α_{rec} skal være for en given værdi af α_{ff}, før alle optræninger forløber uden konditionerings-problemer for Hessian-approksimationen. Den 'sporadiske' sampling af α_{rec} samt α_{ff} skyldes netop søgen efter overgangene, hvor disse problemer elimineres; det var ofte temmelig uklart, for hvilke værdier af α_{rec} overgangen skulle findes for fastholdt α_{ff}. Overgangene er vigtige ved bestemmelse af weight decay parametre for netværk, der efter endt optræning skal udsættes for beskæring, da gradienterne for disse netværk skal være små i overensstemmelse med de antagelser, der gøres for beskæringsalgoritmerne i kapitel 4. For $\alpha_{ff} = 10^{-4}$ ses denne overgang først at fremkomme for $\alpha_{rec} \approx 7.75$, et *meget* stort weight decay. Alligevel ses middeltestfejlen for denne værdi at være en anelse bedre end for feedforward netværket, middeltræningsfejlen en hel del bedre. For de pågældende netværk er de rekursive vægte som forventet meget små, men har tilsyneladende alligevel betydning for netværkets 'ydeevne'.



Figur 6.26: Omtrentlige kombinationer af de to weight decays, for hvilke konditioneringsproblemet elimineres.

Efterhånden som α_{ff} øges, falder den værdi af α_{rec}, hvor overgangen indtræffer, drastisk. I figur 6.26 er de to værdier plottet mod hinanden, og de ligger overraskende nok tilnærmelsesvis på en ret linie i et koordinatsystem med logaritmisk skala på akserne; på grafen er indtegnet kurven beskrevet ved udtrykket $\alpha_{ff} \cdot \alpha_{rec}^{1.4} = 1.758 \cdot 10^{-3}$. For kombinationer af de to α-værdier, der ligger *over* denne linie, er konditioneringsproblemet elimineret.

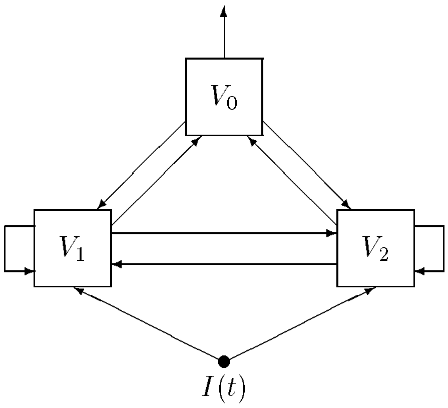
I figur 6.24 bemærkes det, at middel-gradientnormerne for feedforward netværkene *ikke* afslører konditioneringsproblemer, de ligger alle på niveau med stopkriteriet. Dette er dog ikke ensbetydende med, at problemerne udelukkende forekommer for feedback netværk, hvilket beskrives i afsnit 6.10; de kræver blot ikke så store weight decays for at blive elimineret.

For at vende tilbage til indledningen af dette afsnit, betrages *forskelligheden* af de netværk, der blev optrænet uden konditioneringsproblemer. For de lavere værdier af α_{ff} ses opsplitningen af weight decayet at have bevirket, at der nu er en større mangfoldighed af løsninger, der kan nås uden konditioneringsproblemer med deraf følgende store gradienter i slutpunktet. De fleste af disse netværk er dog ret dårlige, middeltestfejlene ligger på et højt niveau. For stigende værdier af α_{ff} falder middeltestfejlen markant, men samtidig mindskes mangfoldigheden af de optrænede netværk. Weight decayet for de bedste netværk er således stadig temmelig restriktivt, på trods af opdelingen af vægtene.

6.8 Lineære afhængigheder mellem vægtene

I dette afsnit skal outputtet fra enhederne i et lagdelt fuldt forbundet feedback netværk undersøges nærmere. I afsnit 2.2.2 blev outputtet fra hver enhed summarisk opskrevet som en funktion af de øvrige enheders output samt de eksterne inputs. I dette afsnit opskrives enhedernes output mere udførligt for et konkret netværk, dvs. med *indsættelse* af udtryk for de øvrige enheders output, for eventuelt at afsløre sammenhænge mellem vægtene, der er uhensigtsmæssige for netværkets optræning.

I figur 6.27 illustreres netværket, der her skal betragtes. For simpelhe-



Figur 6.27: Det betragtede netværk med inputs $I(t)$.

dens skyld repræsenterer $I(t)$ dels de eksterne inputs til tiden t, dels bias til de enkelte enheder. Det bemærkes, at netværket ikke har lokalt feedback i outputenheden, i overensstemmelse med beskrivelsen i afsnit 6.2. Outputenheden modtager som de testede netværk i afsnit 6.7 ikke det eksterne input, hvilket vil *lette overskueligheden*. I det følgende benyttes notationen w_{ij}, der angiver vægten til enhed i fra enhed/input j.

Det her betragtede netværk antages at have lineær outputenhed. Outputtet fra netværket i figur 6.27 til tiden t bliver således

$$V_0(t) = w_{01}V_1(t) + w_{02}V_2(t) \tag{6.11}$$

I dette udtryk indgår outputtet fra de to skjulte enheder til tiden t. For den første skjulte enhed haves:

$$V_1(t) \;=\; f_1\,[\,w_{11}V_1(t-1) + w_{10}V_0(t-1) + w_{12}V_2(t-1) + w_{1I}I(t)\,]$$

hvori indsættes outputtet $V_0(t)$ fra outputenheden til tiden $t-1$:

$$\begin{aligned} V_1(t) &= f_1\,[\,w_{11}V_1(t-1) + w_{10}\,(\,w_{01}V_1(t-1) + w_{02}V_2(t-1)\,) \\ &\qquad + w_{12}V_2(t-1) + w_{1I}I(t)\,] \\ &= f_1\,[\,(w_{11} + w_{10}\,w_{01})\,V_1(t-1) + (w_{12} + w_{10}\,w_{02})\,V_2(t-1) \\ &\qquad + w_{1I}I(t)\,] \end{aligned} \tag{6.12}$$

Tilsvarende kan outputtet fra den anden skjulte enhed opskrives. I udtrykket (6.12) for $V_1(t)$ ses det, at inputtet fra $V_1(t-1)$ ialt vægtes med $(w_{11} + w_{10}\,w_{01})$ og at inputtet fra $V_2(t-1)$ vægtes med ialt $(w_{12} + w_{10}\,w_{02})$. $V_1(t)$ kan nu skrives som

$$V_1(t) \;=\; f_1\,[\,k_1V_1(t-1) + k_2V_2(t-1) + w_{1I}I(t)\,]$$

hvor k_1 samt k_2 for et givet netværk er konstante,

$$\begin{aligned} k_1 &= w_{11} + w_{10}\,w_{01} \\ k_2 &= w_{12} + w_{10}\,w_{02} \end{aligned} \tag{6.13}$$

Vægtene w_{01} og w_{02} er fastlagt af outputtet, og kan ikke ændres uden at netværkets output også ændres. Derimod kan w_{10} vælges til en arbitrær værdi, hvorefter w_{11} samt w_{12} kan justeres således, at k_1 og k_2 i (6.13) stadig er konstante, og netværkets output uændret. Hvis feks. w_{10} elimineres i ovenstående udtryk, fås følgende lineære sammenhæng (de øvrige elementer er for et givet netværk konstante) mellem w_{11} og w_{12}:

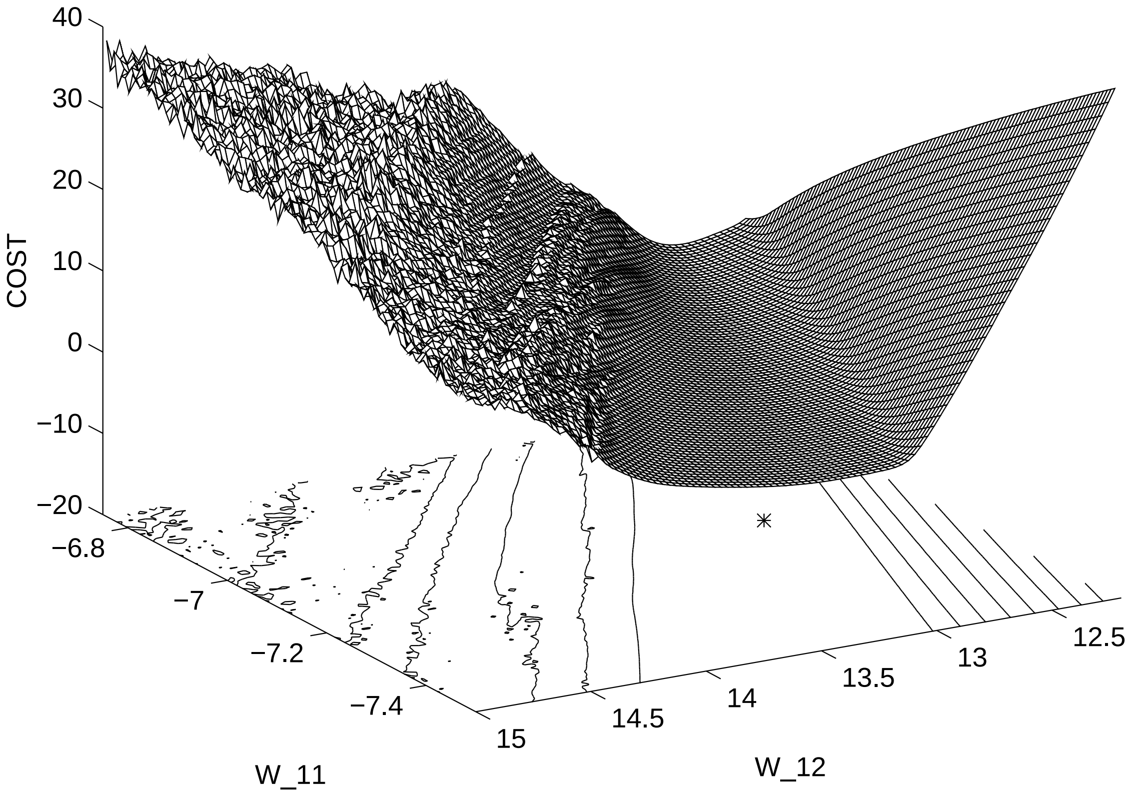
$$w_{11} = w_{12} \cdot \frac{w_{01}}{w_{02}} + k_1 - \frac{k_2 \cdot w_{01}}{w_{02}} \tag{6.14}$$

For værdier af w_{11} og w_{12} på denne linie vil netværkets output være uændret, hvis

$$w_{10} = \frac{k_2 - w_{12}}{w_{02}} \tag{6.15}$$

Dette er særdeles uheldigt, da der til ethvert tidspunkt under optræningen af netværket så vil være en retning i vægtrummet, i hvilken den kvadratiske cost funktion er konstant. Dette vil give problemer, hvis der anvendes Gauss-Newtons eller Newtons metode til optræningen. Hessianmatricen med de anden afledede vil være singulær, da krumningen i den konstante retning vil være nul; Newtonskridtet er derfor ubestemt. For Gauss-Newtons metode vil gælde, at Hessian-approksimationen vil være tæt på at være singulær, med dårlig konditionering og heraf dårligt bestemt søgeretning til følge. Yderligere vil Gauss-Newtons metode resultere i store 'skridt', der dog begrænses af liniesøgningen, hvilket forklarer de små værdier af η, der følger med den dårlige konditionering. Retningen af disse skridt vil (omtrent) være i den konstante retning.

I figur 6.28 illustreres det, hvordan en sådan retning ser ud som funktion af to vægte. Plottet stammer fra en cost funktion uden weight decay for et netværk med to skjulte enheder samt et enkelt eksternt input, som outputenheden dog også modtager. Netværket er indledningsvis optrænet med både gradient descent samt pseudo Gauss-Newton, og plottet viser et udsnit af cost funktionen som funktion af vægtene w_{11} samt w_{12} efter at der er foretaget 150 iterationer med Gauss-Newtons metode. Cost funktionens værdi er her konstant, men vægtene undergår store ændringer gennem iterationerne svarende til, at iterationspunktet bevæger sig i en retning, hvor cost funktionen er konstant, som beskrevet ovenfor. I appendiks F er det her omtalte forsøg beskrevet mere detaljeret, med angivelse af vægtene samt illustration



Figur 6.28: Cost funktion *uden* weight decay efter 100 iterationer gradient descent, 25 iterationer pseudo Gauss-Newton samt 150 iterationer Gauss-Newton.

af cost funktionen på forskellige stadier af optræningsforløbet. Efterfølgende gentages forsøget med weight decay, og den positive effekt af dette illustreres. Weight decayet medfører, at der ikke længere er nogen retninger, i hvilke cost funktionen er konstant, da cost funktionen nu vil vokse for større vægte. I retninger, hvor cost funktionen uden weight decay var konstant, vil den nu beskrive en parabel.

Et væsentligt punkt at bemærke omkring den lineære afhængighed vægtene imellem er, at gradient descent samt pseudo Gauss-Newton *ikke* lider under de samme problemer med retningerne, hvor cost funktionen er konstant, som Newtons samt Gauss-Newtons metode gør. For de to førstnævnte metoder bestemmes søgeretningen separat for hver vægt, de 'opdager' således ikke, at der er afhængigheder imellem vægtene. For de to sidstnævnte andenordens metoder vil søgeretningen derimod søges bestemt ved at løse et ligningssystem, der er (tæt på at være) singulært.

Ovenstående betragtninger omkring lineære afhængigheder mellem vægtene og deraf følgende retninger, i hvilke cost funktionen er konstant, er foretaget på baggrund af en enkelt skjult enhed. For hver af de øvrige enheder vil der naturligvis gælde tilsvarende, således at antallet af retninger øges med antallet af skjulte enheder.

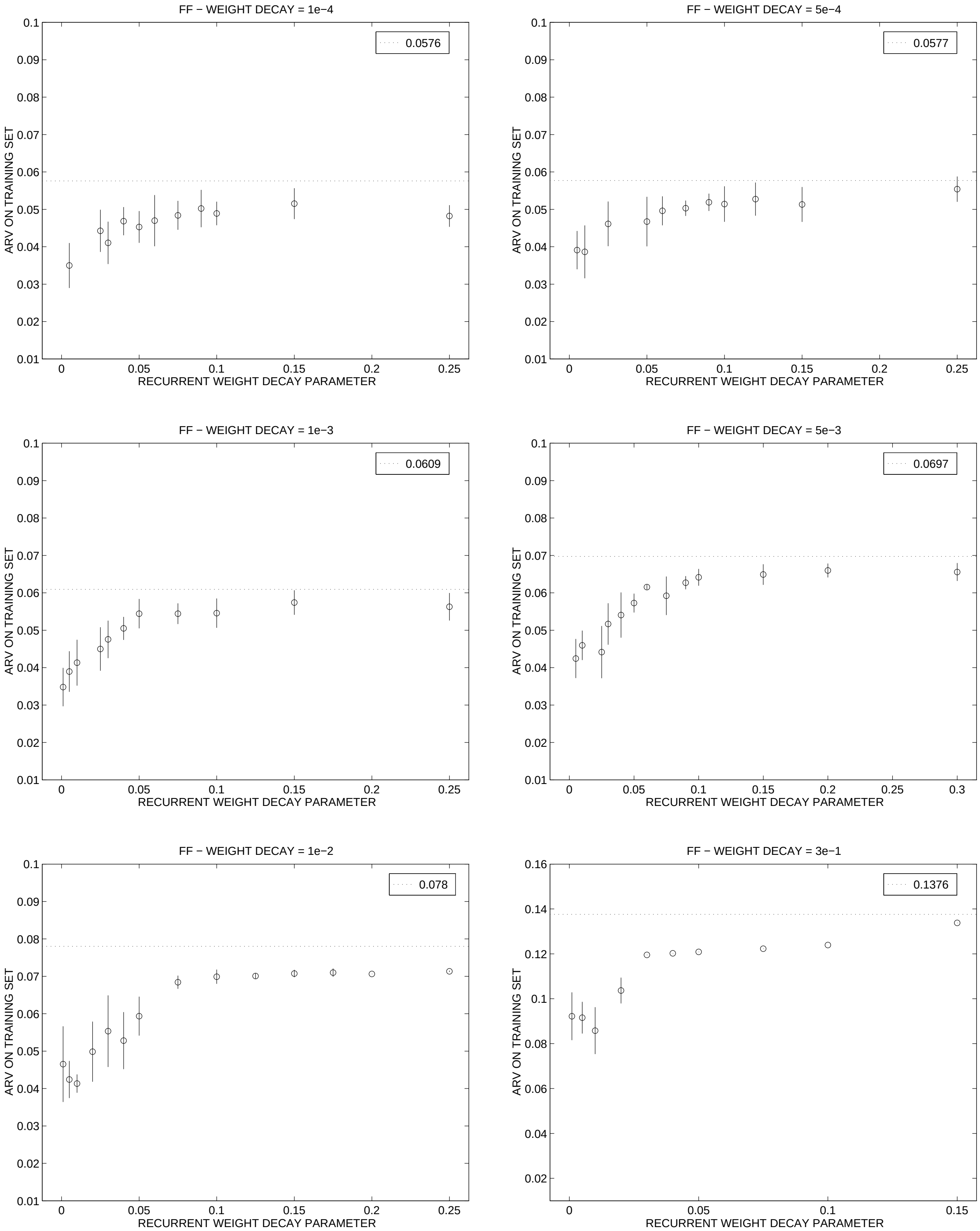
I udtrykkene (6.13) bemærkes det, at vægten w_{10} fra outputenheden til den skjulte enhed er parameteren, der *knytter de øvrige sammen*. Hvis vægtene fra outputenheden til de skjulte enheder fjernes, ses de lineære afhængigheder imellem vægtene således at blive elimineret; det skal derfor her anbefales, at der ikke benyttes feedback fra outputenheden til de skjulte enheder, når outputtet er lineært. Nedenfor beskrives et større forsøg, der illustrerer, hvilken effekt det har på størrelsen af det krævede weight decay, at eliminere disse feedback vægte fra outputenheden til de skjulte enheder.

6.9 Opdeling uden lineær afhængighed

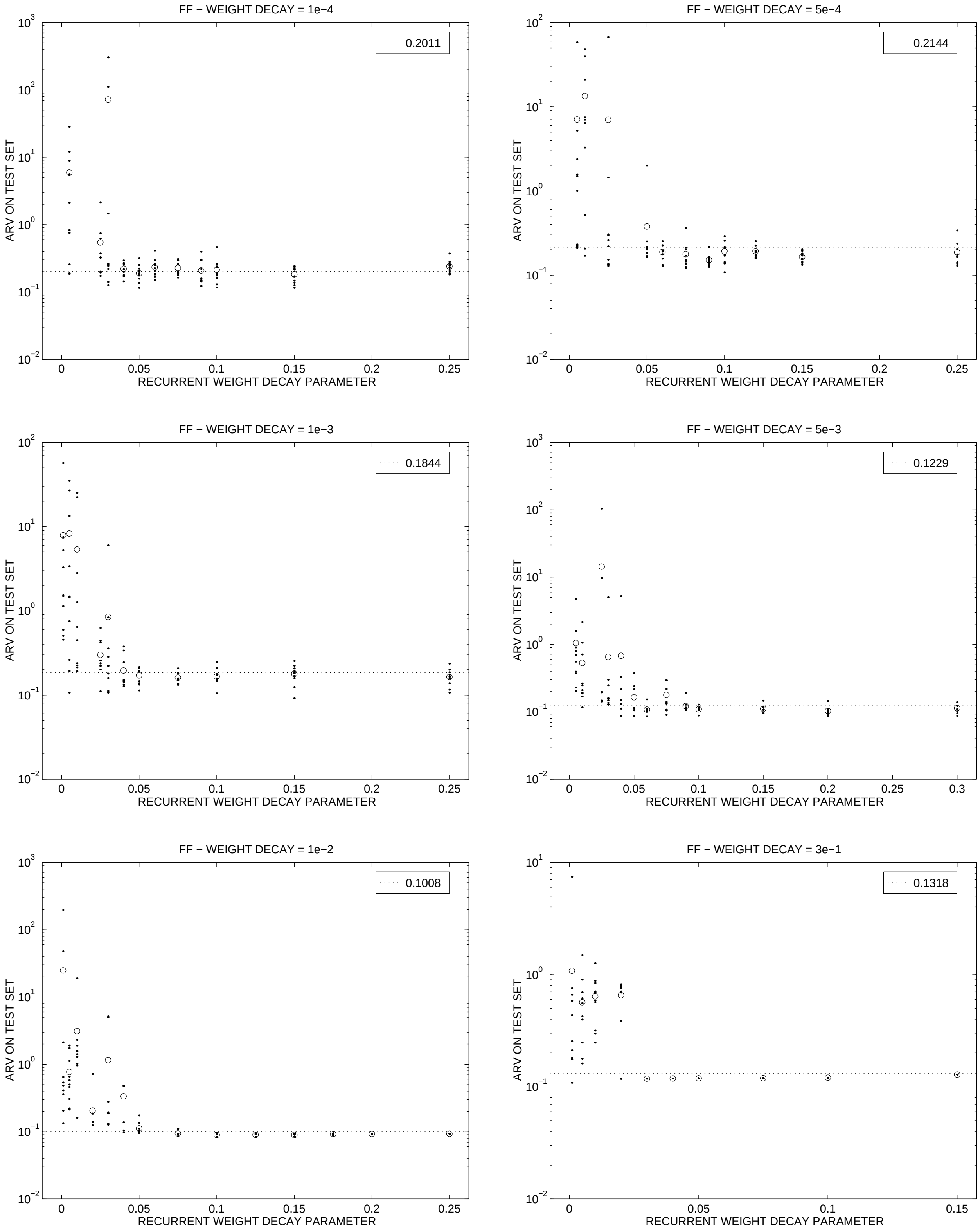
I afsnit 6.7 blev der beskrevet et større forsøg, hvor vægtene blev opdelt i to grupper med hvert sit weight decay, hhv. α_{ff} samt α_{rec}. I det følgende beskrives et lignende forsøg med stort set samme netværk, som blev benyttet i afsnit 6.7, blot er feedback vægtene fra outputenheden til de skjulte enheder nu elimineret, som det anbefales i afsnit 6.8. Det afprøvede netværk har således 12 eksterne inputs til de fire skjulte enheder; outputenheden modtager ikke de eksterne inputs og har ikke lokalt feedback, ligesom der ikke er forbindelse *fra* outputenheden *til* de skjulte enheder. De afprøvede netværk indeholder 73 vægte, fire mindre end de i afsnit 6.7 betragtede.

Som tidligere indledes optræningen med 50 iterationer gradient descent efterfulgt af maksimalt 550 iterationer med Gauss-Newtons metode. Vægtene initialiseres med tilfældige værdier i intervallet $[-0.2\,;\,0.2]$, som det også blev gjort i afsnit 6.7.

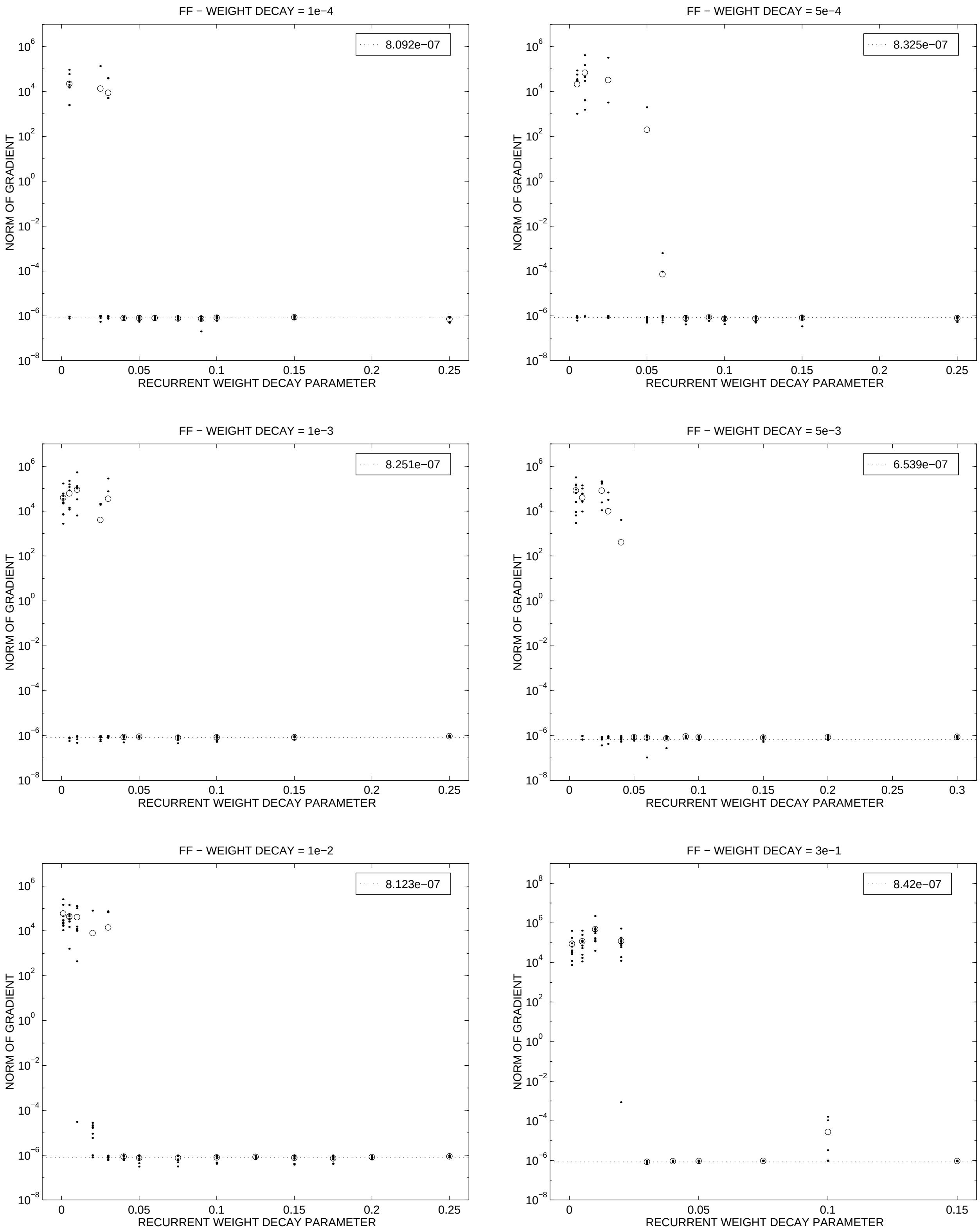
På figurene 6.29–6.31 gengives resultaterne fra forsøgsrækken. Det før-



Figur 6.29: Træningsfejl for et 12-4-1 feedback netværk *uden* feedback fra outputenheden til de skjulte enheder. Den horisontale stiplede linie angiver middel for et tilsvarende feedforward netværk.



Figur 6.30: Testfejl for et 12-4-1 feedback netværk *uden* feedback fra outputenheden til de skjulte enheder. Den horisontale stiplede linie angiver middel for et tilsvarende feedforward netværk. Bemærk den logaritmiske skala på ordinat-aksen.



Figur 6.31: Gradientnorm for et 12-4-1 feedback netværk *uden* feedback fra outputenheden til de skjulte enheder. Den horisontale stiplede linie angiver middel for et tilsvarende feedforward netværk.

ste, der skal bemærkes omkring disse, er niveauet for de anvendte weight decays α_{rec} for de rekursive vægte, for fastholdt α_{ff}. Niveauerne er langt lavere end de tilsvarende i figur 6.22–6.24. Årsagen til dette findes i figur 6.31. Som i afsnit 6.7 er det for hver fastholdt α_{ff} søgt at lokalisere de overgange for α_{rec}, hvor optræningen forløber uden konditioneringsproblemer for Hessian-approksimationen, illustreret ved de store gradientnormer i slutpunktet. Hvor det krævede α_{rec} i afsnit 6.7 var meget svingende, med værdier helt op til $\alpha_{rec} = 7.75$, ses overgangene nu for *alle* de afprøvede værdier af α_{ff} at ligge omkring $\alpha_{rec} = 0.04 - 0.05$.

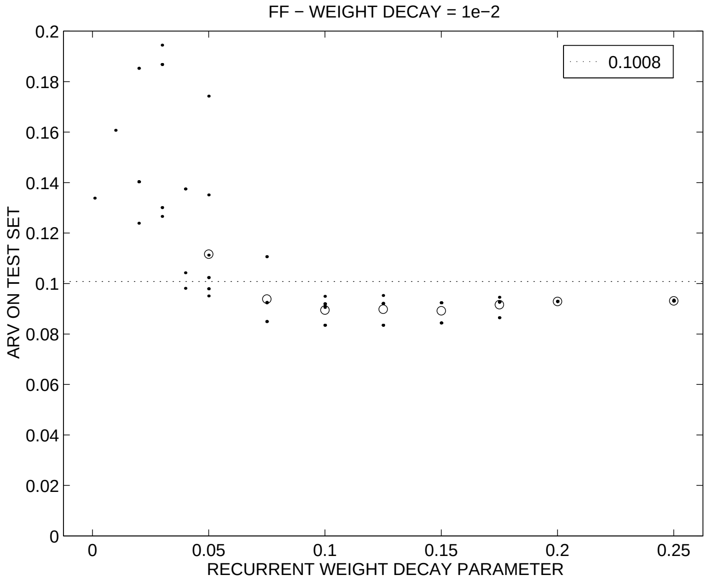
Som beskrevet ovenfor, er den eneste ændring i forhold til de afprøvede netværk i afsnit 6.7, at de fire vægte fra outputenheden til de skjulte enheder nu er elimineret. Eftersom de fire vægte kun udgør omkring 5 % af det samlede antal parametre, kan ændringerne af overgangenes placering ikke tilskrives reduktionen i antallet af vægte. De store reduktioner i de nødvendige α_{ff} for problemfri optræning må derfor skyldes, at der ikke længere er lineær afhængighed imellem nogle af vægtene, som det blev beskrevet i afsnit 6.8, med deraf følgende singularitetsproblemer.

Træningsfejlene i figur 6.29 ses at opføre sig nogenlunde som på figur 6.22 i afsnit 6.7. For små værdier af α_{rec} ligger middeltræningsfejlen væsentligt under middelfejlen for det tilsvarende feedforward netværk, men nærmer sig dette niveau for stigende α_{rec}, som det beskrives i afsnit 6.7. Ligeledes ses standardafvigelsen at aftage for stigende værdier af $(\alpha_{ff}, \alpha_{rec})$.

De opnåede middeltestfejl i figur 6.30 opnår generelt samme niveau som de tilsvarende på figur 6.23 i afsnit 6.7. Dog er det på figur 6.30 for alle de afprøvede α_{ff} nu tydeligere, at middeltestfejlen for stigende α_{rec} først bliver *mindre* end middeltestfejlen for det tilsvarende feedforward netværk, for derefter at konvergere mod denne. Som i afsnit 6.7 opnås den bedste generaliseringsevne for $\alpha_{ff} = 10^{-2}$. På figur 6.32 gengives middeltestfejlene for denne værdi med lineær skala på ordinat-aksen. Den mindste middelfejl opnås for $\alpha_{rec} = 0.15$, hvor $\langle ARV_{test} \rangle = 0.0892$. Denne mindste værdi opnås for en *større* værdi af α_{rec}, end det var tilfældet i afsnit 6.7, hvilket der dog ikke skal lægges for meget i, da det på figur 6.32 ses, at middeltestfejlene i intervallet $\alpha_{rec} \in [0.1\,;0.15]$ stort set er ens.

Elimineringen af vægtene fra outputenheden til de skjulte enheder vil være en fordel ved efterfølgende beskæring af netværket. Hvis disse vægte ikke elimineres, vil de store α_{rec}, der i så fald må anvendes for at sikre problemfri indledende optræning til et (lokalt) minimum for cost funktionen, virke stærkt hæmmende på netværket, hvis de omtalte vægte bliver fjernet under beskæringen. Dette problem ses dog ved sammenligning af figur 6.24 med figur 6.31 kun at være et reelt problem for de afprøvede værdier af α_{ff}, der er mindre end 10^{-2}, da de nødvendige værdier af α_{rec} for denne værdi af α_{ff} stort set er ens på de to figurer.

Selvom eliminering af den lineære afhængighed vægtene imellem har reduceret konditionsproblemet væsentligt, er det dog stadig i høj grad tilstede.



Figur 6.32: Middeltestfejl samt datapunkter for optimalt feedforward α, *uden* feedback fra outputenheden til de skjulte enheder.

I næste afsnit beskrives det, hvad der kan forårsage den resterende del af problemet.

6.10 Analyse af Jacobi-matrix

I dette afsnit foretages en analyse af, hvad der kan bevirke, at den approksimation til Hessianmatricen, der anvendes i Gauss-Newtons metode, bliver dårligt konditioneret.

Fra kapitel 1 erindres det, at den kvadratiske cost funktion

$$E(\mathbf{w}) = \frac{1}{2}\sum_{t=1}^{T}[x(t) - V^o(t)]^2 = \frac{1}{2}\sum_{t=1}^{T}[e(t)]^2 \tag{6.16}$$

har de anden afledede

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} = -\sum_{t=1}^{T}\left[e(t)\frac{\partial^2 V^o(t)}{\partial w_i \partial w_j} - \frac{\partial V^o(t)}{\partial w_i}\cdot\frac{\partial V^o(t)}{\partial w_j}\right] = A_{ij} \tag{6.17}$$

der samles i Hessianmatricen $\mathbf{A}$. Denne matrix kan skrives som

$$\mathbf{A} = \mathbf{J}^{\mathrm{T}}\mathbf{J} + \mathbf{B} \tag{6.18}$$

hvor $\mathbf{J}$ kaldes *Jacobi matricen*. Hvis der er T træningseksempler og n_w vægte i netværket, har $\mathbf{J}$ dimension $T \times n_w$, og

$$J_{tj} = \frac{\partial V^o(t)}{\partial w_j} \tag{6.19}$$

Hver række i $\mathbf{J}$ indeholder således de partielle afledede af netværkets outputfunktion for det pågældende eksempel til tiden t. $\mathbf{B}$ indeholder de anden afledede af netværkets output, med

$$B_{ij} = -\sum_{t=1}^{T} e(t) \frac{\partial^2 V^o(t)}{\partial w_i \partial w_j} \tag{6.20}$$

Gauss-Newton metoden ser bort fra $\mathbf{B}$, og benytter approksimationen

$$\mathbf{A} \approx \mathbf{J}^{\mathrm{T}}\mathbf{J} \tag{6.21}$$

Som enhver anden matrix kan Jacobimatricen opskrives som *Singular Value Dekompositionen* (SVD)

$$\mathbf{J} = \mathbf{U}\,\mathbf{D}\,\mathbf{V}^{\mathrm{T}} \tag{6.22}$$

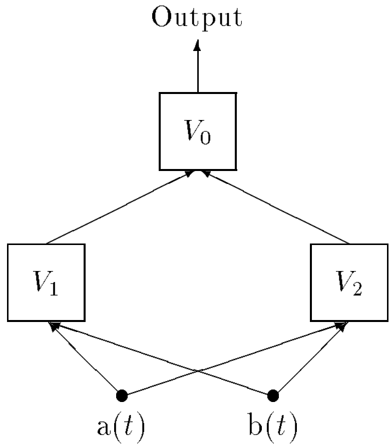
hvor $\mathbf{U}$ med dimension $T \times n_w$ samt $\mathbf{V}$ med dimension $n_w \times n_w$ er ortogonale matricer, og $\mathbf{D}$ er en $n_w \times n_w$ diagonalmatrix med elementer $D_{ii} \geq 0$, der kaldes de singulære værdier for $\mathbf{J}$. SVD er beskrevet indgående i feks. [Press 92], og vil her ikke blive beskrevet yderligere. Det skal blot nævnes, at de singulære værdier er *kvadratroden* af den numeriske værdi af egenværdierne for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$, og at antallet af singulære værdier, der er forskellige fra nul, angiver *rangen* af $\mathbf{J}$. Ved rangen af $\mathbf{J}$ forstås antallet af søjler, der er *lineært uafhængige*.

Hvis nogle af søjlerne i $\mathbf{J}$ er *meget tæt* på at være lineært afhængige, vil nogle af de singulære værdier i stedet for at være nul blive meget små, hvilket medfører, at nogle af egenværdierne for $\mathbf{J}^T\mathbf{J}$ bliver meget små, hvilket igen ifølge beskrivelsen i appendiks D betyder, at Hessian-approksimationen $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ bliver dårligt konditioneret.

For neurale netværk vil det ofte forekomme, at nogle af søjlerne i $\mathbf{J}$ er meget tæt på at være lineært afhængige ([Saarinen 93]). I denne artikel etableres der ved hjælp af diverse sætninger omkring nedre samt øvre grænser for konditionstal for matricer en sammenhæng mellem små vinkler imellem nogle af søjlerne i $\mathbf{J}$ og et stort konditionstal for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$.

Efter at have etableret denne sammenhæng, opskrives de partielle afledede for et feedforward netværk med to lag skjulte enheder svarende til søjlerne i $\mathbf{J}$, og det analyseres, under hvilke betingelser disse afledede vil være tæt på lineær afhængighed. Det viser sig, at den latente lineære afhængighed søjlerne imellem er stærkt knyttet til de benyttede sigmoid-funktioner for de skjulte enheder. Det illustreres, hvorledes de indgående kombinationer af sigmoid-funktioner samt deres afledede i søjlerne i $\mathbf{J}$ stort set er konstante for selv moderate numeriske størrelser af inputtet til de skjulte enheder, hvilket fører til lineær afhængighed. Udledningerne følges op af et større antal eksempler for forskellige størrelser netværk, der illustrerer de opstående lineære afhængigheder mellem søjlerne i $\mathbf{J}$, og de deraf følgende store konditionstal for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$.

For at illustrere ræsonnementet i [Saarinen 93], opskrives de partielle afledede for et simpelt feedforward netværk med to inputs a og b samt to skjulte



Figur 6.33: Simpelt feedforward netværk.

enheder, som afbildet på figur 6.33. I det følgende vil w_{ij} angive vægten til enhed i fra input/enhed j. Outputenheden vil have lineær aktiveringsfunktion, de skjulte enheder vil have sigmoid-aktiveringsfunktionerne $\sigma_1(t)$ og $\sigma_2(t)$, med de afledede hhv. $\sigma_1'(t)$ og $\sigma_2'(t)$. Outputtet fra netværket til tiden t kan nu opskrives som

$$\begin{aligned} V_0(t) &= w_{01}\,\sigma_1\left[s_1(t)\right] + w_{02}\,\sigma_2\left[s_2(t)\right] + w_{0,bias} \\ &= w_{01}\,\sigma_1\left[w_{1a}\,\mathrm{a}(t) + w_{1b}\,\mathrm{b}(t) + w_{1,bias}\right] \\ &\quad + w_{02}\,\sigma_2\left[w_{2a}\,\mathrm{a}(t) + w_{2b}\,\mathrm{b}(t) + w_{2,bias}\right] + w_{0,bias} \end{aligned} \tag{6.23}$$

De partielle afledede bestemmes nu til følgende:

$$\frac{\partial V_0(t)}{\partial w_{01}} = \sigma_1\left[w_{1a}\,\mathrm{a}(t) + w_{1b}\,\mathrm{b}(t) + w_{1,bias}\right] \tag{6.24}$$

$$\frac{\partial V_0(t)}{\partial w_{02}} = \sigma_2\left[w_{2a}\,\mathrm{a}(t) + w_{2b}\,\mathrm{b}(t) + w_{2,bias}\right] \tag{6.25}$$

$$\frac{\partial V_0(t)}{\partial w_{0,bias}} = 1 \tag{6.26}$$

$$\frac{\partial V_0(t)}{\partial w_{1a}} = w_{01}\,\sigma_1'(t)\,\mathrm{a}(t) \tag{6.27}$$

$$\frac{\partial V_0(t)}{\partial w_{1b}} = w_{01}\,\sigma_1'(t)\,\mathrm{b}(t) \tag{6.28}$$

$$\frac{\partial V_0(t)}{\partial w_{1,bias}} = w_{01}\,\sigma_1(t) \tag{6.29}$$

$$\frac{\partial V_0(t)}{\partial w_{2a}} = w_{02}\,\sigma_2'(t)\,\mathrm{a}(t) \tag{6.30}$$

$$\frac{\partial V_0(t)}{\partial w_{2b}} = w_{02}\,\sigma_2'(t)\,\mathrm{b}(t) \tag{6.31}$$

$$\frac{\partial V_0(t)}{\partial w_{2,bias}} = w_{02}\,\sigma_2'(t) \tag{6.32}$$

Hvis der er lineær afhængighed mellem de eksterne inputs $\mathrm{a}(t)$ og $\mathrm{b}(t)$, vil der være lineær afhængighed mellem søjlerne i $\mathbf{J}$ bestemt ved (6.27) og (6.28), samt mellem søjlerne bestemt ved (6.30) og (6.31). Hvis dette ikke er tilfældet, kan der dog alligevel opstå situationer, hvor to eller flere rækker i $\mathbf{J}$ er tæt på at være lineært afhængige. Til at beskrive disse, benyttes notationerne $\underline{\sigma_1}$, $\underline{\sigma_1}'$, $\underline{\sigma_2}$ samt $\underline{\sigma_2}'$, der angiver vektorer indeholdende de pågældende funktionsværdier for $t = 1 \ldots T$.

I [Saarinen 93] anføres nogle muligheder for, at søjler i $\mathbf{J}$ vil være tæt på at være lineært afhængige. I det følgende relateres disse muligheder til de partielle afledede (6.24)–(6.32) for netværket på figur 6.33:

1. Hvis $\underline{\sigma_1}$ eller $\underline{\sigma_2}$ er omtrent proportionale med $\mathbf{e} = (1, \ldots, 1)^{\mathrm{T}}$, vil (6.24) eller (6.25) være omtrent proportionale med (6.26).

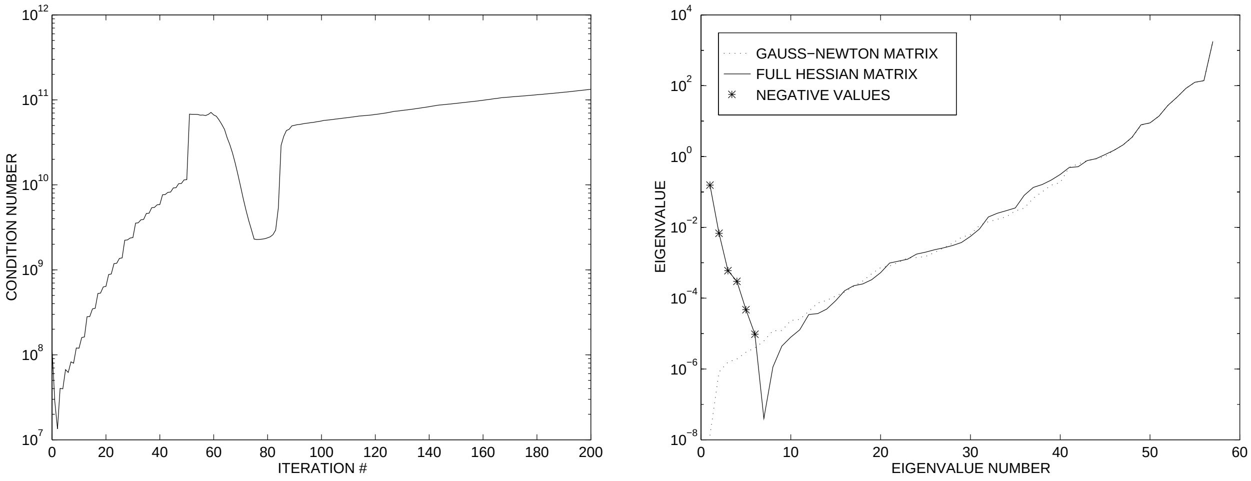
2. Hvis vinklen θ mellem $\underline{\sigma_1}'$ og $\underline{\sigma_2}'$ er lille, vil søjlerne i $\mathbf{J}$ bestemt ved (6.27) og (6.30), (6.28) og (6.31) samt (6.29) og (6.32) være tæt på at være lineært afhængige.

3. Hvis $\underline{\sigma_1}$ og $\underline{\sigma_2}$ er proportionale, vil (6.24) og (6.25) være proportionale.

For at illustrere ovenstående, optrænes et 12–4–1 feedforward netværk. I afsnit 6.7 samt 6.9 blev dette netværk benyttet som reference for tilsvarende rekursive layered RTRL-netværk. Med weight decay parametre ned til $\alpha = 10^{-4}$ var der ingen konditioneringsproblemer at spore for feedforward netværket, men i figur 6.34 vises konditionstallet for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ som funktion af iterationsantallet for $\alpha = 10^{-8}$. Netværket er trænet op med 50 iterationer gradient descent efterfulgt af 150 iterationer med Gauss-Newtons metode.

Som det er observeret for de rekursive netværk i de foregående afsnit, ses feedforward netværket på figur 6.34 ligeledes at lide under konditioneringsproblemer under optræningen. På den højre graf i figur 6.34 vises de numeriske værdier af egenværdierne sorteret efter størrelse for den fulde Hessian $\mathbf{A}$ samt $\mathbf{J}^{\mathrm{T}}\mathbf{J}$; da $\mathbf{A}$ ikke er positivt definit, er dennes negative egenværdier specielt markeret pga. den logaritmiske skala på ordinat-aksen. Denne illustration af egenværdierne har stor lighed med tilsvarende illustrationer i [Saarinen 93] for de der testede netværk.

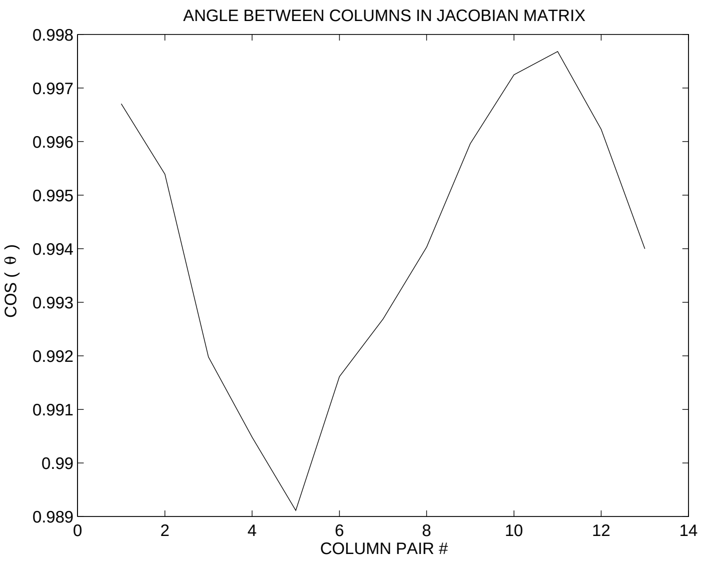
Ifølge beskrivelsen ovenfor af udledningerne i [Saarinen 93] skyldes den dårlige konditionering af Hessianen, at nogle af søjlerne i $\mathbf{J}$ er meget tæt på at være lineært afhængige. Hvis man for det optrænede netværk ovenfor betragter cosinus til vinklerne mellem søjlerne i Jacobi-matricen, vil det fremgå, at et antal af disse er meget tæt på een. De pågældende søjler svarer til situationen (2) af de ovenfor opremsede muligheder for omtrentlig lineær afhængighed svarende til, at $\underline{\sigma_1}'$ samt $\underline{\sigma_2}'$ er omtrent proportionale. I



Figur 6.34: Konditionstal for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ for et 12–4–1 feedforward netværk under optræningen samt spektret af egenværdier i slutpunktet.

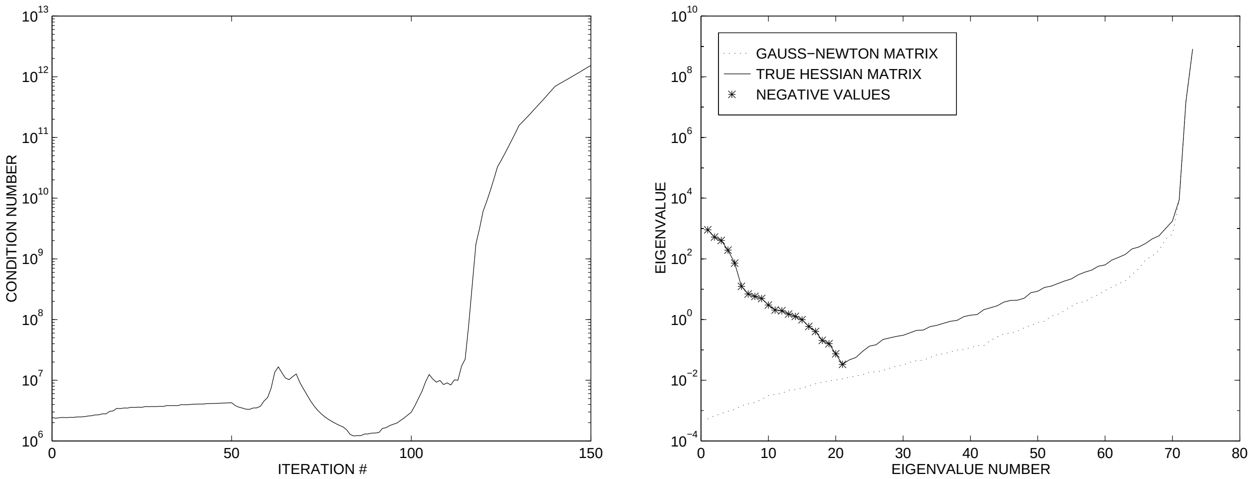
figur 6.35 vises cosinus til vinklen mellem de søjler i $\mathbf{J}$, der således vil være parvis tæt på lineært afhængige, i overensstemmelse med punkt (2) i opremsningen ovenfor. Da der for det her betragtede netværk er 12 eksterne inputs samt et bias, er ialt 13 søjler i $\mathbf{J}$ tæt på at være lineært afhængige.

For rekursive neurale netværk vil der kunne gøres betragtninger tilsvarende dem, der ovenfor er gjort for feedforward netværk. Dette vil man kunne overbevise sig om, hvis samtlige partielle afledede opskrives for et givent layered RTRL netværk, hvad der dog ikke vil blive gjort her. Afslutningsvis



Figur 6.35: Vinkler mellem søjler i $\mathbf{J}$ svarende til situation (2).

illustreres i figur 6.36 konditionstallet for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ som funktion af iterationsantallet samt egenværdi-spektret i slutpunktet for et 12–4–1 layered RTRL netværk magen til det i afsnit 6.9 testede. De benyttede weight decays er $\alpha_{ff} = 10^{-4}$ samt $\alpha_{rec} = 10^{-2}$.



Figur 6.36: Konditionstal for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ for et 12–4–1 layered RTRL netværk under optræningen samt spektret af egenværdier i slutpunktet.

Det bemærkes, at de mindste egenværdier for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ er større end det mindste anvendte weight decay, $\alpha_{ff} = 10^{-4}$. Til gengæld er de største egenværdier meget store, hvilket giver den dårlige konditionering. De store egenværdier skyldes den rekursive beregning af de partielle afledede. Disse beregnes for hvert eksempel som en vægtet sum af de for det foregående eksempel beregnede afledede, hvilket betyder, at elementerne i Jacobi-matricen og dermed i $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ kan blive meget store. Dette kan forklare, hvorfor det er nødvendigt med det store weight decay for de *rekursive* vægte for at undgå konditioneringsproblemer. Weight decayet begrænser disse vægtes numeriske størrelser, hvorfor den vægtede sum af tidligere beregnede partielle afledede uvægerligt vil blive mindre, hvilket man kan overbevise sig om ved at betragte udtrykkene for de partielle afledede for layered RTRL i afsnit 2.2.2. Her ses det, at en overvejende del af de partielle afledede for outputenheden består af en vægtet sum af de *skjulte* enheders partielle afledede; disse beregnes igen som en sum af tidligere afledede, vægtet netop med de rekursive vægte. Samtidig begrænser weight decayet egenværdierne for $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ nedad til (se appendiks D) med bedre konditionering til følge.

6.11 Sammenfatning

Dette kapitel har behandlet optræning af layered RTRL netværk. Tilstrækkelig optræning af et givent netværk er naturligvis helt essentiel for netværkets evne til at udføre den givne opgave, men også i forbindelse med efterfølgende beskæring er det af stor vigtighed, at netværket er trænet til et minimum for cost funktionen.

Optræningsmetoder

I afsnit 6.4 samt afsnit 6.5 afprøves på solpletserien fem forskellige metoder til optræning af netværkene, to gradientmetoder samt tre andenordens metoder. Denne afprøvning afslører store kvalitative forskelle metoderne imellem.

De to gradientmetoder, RTRL samt gradient descent med adaptiv bestemmelse af learning rate η, har store problemer med overhovedet at konvergere mod et (lokalt) minimum for cost funktionen. Udfra de illustrerede iterationsforløb kan man stille sig tvivlende overfor, om de overhovedet vil kunne opnå samme niveau for træningsfejlen, som nås med andenordens metoderne, inden for endelig tid. Denne langsomme konvergens skyldes de store konditionstal for Hessianmatricen, der senere konstateres. Selv med virkeligt store weight decays ligger de laveste konditionstal omkring $\kappa_2(\mathbf{H}) = 10^4$, hvilket ifølge afsnit 1.3 *kan* være ødelæggende for konvergensen af en gradientmetode.

Pseudo Gauss-Newton henregnes til andenordens metoderne, men er egentlig en hybrid imellem første- og andenordens metoder, grundet den grove approksimation til Hessianmatricen. Det opnåede niveau for træningsfejlene ligger omkring 25 % under det tilsvarende opnået med gradientmetoderne, hvorfor pseudo Gauss-Newton betragtes som væsentligt bedre end disse. I betragtning af, at denne forbedring opnås blot ved i forhold til gradient descent at udføre en enkelt multiplikation samt division ekstra pr. element i søgeretningsvektoren, må pseudo Gauss-Newton klart anbefales fremfor gradient descent. Pseudo Gauss-Newton har dog også problemer med at konvergere mod et minimum for cost funktionen, hvilket fremgår dels af niveauet for træningsfejlen sammenlignet med de øvrige andenordensmetoder, dels af gradientnormen i slutpunkterne.

For at træne netværkene 'helt i bund' er det tilsyneladende nødvendigt at benytte Gauss-Newtons metode. Niveauet for de her opnåede træningsfejl ligger yderligere 25% under det tilsvarende niveau for pseudo Gauss-Newton, ligesom gradient-normen for de større værdier af weight decayet ligger på et niveau, der viser, at de bestemte punkter rent faktisk *er* (lokale) minima.

Succesfuld anvendelse af Newtons metode fordrer, at man er 'tæt' på et (lokalt) minimum. Er dette opfyldt, kan man opnå en markant forbedring af konvergensen fremfor med Gauss-Newtons metode, målt i antal iterationer. Forskellen er dog langtfra så markant, som det var tilfældet mellem Gauss-Newton og gradient descent/pseudo Gauss-Newton, hvor det reelt var et spørgsmål om konvergens mod et (lokalt) minimum eller ej; mellem Newtons og Gauss-Newtons metode er der ingen kvalitative forskelle mellem de resulterende bestemte minimaer.

I appendiks G.2 er der for et netværk foretaget en sammenligning af effektiviten imellem de her benyttede optræningsmetoder. Både opnået træningsfejl, benyttet antal iterationer samt forbrugt tid sammenlignes. Af sammenligningen konkluderes det, at Gauss-Newtons metode er den *mest effektive* metode at anvende for netværk af de i dette arbejde benyttede størrelser, da den tidsmæssigt er hurtigst og samtidig opnår det laveste niveau for træningsfejlen. Newtons metode er hurtigst målt i antal forbrugte iterationer, men beregning af søgeretningen udfra den fulde Hessianmatrix vises i appendiks G.1 at være så tidskrævende, at metoden ikke kan svare sig.

Konditioneringsproblemer

Anvendelse af Gauss-Newtons metode til optræning har afsløret store konditioneringsproblemer for cost funktionens Hessianmatrix. Disse problemer har deres rod i flere forhold omkring netværkene. I afsnit 6.8 vises det, at hvis der benyttes feedback fra et lineært output til de skjulte enheder, opstår der lineære afhængigheder imellem nogle af vægtene, hvilket uvægerligt vil medføre singularitet i Hessianmatricen. Det viser sig, at eliminering af disse vægte dog ikke løser problemet helt, men blot reducerer det.

Den resterende del af konditioneringsproblemerne vises i afsnit 6.10 delvis at stamme fra tilnærmelsesvis lineær afhængighed mellem søjlerne i Jacobimatricen med de partielle afledede for hvert eksempel, hvilket kan medføre meget små egenværdier. Denne afhængighed skyldes egenskaber ved de skjulte enheders sigmoid-funktioner. Det vises også, at der kan forekomme meget *store* egenværdier for de rekursive netværk. Disse store værdier forårsages af meget store værdier for de partielle afledede, der igen skyldes en kombination af store vægte samt rekursiv beregning af de afledede.

Weight decay

Udvidelse af cost funktionen med weight decay synes at fungere som et universalmiddel imod alle de problemer med optræningen, der er konstateret for de rekursive netværk; især er det vigtigt med betydeligt weight decay for de rekursive vægte.

Weight decayet begrænser vægtenes numeriske størrelse. Dette modvirker, at der opstår stabilitetsproblemer for de skjulte enheder, med store fejl til følge, som det vises i figur 6.14, hvor de skjulte enheder går i mætning. De begrænsede vægte kan også betyde, at de partielle afledededes numeriske størrelse begrænses, som det nævnes i afsnit 6.10; dette medfører, at de største egenværdier for Hessianmatricen, og dermed dennes konditionstal, reduceres.

Weight decayet begrænser de singularitetsproblemer, der uvægerligt vil opstå for neurale netværk, beskrevet i afsnit 6.10. Dette opnås ved at forøge de mindste egenværdier, hvorved der i hvert fald for den positivt definite approksimation til Hessianmatricen, anvendt af Gauss-Newtons metode, vil opnås bedre konditionering.

De udførte forsøg viser, at weight decayet også har en gavnlig indflydelse på netværkenes generaliseringsevne. Talrige forsøg illustrerer, at den gennemsnitlige generaliseringsfejl først falder med øget weight decay grundet mindsket overtræning, hvorefter den begynder at stige, svarende til at de pålagte restriktioner for vægtene bliver *for* kraftige.

Ved sammenligning af de rekursive netværk med feedforward netværk i afsnit 6.9 fremgår det tydeligt, at et rekursivt netværk (i middel) kan bringes til at generalisere væsentligt bedre end det tilsvarende feedforward netværk med samme antal inputs samt antal skjulte enheder, ved passende valg af weight decayet for de rekursive vægte. Det skal dog her tages med i betragtning, at de to sammenlignede netværk ikke indeholder samme antal parametre. De mindste middeltestfejl observeret i afsnit 6.9 kan dog ikke måle sig med de i kapitel 5 oplyste bedste resultater, der er opnået med feedforward netværk. Dette skyldes sandsynligvis, at den anvendte struktur af netværket ikke er optimal for det givne problem, hvad efterfølgende beskæring muligvis kan rette op på.

Kapitel 7

Forsøg med beskæring

Dette kapitel indeholder en beskrivelse af de forsøg, der er udført med beskæringsalgoritmerne *OBD*, *OBS* samt *magnitude based* beskæring, for hvilke det teoretiske fundament blev udledt i kapitel 4.

De i dette kapitel udførte forsøg skal betragtes som en *indledning* til beskæring af fuldt forbundne feedback netværk med metoderne Optimal Brain Damage samt Optimal Brain Surgeon. Forsøgsrækken er langt fra så omfattende, som det var tilfældet for optræningen i kapitel 6, hvilket skyldes, at der i dette arbejde er lagt størst vægt på at sikre effektiv optræning, hvilket må være en forudsætning for effektiv anvendelse af beskæringsalgoritmerne.

7.1 Beskrivelse af netværk

I afsnit 6.9 blev der udført en forsøgsrække for et layered RTRL netværk med 73 vægte, der forbandt fire skjulte enheder, som hver modtog 12 eksterne inputs. Outputenheden modtog ikke de eksterne inputs, og havde hverken lokalt feedback eller forbindelse *tilbage* til de skjulte enheder. Vægtene var delt op i to grupper med hvert sit weight decay, α_{ff} for *feedforward* vægtene og α_{rec} for *feedback* vægtene. Forsøgene illustrerede, for hvilke kombinationer af de to weight decays optræning af netværkene med Gauss-Newtons metode kunne forløbe uden betydende konditioneringsproblemer for Hessian-approksimationen. Forsøgene viste yderligere, hvilket niveau testfejlen ligger på for den benyttede arkitektur, for forskellige kombinationer af de to weight decays.

De laveste middeltestfejl blev opnået med $\alpha_{ff} = 0.01$, og er afbildet i figur 6.32. I det følgende beskrives en forsøgsrække, hvor de lave testfejl med $\alpha_{ff} = 0.01$ er forsøgt reduceret yderligere ved at beskære netværkene med OBD og OBS; desuden er netværkene beskåret med *Magnitude based pruning* for at se, hvordan denne simple metode vil klare sig sammenlignet med de to andre. Til at bestemme det *optimale* netværk benyttes Akaike's FPE-estimat. For alle tre metoder elimineres een vægt ad gangen, selvom OBD samt Magnitude based metoden åbner op for eliminering af flere vægte på een gang (se kapitel 4).

For $\alpha_{ff} = 0.01$ er der for forskellige værdier af α_{rec} optrænet ti netværk til et (lokalt) minimum. Nedenfor gengives de opnåede middeltestfejl med angivelse af standardafvigelse efter denne optræning:

α_{ff}	α_{rec}	ARV_{test}
0.01	0.05	0.1108 ± 0.0129
0.01	0.075	0.0927 ± 0.0002
0.01	0.1	0.0920 ± 0.0161
0.01	0.125	0.0884 ± 0.0068
0.01	0.15	0.0908 ± 0.0064

Tabel 7.1: Testfejl inden beskæring.

Netværkene er herefter beskåret med hver af de tre metoder; hver metode er således blevet afprøvet på præcis de samme netværk, for at give lige begyndelsesbetingelser.

Det skal her bemærkes, at nogle af de optrænede netværk for samme kombination af weight decays havde nøjagtig den samme trænings- samt testfejl, som det også er tilfældet for tilsvarende netværk på figur 6.29 samt figur 6.30 i kapitel 6.9. Dette skyldes det relativt store anvendte weight decay, der bevirker, at de optrænede netværk bliver ens pånær en permutation af de skjulte enheder. Dette havde den betydning, at forløbene af fejlene for disse netværk under beskæringen var helt ens. De beskårede netværk for to sådanne helt ens forløb af fejlene blev skitseret ved siden af hinanden, og det viste sig netop, at de var ens pånær en permutation af enhederne.

7.2 Genoptræning

Efter at en vægt er skåret væk, *genoptrænes* det reducerede netværk med Gauss-Newtons metode, indtil stopkriteriet $\|C(\mathbf{w})\|_2 < 10^{-6}$ er opfyldt; dette stopkriterie blev i afsnit 6.9 opfyldt uden problemer for alle netværk med de her benyttede kombinationer af α_{ff} og α_{rec}. Det maksimalt tilladte antal iterationer for at opfylde stopkriteriet er sat til 100, men stopkriteriet opfyldes som oftest efter 10–20 iterationer. Efter Gauss-Newton optræningen foretages der nogle få iterationer med Newtons metode for at reducere gradient-normen yderligere. I de fleste tilfælde indebærer disse iterationer, at gradient-normen bliver bragt adskillige størrelsesordner *under* 10^{-6}. Således sikres det, at cost funktionen efter hver vægt-eliminering igen er genoptrænet til et punkt, der ligger meget tæt på et (lokalt) minimum.

Med velvalgt weight decay samt optræning med Gauss-Newtons/Newtons metode spiller antallet af maksimalt tilladte iterationer til genoptræning ikke den samme betydende rolle, som det vil gøre for feks. gradient descent eller pseudo Gauss-Newton, hvor konvergensen mod et minimum er *meget* langsom, som det blev vist i henholdsvis afsnit 6.4 og afsnit 6.5. For gradientmetoder vil det i praksis altid være muligt at presse træningsfejlen en *lille* smule nedad ved at anvende *flere* iterationer, mens Gauss-Newton/Newtons metode i få iterationer når så tæt på et (lokalt) minimum, at yderligere iterationer ikke har nogen mening, da gradienten, og dermed Newtonskridtet, er *meget* tæt på nul.

De lave gradient-normer efter genoptræning betyder, at de antagelser, der for OBD samt OBS gøres om, at cost funktionen er optrænet til et (lokalt) minimum, således at gradienten kan negligeres, er opfyldte. Det er derfor ikke aktuelt at tage de udtryk for OBD samt OBS i brug, der udledes i appendiks B, hvor det *ikke* antages, at gradienten kan negligeres.

7.3 Resultater

På figur 7.1–figur 7.3 illustreres de opnåede resultater med beskæringen af de i afsnit 7.1 omtalte netværk. Tabellen i hver figur indeholder angivelse af de opnåede middeltræningsfejl samt middeltestfejl med den pågældende beskæringsmetode. Disse fejl er bestemt udfra de netværk, der blev udpeget af Akaike's FPE-estimat[1]. Dernæst angives antallet af parametre, som de af FPE udpegede netværk indeholdt. Tabellerne for OBD samt OBS indeholder derudover en søjle, der angiver, hvor stor en del af de vægte, der blev valgt til at blive fjernet, som også havde lavest 'sand' saliency (se kapitel 4), dvs. rent faktisk *var* de vægte, der ved bortskæring (og for OBS deraf følgende genoptræning af de tilbageblevne vægte) ville forårsage de mindste stigninger i *træningsfejlen*. Tallene angiver andelen af korrekt udpegede vægte for alle ti beskæringsforløb.

Foruden tabellen indeholder hver figur for $\alpha_{ff} = 0.01$ samt $\alpha_{rec} = 0.125$ tre eksempler på fejlene samt FPE-estimatet som funktion af antallet af parametre i netværket. Netværket, der af FPE-estimatet udpeges til at være 'det bedste', er specielt markeret. Til sidst illustreres 2-normen af gradienterne efter genoptræning for alle ti forsøg. Det illustrerede weight decay er tilfældigt valgt, forløbene for de andre kombinationer havde tilsvarende udseende.

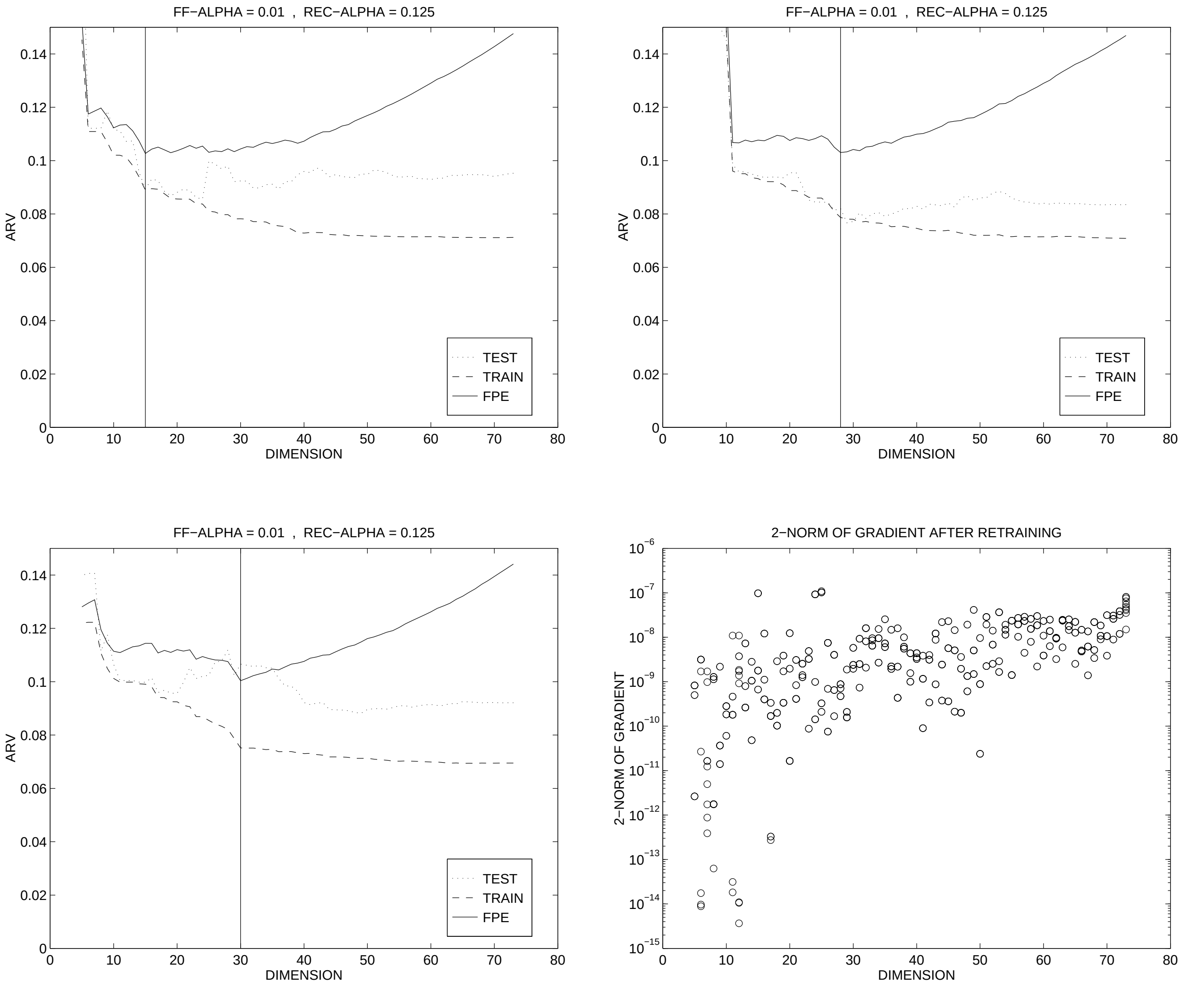
Ved at sammenligne de opnåede middeltestfejl for de forskellige weight decays i figur 7.1 – 7.3 med middeltestfejlene *inden* beskæring i tabel 7.1 ovenfor, fremgår det, at generaliseringsevnen for alle de afprøvede kombinationer af weight decayet gennemsnitlig er blevet *forringet*, uanset hvilken metode, der er anvendt til beskæringen. Generelt må det derfor konstateres, at forsøgene på at forbedre netværkenes generaliseringsevne er slået fejl.

En af årsagerne til dette er, at FPE-estimatet meget ofte udpeger et netværk, der *ikke* har minimal testfejl, hvilket man kan få et indtryk af ved at

[1]Som nævnt i afsnit 4.3 tages der her ikke højde for weight decay, der benyttes $N_{eff} = N$.

Resultater med Optimal Brain Damage

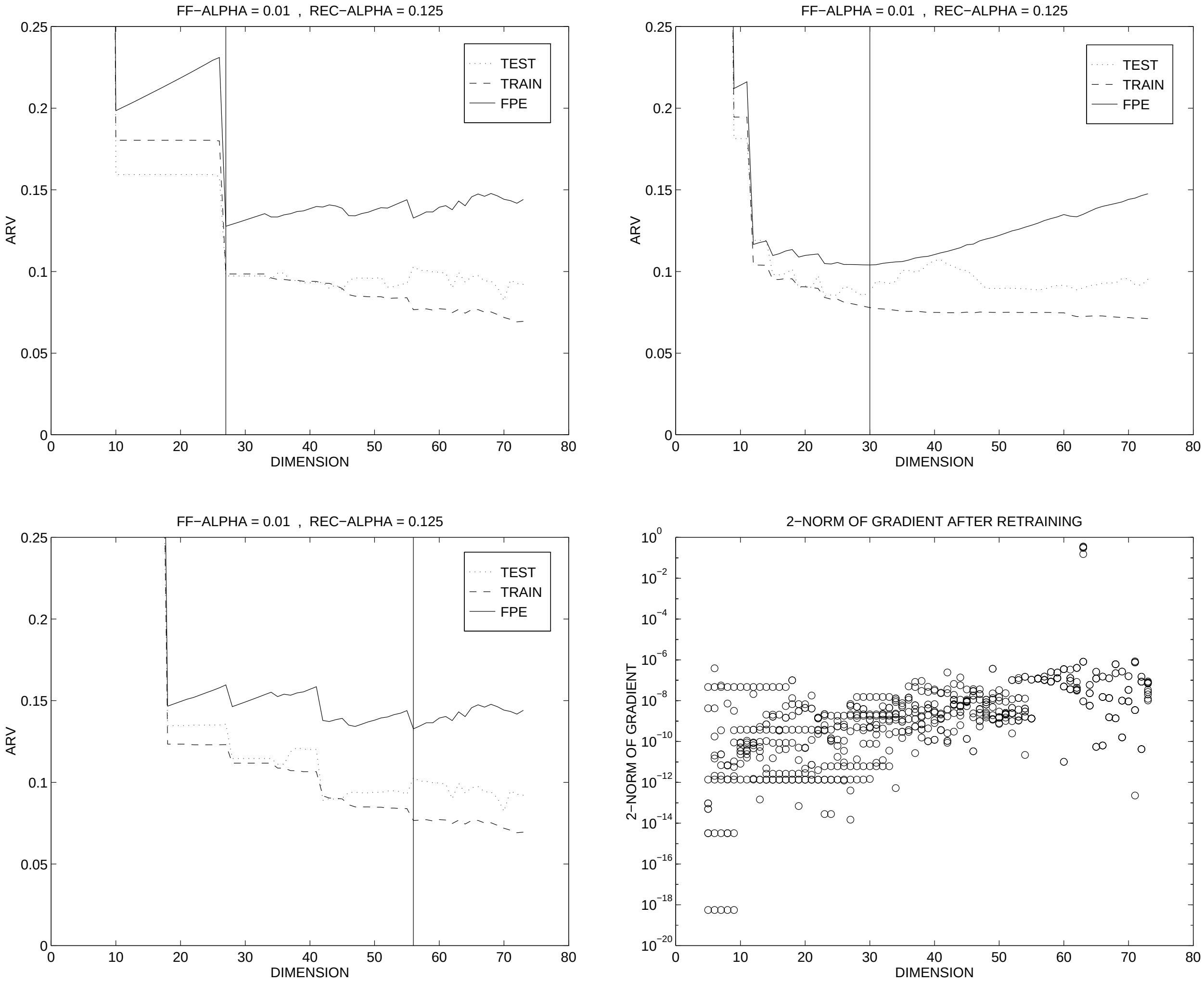
α_{ff}	α_{rec}	ARV_{train}	ARV_{test}	Parametre	% korrekte valg
0.01	0.05	0.0646 ± 0.0019	0.1149 ± 0.0014	$15 - 41$	81.2
0.01	0.075	0.0774 ± 0.0012	0.0935 ± 0.0022	$24 - 34$	89.1
0.01	0.1	0.0779 ± 0.0022	0.1003 ± 0.0064	$21 - 30$	93.8
0.01	0.125	0.0796 ± 0.0092	0.0909 ± 0.0015	$15 - 30$	90.3
0.01	0.15	0.0773 ± 0.0062	0.0988 ± 0.0171	$23 - 33$	93.5



Figur 7.1: Beskæring med Optimal Brain Damage. Samlede resultater samt eksempler på forløb af fejlene samt gradient-normen i slutpunktet efter genoptræning for en given kombination af weight decays. Den lodrette linie markerer det 'optimale' netværk udpeget af FPE-estimatet.

Resultater med Optimal Brain Surgeon

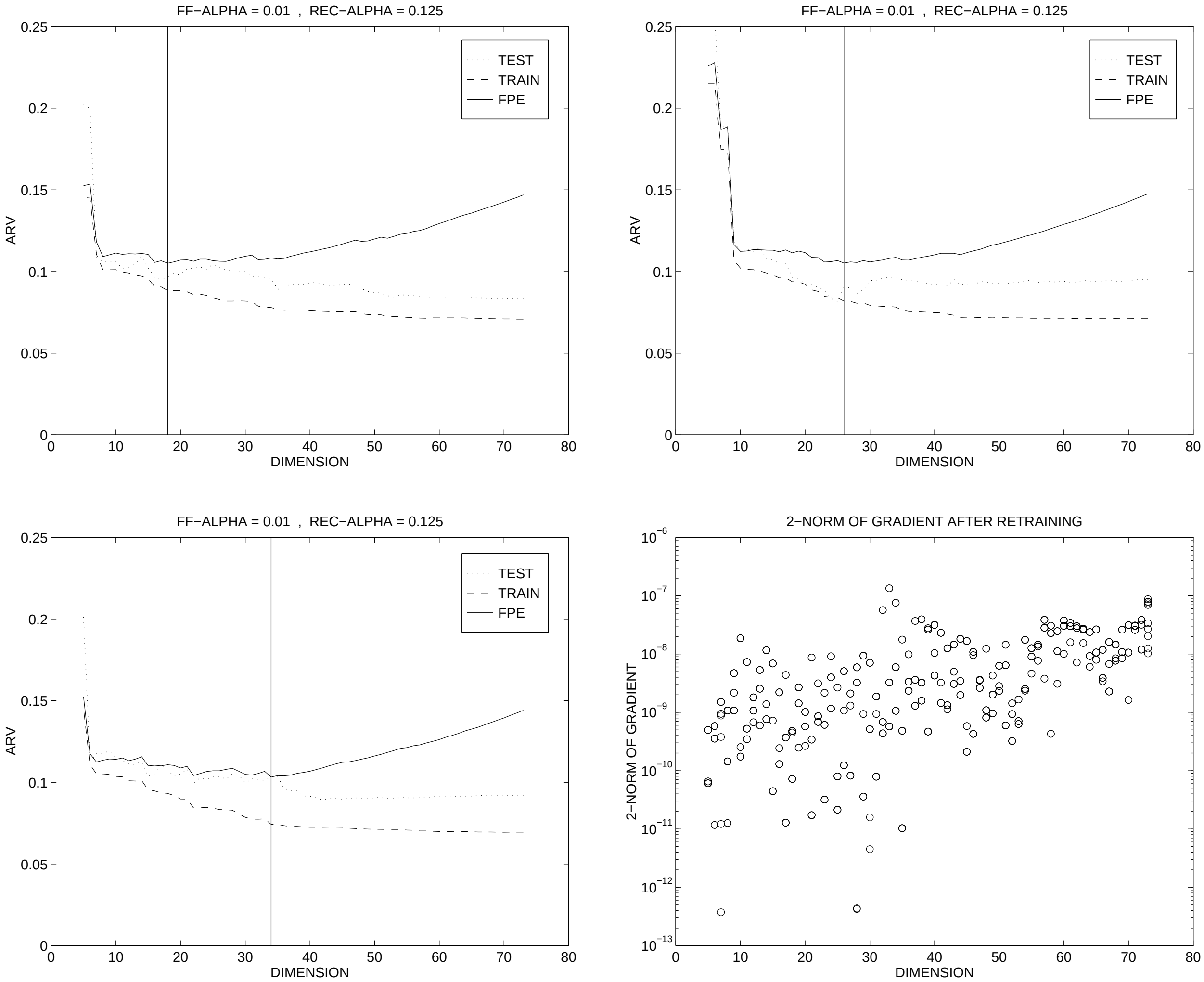
α_{ff}	α_{rec}	ARV_{train}	ARV_{test}	Parametre	% korrekte valg
0.01	0.05	0.0635 ± 0.0127	0.1748 ± 0.0371	$23 - 70$	26.3
0.01	0.075	0.0793 ± 0.0027	0.0991 ± 0.0030	$38 - 50$	14.4
0.01	0.1	0.0788 ± 0.0009	0.0942 ± 0.0066	$25 - 50$	16.9
0.01	0.125	0.0797 ± 0.0037	0.0981 ± 0.0072	$27 - 56$	12.9
0.01	0.15	0.0871 ± 0.0101	0.1024 ± 0.0016	$17 - 57$	6.3



Figur 7.2: Beskæring med Optimal Brain Surgeon. Samlede resultater samt eksempler på forløb af fejlene samt gradient-normen i slutpunktet efter genoptræning for en given kombination af weight decays. Den lodrette linie markerer det 'optimale' netværk udpeget af FPE-estimatet.

Resultater med Magnitude Based Pruning

α_{ff}	α_{rec}	ARV_{train}	ARV_{test}	Parametre
0.01	0.05	0.0664 ± 0.0021	0.1268 ± 0.0277	$22-35$
0.01	0.075	0.0789 ± 0.0027	0.1071 ± 0.0071	$13-32$
0.01	0.1	0.0760 ± 0.0020	0.0997 ± 0.0037	$23-34$
0.01	0.125	0.0815 ± 0.0069	0.0981 ± 0.0013	$18-34$
0.01	0.15	0.0782 ± 0.0102	0.1013 ± 0.0045	$18-34$



Figur 7.3: Magnitude baseret beskæring. Samlede resultater samt eksempler på forløb af fejlene samt gradient-normen i slutpunktet efter genoptræning for en given kombination af weight decays. Den lodrette linie markerer det 'optimale' netværk udpeget af FPE-estimatet.

betragte illustrationerne i figur 7.1–figur 7.3. FPE vælger netværk, der ligger umiddelbart før en markant forøgelse af hældningen af kurven for træningsfejlen. Dette stemmer overens med udtrykket (4.23) for FPE-estimatet; hvis træningsfejlen stiger mere end den foranstående faktor formindskes, opstår der et minimum for estimatet. Uheldigvis er disse minima her ikke sammenfaldende med testfejlenes minima.

En anden årsag til den forværrede middeltestfejl er, at for nogle af beskæringsforløbene 'dykker' testfejlen på intet tidspunkt ned *under* begyndelsesværdien. Det er således ikke *udelukkende* FPE-estimatets manglende evne til at udpege det korrekte netværk, der er årsag til, at testfejlene forværres.

I [Svarer 93] genoptrænedes de beskårne netværk *uden* weight decay. Dette blev gjort ud fra den betragtning, at FPE-estimatet havde udpeget den 'optimale' arkitektur, hvorfor der ikke længere var behov for at begrænse netværkets frihedsgrader for at modvirke overtræning. Optræning af de udpegede netværk uden weight decay rapporteredes at forbedre netværkenes generaliseringsevne yderligere.

Eliminering af weight decayet med efterfølgende optræning med Gauss-Newtons metode er også forsøgt her for *nogle* af de beskårede netværk, men træningsfejlen bliver ikke reduceret i alle tilfælde. Hvis netværket indeholder feedback vægte, resulterer optræningen uden weight decay i de *fleste* tilfælde i overtræning; træningsfejlen falder markant, samtidig med at testfejlen stiger. For de større udpegede netværk kan der tillige opstå konditioneringsproblemer for Hessian-approksimationen, som det er beskrevet i kapitel 6. Det faktum, at netværkene ved yderligere optræning uden weight decay stadig overtræner, tages som udtryk for, at de udpegede netværk ikke er tilstrækkeligt godt beskåret; de indeholder stadig for mange ressourcer i form af parametre i forhold til, hvad der er nødvendigt for at modellere solpletserien.

Ovenstående problemer med yderligere optræning uden weight decay er årsagen til, at de rapporterede middeltestfejl i figur 7.1–figur 7.3 *alle* er for netværkene, hvor der *ikke* er foretaget yderligere optræning uden weight decay. For nogle få af netværkene er det forsøgt at *ændre* på weight decayet, og derefter foretage yderligere optræning med Gauss-Newtons metode. I de fleste tilfælde er det lykkes at finde en kombination af α_{ff} samt α_{rec}, for hvilken generaliseringsevnen forbedres, men kun for få netværk er forbedringen markant.

Valget af de optimale weight decays for de beskårede netværk er ikke simpelt, og vil i praksis kræve, at der for *hvert* af dem bliver udført lige så omfattende tests, som det blev gjort for netværkene i afsnit 6.7 samt afsnit 6.9, hvilket er uoverkommeligt. En sådan fremgangsmåde svarer dog til at vælge de optimale weight decays udfra fejlen på det *uafhængige* testsæt, hvilket ikke er 'tilladt'. Snarere burde weight decay'et bestemmes adaptivt som i feks. [Hansen 94a].

I afsnit 7.4 gives der et eksempel på et netværk, der genoptrænes med ændret weigh decay.

I figur 7.1 for Optimal Brain Damage viser tabellen, at OBD i omkring 90 % af alle tilfælde korrekt har bestemt den vægt, hvis eliminering forårsager mindst stigning i træningsfejlen. Dette er overraskende højt, i betragtning af cost funktionens ikke-kvadratiske udseende, jvf. figur 6.8. På de afbildede forløb ses træningsfejlen da også at have en jævn, svag stigning, indtil antallet af parametre i netværket begynder at blive kritisk for netværkets evne til at modellere træningssættet, hvor stigningen bliver mere markant.

Testfejlene har derimod et mere ujævnt forløb, ind imellem stiger de, ind mellem falder de. Der observeres langt fra det jævnt faldende forløb, der vises i [Svarer 93] for beskæring af et feedforward netværk. Det anvendte weight decay har dog stor indflydelse på testfejlenes forløb under beskæringen, hvilket illustreres i afsnit 7.5. For alle tre afbildede forløb af testfejlen ses det, at denne på et eller andet tidspunkt i forløbet er faldet til en smule *under* begyndelsesniveauet; for nogle af de øvrige forløb gælder det derimod, at fejlen i begyndelsespunktet er den mindste, der observeres. Generaliseringsevnen kan således ikke hver gang forbedres ved at beskære netværket. Antallet af parametre i de udpegede netværk ligger mellem 15–41. I [Svarer 93] var det tilsvarende antal for feedforward netværk 12–16, hvilket indikerer, at netværkene her ikke er beskåret tilstrækkeligt.

I figur 7.2 viser tabellen, at OBS-metoden er temmelig dårlig til at udpege de vægte, der ved eliminering samt heraf resulterende ændring af de tilbageblevne vægte vil resultere i den mindste forøgelse af træningsfejlen. OBS anvender den inverterede af *hele* Hessianmatricen. Som det blev gjort i [Hassibi 93] benyttes der ved beskæringerne her blot Gauss-Newton approksimationen til Hessianmatricen, de anden afledede af cost funktionen negligeres. Forsøg med den *fulde* Hessianmatrix gav dog ikke bedre resultater.

Resultatet af den dårlige udpegelse fremgår af de afbildede forløb af træningsfejlen. Ofte resulterer det forkerte valg af vægt i, at træningsfejlen, og med den også FPE-estimatet, foretager nogle mindre 'spring'. Ind imellem resulterer det forkerte valg af vægt, at træningsfejlen foretager et endog meget stort spring opad, der reelt ødelægger netværkets funktion. Disse spring er årsagen til, at FPE-estimatet udpeger netværk, hvis antal parametre er væsentligt større, end det var tilfældet med OBD; antallet af vægte ligger imellem 17–70!

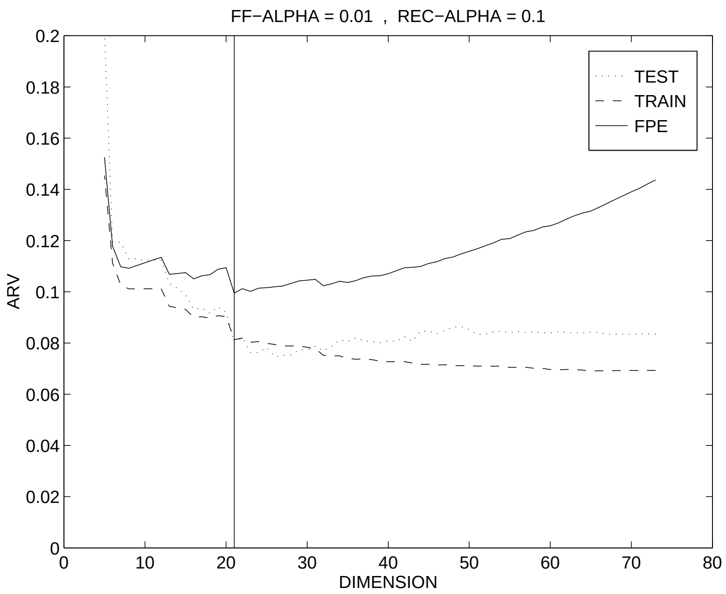
De mange forkerte valg af vægten, der skal beskæres, skyldes muligvis, at cost funktionen er kraftigt ikke-kvadratisk, som det fremgår af figur 6.8. OBS stiller forholdsvis store krav til den kvadratiske approksimation af cost funktionen, idet ændringen i træningsfejlen ikke blot estimeres ved nulstilling af en *enkelt* vægt som for OBD, men ved ændring af *samtlige* vægte. På trods af, at gradienten i udviklingspunktet som antaget er omtrent nul, er disse krav tilsyneladende for store, at dømme udfra de mange forkerte valg af vægten til beskæring.

I [Hassibi 93] lægges der op til, at den 'indbyggede' optræning af de øvrige vægte er tilstrækkelig, yderligere genoptræning er overflødig. Dette er langt fra tilfældet for rekursive netværk. Hvis den udførte yderligere genoptræning med Gauss-Newtons metode udelades, bryder beskæringen sammen, efter at ganske få vægte er fjernet, da den indbyggede genoptræning ikke opnår et minimum for den *rigtige* cost funktion, men kun for den kvadratiske approksimation. Gradienten for cost funktionen er således ikke nul i det nye punkt, hvorfor den anvendte kvadratiske approksimation bliver endnu dårligere, hvilket igen øger sandsynligheden for, at der vælges en vægt til beskæring, som bevirker en kraftig stigning i cost funktionen.

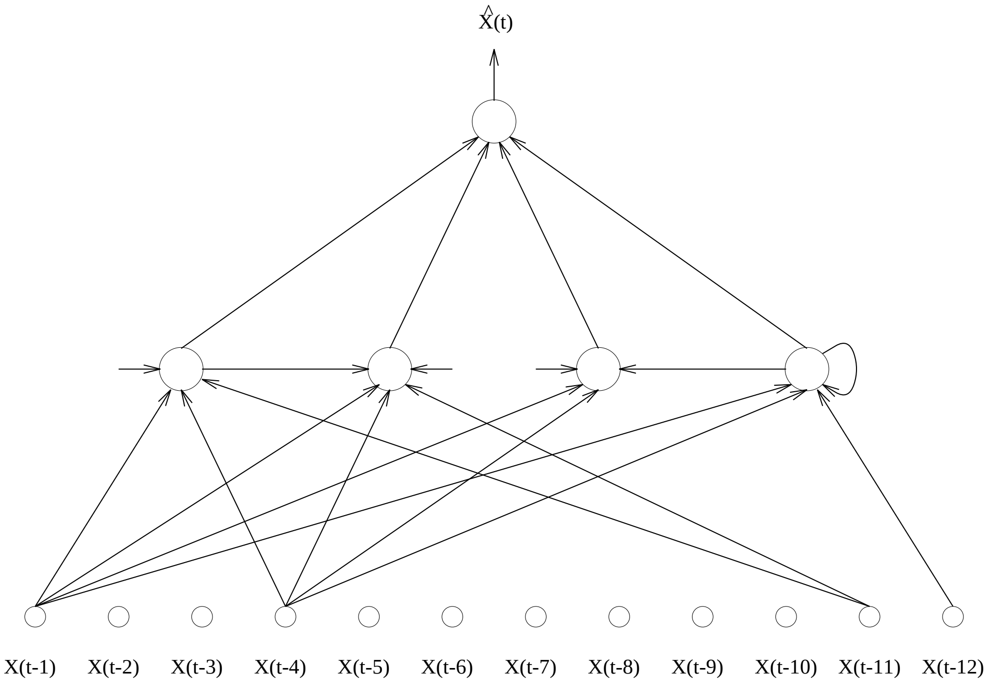
Figur 7.3 viser de opnåede resultater med den simple magnitude based metode. Forløbene af træningsfejlene ses at være meget lig de tilsvarende forløb for OBD, svagt stigende. Antallet af parametre i de af FPE-estimatet udpegede netværk ligger derfor på niveau med OBD, hvorimod middeltestfejlen ligger lidt højere, end det er tilfældet for både OBD og OBS. Dette skyldes, at det for magnitude metoden oftere forekommer, at den mindste testfejl i beskæringsforløbet er den, som netværket har *inden* der bliver fjernet vægte. Dette kan tyde på, at de mindste vægte ikke er dem, der betyder *mindst* for netværkets *generaliseringsevne*.

7.4 Eksempel på netværk

I dette afsnit vises et eksempel på genoptræning med ændret weight decay af et netværk med *lille* testfejl udpeget af FPE-estimatet. Cost funktionen var for det pågældende netværk udvidet med $\alpha_{ff} = 0.01$ samt $\alpha_{rec} = 0.1$, og



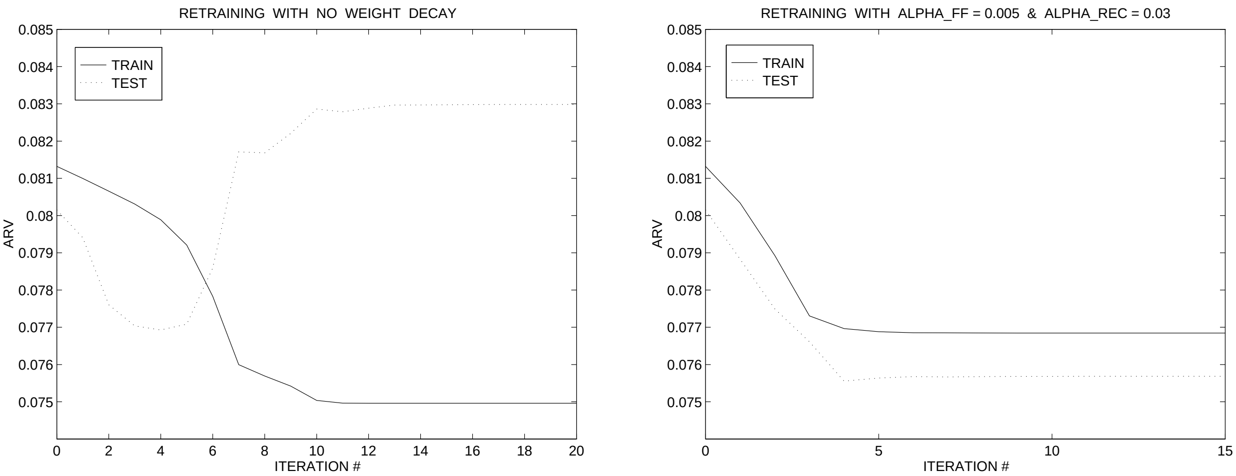
Figur 7.4: Beskæringsforløb med Optimal Brain Damage.



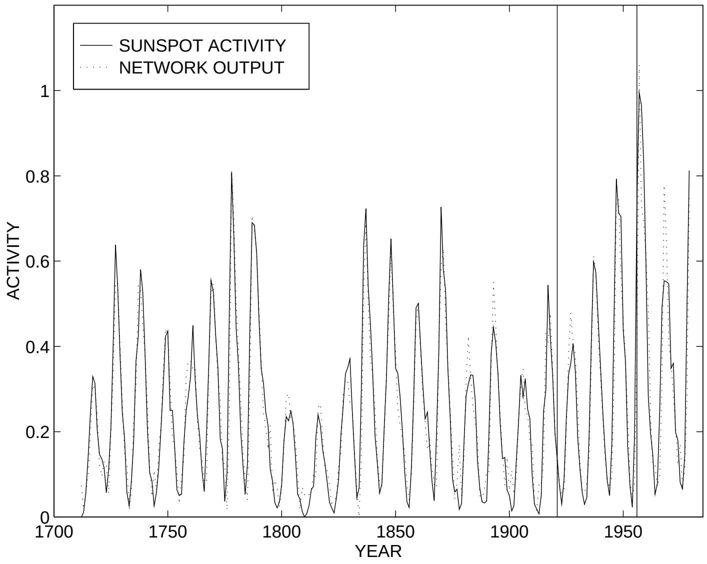
Figur 7.5: Arkitektur for beskåret netværk.

forløbet af beskæringen vises i figur 7.4. Det af FPE udpegede 'optimale' netværk har ikke den *mindste* testfejl, der observeres under beskæringsforløbet, men er 'tæt' på. Det udpegede netværk har 21 vægte, og er skitseret i figur 7.5. Det bemærkes, at netværket kun gør brug af nogle få elementer i inputvektoren, som det også var tilfældet i [Svarer 93].

Netværket er først genoptrænet *uden* weight decay. Forløbet af denne optræning er gengivet på den venstre graf i figur 7.6. Det bemærkes, at net-



Figur 7.6: Genoptræning af netværk med forskellige weight decays. Bemærk det snævre interval på ordinat-aksen.



Figur 7.7: Et-skridts prædiktion af solpletserien med beskåret netværk genoptrænet med mindre weight decay.

værket *overtrænes*, da træningsfejlen falder, og testfejlen ret hurtigt begynder at stige. Arkitekturen af det udpegede netværk er således ikke 'optimal', da netværket i så fald *ikke* vil overtræne; netværkets ressourcer er stadig for rigelige.

For at begrænse netværkets ressourcer er det herefter optrænet med varierende værdier af α_{ff} samt α_{rec}. Grafen til højre i figur 7.6 viser forløbet af optræningen med $\alpha_{ff} = 0.005$ samt $\alpha_{rec} = 0.03$, for hvilke netværket praktisk taget *ikke* overtrænede. Dette netværks output er i figur 7.7 vist sammen med solpletserien. De opnåede fejl blev:

α_{ff}	α_{rec}	ARV_{train}	ARV_{test}	Parametre
0.005	0.03	0.0768	0.0757	21

hvilket er sammenligneligt med de opnåede resultater for feedforward netværk i [Svarer 93]. De her benyttede weight decays er *mindre* end de oprindeligt anvendte, hvilket kan tolkes som at det beskårede netværk er 'tættere på' at være optimalt; begrænsningen af de frie parametre skal ikke være helt så kraftig for at undgå overtræning.

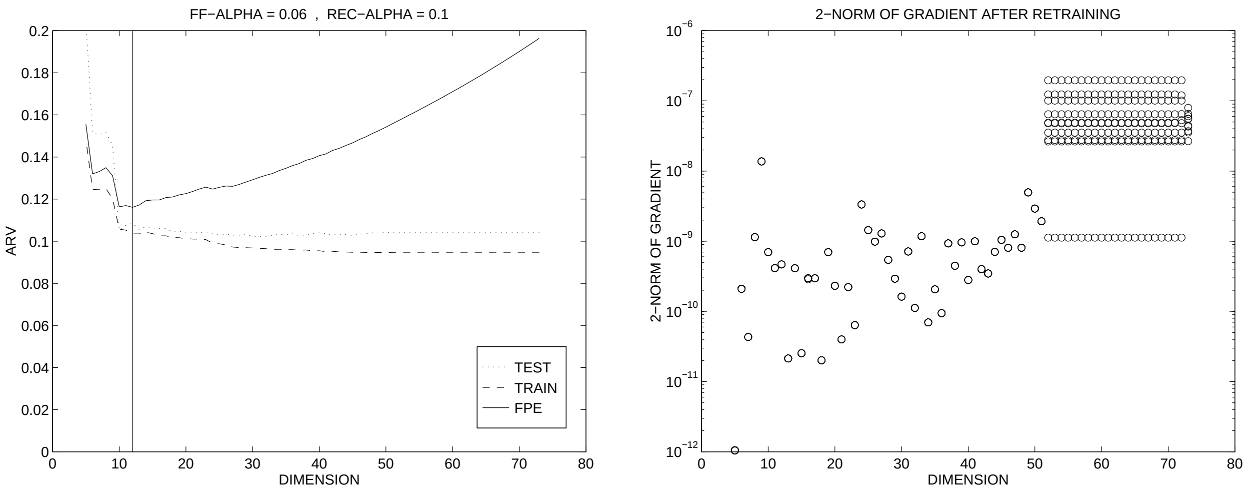
7.5 Større weight decay

I artiklen, der er gengivet i appendiks I, vises et eksempel på beskæring af et layered RTRL netværk med OBS. Det bemærkes, at forløbet af især testfejlen er mere 'glat', end det er tilfældet for nogen af de i figur 7.1–figur 7.3 afbildede forløb. Dette skyldes til dels, at der er benyttet 'sand' saliency fremfor den estimerede saliency ved valget af den vægt, der skæres bort; i langt højere grad skyldes det dog, at der er blevet benyttet et meget stort weight decay, $\alpha = 0.08$, for *alle* vægte. Weight decayet lægger store begrænsninger på hver vægt, så de hver især kun har lille indflydelse på netværkets output; dette bevirker det meget jævne forløb af fejlene i beskæringsforløbet. Akaike's FPE-estimat har således bedre muligheder for at udpege det 'optimale' netværk, da testfejlen i det store hele følger træningsfejlen. Prisen for det mere jævne forløb af fejlene er et højt niveau for disse; succesfuld retraining med ændret weight decay af netværk med så høje fejl er meget vanskelig.

I figur 7.8 gengives resultatet af 10 beskæringer af den i dette kapitel benyttede arkitektur, når der benyttes OBD med større weight decay på *feedforward* vægtene, end der blev benyttet i forsøgene ovenfor. Tabellen afslører, at der ikke er nogen varians på fejlene, hvilket er en konsekvens af det store weight decay; alle forløb af beskæringerne er identiske med det i figur 7.8 afbildede. Forløbet af fejlene er langt mere jævnt, end det er tilfældet for de afbildede forløb i figur 7.1, og har lighed med de i [Svarer 93] afbildede tilsvarende forløb for feedforward netværk. Testfejlen ses at ligge konstant på et relativt højt niveau under hele beskæringsforløbet; den falder ikke, som det ofte var tilfældet med de mindre weight decays. Træningsfejlen er ligeledes meget monoton, hvilket er årsagen til, at FPE-estimatet udpeger

Resultater med Optimal Brain Damage – stort weight decay

α_{ff}	α_{rec}	ARV_{train}	ARV_{test}	Parametre	% korrekte valg
0.06	0.1	0.1035 ± 0.0000	0.1085 ± 0.0000	12	68.7



Figur 7.8: Beskæring med Optimal Brain Damage med større weight decay for feedforward vægtene.

netværk med meget få vægte.

Det udpegede netværk indeholder 12 vægte, hvoraf kun en enkelt er en *feedback* vægt. Netværket er optrænet *uden* weight decay, hvilket (uden overtræning) resulterede i fejlene

Uden weight decay:

ARV_{train}	ARV_{test}
0.1004	0.0973

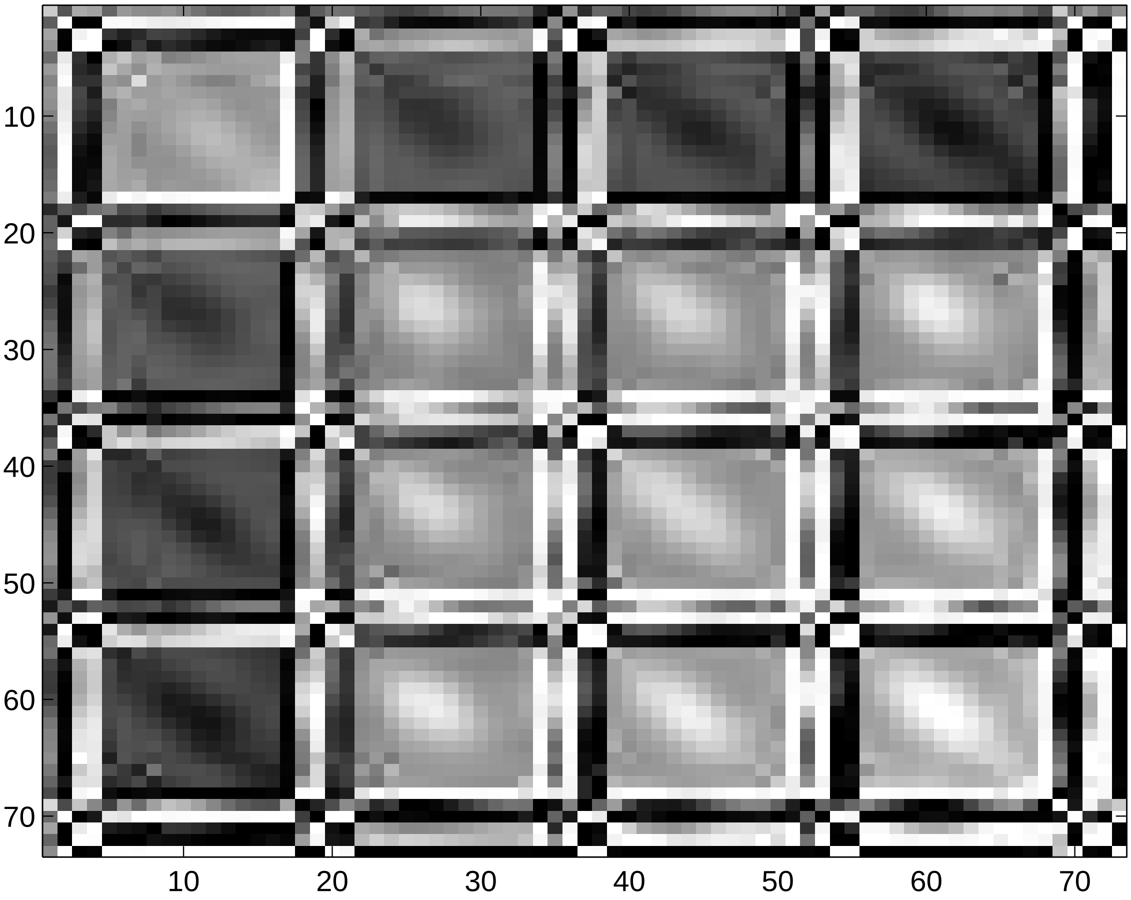
Forbedringen af generaliseringsevnen ses at være marginal, og langt fra det niveau, der blev opnået for eksemplet i afsnit 7.4. Selvom netværket ikke indeholder flere parametre end de i [Svarer 93] angivne, er generaliseringsevnen væsentlig dårligere, hvilket muligvis kan skyldes, at det store weight decay har 'sløret' vægtenes betydning så meget, at de 'forkerte' er blevet skåret bort.

Hensigten med dette lille forsøg har været at vise, at forløbet af fejlene under beskæringen er stærkt afhængig af de benyttede weight decays. Sandsynligvis vil det ved *passende* valg af disse kunne lade sig gøre at opnå et til [Svarer 93] tilsvarende forløb af testfejlen, hvor denne først falder monotont med de færre parametre, for derefter at stige igen, når ressoucerne i form af vægte bliver for sparsomme. Der er dog meget, der taler for, at et sådant *optimalt* weight decay vil lægge store begrænsninger på vægtene; beskæringsforløbene vil derfor, ved løbende genoptræning med de her benyttede andenordens metoder, være ens.

7.6 Illustration af Hessian

I dette afsnit præsenteres 'udseendet' af en Hessianmatrix, da Hessianen er central for beskæringsmetoderne OBD samt OBS. I figur 7.9 afbildes Gauss-Newton approksimationen til en Hessianmatrix for et optrænet netværk med den i dette kapitel anvendte arkitektur, *inden* der foretages beskæring. Matricen er afbildet med *image*-funktionen i MATLAB, der skalerer matricens elementer så de kan benyttes til opslag i en gråtone-tabel. Figur 7.9 giver således ikke information om størrelsen af de *enkelte* elementer, men kan illustrere det *relative* størrelsesforhold imellem disse, og dermed relationer elementerne imellem. Disse relationer vil det dog føre for vidt at komme ind på her, der skal blot gives en beskrivelse af elementernes relation til netværket.

Hvert element i matricen svarer til Gauss-Newton approksimationen til de anden afledede af cost funktionen, $\frac{\partial C(\mathbf{w})}{\partial w_{ij} \partial w_{pq}}$. De første fire søjler svarer til vægtene *til* den første skjulte enhed, *fra* hver af de ialt fire skjulte enheder. Herefter følger vægte for de 12 eksterne inputs til enheden, og søjle 17 er biasvægt for den første skjulte enhed. Herefter følger tilsvarende vægte for de skjulte enheder to, tre og fire. De sidste fem søjler svarer til vægte *til* outputenheden *fra* de fire skjulte enheder samt outputenhedens bias. Hessianmatricen er symmetrisk, så tilsvarende gælder naturligvis for rækkerne.



Figur 7.9: Hessianmatrix for optrænet 12-4-1 layered RTRL netværk.

7.7 Sammenfatning

Beskæringerne i dette afsnit har ikke været en ubetinget succes. Middeltestfejlen blev for både OBD, OBS samt magnitude base metoden *forværret* i samtlige forsøgsrækker.

Dette skyldes til dels, at FPE-estimatet ikke var i stand til at udpege de netværk, der havde den mindste testfejl; ofte var det temmelig langt fra at være tilfældet. En mulig forklaring på dette er, at de gjorte antagelser omkring anvendelse af FPE-estimatet til tidsserieprædiktion med feedback netværk er *for* grove.

Som oftest indeholdt de udpegede netværk for mange parametre, hvilket gav sig udslag i, at de overtrænede, når der blev forsøgt genoptræning *uden* weight decay, som det blev gjort i [Svarer 93].

Netop weight decayet synes at spille en meget stor rolle for forløbet af netværkenes fejl under beskæringen. De her anvendte weight decays viste sig ikke så hensigtsmæssige for beskæringen, selvom de gav de laveste testfejl i afsnit 6.9, testfejlen varierede for voldsomt. Dette kan dog reguleres med weight decayet, med *passende* valg af dette vil det sandsynligvis være muligt at opnå omtrent samme forløb for fejlene under beskæringen, som det i [Svarer 93] illustrerede for feedforward netværk.

På baggrund af de her udførte forsøg konkluderes det, at OBD er den mest anvendelige af de her sammenlignede metoder. De bedste netværk, målt både i størrelse samt testfejl, blev fundet ved beskæring med OBD, der ydermere i omkring 90 % af tilfældende udpegede vægten med lavest saliency.

I teorien bør OBS være mere hensigtsmæssig at anvende fremfor OBD, da saliency for OBS bestemmes som ændringen i træningsfejlen *efter* en genoptræning af de øvrige vægte. Dette vil da også være tilfældet, hvis den betragtede funktion er kvadratisk, hvor de gjorte andenordens approksimationer vil være eksakte. Imidlertid er cost funkionen for rekursive netværk kraftigt *ikke*-kvadratisk, hvilket lader til at have katastrofale konsekvenser for OBS. Kun i meget få tilfælde udpeges vægten med lavest 'sand' saliency, hvilket ofte fører til, at der bortskæres en vægt med så stor ændring i træningsfejlen til følge, at ændringen ikke kan udlignes ved genoptræning. Resultatet er store spring opad i både trænings- samt testfejl.

Magnitude based beskæring er meget simpel at implementere, og kan da heller ikke totalt afvises på baggrund af middeltestfejlene for netværkene udpeget af FPE-estimatet i de relativt få forsøg, der her er udført. Imidlertid lader det til, at testfejlen ved anvendelse af magnitude based metoden har en kraftig tendens til at stige i forhold til begyndelsesniveauet, uden senere at 'dykke' ned *under* dette, i hvilke tilfælde fordelen ved beskæring er stærkt begrænset. Flere forsøg bør dog udføres for at verificere denne iagttagelse.

Det faktum, at der *blev* opnået gode resultater med *nogle* af de beskårede netværk, tages som udtryk for, at beskæring af rekursive netværk *kan* foretages med de samme metoder, som der anvendes for feedforward netværk. Succesfuld beskæring med disse metoder afhænger imidlertid i høj grad af et 'velvalgt' weight decay. Da det ikke er 'tilladt' at vælge weight decays udfra fejlen på det uafhængige testsæt, bør det gøres *adaptivt*, feks. som i [Hansen 94a]. Dette vil ved succesfuld implementering ligeledes øge den generelle anvendelighed af beskæringsmetoderne.

Kapitel 8

Konklusion

I dette projektarbejde er der introduceret en ny metode til opdatering af fuldt rekursive netværk, der kaldes *layered RTRL*. Til forskel fra traditionelle fuldt rekursive netværk, hvor alle enheder i netværket opdateres synkront, opererer den her indførte arkitektur med *lag-deling* af enhederne i et skjult lag og et output lag, for hvilke opdateringen foregår som for et feedforward netværk. Dette har fordele for netværkets modelleringsevne, ligesom netværket ved eventuel beskæring kan reduceres til et sædvanligt feedforward netværk, ifald dette skulle vise sig optimalt.

For layered RTRL samt for de øvrige beskrevne arkitekturer er der udledt de anden afledede af cost funktionen til anvendelse ved optræning med andenordens metoder samt beskæring. Det må konstateres, at udbyttet af dette arbejde ikke helt har stået mål med anstrengelserne, idet den langt simplere Gauss-Newton approksimation til Hessianmatricen har vist sig mere praktisk anvendelig. Denne konstatering er vigtig, da man ved fremtidige anvendelser af andenordens metoder til optræning samt af beskæringsalgoritmer på denne baggrund kan undgå en meget vanskelig implementering af de anden afledede af cost funktionen.

Der er med succes benyttet andenordens metoderne Gauss-Newton samt Newtons metode til optræning af layered RTRL anvendt til tidsserieprædiktion på solpletserien. Disse metoder har vist sig at være særdeles effektive, da de både er tidsmæssigt hurtige samt i stand til at opnå et meget lavere niveau for cost funktionen, end det har vist sig praktisk muligt med med de hidtil anvendte rene gradientmetoder samt med 'hybriden' pseudo Gauss-Newton. Gauss-Newton er at foretrække fremfor Newtons metode, da den *tidsmæssigt* har vist sig at være hurtigst; det opnåede niveau for cost funktionen er det samme for de to metoder. Newtons metode har den ulempe, at den kun kan anvendes, når man er forholdsvis 'tæt' på et (lokalt) minimum, ligesom implementeringen af de anden afledede er meget vanskelig, som nævnt ovenfor. Gauss-Newtons metode besidder i modsætning hertil i en vis udstrækning globale konvergensegenskaber. Implementeringen af Gauss-Newtons metode er forholdsvis simpel at foretage i forhold til rene gradientmetoder, da den kun yderligere kræver det ydre produkt af gradienten med sig selv samt løsning af et lineært ligningssystem.

Netop løsning af det lineære ligningssystem kan være årsag til store problemer for andenordens metoderne, da der let opstår store konditioneringsproblemer for Hessianmatricen samt Gauss-Newton approksimationen til denne. Det er vist, at feedback fra et lineært output til de skjulte enheder medfører lineære afhængigheder imellem vægtene, med deraf følgende singularitet af Hessianmatricen. Ydermere er det illustreret, hvorledes omtrentlig lineær afhængighed mellem de partielle afledede kan give konditioneringsproblemer for Gauss-Newton approksimationen til Hessianmatricen. Store værdier af vægtene kan medføre store elementer i Hessianmatricen med deraf følgende store egenværdier, der igen kan betyde dårlig konditionering.

Udvidelse af cost funktionen med weigh decay har vist sig afgørende for håndtering af problemet med de store konditionstal for Hessianmatricen. Udover at begrænse egenværdierne nedadtil, begrænser weight decayet også egenværdierne opadtil i kraft af vægtenes begrænsede størrelse, som nævnt ovenfor. Ydermere har weight decay haft gavnlig indflydelse på netværkenes generaliseringsevne.

De store konditionstal for Hessianmatricen kan give en forklaring på, hvorfor konvergensen for gradientmetoderne generelt er så dårlig, da denne er afhængig af forholdet mellem største og mindste egenværdi for Hessianmatricen. Selv med forholdsvis stort weight decay er konditionstallet så stort, at konvergensen for disse metoder er *meget* langsom.

De udførte forsøg ved undersøgelse af optræningsmetoderne har ledt frem til, at *fuldstændigt* forbundne netværk ikke er hensigtsmæssige, når der benyttes lineært output, som det kan være nødvendigt for at sikre arbitrært dynamisk område ved tidsserieprædiktion. I disse tilfælde bør outputenheden være rent feedforward, da lokalt feedback i outputtet kan medføre ustabilitet for netværket, og feedback til de skjulte enheder som nævnt ovenfor medfører singularitet for de anden afledede. Ydermere bør outputenheden heller ikke modtage de eksterne inputs, for at forhindre, at netværket 'degenererer' til en lineær model.

I forsøget på at optimere arkitekturen er der udført nogle indledende forsøg med beskæringsmetoderne Optimal Brain Damage, Optimal Brain Surgeon samt magnitude based beskæring med anvendelse af Akaike's FPE-estimat til bestemmelse af det 'optimale' netværk. FPE-estimatet har vist sig at have besvær med at udpege netværket med den laveste testfejl. Dette skyldes til dels, at beskæringsmetoderne har udvist stor afhængighed af et 'velvalgt' weight decay. Der er ikke opnået kvantitativt gode resultater med nogen af metoderne, men OBD virker som den mest lovende af de tre.

Blandt de resultater, der nok mere har teoretisk interesse, er udvidelsen af layered RTRL til et arbitrært antal skjulte lag, *multi-layered RTRL*. Enhederne inddeles i lag, der opdateres i stil med lagene i et multi-layer feedforward netværk.

OBD samt OBS er udledt for tilfælde, hvor gradienten ikke kan negligeres, som for eksempel et utilstrækkeligt optrænet neuralt netværk. For OBD er udvidelsen meget simpel at implementere, mens de resulterende udtryk for OBS er temmelig komplicerede, hvilket nok vil begrænse anvendelsen.

Endelig er der redegjort for, i hvilke tilfælde det for et neuralt netværk kan betale sig at anvende en iterativ metode til matrixinvertering, der især benyttes indenfor system identifikation.

Der er mange emner, som vil være relevante at beskæftige sig med i fremtidige arbejder om fuldt rekursive netværk. Et af de vigtigste, og samtidig mest udfordrende, forekommer at være indførelse af *adaptivt* weight decay. 'Velvalgt' weight decay har vist sig at have stor gavnlig indflydelse på både optræning samt beskæring. Det 'optimale' weight decay er afhængigt af netværkets arkitektur, hvorfor adaptiv bestemmelse vil være til stor hjælp under beskæring. Weight decayet bør være individuelt for hver vægt, da det i dette arbejde har vist sig, at i hvertfald to grupper af netværkets vægte kræver forskelligt weight decay.

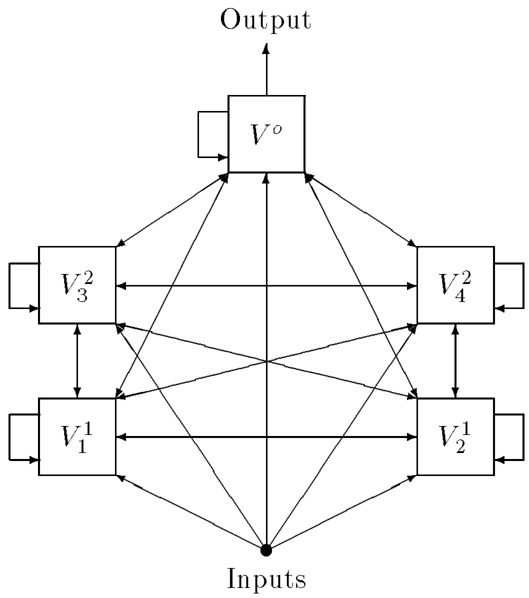
Der bør i fremtidige arbejder udføres mere omfattende forsøg med de forskellige beskæringsmetoder, end det er gjort her, for at give et mere vægtigt fundament for udpegelsen af den 'bedste' metode. Ligeledes bør det søges at give en kvalitativ analyse af, hvad der kan bevirke, at OBD samt OBS udpeger den 'forkerte' vægt til beskæring.

Af væsentlig betydning for beskæringens anvendelighed er metoder til udpegelse af de optimale netværk. Det vil i denne sammenhæng være særdeles relevant at udlede udtryk for Akaike's Final Prediction Error estimat gældende for feedback netværk anvendt til tidsserieprædiktion.

Appendiks A

Multi-layered RTRL

I afsnit 2.2.2 blev der udledt udtryk for opdatering samt optræning for en lag-delt fuldt forbundet feedback-arkitektur, der blev kaldt *layered RTRL*. Til forskel fra den oprindelige arkitektur introduceret af Williams&Zipser, hvor alle enheder tilhører samme lag, er layered RTRL inspireret fra sædvanlige feedforward netværk. Udledningen i afsnit 2.2.2 er begrænset til et enkelt lag skjulte enheder, hvilket vil være tilstrækkeligt ved mange anvendelser, som det også er tilfældet for almindelige feedforward netværk, se feks. [Weigend 90, Hertz 91, Svarer 93].



Figur A.1: Fuldt forbundet layered feedback netværk med to lag.

Der er dog ikke noget i vejen for at generalisere udtrykkene fra afsnit 2.2.2 til at omfatte multiple lag af skjulte fuldt forbundne feedback-enheder, og således opnå *multi-layered RTRL*. Denne udvidelse beskrives i det følgende for det generelle fuldt forbundne tilfælde, hvor alle enheder i netværket er forbundet med alle enheder, og alle enheder modtager de eksterne inputs. Et eksempel på et sådant net er givet i figur A.1, hvor der er to lag skjulte enheder med to enheder i hvert lag. Af overskuelighedsmæssige årsager er forbindelserne imellem enhederne angivet som dobbeltpile; disse skal således forstås som en seperat forbindelse i hver retning med hver sin vægt. Eftersom det afbildede netværk er fuldt forbundet, er det selvsagt ikke den indbyrdes placering af enhederne, der afgør, om netværket er multi-layered; lagdelingen fremgår udelukkende af den rækkefølge, hvori enhedernes output opdateres, som indekseringen af disse antyder. I udledningen nedenfor benyttes for enkelhedens skyld som tidligere kun en enkelt output-enhed, men udvidelsen til flere er ligefrem at foretage.

Antag, at netværket har n skjulte lag. Lad $\mathbf{x}(t)$ angive en vektor med de eksterne inputs til tiden t, og lad $\mathbf{V}(t)$ angive vektoren indeholdende samtlige skjulte enheders samt outputenhedens output til tiden t. Indeks for $\mathbf{x}(t)$ og $\mathbf{V}(t)$ tænkes arrangeret således, at mængden I angiver indeks for de eksterne inputs; U^h , $h = \{1 \dots n\}$ angiver indeks for enhederne i hvert af de n skjulte lag, og O angiver indeks for outputenheden. Inputvektoren til enhederne i det første skjulte lag benævnes $\mathbf{z}^1(t)$, og elementerne i denne defineres til tiden t som

$$z_k^1(t) = \begin{cases} x_k(t) & , \ k \in I \\ V_k^h(t-1) & , \ k \in U^h \ , \ h = \{1 \dots n\} \\ V^o(t-1) & , \ k \in O \end{cases} \tag{A.1}$$

Her er der for overskuelighedens skyld tilføjet superscript h til outputtet fra de skjulte enheder, der angiver, i hvilket lag den pågældende enhed er placeret. Når outputtet fra de skjulte enheder i det første lag er bestemt til tiden t, føres det frem og benyttes som input til det andet, og så fremdeles. Inputvektoren $\mathbf{z}^m(t)$ til de skjulte enheder i det m'te lag bliver således

$$z_k^m(t) = \begin{cases} x_k(t) & , \ k \in I \\ V_k^h(t) & , \ k \in U^h \ , \ h = \{1 \dots m-1\} \\ V_k^h(t-1) & , \ k \in U^h \ , \ h = \{m \dots n\} \\ V^o(t-1) & , \ k \in O \end{cases} \tag{A.2}$$

Endelig bliver inputvektoren $\mathbf{z}^o(t)$ til outputenheden

$$z_k^o(t) = \begin{cases} x_k(t) & , \ k \in I \\ V_k^h(t) & , \ k \in U^h \ , \ h = \{1 \dots n\} \\ V^o(t-1) & , \ k \in O \end{cases} \tag{A.3}$$

Outputtet fra enhederne i det første lag defineres nu som

$$\begin{aligned} V_k^1(t) &= f_k^1\left[s_k^1(t)\right] \\ &= f_k^1\left[\sum_{j \in I \cup U^1 \dots U^n \cup O} w_{kj} z_j^1(t)\right] \\ &= f_k^1\left[\sum_{j \in I} w_{kj} x_j(t) + \sum_{h=1}^{n} \sum_{j \in U^h} w_{kj} V_j^h(t-1) + w_{ko} V^o(t-1)\right] \end{aligned}$$

hvor $k \in U^1$, og outputtet fra de efterfølgende $n-1$ skjulte lag bliver

$$\begin{aligned}
V_k^m(t) &= f_k^m\left[s_k^m(t)\right] \\
&= f_k^m\left[\sum_{j \in I \cup U^1 \ldots U^n \cup O} w_{kj} z_j^m(t)\right] \\
&= f_k^m\left[\sum_{j \in I} w_{kj} x_j(t) + \sum_{h=1}^{m-1} \sum_{j \in U^h} w_{kj} V_j^h(t)\right. \\
&\qquad \left. + \sum_{h=m}^{n} \sum_{j \in U^h} w_{kj} V_j^h(t-1) + w_{ko} V^o(t-1)\right]
\end{aligned}$$

hvor $k \in U^m$, $m \in \{2, \ldots, n\}$. Endelig bliver outputtet fra outputenheden

$$\begin{aligned}
V^o(t) &= f^o\left[s^o(t)\right] \\
&= f^o\left[\sum_{j \in I \cup U^1 \ldots U^n \cup O} w_{oj} z_j^o(t)\right] \\
&= f^o\left[\sum_{j \in I} w_{oj} x_j(t) + \sum_{h=1}^{n} \sum_{j \in U^h} w_{oj} V_j^h(t) + w_{oo} V^o(t-1)\right]
\end{aligned}$$

Til optimering af den kvadratiske cost funktion

$$E(\mathbf{w}) = \frac{1}{2} \sum_{t=1}^{T} [e(t)]^2 \quad , \quad e(t) = x(t) - V^o(t)$$

er der brug for de afledede af outputtet $V^o(t)$, der beregnes til

$$\frac{\partial V^o(t)}{\partial w_{ij}} = \frac{\partial f^o\left[s^o(t)\right]}{\partial s^o(t)} \cdot \frac{\partial s^o(t)}{\partial w_{ij}}$$

hvor

$$\begin{aligned}
\frac{\partial s^o(t)}{\partial w_{ij}} &= \delta_{oi} z_j^o(t) + \sum_{j' \in U^1, \ldots, U^n, O} w_{oj'} \frac{\partial z_{j'}^o(t)}{\partial w_{ij}} \\
&= \delta_{oi} z_j^o(t) + \sum_{h=1}^{n} \sum_{j' \in U^h} w_{oj'} \frac{\partial V_{j'}^h(t)}{\partial w_{ij}} + w_{oo} \frac{\partial V^o(t-1)}{\partial w_{ij}}
\end{aligned}$$

I dette udtryk indgår de afledede af enhederne i de skjulte lag. De afledede af enhederne i det første skjulte lag beregnes til

$$\frac{\partial V_k^1(t)}{\partial w_{ij}} = \frac{\partial f_k^1\left[s_k^1(t)\right]}{\partial s_k^1(t)} \cdot \frac{\partial s_k^1(t)}{\partial w_{ij}}$$

hvor

$$\begin{aligned}
\frac{\partial s_k^1(t)}{\partial w_{ij}} &= \delta_{ki} z_j^1(t) + \sum_{j' \in U^1, \ldots, U^n, O} w_{kj'} \frac{\partial z_{j'}^1(t)}{\partial w_{ij}} \\
&= \delta_{ki} z_j^1(t) + \sum_{h=1}^{n} \sum_{j' \in U^h} w_{kj'} \frac{\partial V_{j'}^h(t-1)}{\partial w_{ij}} + w_{ko} \frac{\partial V^o(t-1)}{\partial w_{ij}}
\end{aligned}$$

hvor $k \in U^1$. De afledede af enhederne i de resterende $n-1$ skjulte lag beregnes til

$$\frac{\partial V_k^m(t)}{\partial w_{ij}} = \frac{\partial f_k^m\left[s_k^m(t)\right]}{\partial s_k^m(t)} \cdot \frac{\partial s_k^m(t)}{\partial w_{ij}}$$

hvor

$$\begin{aligned}\frac{\partial s_k^m(t)}{\partial w_{ij}} &= \delta_{ki} z_j^m(t) + \sum_{j' \in U^1,\ldots,U^n,O} w_{kj'} \frac{\partial z_{j'}^m(t)}{\partial w_{ij}} \\ &= \delta_{ki} z_j^m(t) + \sum_{h=1}^{m-1} \sum_{j' \in U^h} w_{kj'} \frac{\partial V_{j'}^h(t)}{\partial w_{ij}} \\ &\quad + \sum_{h=m}^{n} \sum_{j' \in U^h} w_{kj'} \frac{\partial V_{j'}^h(t-1)}{\partial w_{ij}} + w_{ko} \frac{\partial V^o(t-1)}{\partial w_{ij}}\end{aligned}$$

hvor $k \in U^m$, $m \in \{2, \ldots, n\}$. Som det er tilfældet for beregning af netværkets output benytter man således også ved opdatering af de afledede til tiden t i det m'te skjulte lag de netop beregnede værdier af de afledede af enhederne i de underliggende $m-1$ lag.

Som tidligere sættes alle enheders output samt deres afledede til nul til tiden $t = 0$. Måtte man have behov for den fulde Hessian matrix, kan denne udledes fuldstændigt analogt til fremgangsmåden for alm. layered RTRL i afsnit 2.2.2; dette gøres ikke her.

Udledningerne ovenfor giver mulighed for fuldstændigt forbundne netværk. Hvis der imidlertid ønskes en lidt simplere arkitektur, kan dette opnås ved blot at eliminere de 'uønskede' vægte. En sådan simplificering kunne eksempelvis bestå i, at det kun var nabolag, der var (fuldt) forbundet med hinanden. En sådan arkitektur vil give højere grad af adskillelse imellem de enkelte lag skjulte enheder, og således udvise større arkitektonisk lighed med multi-layer feedforward netværk.

Appendiks B

Udvidet OBD & OBS

Ved utilstrækkelig optræning af et neuralt netværk vil det ikke altid kunne antages, at gradienten for cost funktionen er nul. I dette appendiks udledes udtryk for saliencies for OBD samt OBS for en regulariseret cost funktion, i hvilke det *ikke* antages, at gradienten er nul. For nærmere forklaring omkring de to metoder henvises til kapitel 4.

B.1 Optimal Brain Damage med gradient

Cost funktionen antages at have formen

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{D}\mathbf{w} \tag{B.1}$$

hvor $\mathbf{D}$ er regulariseringsmatricen. For simpelt weight decay er $\mathbf{D}$ proportional med enhedsmatricen, $\mathbf{D} = \alpha\mathbf{I}$. Saliency udledes nu som følger:

Lad det lokale ekstremum for $C(\mathbf{w})$ være $\mathbf{w}_0$. I dette punkt approksimeres den oprindelige cost funktion $E(\mathbf{w})$ med Taylorudviklingen

$$E(\mathbf{w}_0 + \delta\mathbf{w}) = E(\mathbf{w}_0) + \delta\mathbf{w}^{\mathrm{T}}\nabla E(\mathbf{w}_0) + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}\mathbf{A}\delta\mathbf{w} \tag{B.2}$$

hvor $\delta\mathbf{w}$ angiver ændringen af vektoren $\mathbf{w}_0$, og $\mathbf{A}$ angiver Hessianmatricen for E i $\mathbf{w}_0$. Hvis gradienten $\mathbf{g}_0 = \nabla C(\mathbf{w}_0)$ for den udvidede cost funktion $C(\mathbf{w})$ i punktet $\mathbf{w}_0$ er forskellig fra nul, haves

$$\begin{gathered} \mathbf{g}_0 = \nabla E(\mathbf{w}_0) + \mathbf{D}\mathbf{w}_0 \\ \Updownarrow \\ \nabla E(\mathbf{w}_0) = \mathbf{g}_0 - \mathbf{D}\mathbf{w}_0 \end{gathered} \tag{B.3}$$

Dette udtryk indsættes i (B.2), hvilket giver

$$\begin{aligned} \delta E &= E(\mathbf{w}_0 + \delta\mathbf{w}) - E(\mathbf{w}_0) \\ &= \delta\mathbf{w}^{\mathrm{T}}(\mathbf{g}_0 - \mathbf{D}\mathbf{w}_0) + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}\mathbf{A}\delta\mathbf{w} \end{aligned} \tag{B.4}$$

Hvis pertubationen $\delta\mathbf{w}$ har karakter af nulstilling af den j'te vægt i $\mathbf{w}_0$, $\delta\mathbf{w} = -(\mathbf{e}_j^{\mathrm{T}}\mathbf{w}_0)\mathbf{e}_j$ hvor $\mathbf{e}_j$ er den j'te enhedsvektor, og regulariseringen har form af simpelt weight decay, kan den resulterende ændring i den oprindelige cost funktion E, saliency, skrives som

$$\begin{aligned}\delta E_j &= -((\mathbf{e}_j^{\mathrm{T}}\mathbf{w}_0)\mathbf{e}_j)^{\mathrm{T}}\mathbf{g} + \alpha((\mathbf{e}_j^{\mathrm{T}}\mathbf{w}_0)\mathbf{e}_j)^{\mathrm{T}}\mathbf{w}_0 + \frac{1}{2}(\mathbf{e}_j^{\mathrm{T}}\mathbf{w}_0)\mathbf{e}_j^{\mathrm{T}}\mathbf{A}(\mathbf{e}_j^{\mathrm{T}}\mathbf{w}_0)\mathbf{e}_j \\ &= -w_{0,j}\cdot g_{0,j} + \left(\alpha + \frac{1}{2}\mathbf{A}_{jj}\right) w_{0,j}^2 \end{aligned} \tag{B.5}$$

hvor $w_{0,j}$ er det j'te element i $\mathbf{w}_0$, $g_{0,j}$ er det j'te element i $\mathbf{g}_0$, og $\mathbf{A}_{jj}$ er det j'te diagonalelement i $\mathbf{A}$. Som det er tilfældet uden gradient-leddet, er de således definerede saliencies additive under de forudsættende antagelser, der er beskrevet i afsnit 4.1.

B.2 Optimal Brain Surgeon med gradient

I dette afsnit udledes udtrykkene for OBS for en *regulariseret* cost funktion, herunder en cost funktion udvidet med weight decay, når det *ikke* antages, at gradienten kan negligeres. Dette har selvsagt konsekvenser for den genoptræning, som OBS indeholder; når det nye optimum flyttes, får dette imidlertid også konsekvenser for den estimerede saliency, som det vil fremgå af udledningen nedenfor.

Lad den oprindelige cost funktion være $E(\mathbf{w})$, hvor $\mathbf{w}$ er vægtvektoren. Den udvidede cost funktion til regulariseret optræning er

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{D}\mathbf{w} \tag{B.6}$$

hvor $\mathbf{D}$ er regulariseringsmatricen.

Lad det lokale ekstremum for $C(\mathbf{w})$ være $\mathbf{w}_0$. Cost funktionen approksimeres med Taylorudviklingen omkring $\mathbf{w}_0$

$$C(\mathbf{w}) \approx C(\mathbf{w}_0) + \delta\mathbf{w}^{\mathrm{T}}\mathbf{g} + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}(\mathbf{A}+\mathbf{D})\delta\mathbf{w} \tag{B.7}$$

hvor $\mathbf{w} = \mathbf{w}_0 + \delta\mathbf{w}$, og $\mathbf{g}$ er gradientvektoren i $\mathbf{w}_0$, $\nabla C(\mathbf{w}_0)$. Det antages, at afledede af orden større end to er uden betydning. $\mathbf{A}$ er matricen med de anden afledede af $E(\mathbf{w})$ i punktet $\mathbf{w}_0$; i det efterfølgende benyttes Hessian matricen for $C(\mathbf{w})$, $\mathbf{H} = \mathbf{A} + \mathbf{D}$.

Herefter nulstilles den j'te vægt, hvilket kan udtrykkes som

$$\delta\mathbf{w}_j^{\mathrm{T}}\mathbf{e}_j + \mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j = 0 \;\Leftrightarrow\; \delta\mathbf{w}_j^{\mathrm{T}}\mathbf{e}_j = -\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j \tag{B.8}$$

hvor $\mathbf{e}_j$ er den j'te enhedsvektor. Udtrykket (B.7) skal nu optimeres under lighedsbibetingelsen (B.8), og til dette formål indføres Lagrange funktionen

$$\tilde{C}_j(\mathbf{w}) = C(\mathbf{w}) + \gamma_j(\delta\mathbf{w}_j + \mathbf{w}_0)^{\mathrm{T}}\mathbf{e}_j \tag{B.9}$$

hvor $C(\mathbf{w})$ er approksimationen (B.7). Koefficienten γ_j er en Lagrange multiplikator, og $\delta\mathbf{w}_j$ samt γ_j skal bestemmes således, at (B.9) er optimalt, samtidig med at bibetingelsen (B.8) er opfyldt.

Først bestemmes $\delta\mathbf{w}_j$, således at der opnås et ekstremum for (B.9). Dette gøres ved at sætte den afledede af udtrykket lig nul:

$$\begin{aligned} &\tfrac{\partial C(\mathbf{w})}{\partial(\delta\mathbf{w}_j)} + \gamma_j\mathbf{e}_j = \mathbf{0} \\ \Updownarrow \\ &\mathbf{g} + \mathbf{H}\delta\mathbf{w}_j + \gamma_j\mathbf{e}_j = \mathbf{0} \\ \Updownarrow \\ &\delta\mathbf{w}_j = -\mathbf{H}^{-1}(\mathbf{g} + \gamma_j\mathbf{e}_j) \end{aligned} \tag{B.10}$$

Til at bestemme det tilhørende γ_j benyttes (B.10) indsat i (B.8):

$$\begin{aligned} &(\mathbf{g} + \gamma_j\mathbf{e}_j)^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j = \mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j \\ \Updownarrow \\ &\gamma_j = \tfrac{\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j - \mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j} \end{aligned} \tag{B.11}$$

'Genoptræningen' efter nulstilling af den j'te vægt indebærer således en ændring af vægt-vektoren, der bestemmes ved indsættelse af (B.11) i (B.10):

$$\delta\mathbf{w}_j = -\mathbf{H}^{-1}\mathbf{g} + \left(\frac{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{g}}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j} - \frac{\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j}\right)\mathbf{H}^{-1}\mathbf{e}_j \tag{B.12}$$

Med dette udtryk er det nu muligt at estimere ændringen i den *oprindelige* cost funktion, dvs. saliency for $E(\mathbf{w})$. Funktionen Taylorudvikles som $C(\mathbf{w})$ omkring $\mathbf{w}_0$, hvilket giver

$$E_j(\mathbf{w}) = E(\mathbf{w}_0) + \delta\mathbf{w}_j^{\mathrm{T}}\nabla E(\mathbf{w}_0) + \frac{1}{2}\delta\mathbf{w}_j^{\mathrm{T}}\mathbf{A}\delta\mathbf{w}_j \tag{B.13}$$

Førsteordensleddet i (B.13) kan bestemmes udfra den regulariserede cost funktion $C(\mathbf{w})$, idet

$$\begin{aligned} &\nabla C(\mathbf{w}_0) = \mathbf{g} = \nabla E(\mathbf{w}_0) + \mathbf{D}\mathbf{w}_0 \\ \Updownarrow \\ &\nabla E(\mathbf{w}_0) = \mathbf{g} - \mathbf{D}\mathbf{w}_0 \end{aligned} \tag{B.14}$$

Dette udtryk indsættes i (B.13), hvilket giver

$$\delta E_j(\mathbf{w}) = \delta\mathbf{w}_j^{\mathrm{T}}(\mathbf{g} - \mathbf{D}\mathbf{w}_0) + \frac{1}{2}\delta\mathbf{w}_j^{\mathrm{T}}\mathbf{A}\delta\mathbf{w}_j \tag{B.15}$$

hvor $\delta E_j(\mathbf{w}) = E_j(\mathbf{w}) - E(\mathbf{w}_0)$. Endelig indsættes ændringen for genoptræningen (B.10) i (B.15), og udtrykket, der approksimerer saliency ved nulstilling af den j'te vægt, bliver

$$\begin{aligned} &\delta E_j(\mathbf{w}) = \\ &\quad(\mathbf{g} + \gamma_j\mathbf{e}_j)^{\mathrm{T}}\mathbf{H}^{-1}(\mathbf{D}\mathbf{w}_0 - \mathbf{g}) + \tfrac{1}{2}(\mathbf{g} + \gamma_j\mathbf{e}_j)^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{A}\mathbf{H}^{-1}(\mathbf{g} + \gamma_j\mathbf{e}_j) \end{aligned} \tag{B.16}$$

Hvis der ikke benyttes regularisering, svarende til at $\mathbf{D} = \mathbf{0}$, reduceres udtrykket (B.16) efter nogen regning til:

$$
\begin{aligned}
\delta E_j(\mathbf{w}) &= -\frac{1}{2}\mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g} + \frac{1}{2}\gamma_j^2\mathbf{e}_j^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{e}_j \\
&= -\frac{1}{2}\mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g} + \frac{1}{2}\frac{(\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j - \mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{e}_j)^2}{\mathbf{e}_j^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{e}_j}
\end{aligned}
\tag{B.17}
$$

Hvis regulariseringen har karakter af simpelt weight decay, således at $\mathbf{H} = \mathbf{A} + \alpha\mathbf{I}$, kan udtrykket (B.16) efter en del regning 'reduceres' til

$$
\begin{aligned}
\delta E_j(\mathbf{w}) &= \alpha\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0 + \alpha\,\gamma_j\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0 - \mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{g} - \gamma_j\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{g} \\
&\quad + \frac{1}{2}\left(\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{A}\mathbf{H}^{-1}\mathbf{g} + \gamma_j^2\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{A}\mathbf{H}^{-1}\mathbf{g}\right) \\
&\quad + \gamma_j\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{A}\mathbf{H}^{-1}\mathbf{g} \\
&= \alpha\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0 - \frac{1}{2}\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{g} - \frac{\alpha}{2}\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-2}\mathbf{g} \\
&\quad + \alpha\left(\frac{\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j - \mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j}\right)\left(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{w}_0 - \mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-2}\mathbf{g}\right) \\
&\quad + \frac{1}{2}\left(\frac{\mathbf{w}_0^{\mathrm{T}}\mathbf{e}_j - \mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j}{\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j}\right)^2\left(\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{e}_j - \alpha\mathbf{e}_j^{\mathrm{T}}\mathbf{H}^{-2}\mathbf{e}_j\right)
\end{aligned}
\tag{B.18}
$$

Hvis gradienten $\mathbf{g} = \mathbf{0}$, ses ovenstående udtryk som forventet at reducere til det tilsvarende resultat (4.21) udledt *uden* gradienten. Udtrykket (B.19) er en del mere kompliceret at beregne fremfor (4.21), og eftersom den kvadratiske approksimation ofte er *meget* lokal, vil 'gevinsten' ved at benytte (B.19) nok være marginal; men der er hermed åbnet op for muligheden.

Appendiks C

Iterativ beregning af invers Hessian

I dette appendiks analyseres beregningsomkostningerne for en standardmetode til iterativ beregning af den inverse Hessian matrix, der bla. anvendes indenfor området *system identifikation* i forbindelse med *online* beregning af den inverse Hessian, se [Ljung 87]. Metoden bliver også anvendt i [Hassibi 93] i forbindelse med *offline* beregning af den inverse Hessian for et neuralt netværk, hvor den anbefales fremfor at beregne Hessian matricen og efterfølgende invertere denne.

Metoden forudsætter, at Hessian matricen kan bringes på følgende form til iterativ beregning:

$$\mathbf{H}_k = \mathbf{H}_{k-1} + u_k \cdot \mathbf{x}_k \mathbf{x}_k^T \quad , \quad k = 1 \ldots p \quad , \quad \mathbf{H}_0 = \mathbf{0} \tag{C.1}$$

hvor p angiver antallet af træningseksempler, og u_k er en konstant. Hvis $\mathbf{H}$ angiver Gauss-Newton approksimationen til Hessian matricen, vil $\mathbf{x}_k$ være gradienten for eksempel k.

Ved hjælp af ovenstående iterative udtryk til beregning af $\mathbf{H}$ kan $\mathbf{H}^{-1}$ nu beregnes under anvendelse af et standard-udtryk til matrixinvertering:

$$(\mathbf{A} + \mathbf{B} \cdot \mathbf{C} \cdot \mathbf{D})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \cdot \mathbf{B} \cdot (\mathbf{C}^{-1} + \mathbf{D} \cdot \mathbf{A}^{-1} \cdot \mathbf{B})^{-1} \cdot \mathbf{D} \cdot \mathbf{A}^{-1} \tag{C.2}$$

Ved passende substitution fås en iterativ algoritme til beregning af $\mathbf{H}^{-1}$, se [Hassibi 93]:

$$\mathbf{H}_k^{-1} = \mathbf{H}_{k-1}^{-1} - \frac{\mathbf{H}_{k-1}^{-1}\, \mathbf{x}_k\, \mathbf{x}_k^T\, \mathbf{H}_{k-1}^{-1}}{u_k^{-1} + \mathbf{x}_k^T\, \mathbf{H}_{k-1}^{-1}\, \mathbf{x}_k} \quad , \quad k = 1 \ldots p \quad , \quad \mathbf{H}_0^{-1} = \alpha^{-1}\mathbf{I}$$

I nogle tilfælde kan det forekomme, at u_k er nul. Dette var tilfældet i [MWP 94], hvor u_k indeholdt den afledede af $\tanh(x)$, der er $1 - \tanh^2(x)$. For numerisk 'store' værdier af x kan den afledede i praksis blive nul grundet begrænset maskinnøjagtighed. I sådanne tilfælde vil det være mere hensigtsmæssigt at benytte omskrivningen

$$\mathbf{H}_k^{-1} = \mathbf{H}_{k-1}^{-1} - \frac{\mathbf{H}_{k-1}^{-1}\, \mathbf{x}_k\, \mathbf{x}_k^T\, \mathbf{H}_{k-1}^{-1}\, u_k}{1 + \mathbf{x}_k^T\, \mathbf{H}_{k-1}^{-1}\, \mathbf{x}_k\, u_k} \tag{C.3}$$

Det bemærkes, at den iterative metode til beregning af $\mathbf{H}^{-1}$ behøver en startværdi $\mathbf{H}_0^{-1}$, her angivet som $\alpha^{-1}\mathbf{I}$, hvor α er en *lille* værdi. Denne startværdi vil naturligvis introducere en unøjagtighed.

Hvis cost funktionen imidlertid udvides med et weight decay, $\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{D}\mathbf{w}$, udgøres startværdien naturligt af $\mathbf{D}^{-1}$, den anden afledede af regulariseringsleddet. Hvis der er tale om simpelt weight decay, er $\mathbf{D} = \alpha\mathbf{I}$, og heraf fås $\mathbf{D}^{-1} = \alpha^{-1}\mathbf{I}$.

I det følgende udledes beregningsomkostningerne for udtrykket (C.3) som det også blev foretaget i [MWP 94]. Ved omkostninger forstås antallet af multiplikationer samt divisioner, der sædvanligvis tager væsentlig længere tid at udføre end addition og subtraktion. I [MWP 94] blev der dog ikke taget højde for, at de indgående matricer er symmetriske, ligesom det heller ikke blev udnyttet, at $(\mathbf{H}^{-1}\mathbf{x})^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}}\mathbf{H}^{-1}$. Dette vil blive udnyttet i følgende udledning, der foretages for den mest hensigtsmæssige måde at beregne $\mathbf{H}_k^{-1}$ på. Hvis n angiver antallet af vægte, bliver omkostningerne for hvert træningseksempel

$$\begin{array}{lcl}
\mathbf{H}^{-1}\,\mathbf{x}\;,\;\mathbf{x}^{\mathrm{T}}\,\mathbf{H}^{-1} & = & n^2 \\
u_k\,(\mathbf{H}^{-1}\,\mathbf{x}) & = & n \\
\mathbf{x}^{\mathrm{T}}\,(\mathbf{H}^{-1}\,\mathbf{x}\,u_k) & = & n \\
(\mathbf{x}^{\mathrm{T}}\,\mathbf{H}^{-1})\,/\,\textit{nævner} & = & n \\
(u_k\,\mathbf{H}^{-1}\,\mathbf{x})\,((\mathbf{x}^{\mathrm{T}}\,\mathbf{H}^{-1})\,/\,\textit{nævner}) & = & n(n+1)/2
\end{array}$$

Sammenlagt koster den iterative beregning af den inverse Hessian Matrix $\mathbf{H}^{-1}$ $p\left(\frac{3}{2}n^2 + \frac{7}{2}n\right)$, hvor p igen er antallet af træningseksempler.

Som alternativ til den iterative beregning af $\mathbf{H}^{-1}$ kan man beregne $\mathbf{H}$ efter (C.1) på sædvanlig vis, hvilket med udnyttelse af symmetri 'koster' $p\left(\frac{1}{2}n(n+1) + n\right) = p\left(\frac{1}{2}n^2 + \frac{3}{2}n\right)$. Hertil kommer så en matrixinvertering med feks. Gauss-Jordan elimination, for hvilken beregningsarbejdet i [Press 92] angives til n^3.

Matrixinverteringen er årsag til, at den iterative metode fremhæves i [Hassibi 93] idet sidstnævnte 'koster' i størrelsesordnen $O(n^2)$, mens sædvanlig matrixinvertering koster $O(n^3)$. Imidlertid benytter man ved optræning af neurale netværk ofte mange eksempler pr. vægt, for at opnå god generaliseringsevne, se [Hertz 91]. Det er således ikke n, der er den dominerende faktor for beregningsomkostningerne, men p. Hvis den iterative metode til beregning af den inverse Hessian matrix $\mathbf{H}^{-1}$ skal være at foretrække fremfor beregning af $\mathbf{H}$ med efterfølgende invertering, skal der gælde, at

$$\begin{array}{c}
n^3 + p(\frac{1}{2}n^2 + \frac{3}{2}n) > p(\frac{3}{2}n^2 + \frac{7}{2}n) \\
\Updownarrow \\
n > \frac{1}{2}p\left(1 + \sqrt{1 + 8/p}\right) \approx p
\end{array}$$

hvilket ved de fleste anvendelser langt fra vil være opfyldt. Det kan således *ikke* anbefales at benytte den iterative metode til *off line* beregning af den inverse Hessian matrix for neurale netværk.

Appendiks D

Konditionstal for matricer

I dette appendiks defineres konditionstallet for en matrix, ligesom det analyseres, hvilken indflydelse det har på løsningen af et lineært ligningssystem, hvor den indgående matrix samt højreside er behæftede med fejl, der feks. skyldes, at der ved beregninger på computer benyttes endelig præcision.

Ved optimering med Gauss-Newtons metode eller Newtons metode bestemmes søgeretningen ved løsning af et lineært ligningssystem. I dette ligningssystem indgår (en approksimation til) Hessianmatricen samt den negative gradient for cost funktionen som højreside. Både gradienten samt matricen beregnes iterativt som sædvanligt for neurale netværk, ved summering af hhv. Hessianen og gradienten for hvert enkelt datapunkt. Begge beregnes med endelig præcision på en computer, og er derfor behæftede med (små) fejl, som det altid vil gælde, når der regnes med endelig præcision.

Disse fejl er for *rekursive netværk* relativt *større* end de tilsvarende for sædvanlige feedforward netværk. Hvor feks. gradienten for et feedforward netværk består af en sum af *uafhængige* bidrag, beregnes gradienten for et rekursivt netværk i en given iteration til dels udfra resultatet fra iterationen før, som det fremgår af udledningen i afsnit 2.2.2. Eftersom resultatet fra hver iteration således hele tiden benyttes ved beregningerne i den næste, vil *den akkumulerede afrundingsfejl* (se feks. [Nielsen 91]) ved beregning af gradient samt Hessianmatrix for rekursive netværk være mere betydende, end det er tilfældet for feedforward netværk.

Ligningssystemet til bestemmelse af en søgeretning skrives som

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{D.1}$$

hvor $\mathbf{A}$ skal opfattes som den *eksakte* Hessianmatrix, dvs. *uden* afrundingsfejl. Tilsvarende opfattes $\mathbf{b}$ som den negative gradient uden afrundingsfejl. Det antages, at $\mathbf{A}$ ikke er singulær, og at der således eksisterer en eksakt løsning $\mathbf{x}_*$ til (D.1).

I det følgende vurderes den relative ændring i $\mathbf{x}_*$ for en given ændring i $\mathbf{A}$ eller $\mathbf{b}$, der feks. skyldes endelig præcision, som beskrevet ovenfor; udledningerne stammer fra [Dennis 83]. Hvis højresiden $\mathbf{b}$ ændres med $\Delta\mathbf{b}$, kan den nye løsning til (D.1) skrives som $\mathbf{x}_* + \Delta\mathbf{x}$, hvor

$$\mathbf{A}(\,\mathbf{x}_* + \Delta\mathbf{x}\,) = \mathbf{b} + \Delta\mathbf{b} = \mathbf{A}\,\mathbf{x}_* + \Delta\mathbf{b}$$

hvoraf haves, at

$$\mathbf{A} \cdot \Delta\mathbf{x} = \Delta\mathbf{b}$$

For enhver vektornorm $\|\cdot\|$ og den tilsvarende *inducerede* matrixnorm (se feks. [Nielsen 72]) vil der gælde

$$\|\Delta\mathbf{x}\| \leq \|\mathbf{A}^{-1}\| \cdot \|\Delta\mathbf{b}\|$$

Fra $\mathbf{A}\,\mathbf{x}_* = \mathbf{b}$ haves tilsvarende

$$\frac{1}{\|\mathbf{x}_*\|} \leq \frac{\|\mathbf{A}\|}{\|\mathbf{b}\|}$$

Der kan nu gives en vurdering af den *relative* fejl på løsningen $\mathbf{x}_*$

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}_*\|} \leq \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|\,\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} \tag{D.2}$$

Tilsvarende vurderes ændringer $\Delta\mathbf{A}$ i venstresiden $\mathbf{A}$. Den nye løsning til (D.1) skrives som $\hat{\mathbf{x}} = \mathbf{x}_* + \Delta\hat{\mathbf{x}}$, således at

$$(\mathbf{A} + \Delta\mathbf{A})\,(\mathbf{x}_* + \Delta\hat{\mathbf{x}}) = \mathbf{b}$$

hvilket giver

$$\begin{aligned} & \mathbf{A}\,\Delta\hat{\mathbf{x}} = -\Delta\mathbf{A} \cdot \hat{\mathbf{x}} \\ \Updownarrow & \\ & \|\Delta\hat{\mathbf{x}}\| \leq \|\mathbf{A}^{-1}\| \cdot \|\Delta\mathbf{A}\| \cdot \|\hat{\mathbf{x}}\| \\ \Updownarrow & \\ & \tfrac{\|\mathbf{A}\,\hat{\mathbf{x}}\|}{\|\hat{\mathbf{x}}\|} \leq \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|\,\tfrac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \end{aligned} \tag{D.3}$$

Udtrykkene (D.2) og (D.3) kan feks. give en vurdering af, hvor stor en indflydelse det vil have på den bestemte søgeretning, at Hessianmatricen samt gradienten er bestemt med endelig præcision, og dermed fejlbehæftede. I både (D.2) og (D.3) afgrænses den relative ændring i $\mathbf{x}$ af den relative ændring i data multipliceret med størrelsen $\|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$. Denne størrelse kaldes *konditionstallet* for $\mathbf{A}$, der angives som $\kappa_p(\mathbf{A})$. p er her 1, 2 eller ∞, alt efter om der benyttes 1-normen, 2-normen eller ∞-normen (se [Nielsen 72]).

Her skal anvendes 2-normen, der for en matrix er defineret som

$$\|\mathbf{A}\|_2 = \sqrt{\max \lambda(\mathbf{A}^{\mathrm{T}}\mathbf{A})}$$

dvs. som kvadratroden af den største egenværdi for $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. Idet de her betragtede matricer alle vil være reelle og symmetriske, vil dette udtryk være ensbetydende med

$$\|\mathbf{A}\|_2 = \max|\lambda(\mathbf{A})|$$

dvs. den numerisk største egenværdi for $\mathbf{A}$. Da der for $\mathbf{A}$ reel samt symmetrisk gælder

$$\|\mathbf{A}^{-1}\|_2 = (\min|\lambda(\mathbf{A})|)$$

kan det her benyttede konditionstal nu skrives

$$\kappa_2(\mathbf{A}) = \left|\frac{\lambda_{max}}{\lambda_{min}}\right| \tag{D.4}$$

hvor λ_{max} samt λ_{min} angiver hhv. største og mindste egenværdi for $\mathbf{A}$. For den konkrete anvendelse her vil $\kappa_2(\mathbf{A})$ angive forholdet mellem den største og den mindste krumning i den approksimation til cost funktionen, som udspændes af $\mathbf{A}$ ([Widrow 85]). Hvis den mindste krumning λ_{min} er nul, svarende til at approksimationen er konstant i en given retning, svarer dette til, at $\mathbf{A}$ er singulær. Konditionstallet for en singulær matrix kan derfor regnes som værende uendeligt ([Dennis 83]); derfor kan konditionstallet for $\mathbf{A}$ benyttes som målestok for, hvor 'tæt' $\mathbf{A}$ er på at være singulær.

Analysen ovenfor af de relative fejl på løsningen til (D.1), blev foretaget for fejl, der skyldes begrænset maskinnøjagtighed og heraf akkumulerede fejl ved beregning af de indgående $\mathbf{A}$ samt $\mathbf{b}$. Selve løsningen af ligningssystemet introducerer dog også en fejl. I [Dennis 83] oplyses det, at ved løsning af systemet $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ på en computer med endelig præcision, vil den opnåede løsning være den eksakte løsning til det forstyrrede system $(\mathbf{A} + \Delta\mathbf{A})\,\mathbf{x} = \mathbf{b}$, for hvilket den relative fejl angives i (D.3). Det angives yderligere, at $\|\Delta\mathbf{A}\|/\|\mathbf{A}\|$ kan begrænses til en konstant k gange maskinnøjagtigheden ϵ. Af (D.3) ses nu, at den relative fejl på løsningen, stammende fra selve løsningen af ligningssystemet, er begrænset af $\kappa(\mathbf{A}) \cdot k \cdot \epsilon$.

Ved løsning af et ligningssystem er løsningens pålidelighed således afhængig af konditionstallet. I [Dennis 83] oplyses det, at hvis $\kappa(\mathbf{A}) \geq \epsilon^{-1}$, vil løsningen være helt upålidelig. Yderligere angives som tommelfingerregel, at hvis $\kappa(\mathbf{A}) > \epsilon^{-\frac{1}{2}}$ er der *mulighed* for, at løsningen er så fejlbehæftet, at den er upålidelig, hvilket i det konkrete tilfælde her vil betyde, at søgeretningen for en optimeringsmetode er dårligt bestemt.

Det her anvendte tal-format er IEEE standarden for 64-bit floating point, hvilket giver maskinnøjagtigheden $\epsilon = 2^{-52} \approx 2.2204 \cdot 10^{-16}$. Ovenfor omtalte tommelfingerregel indebærer således, at den bestemte søgeretning *kan* være upålidelig, hvis

$$\kappa_2(\mathbf{A}) > 6.71 \cdot 10^7 \tag{D.5}$$

En måde at forbedre en matrix $\mathbf{A}$'s konditionstal på er at forøge de mindste egenværdier. Dette kan gøres ved at *regularisere* $\mathbf{A}$ (se feks. [Ljung 87]), dvs. ved at addere en (lille) størrelse proportional med enhedsmatricen, $\alpha\mathbf{I}$. Der gælder, at hvis $\mathbf{A}$ har egenværdier λ_j, vil $\mathbf{A} + \alpha\mathbf{I}$ have egenværdier $\lambda_j + \alpha$ ([Dennis 83]). Denne metode svarer netop til at udvide den kvadratiske cost funktion med et simpelt weight decay, som beskrevet i afsnit 1.1. Metodens garanterede succes er dog betinget af, at den benyttede **A** er positivt (semi-) definit, hvilket for reelle, symmetriske matricer er ensbetydende med, at alle egenværdier er større end eller lig med nul; i modsat fald kan konditionstallet risikeres at forværres. Dette er netop tilfældet med Gauss-Newtons metode til optimering, hvor den benyttede **A** altid er positivt definit (se afsnit 1.4.2).

Appendiks E

Yderligere tests med Gauss-Newton

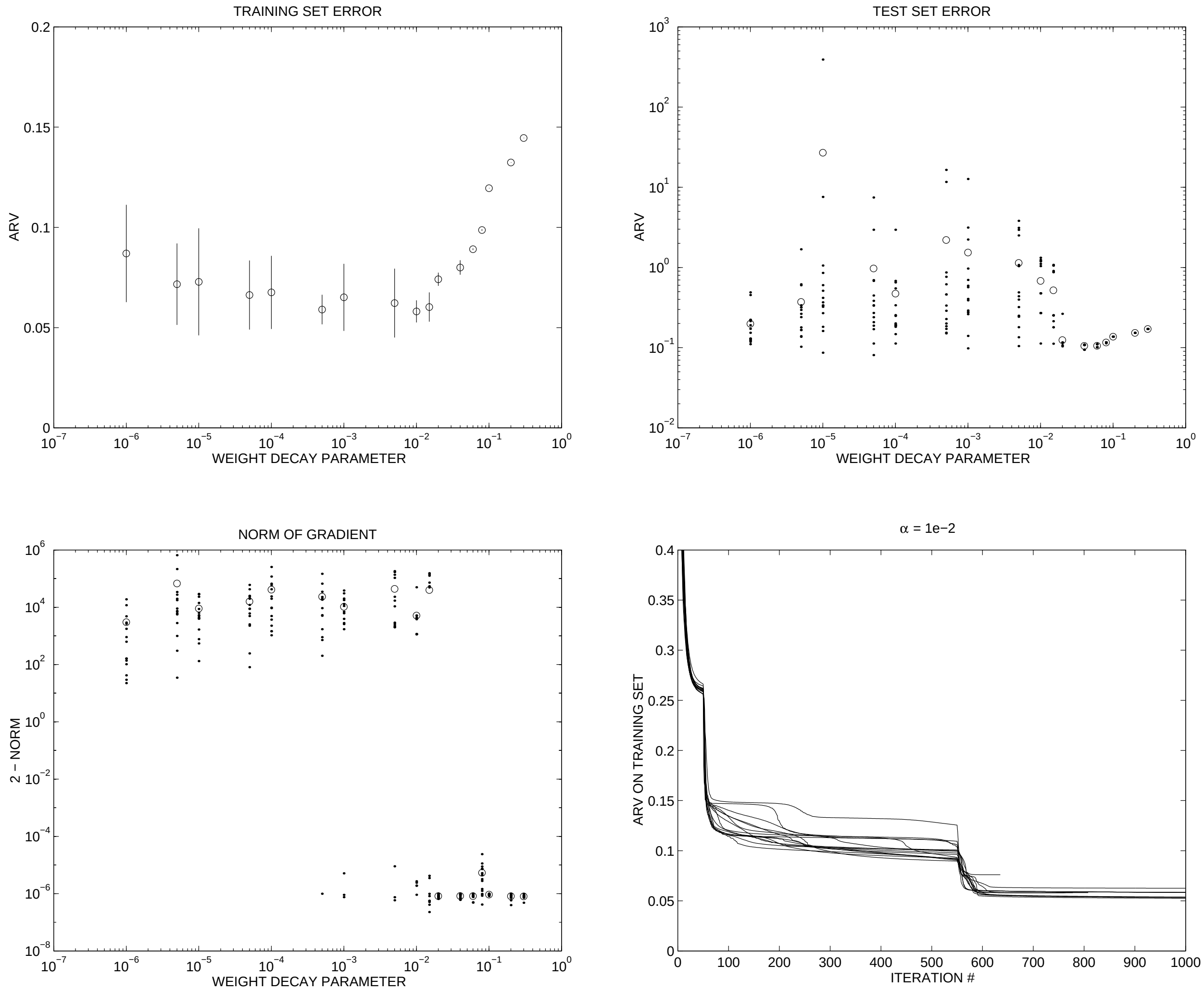
Dette appendiks indeholder forsøg med Gauss-Newtons metode, der supplerer dem beskrevet i afsnit 6.5.2. I afsnit E.1 illustreres det, at det ikke har nogen effekt på konditioneringen af Hessian-approksimationen endsige på niveauet af de opnåede middelfejl at kombinere de tre optimeringsmetoder gradient descent, pseudo Gauss-Newton samt Gauss-Newtons metode. I afsnit E.2 optrænes et layered RTRL netværk med flere eksterne inputs med Gauss-Newtons metode, og det konstateres, at der ikke er nogen forskel på, om outputenheden modtager disse eksterne inputs eller ej.

E.1 Kombineret optræning

I afsnit 6.5.2 beskrives en forsøgsrække på et layered RTRL netværk, der har fire skjulte enheder samt et enkelt eksternt input, men intet lokalt feedback i outputtet. Optræningen indledes med 50 iterationer gradient descent, hvorefter Gauss-Newtons metode benyttes. For forskellige weight decay parametre α optrænes der for hver værdi 15 netværk. For disse forsøg blev der i afsnit 6.5.2 konstateret problemer med store gradienter for cost funktionen i de bestemte 'minima', hvilket skyldes konditioneringsproblemer for Hessian-approksimationen under optræningen. De store gradienter er ikke hensigtsmæssige, hvis netværket efter optræning skal beskæres med OBD eller OBS, hvor det sædvanligvis antages, at gradienterne er negligible.

I dette appendiks undersøges det, hvorvidt det har nogen effekt at bringe cost funktionen tættere på et minimum, inden Gauss-Newtons metode tages i anvendelse, hvilket *muligvis* kan konditionere Hessianen bedre ([Dennis 83]). Der foretages derfor et lignende forsøg, hvor der indledes med 50 iterationer gradient descent, derefter 500 iterationer pseudo Gauss-Newton efterfulgt af maksimalt 450 iterationer Gauss-Newton. I figur E.1 gengives resultaterne af dette forsøg, og de er stort set identiske med figur 6.13.

Middel-træningsfejlene ligger på samme niveau som i figur 6.13, dog med lidt større standardafvigelse. Den mindste middelfejl er ARV_{train} =



Figur E.1: 4-1 layered RTRL net trænet med 50 batch, 500 pseudo Gauss-Newton efterfulgt af 450 iterationer Gauss-Newton for forskellige weight decay parametre α.

0.0581 ± 0.0055, der opnås for $\alpha = 10^{-2}$. For denne værdi af α er forløbet af træningsfejlene afbildet som funktion af iterationsantallet, og det ses tydeligt, at træningsfejlen falder markant ved påbegyndelse af en ny optimeringsmetode. Ved sammenligning med den tilsvarende afbildning i figur 6.13 ses det, at niveauet for træningsfejlen ved påbegyndelse af Gauss-Newtons metode sænkes fra ca. 0.25 til omkring 0.1. Nu er træningsfejlen er ikke proportional med den udvidede cost funktion, men giver dog en ide om niveauforskellen for denne.

Middel-testfejlene forløber for de større weight decays identisk med de tilsvarende i figur 6.13, og der er igen minimum for $\alpha = 0.04$, hvor $\langle ARV_{test} \rangle = 0.1054$

Vigtigst er dog konstateringen af, at de store gradienter stadig er tilstede. Skellet, hvor regulariseringen af cost funktionen 'løser' konditioneringsproblemet, ses at ligge omkring samme værdi som i figur 6.13, $\alpha = 0.02$. Konditioneringsproblemet kan således ikke mindskes med grundigere indledende optræning forud for anvendelse af Gauss-Newtons metode, men må afhjælpes ad anden vej.

E.2 Eksterne inputs til outputtet

I modsætning til sædvanlige feedforward netværk åbner definitionen af fuldt forbundne feedback netværk op for muligheden for, at outputenheden modtager de eksterne inputs til netværket på lige fod med de skjulte enheder, se kapitel 2. I dette appendiks gengives to forsøgsrækker for et layered RTRL netværk med fire skjulte enheder samt fire eksterne inputs, uden feedback i outputenheden.

For den første forsøgsrække gælder, at outputenheden *modtager* de eksterne inputs, for den anden gælder, at outputenheden *ikke* modtager de fire eksterne inputs. Netværkene optrænes med 50 iterationer gradient descent efterfulgt af maksimalt 450 iterationer Gauss-Newton. For hver værdi af weight decay parameteren α er der udført 15 optræninger.

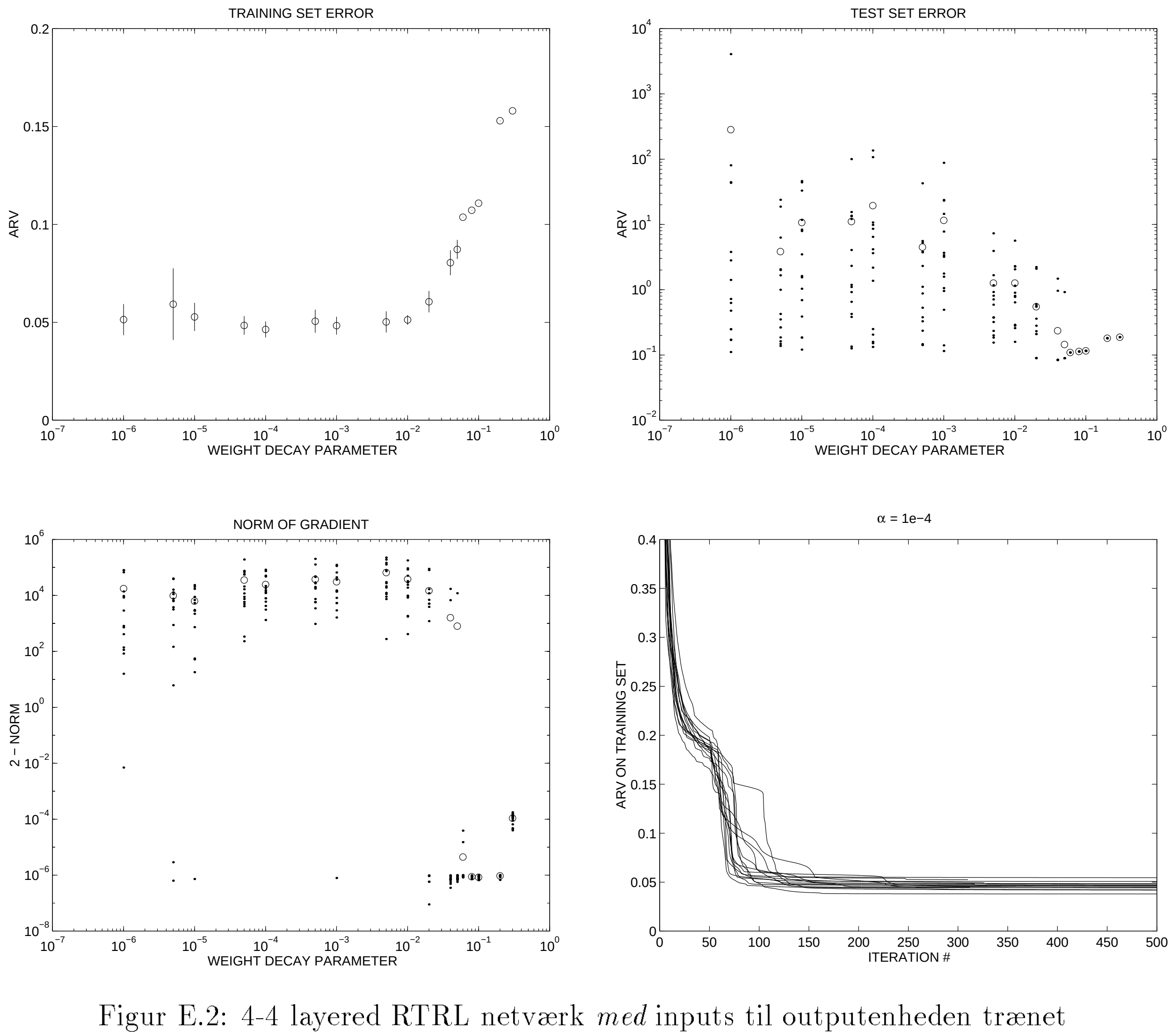
I figur E.2 vises resultaterne for netværket, hvor outputenheden modtager de fire eksterne inputs. Den laveste middeltræningsfejl $\langle ARV_{train} \rangle = 0.0464 \pm 0.0041$ opnås for $\alpha = 10^{-4}$. Dette er en del lavere end den laveste middeltræningsfejl observeret i figur 6.13, hvor der dog kun blev benyttet et enkelt eksternt input. Forskellen skyldes primært det øgede antal parametre, med eet input bruges 34 parametre, med fire bruges 49.

Den laveste middeltestfejl $\langle ARV_{test} \rangle = 0.1091$ opnås for $\alpha = 0.06$. Denne middelfejl er noget højere end den laveste, der opnås i figur 6.13, hvilket sandsynligvis skyldes det større nødvendige weight decay for at modvirke overparameterisering og heraf følgende overtræning.

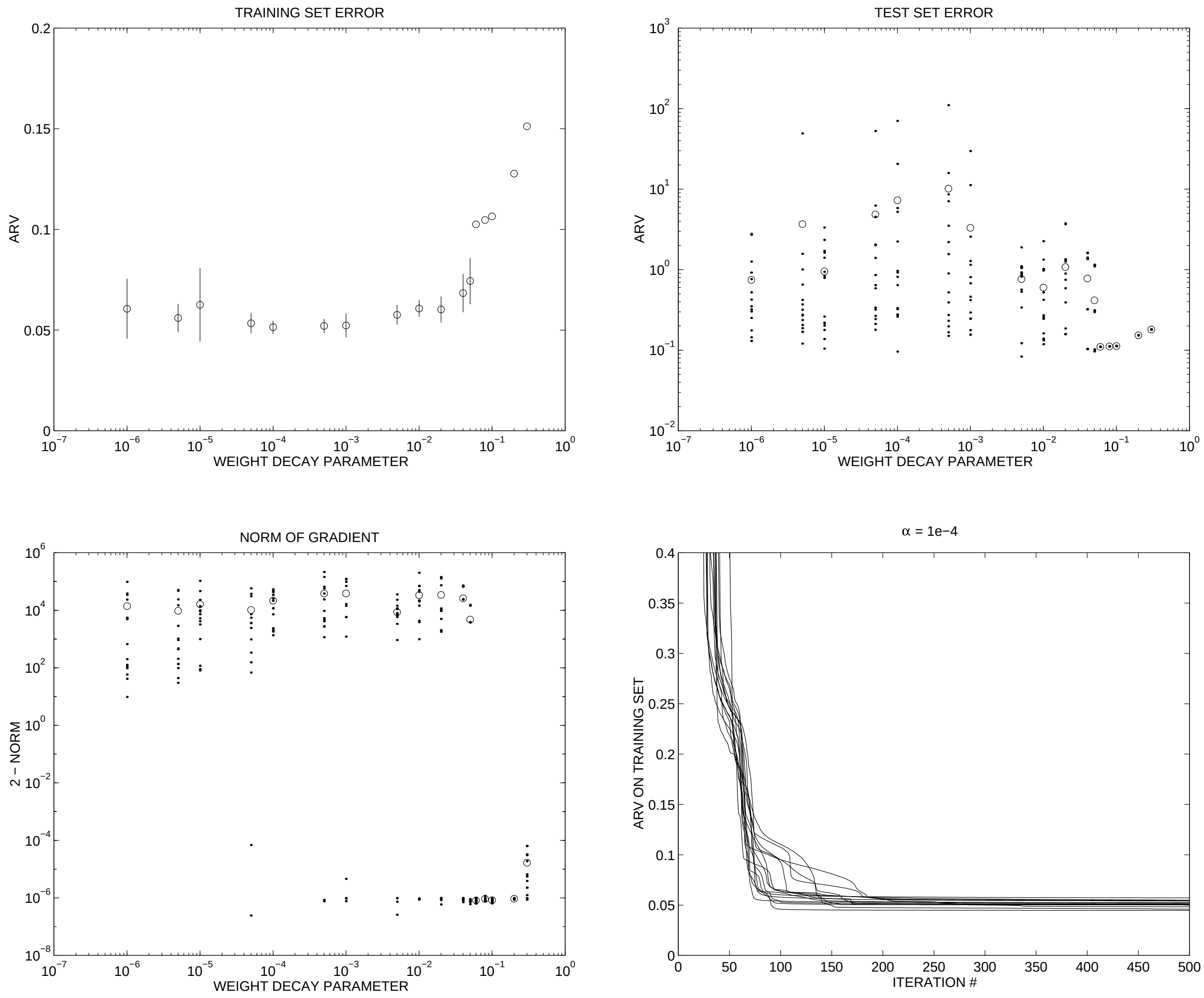
Konditioneringsproblemet illustreret ved de store gradienter i slutpunktet ses at kræve et α omkring 0.06 for at blive håndteret, hvilket er en del højere end for netværkene i figur 6.13, hvor den tilsvarende skillelinie ligger omkring $\alpha = 0.02$. Dette tages som udtryk for, at større netværk kræver større regularisering for at modvirke konditioneringsproblemet for Hessian-approksimationen ved Gauss-Newton optimering.

I figur E.3 på næste side gengives resultaterne fra forsøgsrækken, hvor outputenheden ikke modtager de fire eksterne inputs. Den laveste middeltræningsfejl opnås igen for $\alpha = 10^{-4}$, og er $ARV_{train} = 0.0515 \pm 0.0033$, en anelse højere, end når outputtet modtager de eksterne inputs.

Den laveste middeltræningsfejl $\langle ARV_{test} \rangle = 0.1101$ opnås ligeledes for samme α-værdi som ovenfor, $\alpha = 0.06$. Forøgelsen af middelfejlen er så lille som 0.002 eller 1.8 %, hvilket dårligt kan betegnes som værende signifikant. Heraf konkluderes det, at det ikke har nogen indflydelse på netværkets generaliseringsevne, om outputenheden modtager de eksterne inputs eller ej, hvorfor disse forbindelser kan udelades, som det er tilfældet for feedforward netværk.



Figur E.2: 4-4 layered RTRL netværk *med* inputs til outputenheden trænet med Gauss-Newtons metode for varierende weight decay parametre α.



Figur E.3: 4-4 layered RTRL netværk *uden* inputs til outputenheden trænet med Gauss-Newtons metode for varierende weight decay parametre α.

Appendiks F

Forsøg: lineær afhængighed

I dette appendiks beskrives et forsøg, der illustrerer den lineære afhængighed, der eksisterer vægtene imellem, når der for et layered RTRL netværk (samt for andre typer feedback netværk) benyttes feedback fra et *lineært* output til de skjulte enheder, som det beskrives i afsnit 6.8. Denne lineære afhængighed vil give problemer, når der anvendes Gauss-Newtons metode til optræningen, uden at der anvendes weight decay. Efterfølgende gentages forsøget med et lille weight decay for at illustrere effekten af dette.

Det benyttede netværk har samme arkitektur som det i figur 6.27 afbildede, med to skjulte enheder samt et enkelt eksternt input. Dog modtager outputtet her også det eksterne input. Optræningen af netværket indledes med 100 iterationer gradient descent, derefter 25 iterationer pseudo Gauss-Newton, hvorefter der benyttes Gauss-Newtons metode. Der benyttes ikke weight decay, og forløbet af optræningen illustreres i figur F.1. Det ses, at træningsfejlen, der er proportional med cost funktionen, forbliver konstant efter ganske få iterationer med Gauss-Newtons metode, mens gradientnormen samt konditionstallet for Hessian-approksimationen vokser kraftigt.

I figur F.2 vises et meshplot af cost funktionen som funktion af w_{11} og w_{12} samt de aktuelle vægte, lige inden Gauss-Newtons metode påbegyndes. Outputtets lokale feedback er sat til nul, idet det ikke benyttes. Det skal bemærkes, at w_{11}-aksen er 'vendt om' på grund af den vinkel, hvorfra fladen betragtes. Det aktuelle punkt er markeret med * i det underliggende konturplot. I retning mod (negativ w_{11}, positiv w_{12}) ses at starte en 'dal', og det er i denne retning, at w_{11} og w_{12} begynder at bevæge sig under den videre optræning med Gauss-Newtons metode.

Af figur F.3 – F.7 fremgår det, at nogle af vægtene, heriblandt w_{11} og w_{12}, vokser voldsomt, samtidig med at cost funktionen i figur F.1 ses at være konstant. Dette skyldes, at iterationspunktet bevæger sig i en retning i vægtrummet, i hvilken cost funktionen er konstant, som det beskrives i afsnit 6.8. Betragtes snittene gennem cost funktionen på figurerne F.3–F.7 får man da også indtrykket af, at vægtene w_{11} og w_{12} bevæger sig gennem en dal, i hvis retning cost funktionen er konstant. På figur F.9 vises et meshplot af cost funktionen efter 150 iterationer Gauss-Newton, der yderligere bestyrker dette indtryk.

Udsnittene af cost funktionen på figurerne F.3–F.7 angiver iterationspunktet med *. Umiddelbart synes det som om, at punktet befinder sig i et minimum, men dette er ikke helt tilfældet, som det fremgår af figur F.8. Her er der 'zoomet' ind omkring iterationspunktet efter 250 iterationer Gauss-Newton, og det ses her tydeligt, at iterationspunktet befinder sig et stykke fra det (lokale) minimum; at iterationspunktet ikke bevæger sig helt ned i 'bunden af dalen' må skyldes, at den samlede Gauss-Newton søgeretning er dårligt bestemt grundet den dårlige konditionering (se appendiks D), samtidig med at den lineære afhængighed vægtene imellem bevirker et meget stort skridt (se afsnit 6.8). De store skridt bevirker de små skridtlængder η (learning rates) i figur F.1. De stadig stigende gradienter kan malende beskrives ved, at iterationspunktet hele tiden rammer lidt forbi minimumet, samtidig med at det bevæger sig gennem den dal, hvori cost funktionen er konstant. 'Siderne' i dalen bliver stadig stejlere, hvilket bevirker den observerede stadigt stigende gradientnorm. I tabellen nedenfor gengives de afledede af cost funktionen mht. w_{11} og w_{12} efterhånden som Gauss-Newton iterationerne skrider frem:

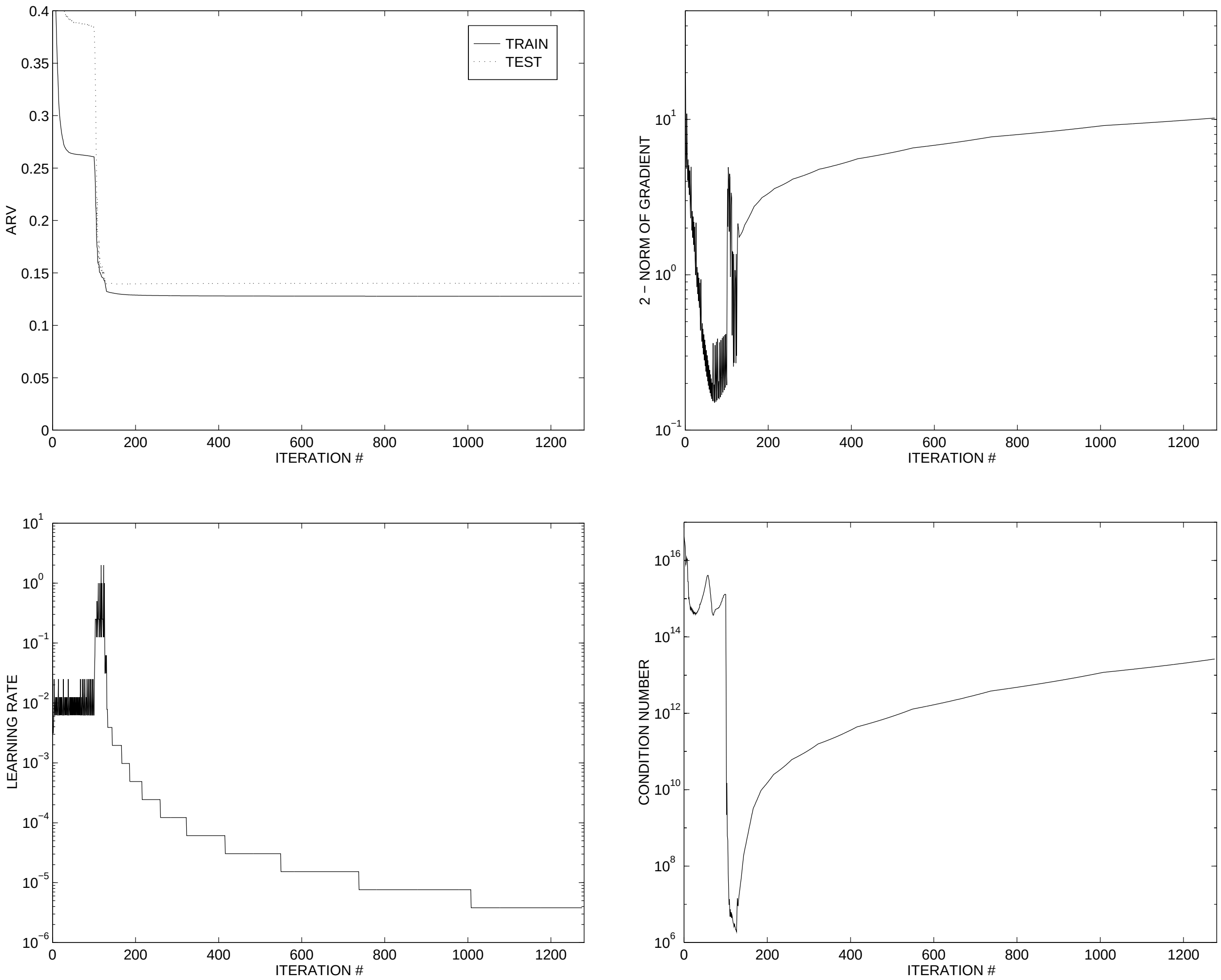
	50 it.	150 it.	250 it.	500 it.	1000 it.
$\frac{\partial C(\mathbf{w})}{\partial w_{11}}$	-0.6212	-1.2941	-1.7235	-2.9148	-3.9070
$\frac{\partial C(\mathbf{w})}{\partial w_{12}}$	-0.4757	-1.0351	-1.4270	-2.5652	-3.5366

De afledede ses at vokse støt i overensstemmelse med ovenstående udredning.

Eksemplet beskrevet ovenfor viser tydeligt, hvilke problemer, der opstår på gund af de lineære afhængigheder imellem vægtene, når der benyttes Gauss-Newton optimering. Betragtes forløbet af træningsfejlen i figur F.1 ses det, at der for de indledende iterationer med gradient descent samt pseudo Gauss-Newton ikke er den samme slags problemer. Dette skyldes, at søgeretningen for disse metoder bestemmes ved at betragte hver vægt for sig, de 'opdager' således ikke den lineære sammenhæng imellem nogle af vægtene, som det beskrives i afsnit 6.8.

Som det fremgår af talrige forsøg i denne rapport, kan weight decay løse ovenstående problemer. Netværket blev derfor optrænet igen, men nu med weight decay, $\alpha = 10^{-3}$. Optræningen blev påbegyndt fra *samme startpunkt* som det tidligere eksempel, og der blev benyttet samme fremgangsmåde ved optræningen, 100 iterationer gradient descent samt 25 iterationer pseudo Gauss-Newton, inden Gauss-Newtons metode blev sat ind. I figur F.10 gengives forløbet af denne optræning. Stopkriteriet for gradientnormen bliver nu opfyldt efter få iterationer med Gauss-Newtons metode.

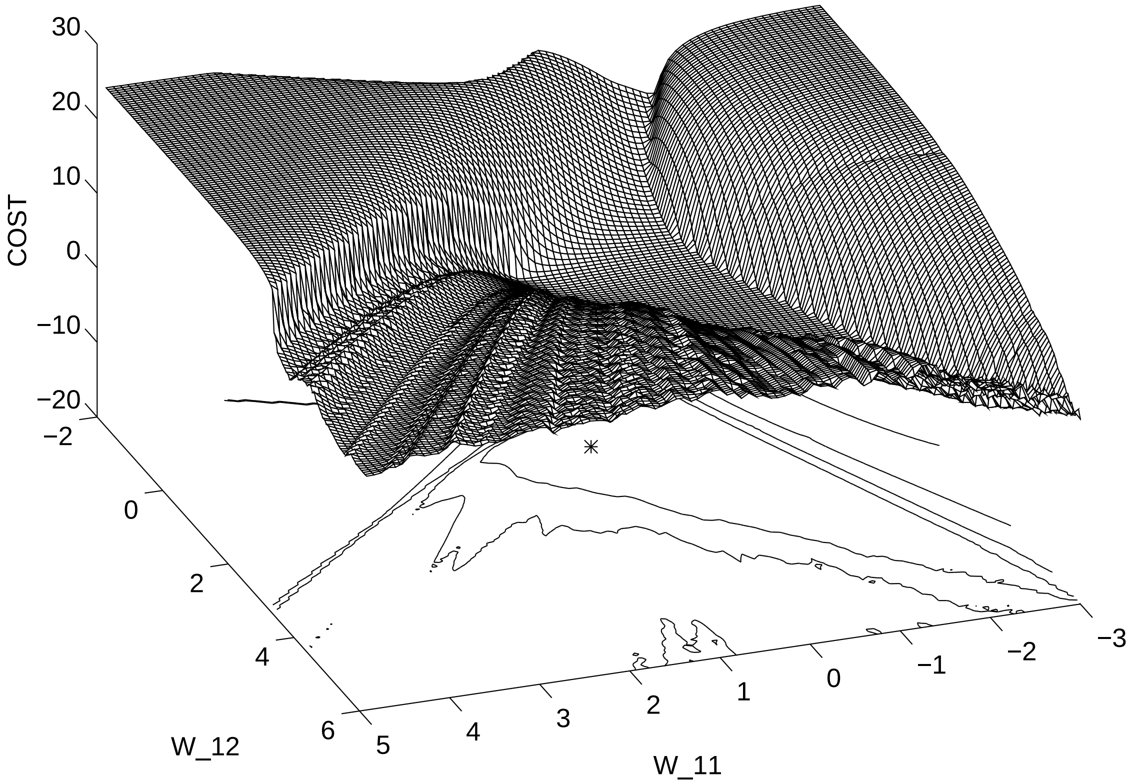
Træningsfejlen ses at nå ned på et noget lavere niveau, Gauss-Newtons metode bliver ikke længere 'fanget' af retninger, hvor cost funktionen er konstant, for de findes ikke længere; leddet med weight decayet sørger for, at cost funktionen vil *stige* i disse retninger. På figur F.11 vises et meshplot af cost funktionen i det bestemte minimum som funktion af w_{11} og w_{12} sammen med vægtene i dette punkt. Det bemærkes, at plottet har stor lighed med figur F.2, blot er det drejet en smule mod urets retning. Hvis man betragter de underliggende konturplots, bemærkes det, at weight decayet i figur F.11 nu har 'lukket' den dal henimod (negativ w_{11}, positiv w_{12}), gennem hvilken vægtene før bevægede sig, ved at hæve niveauet for denne. Som en konsekvens af, at weight decayet har elimineret retningerne, hvor cost funktionen før var konstant, samtidig med at det har forøget de mindste egenværdier (se appendiks D), ses konditionstallet for Hessian-approksimationen i figur F.10 at være blevet væsentligt reduceret.



Figur F.1: Forløb af optræning af 2-1 netværk *uden* weight decay. Illustration af hhv. trænings- & testfejl, gradientnorm, η-værdi samt konditionstal for Hessian-approksimationen.

Vægte lige inden Gauss-Newton påbegyndes:

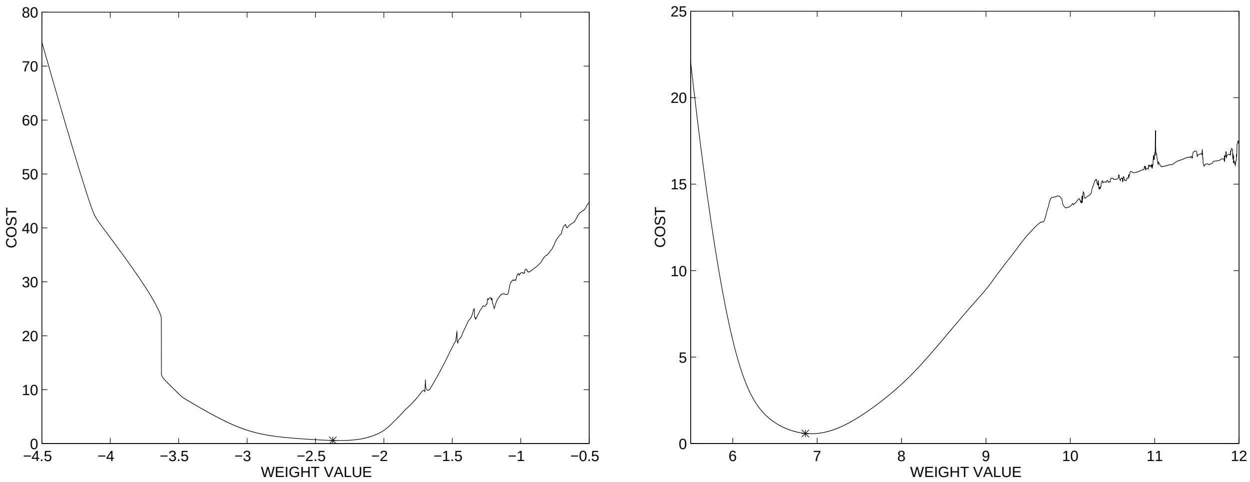
$^{fra \rightarrow}_{\downarrow\, til}$	$h1$	$h2$	$output$	$input$	$bias$
$h1$	0.4202	0.4787	-0.2543	0.1526	0.0397
$h2$	-0.3943	0.7088	-0.4705	-0.5009	0.0936
out	0.4186	0.0223	0	1.0431	0.0506



Figur F.2: Vægte samt cost funktion *uden* weight decay efter 100 iterationer gradient descent samt 25 iterationer pseudo Gauss-Newton, lige inden Gauss-Newtons metode påbegyndes.

Vægte efter 50 iterationer Gauss-Newton:

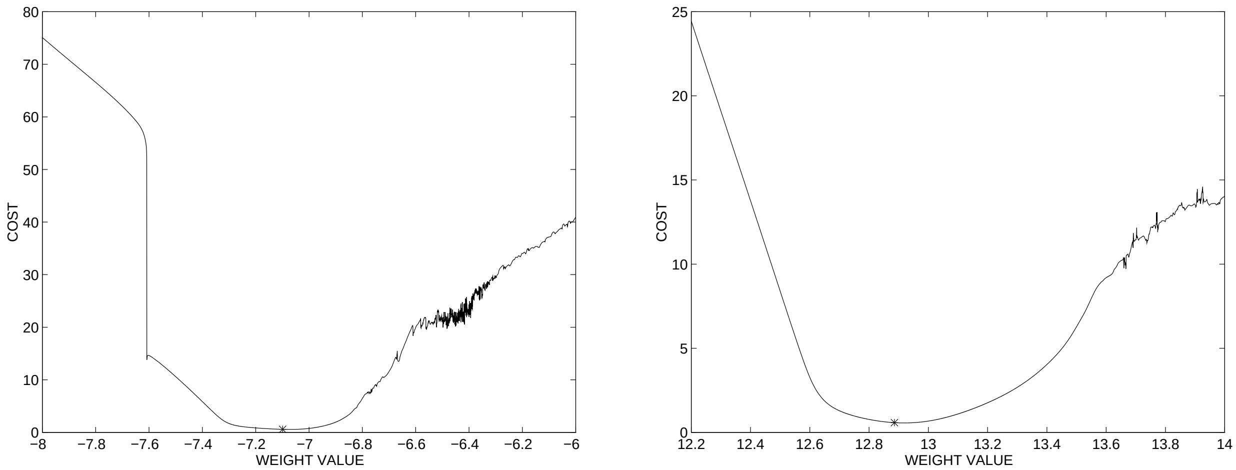
$\overset{fra\rightarrow}{\downarrow til}$	$h1$	$h2$	$output$	$input$	$bias$
$h1$	-2.3732	6.8597	2.7360	-0.0957	0.1043
$h2$	-1.8659	4.5333	2.0998	-0.6810	0.0173
out	0.7041	-1.4156	0	0.3184	0.0106



Figur F.3: Vægte samt udsnit af cost funktionen for hhv. w_{11} (venstre) samt w_{12} (højre) efter 50 iterationer Gauss-Newton.

Vægte efter 150 iterationer Gauss-Newton:

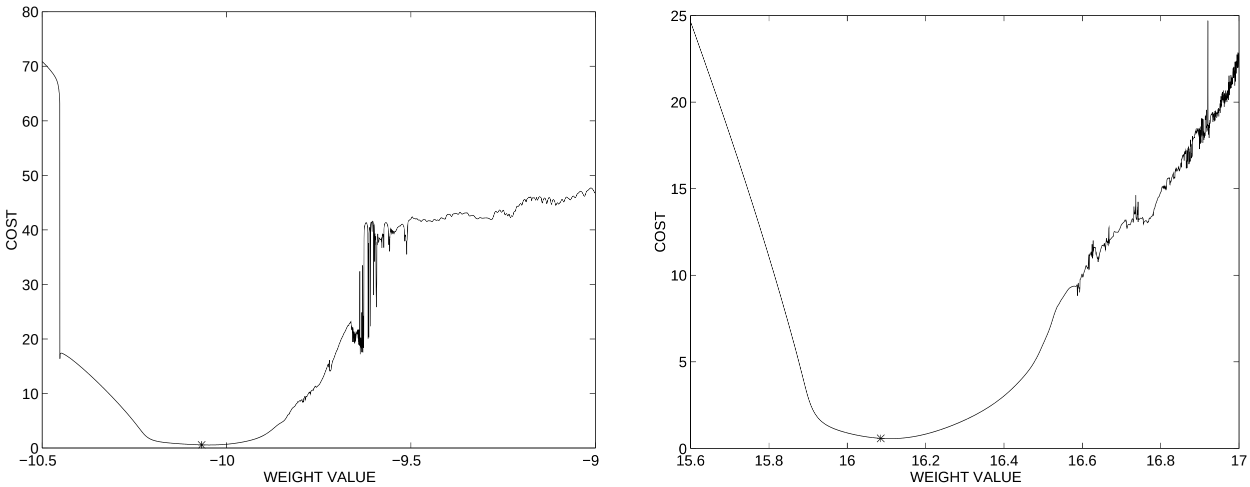
$\overset{fra\rightarrow}{\downarrow til}$	$h1$	$h2$	$output$	$input$	$bias$
$h1$	-7.0992	12.8855	4.0722	-0.2385	0.0866
$h2$	-5.5304	9.6594	3.2140	-0.5919	0.0319
out	1.3881	-2.2060	0	0.2444	0.0077



Figur F.4: Vægte samt udsnit af cost funktionen for hhv. w_{11} (venstre) samt w_{12} (højre) efter 150 iterationer Gauss-Newton.

Vægte efter 250 iterationer Gauss-Newton:

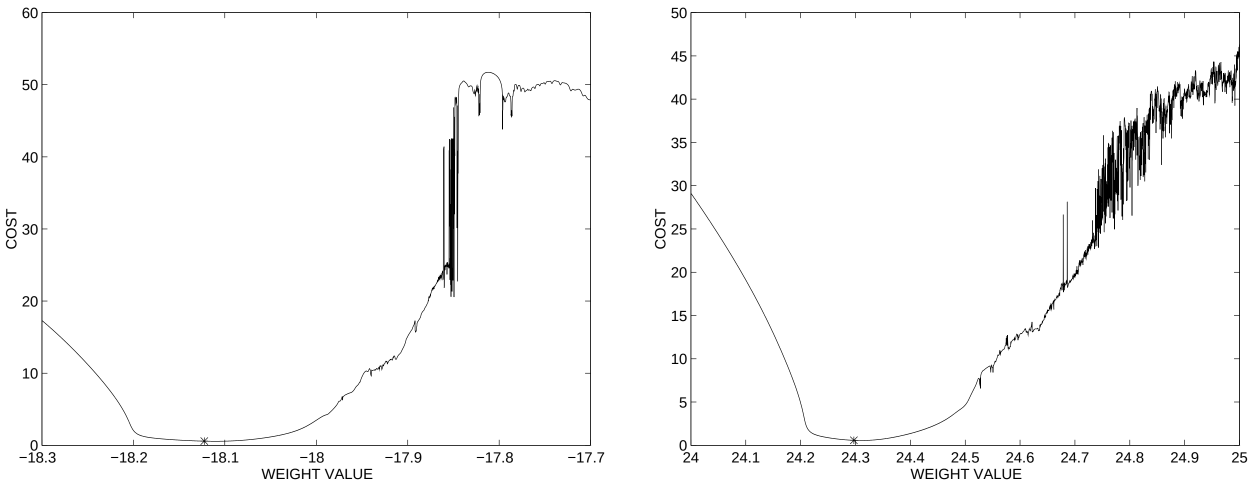
$\substack{fra\rightarrow \\ \downarrow til}$	$h1$	$h2$	$output$	$input$	$bias$
$h1$	-10.0675	16.0855	4.3453	-0.2812	0.0809
$h2$	-8.1450	12.7308	3.5642	-0.5627	0.0370
out	1.8423	-2.6780	0	0.2310	0.0072



Figur F.5: Vægte samt udsnit af cost funktionen for hhv. w_{11} (venstre) samt w_{12} (højre) efter 250 iterationer Gauss-Newton.

Vægte efter 500 iterationer Gauss-Newton:

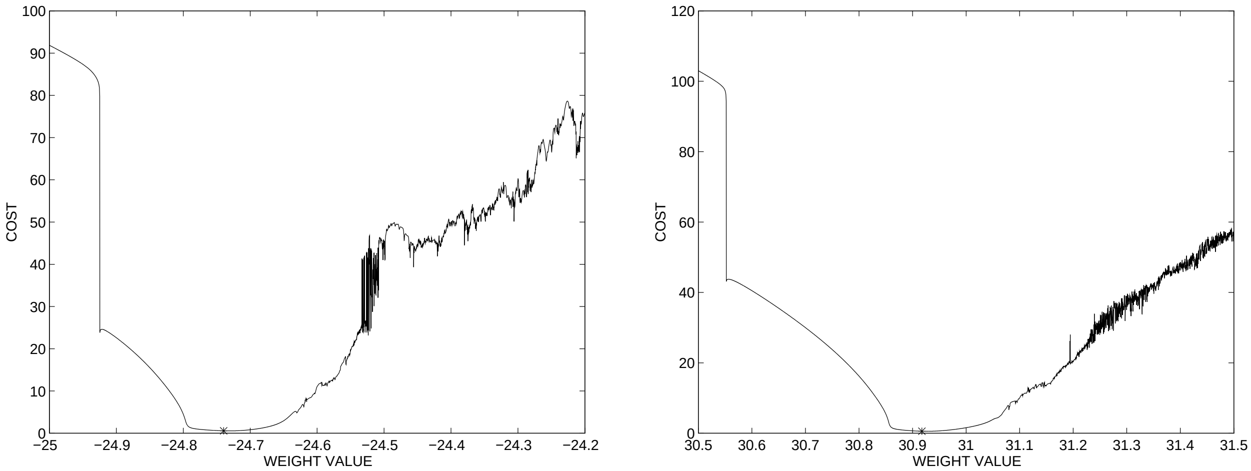
$\substack{fra\rightarrow \\ \downarrow til}$	$h1$	$h2$	$output$	$input$	$bias$
$h1$	-18.1225	24.2971	4.5825	-0.3400	0.0726
$h2$	-15.7133	20.9001	4.0176	-0.5187	0.0446
out	3.1375	-3.9900	0	0.2178	0.0068



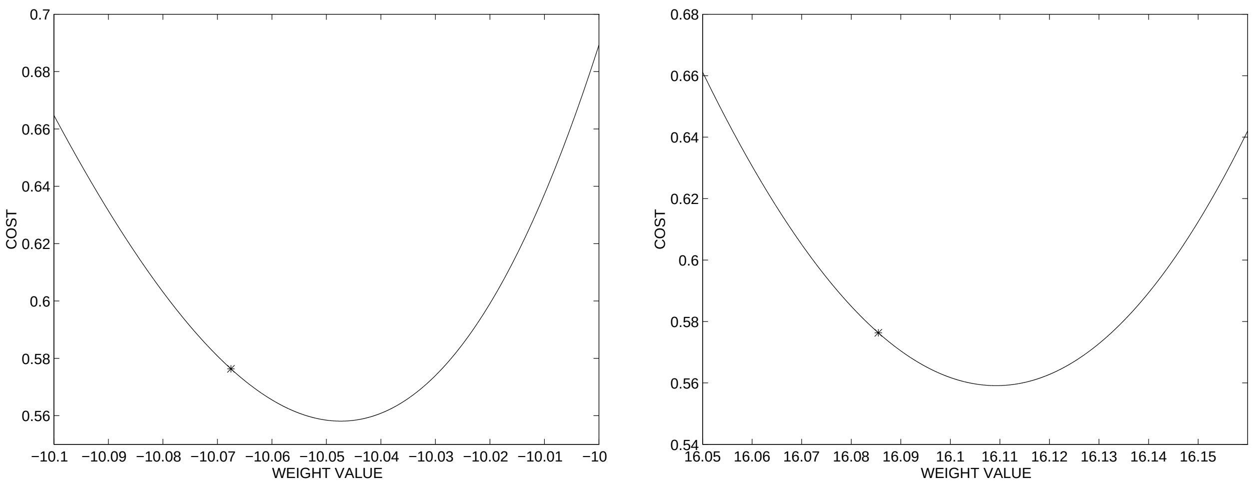
Figur F.6: Vægte samt udsnit af cost funktionen for hhv. w_{11} (venstre) samt w_{12} (højre) efter 500 iterationer Gauss-Newton.

Vægte efter 1000 iterationer Gauss-Newton:

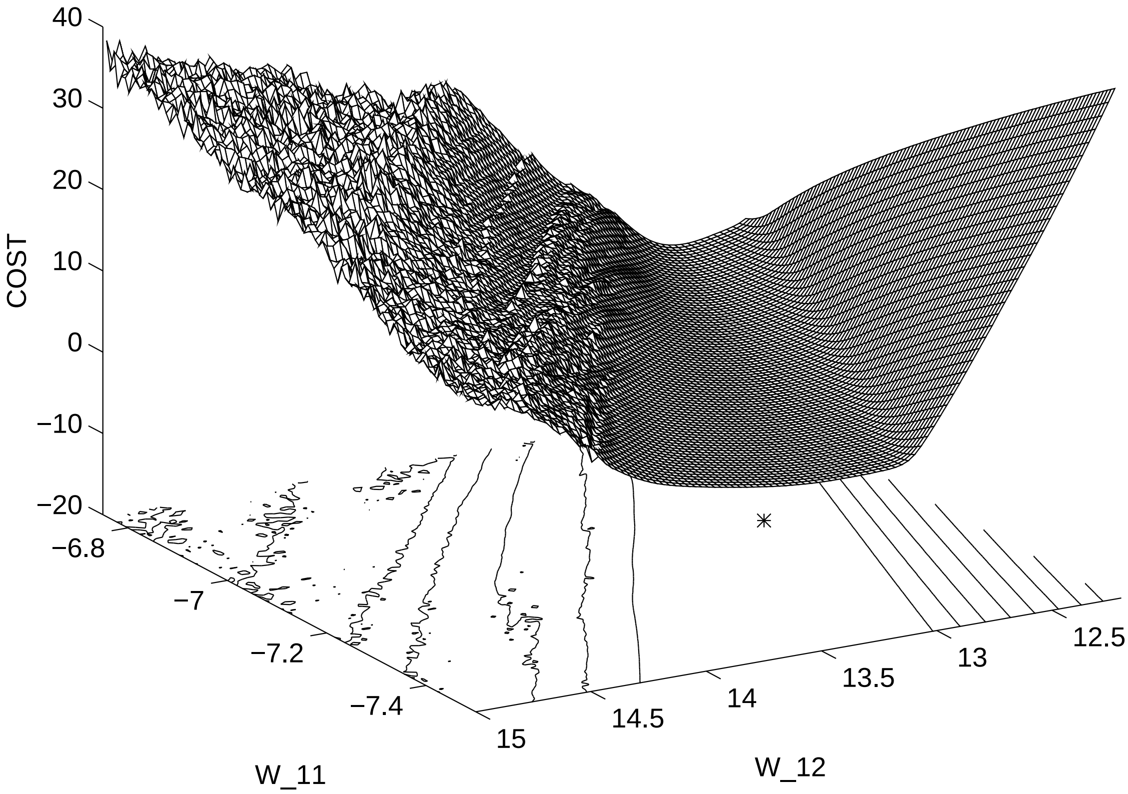
$fra\rightarrow$ $\downarrow til$	$h1$	$h2$	$output$	$input$	$bias$
$h1$	-24.7397	30.9174	4.6252	-0.3632	0.0692
$h2$	-22.1454	27.5510	4.1780	-0.4997	0.0478
out	4.2373	-5.0944	0	0.2142	0.0066



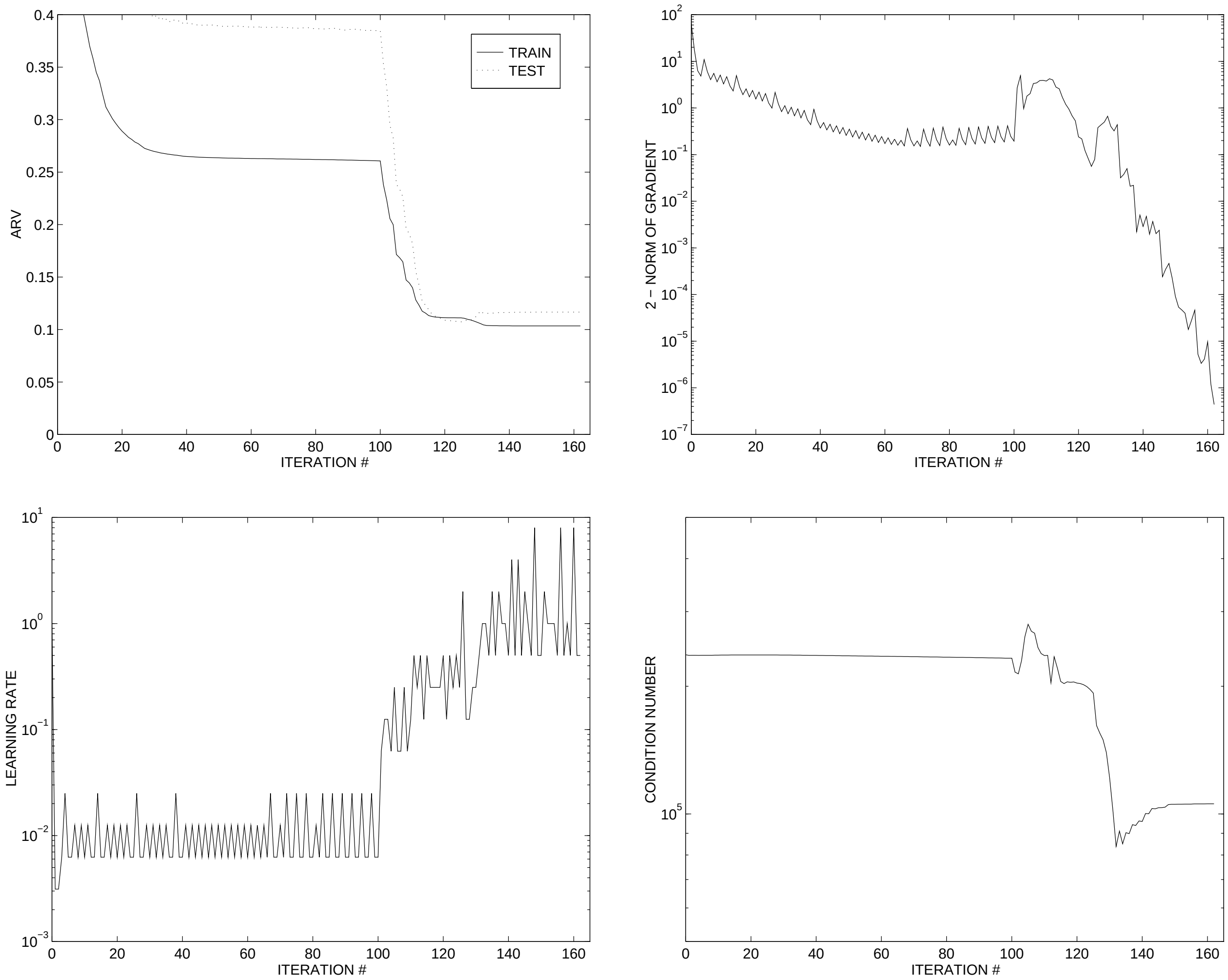
Figur F.7: Vægte samt udsnit af cost funktionen for hhv. w_{11} (venstre) samt w_{12} (højre) efter 1000 iterationer Gauss-Newton.



Figur F.8: Zoom-in på udsnittene af cost funktionen for w_{11} og w_{12} efter 250 iterationer Gauss-Newton.



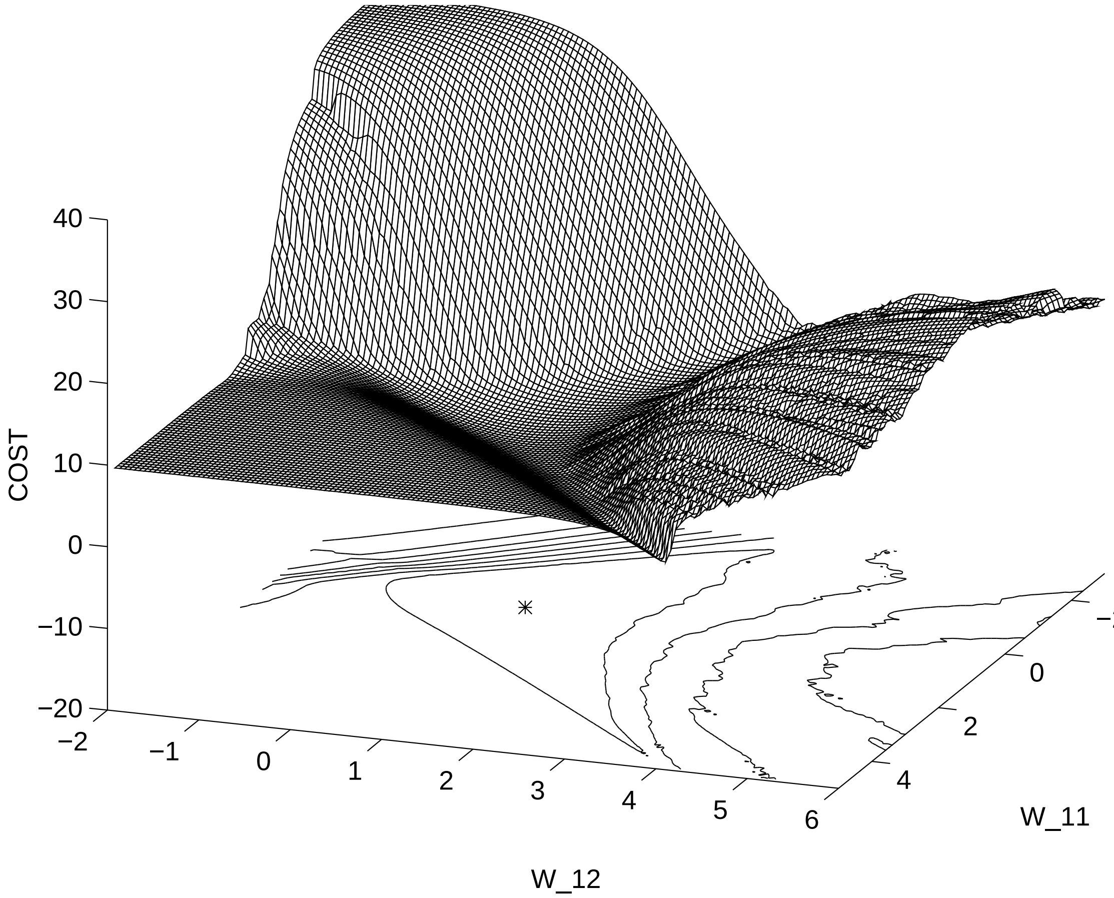
Figur F.9: Cost funktion *uden* weight decay efter 150 iterationer Gauss-Newton.



Figur F.10: Forløb af optræning af 2-1 netværk *med* weight decay. Illustration af hhv. trænings- & testfejl, gradientnorm, η-værdi samt konditionstal for Hessian-approksimationen.

Vægte efter optræning *med* weight decay:

$\substack{fra\rightarrow \\ \downarrow til}$	$h1$	$h2$	$output$	$input$	$bias$
$h1$	0.1531	0.8065	-1.8284	2.4646	-1.0596
$h2$	-0.9030	0.6475	-1.4613	-1.6006	0.1249
out	0.4728	-0.1691	0	0.4519	0.4578



Figur F.11: Vægte samt cost funktion *med* weight decay efter endt optræning.

Appendiks G

Beregningstider for layered RTRL

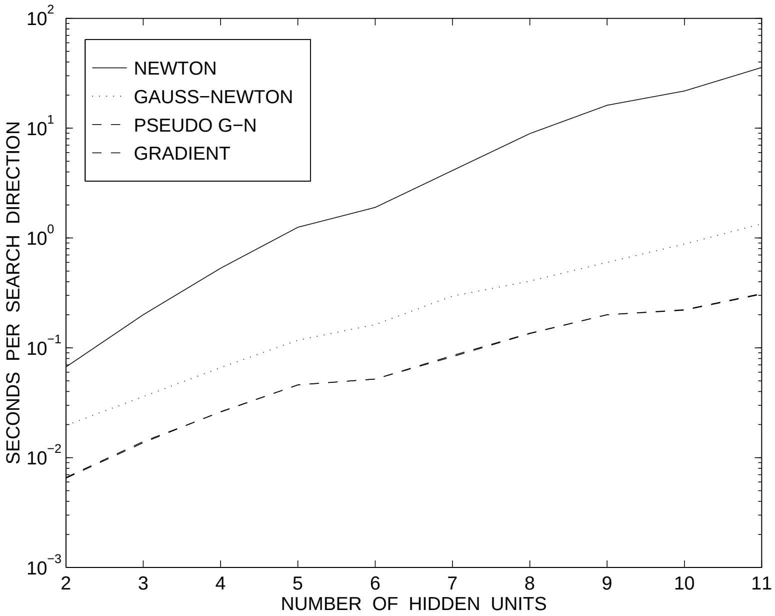
I dette appendiks vises det, hvor lang tid det tager at beregne søgeretningen for de forskellige anvendte optræningsmetoder; se kapitel 1. Beregningstiden er bestemt for voksende netværksstørrelse. Ydermere illustreres de kvantitative såvel som de kvalitative forskelle på optræningsmetoderne med et eksempel for et givent netværk. Specielt fremgår det, at selvom Newtons metode konvergerer mod et (lokalt) minimum i færre iterationer end Gauss-Newtons metode, er den alligevel for dyr at benytte rent tidsmæssigt.

G.1 Beregningstid for søgeretning

I afsnit 2.3 er der foretaget et overslag over, hvordan beregningsbyrden ved beregning af første- og anden afledede for et fuldt forbundet feedback netværk afhænger af netværkskompleksiteten. For at give en fornemmelse for *tidsforbruget* under optræningen, er det for de fire optimeringsmetoder *gradient descent, pseudo Gauss-Newton, Gauss-Newton* samt *Newtons metode* bestemt, hvor lang tid det for hver af metoderne tager at beregne en *søgeretning* for en given netværkskompleksitet.

Resultatet er afbildet i figur G.1. Det afprøvede netværk havde et enkelt eksternt input, hvilket med bias svarer til $I = 2$ i afsnit 2.3. For øget antal skjulte enheder, svarende til øget K, blev det bestemt, hvor lang tid det tog at beregne søgeretningen for hver af de fire optimeringsmetoder nævnt ovenfor. Tiderne blev fundet ved på en iøvrigt ubelastet HP 735 at foretage et større antal søgeretningsbestemmelser, og derefter midle den forbrugte tid.

Funktionen, der blev benyttet til tidsbestemmelse, havde en usikkerhed på ± 1 sekund, men da alle beregningstider er bestemt udfra kørsler af mere end eet minuts varighed, er usikkerheden stammende herfra begrænset til nogle få procent. Usikkerhed er også introduceret af variationer i den del af CPU-tiden, som programmets processer blev tildelt, samt af kommunikation mellem en program-proces og en MATLAB-proces. Disse usikkerheder burde dog være forholdsvis små, da HP 735'eren som nævnt udelukkende afviklede

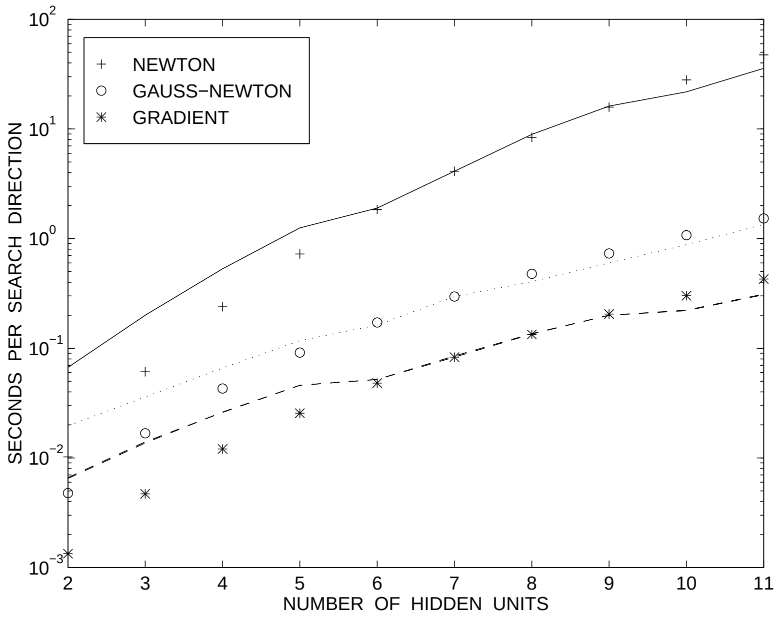


Figur G.1: Beregningstider for søgeretninger med $I = 2$ og voksende K.

mine test-applikationer.

Ved bestemmelse af søgeretningen for Newtons samt Gauss-Newtons metode skal der løses et ligningssystem, se kapitel 1. I den her benyttede implementering foretages denne invertering ved at overføre ligningssystemet til MATLAB, løse systemet her, og hente resultatet tilbage til den pågældende funktion i C–programmet (se appendiks H). Denne overførsel introducerer et (lille) ekstra *overhead* fremfor gradient descent samt pseudo Gauss-Newton, hvilket der lige skal gøres opmærksom på. For alle metoder gælder det, at der er overhead forbundet med kald af funktioner, initialisering af variable m. m.

I figur G.1 bemærkes det, at beregningstiden for søgeretningerne vokser voldsomt for øget netværkskompleksitet. Forskellen mellem gradient descent samt pseudo Gauss-Newton er meget lille, hvorfor kurverne for disse er sammenfaldende; forskellen består blot i, at for pseudo Gauss-Newton skal der for hvert element i søgeretningen foretages en addition, en multiplikation samt en division mere end for gradient descent (se afsnit 1.4.3). Afstanden mellem kurverne for gradient descent og Gauss-Newtons metode er stort set konstant, svarende til at beregningsomkostningerne for de to metoder er omtrent proportionale; dette fremgår også af overslagene i afsnit 2.3. For større netværk øges afstanden, hvilket skyldes, at overheadet fra MATLAB-kaldet bliver mere betydeligt. Det ses, at beregningstiderne for Newtons metode ligger væsentligt over de øvrige; dette skyldes, at Newtons metode benytter den *fulde* Hessian-matrix, og dermed kræver beregning af de anden afledede, hvilket er beregningsmæssigt meget 'dyrt', også selvom det i implementeringen er udnyttet, at Hessianen er symmetrisk.



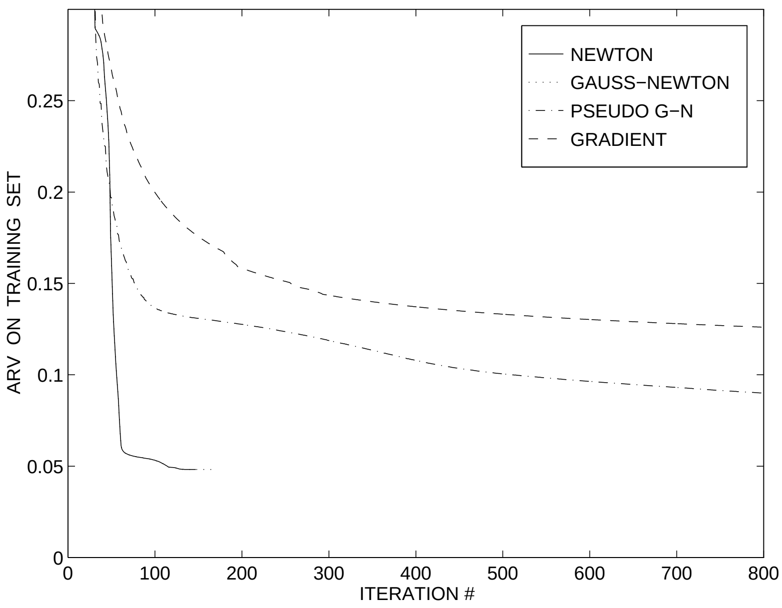
Figur G.2: Angivelse af den teoretiske vækst i beregningstiden.

I figur G.2 er beregningstiderne vist igen, nu med angivelse af den *estimerede* tid for hver af disse. De estimerede tider for metoderne er bestemt som en faktor multipliceret med de tilsvarende udtryk angivet i afsnit 2.3. Faktoren er bestemt således, at de estimerede tider skærer kurven med de faktiske tider i punktet $K = 7$, da overheadet for mindre værdier af K er *for* betydeligt.

De faktiske tider stemmer omtrent overens med de estimerede. For de mindre værdier af antallet af skjulte enheder K ligger de faktiske tider en del over de estimerede, hvilket skyldes større overhead. For de største værdier af K ligger de faktiske tider en smule *under* de estimerede, hvilket muligvis kan tilskrives, at den benyttede compiler i stigende grad kan udnytte effektiviteten af den anvendte maskines arkitektur, herunder udførelse af flere intruktioner samtidigt. Disse forhold kunne eventuelt undersøges nærmere, men udelades her.

G.2 Effektivitet af optræningsmetoder

I dette afsnit foretages en sammenligning af effektiviteten af de forskellige optimeringsmetoder, der er undersøgt i afsnit 6.4 samt afsnit 6.5. Sammenligningen foretages for et layered RTRL netværk med samme arkitektur, som bliver benyttet i afsnit 6.9, dvs. fire skjulte enheder, 12 eksterne inputs; outputenheden har ikke lokalt feedback, modtager ikke de eksterne inputs og forbindes ikke tilbage til de skjulte enheder. Under optræningen var cost funktionen udvidet med weight decay, $\alpha_{ff} = 10^{-4}$ samt $\alpha_{rec} = 0.05$. De fire optræninger startede naturligvis i *samme startpunkt*.

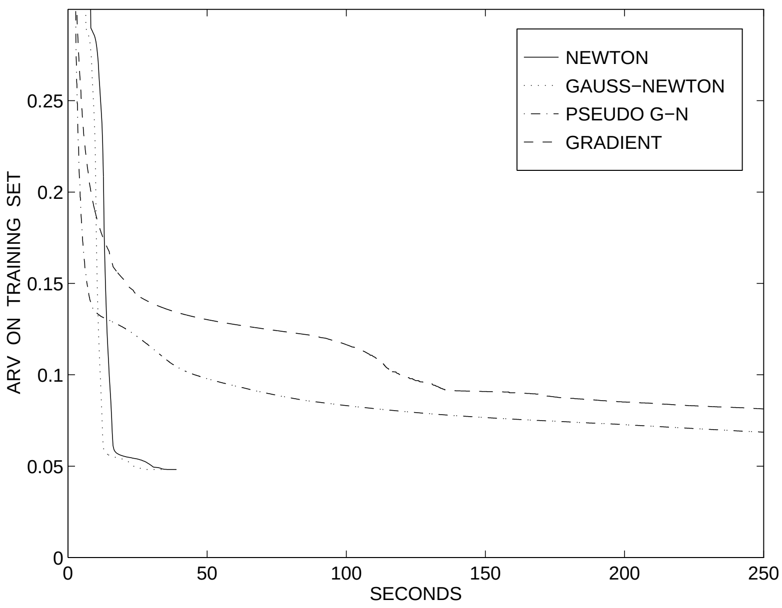


Figur G.3: Træningsfejl som funktion af iterationsantallet for forskellige optimeringsmetoder.

I figur G.3 er for de fire metoder vist træningsfejlen som funktion af iterationsantallet. Alle fire optræninger blev indledt med 30 iterationer gradient descent. Herefter blev der udført 5000 iterationer gradient descent samt 5000 iterationer pseudo Gauss-Newton; stopkriteriet $\|C(\mathbf{w})\|_2 < 10^{-6}$ blev for disse metoder aldrig opfyldt. Dette stopkriterie blev efter de indledene 30 iterationer opfyldt med 139 iterationer af Gauss-Newtons metode. Newtons metode kan først benyttes, når man er tilstrækkelig tæt på et (lokalt) minimum, hvorfor metoden indledtes med Gauss-Newton iterationer indtil $\|C(\mathbf{w})\|_2 < 10^{-2}$, som det også blev gjort i afsnit 6.5.3; dette forklarer, hvorfor kurverne for de to metoder er sammenfaldende. Iterationsantallet fordelte sig her med 113 iterationer Gauss-Newton og 4 iterationer med Newtons metode for opfyldelse af stopkriteriet $\|C(\mathbf{w})\|_2 < 10^{-6}$; denne sammensætning af optimeringsmetoderne 'sparede' således 22 iterationer fremfor Gauss-Newton alene.

Af figur G.3 fremgår det, at andenordens metoderne Gauss-Newton samt Newtons metode kvalitativt er langt bedre end de to andre metoder, idet de er i stand til at 'nå ned' til et (lokalt) minimum for cost funktionen. Som nævnt ovenfor opfyldte gradient descent og pseudo Gauss-Newton aldrig stopkriteriet, ligesom niveauet for træningsfejlen efter de 5000 iterationer lå væsentligt højere end for andenordens metoderne.

I figur G.4 vises træningsfejlene igen, men nu som funktion af den forbrugte tid til optræningen, midlet over *hele* optræningsforløbet. Figuren giver således udtryk for effektiviteten af de anvendte optræningsmetoder. I tabellen nedenfor gengives for de fire metoder den tid, det tog at udføre det



Figur G.4: Træningsfejl som funktion af den benyttede tid til optræning.

optræningsforløb, der blev beskrevet ovenfor, ligesom antallet af iterationer samt opnået træningsfejl angives:

Metode	Tid	Iterationer	ARV_{train}
Gradient descent	415 sek.	5000	0.0778
Pseudo Gauss-Newton	446 sek.	5000	0.0596
Gauss-Newton	35 sek.	169	0.0482
Newton	39 sek.	147	0.0482

I figur G.4 bemærkes det, at Gauss-Newton samt Newton også rent tidsmæssigt, eller kvantitativt, hurtigt bliver langt mere effektive end de to andre metoder. Af figuren samt af tabellen ovenfor fremgår det, at det *ikke* kan betale sig at benytte Newtons metode til optimering for den her undersøgte arkitektur. Selvom der blev sparet 22 iterationer i forhold til Gauss-Newtons metode, tog optræningen alligevel længere tid. Dette skyldes, at beregning af den fulde Hessian er langt dyrere end blot at beregne Gauss-Newton approksimationen, jvf. afsnit G.1. Så selvom Newtons metode rent konvergensmæssigt er mere effektiv end Gauss-Newtons metode, står den øgede effektivitet ikke mål med den øgede beregningstid. På denne baggrund må det derfor konstateres, at det *ikke* kan betale sig at implementere den *fulde* Hessian til optimeringsformål for de i dette arbejde benyttede arkitekturer.

Den store effektivitet af Newtons samt Gauss-Newtons metode er naturligvis helt afhængig af, at det betragtede netværk har en overkommelig størrelse. Bliver netværket for stort, bliver det praktisk umuligt at beregne disse metoders søgeretning, hvilket vil fremgå af figur G.1. I dette tilfælde må det udfra figur G.4 anbefales at bruge pseudo Gauss-Newton, idet beregningsbyrden ikke er meget større end for gradient descent, men konvergensen mod et (lokalt) minimum derimod en del bedre.

Appendiks H

Programbeskrivelse

Dette appendiks indeholder en kortfattet beskrivelse af det udviklede programmel.

Til at afprøve layered RTRL-arkitekturen beskrevet i denne rapport er der udviklet programmel i C. Programmellet består af et større antal funktioner til bla. initialisering, optræning, beskæring samt test, der alle beskrives kort nedenfor.

Programmet er meget generelt. Hvilke af programmets mange muligheder der ønskes benyttet, samt parametre programmet skal anvende, fastlægges inden kompilering ved brug af `# define`'s. Dette er ikke det mest brugervenlige interface, men til brug for de i dette arbejde udførte forsøg har det været fuldt ud tilstrækkeligt, og tillige simpelt at implementere samt udvide med nye features.

Det er ikke blot programmets funktionelle muligheder, der er meget generelle. Programmet åbner op for muligheden for at simulere netværk med *vilkårlige* forbindelser mellem enhederne. Dette er en direkte konsekvens af, at programmet giver mulighed for at beskære netværkene. Det er feks. på denne måde, at forsøgene med feedforward netværk samt de 'amputerede' fuldt rekursive netværk i denne rapport er blevet udført.

Helt essentiel for andenordens metoderne er løsning af lineære ligningssystemer, ligesom Optimal Brain Surgeon behøver den inverse Hessianmatrix. I programmet løses disse opgaver ved at starte en *MATLAB engine* op, hvilket svarer til at starte en sædvanlig MATLAB session; der åbnes blot ikke et kommandovindue. Kommunikation med MATLAB processen foregår via nogle af MathWorks leverede kommunikationsfunktioner, som *linkes* på programmet, og der kan vha. disse sendes data frem og tilbage imellem MATLAB og C–programmet, ligesom almindelige MATLAB kommandoer kan udføres ved at sende en streng med kommandoerne til MATLAB processen, hvor de udføres på sædvanlig vis.

Anvendelse af MATLAB på denne måde er en stor fordel ved undersøgelse af interne forhold i netværket under optræning samt beskæring. Data for netværket opsamles løbende og sendes til MATLAB, hvor de gemmes i passende vektorer. Ved programmets afslutning udføres der en `save`–kommando fra MATLAB processen, hvorved alle de lagrede variable gemmes på sædvanlig vis. Herefter kan de hentes ind i en ordinær MATLAB session og visualiseres samt behandles her.

Programmet anvender i vid udstrækning globale variable. Dette er imod al god programmeringsskik, men strategien er her benyttet alligevel for at undgå funktioner med et overdrevent antal argumenter. Det har dog heller ikke på noget tidspunkt været hensigten at udvikle generelle funktioner, snarere at udvikle et *lukket* miljø med *alle* relevante funktioner indbygget.

Nogle af de implementerede funktioner har meget stor kompleksitet. Dette er specielt tilfældet for beregning af gradienten og i særdeleshed af de anden afledede af cost funktionen. Sådanne funktioner er meget disponerede for banale programmeringsfejl, der med sædvanlige debugging værktøjer er umulige at opdage. For at undgå denne type fejl, er der udviklet fuldstændigt identiske funktioner i MATLAB, hvor overskueligheden er langt større. Ved at sammenligne outputtet fra hhv. MATLAB og C–funktionerne er alle banale fejl med glemt pointer-opdatering mv. elimineret fuldstændigt.

Den parallelle implementering i MATLAB viste med al ønskelig tydelighed fordelen ved at anvende C. C–programmet var i hvert fald 20 gange hurtigere end et (dog uoptimeret) tilsvarende MATLAB program. Ydermere tillader de begrænsede muligheder for at opbygge datastrukturer i MATLAB heller ikke at økonomisere med lagerforbruget i samme omfang, som det er gjort i C–programmet.

Programmellets funktioner er distribueret på følgende filer:

rtrl.c Indeholder hovedprogrammet samt erklæringer af samtlige globale variable samt prototyper på alle funktioner. Filen indeholder yderligere funktioner, der opbygger samt initialiserer datastrukturerne for de globale variable. I denne fil sammensættes programmets funktion vha. `#define`'s.

mat_tool.h Indeholder funktioner, der opretter samt nedlægger en MATLAB proces samt hjælpefunktioner, der kommunikerer med MATLAB-processen.

gradient.h Indeholder funktioner, der beregner gradienten af cost funktionen for et layered RTRL netværk.

training.h Indeholder funktioner, der implementerer de forskellige benyttede optræningsmetoder.

pseudo.h Indeholder funktioner, der beregner approksimationerne til Hessianmatricen anvendt af pseudo Gauss-Newton samt Gauss-Newtons metode.

hessian.h Indeholder funktioner til beregning af den *fulde* Hessian for et layered RTRL netværk.

prune.h Indeholder funktioner, der implementerer de benyttede beskæringsalgoritmer.

toolbox.h Indeholder hjælpefunktioner, der benyttes til undersøgelse af et givet netværk.

I det følgende gives en oversigt over funktionerne i de enkelte filer.

rtrl.c

Hovedfilen indeholder funktionerne:

```
irand();
rand250();
init_r250();
rand250();
frand250();
randnum();
```
Funktioner fra *Dr. Dobb's Journal* til generering af tilfældige tal.

```
calc_var();
```
Beregner variansen af det anvendte datasæt.

```
setup();
```
Opbygger de globale datastrukturer.

```
design_net();
```
Initialiserer de tilstedeværende vægte samt deres weight decays.

```
main();
```
Hovedfunktionen. Koordinerer optræning og beskæring af netværket.

mat_tool.h

Filen med MATLAB-interface funktioner indeholder funktionerne:

```
open_engine();
```
Starter en MATLAB proces op, som C–programmet kan kommunikere med.

```
kill_engine();
```
Nedlægger en MATLAB proces.

```
mat_eval_str();
```
Sender en streng med MATLAB kommandoer til MATLAB processen, hvor strengen evalueres.

```
mat_put_vec();
```
Sender en vektor fra C–programmet til MATLAB processen under et specificeret navn.

`mat_get_vec();`
Overfører en navngiven vektor fra MATLAB processen til en vektor i C–programmet.

`mat_put_matrix();`
Sender en matrix fra C–programmet til MATLAB processen under et specificeret navn.

`mat_get_matrix();`
Overfører en navngiven matrix fra MATLAB processen til en matrix i C–programmet.

`solve_system();`
Løser et lineært ligningssystem ved at overføre matricen samt højresiden til MATLAB processen, hvor ligningssystemet løses. Resultatet hentes tilbage til en specificeret vektor i C–programmet.

gradient.h

Denne fil indeholder funktioner til beregning af gradienten:

`calc_grad();`
Beregner gradienten for den udvidede kvadratiske cost funktion for et layered RTRL netværk.

`calc_out_and_deriv();`
Beregner outputtet fra samt de afledede af aktiveringsfunktionerne for et enkelt eksempel for et layered RTRL netværk.

`calc_out();`
Beregner de skjulte enheders output for et givet input samt tidligere tilstand.

training.h

Denne fil indeholder hovedsalig funktioner til optræning af netværket:

`grad_desc();`
Optimerer på cost funktionen med *Gradient Descent* metoden.

`pseudo_newton1();`
Optimerer på cost funktionen med *pseudo Gauss-Newton*.

`pseudo_newton2();`
Optimerer på cost funktionen med *Gauss-Newtons* metode.

`train_newton();`
Optimerer på cost funktionen med *Newtons* metode.

`evaluate();`
Beregner *ARV*'s på trænings- samt testsæt.

`calc_E();`
Returnerer værdien af den *kvadratiske* cost funktion $E(\mathbf{w})$ for de aktuelle vægte.

`calc_C();`
Returnerer værdien af den *udvidede* cost funktion $C(\mathbf{w})$ (adderer weight decay).

pseudo.h

Funktionerne i denne fil beregner approksimationer til Hessianmatricen:

`diag_pseudo_hess();`
Beregner gradienten samt diagonal-approksimationen til Gauss-Newton Hessianen for den udvidede cost funktion for et layered RTRL netværk.

`pseudo_hess();`
Beregner gradienten samt Gauss-Newton approksimationen til Hessianmatricen for den udvidede cost funktion for et layered RTRL netværk.

hessian.h

Med disse funktioner beregnes den *fulde* Hessian matrix for cost funktionen for et layered RTRL netværk:

`inv_hessian();`
Returnerer den *inverse* af den *fulde* Hessianmatrix i den globale matrix `hessian`.

`calc_hessian();`
Beregner den *fulde* Hessianmatrix samt gradienten for den udvidede cost funktion for et layered RTRL netværk, og lagrer resultaterne i de globale datastrukturer.

`calc_sec_deriv();`
Beregner de anden afledede af aktiveringsfunktionerne for enhederne i netværket.

`inverse();`
Inverterer Hessianmatricen i MATLAB.

prune.h

Med disse funktioner beskæres netværket:

`obs_pruning();`
Beskærer det aktuelle netværk med Optimal Brain Surgeon. Netværket genoptræ-

nes, hvis dette er specificeret, og informationer omkring beskæringsforløbet sendes til MATLAB processen.

`obd_pruning();`
Beskærer det aktuelle netværk med Optimal Brain Damage. Netværket genoptrænes, hvis dette er specificeret, og informationer omkring beskæringsforløbet sendes til MATLAB processen.

`mag_pruning();`
Beskærer det aktuelle netværk med Magnitude Based metoden. Netværket genoptrænes, hvis dette er specificeret, og informationer omkring beskæringsforløbet sendes til MATLAB processen.

toolbox.h

Denne fil indeholder diverse hjælpeværktøjer til netværksanalyse:

`plot_onestep();`
Beregner et-skridts prædiktion for netværket, og gemmer tidsserien, netværkets output samt de skjulte enheders output i filen *comp_one.dat*.

`plot_iterate();`
Beregner itereret prædiktion af testsættet, og gemmer tidsserien, netværkets output samt de skjulte enheders output i filen *comp_ite.dat*.

`impulse();`
Beregner outputtets samt de skjulte enheders svar på en impuls af given størrelse, og gemmer impulssvarene i filen *impulse.dat*.

`dump_weights();`
Gemmer de aktuelle vægte i MATLAB format i filen *weights.mat* i en variabel med navnet `weights`.

`norm();`
Returnerer 2–normen af en specificeret vektor.

`save_jacob();`
Beregner *Jacobi-matricen* for cost funktionen i det aktuelle punkt, og lagrer den i MATLAB processen under navnet `jacob`.

Appendiks I

Artikel til NIPS'94

Dette appendiks indeholder en artikel, som jeg i forbindelse med mit eksamensprojekt har udarbejdet i samarbejde med min vejleder Lars Kai Hansen. Artiklen er optaget til præsentation ved konferencen NIPS'94, der afholdes ultimo november 1994 i Denver, Colorado, og er et udkast til den endelige artikel.

Recurrent Networks: Second Order Properties and Pruning

Morten With Pedersen and **Lars Kai Hansen**
CONNECT, Electronics Institute
Technical University of Denmark B349
DK-2800 Lyngby, DENMARK
emails: with,lkhansen@ei.dtu.dk

Abstract

Second order properties of cost functions for recurrent networks are investigated. We analyze a layered fully recurrent architecture, the virtue of this architecture is that it features the conventional feedforward architecture as a special case. A detailed description of recursive computation of the *full Hessian* of the network cost function is provided. We discuss the possibility of invoking simplifying approximations of the Hessian and show how weight decays *iron* the cost function and thereby greatly assist training. We present tentative pruning results, using Hassibi et al.'s *Optimal Brain Surgeon*, demonstrating that recurrent networks can construct an efficient internal memory.

1 LEARNING IN RECURRENT NETWORKS

Time series processing is an important application area for neural networks and numerous architectures have been suggested, see e.g. (Weigend and Gershenfeld, 94). The most general structure is a fully recurrent network and it may be adapted using *Real Time Recurrent Learning* (RTRL) suggested by (Williams and Zipser, 89). By invoking a recurrent network, the *length* of the network memory can be adapted to the given time series, while it is fixed for the conventional lag-space net (Weigend et al., 90). In forecasting, however, feedforward architectures remain the most popular structures; only few applications are reported based on the Williams&Zipser approach. The main difficulties experienced using RTRL are slow convergence and lack of generalization. Analogous problems in feedforward nets are solved using second order methods for training and pruning (LeCun et al., 90; Hassibi et al., 92; Svarer et al., 93). Also, regularization by weight decay significantly improves training and generalization. In this work we initiate the investigation of second order properties for RTRL; a detailed calculation scheme for the cost function Hessian is presented, the importance of weight decay is demonstrated, and preliminary pruning results using Hassibi et al.'s Optimal Brain Surgeon (OBS) are presented. We find that the recurrent network discards the available lag space and constructs its own efficient internal memory.

1.1 REAL TIME RECURRENT LEARNING

The fully connected feedback nets studied by Williams&Zipser operate like a state machine, computing the outputs from the internal units according to a state vector $\mathbf{z}(t)$ containing *previous* external inputs and internal unit outputs. Let $\mathbf{x}(t)$ denote a vector containing the external inputs to the net at time t, and let $\mathbf{y}(t)$ denote a vector containing the outputs of the units in the net. We now arrange the indices on $\mathbf{x}$ and $\mathbf{y}$ so that the elements of $\mathbf{z}(t)$ can be defined as

$$z_k(t) = \begin{cases} x_k(t) & , \quad k \in I \\ y_k(t) & , \quad k \in U \end{cases}$$

where I denotes the set of indices for which z_k is an input, and U denotes the set of indices for which z_k is the output of a unit in the net. Thresholds are implemented using an input permanently clamped to unity. The k'th unit in the net is now updated according to

$$y_k(t+1) = f_k[s_k(t)] = f_k\left[\sum_{j\in I} w_{kj}x_j(t) + \sum_{j\in U} w_{kj}y_j(t)\right] = f_k\left[\sum_{j\in I\cup U} w_{kj}z_j(t)\right]$$

where w_{kj} denotes the weight to unit k from input/unit j and $f_k(\cdot)$ is the activation function of the k'th unit.

When used for time series prediction, the input vector (excluding threshold) is usually defined as $\mathbf{x}(t) = [x(t), \ldots, x(t-L+1)]$ where L denotes the dimension of the *lag space*. One of the units in the net is designated to be the output unit y_o, and its activating function f_o is often chosen to be linear in order to allow for arbitrary dynamical range. The prediction of $x(t+1)$ is $\hat{x}(t+1) = f_o[s_o(t)]$. Also, if the first prediction is at $t = 1$, the first example is presented at $t = 0$ and we set $\mathbf{y}(0) = \mathbf{0}$. We analyse here a modification of the standard Williams&Zipser construction that is appropriate for forecasting purposes. The studied architecture is *layered*. Firstly, we remove the external inputs from the linear output unit in order to prevent the network from getting trapped in a linear mode. The output then reads

$$\hat{x}(t+1) = y_o(t+1) = \sum_{j\in U} w_{oj}y_j(t) + w_{\mathrm{thres},o} \tag{1}$$

Since $\mathbf{y}(0) = \mathbf{0}$ we obtain a first prediction yielding $\hat{x}(1) = w_{\mathrm{thres},o}$ which is likely to be a poor prediction, and thereby introducing a significant error that is fed back into the network and used in future predictions. Secondly, when pruning a fully recurrent feedback net we would like the net to be able to reduce to a simple two-layer feedforward net if necessary. Note that this is *not* possible with the conventional Williams&Zipser update rule, since it doesn't include a layered feedforward net as a special case. In a layered feedforward net the output unit is disconnected from the external inputs; in this case, cf. (1) we see that $\hat{x}(t+1)$ is based on the internal 'hidden' unit outputs $y_k(t)$ which are calculated on the basis of $\mathbf{z}(t-1)$ and thereby $\mathbf{x}(t-1)$. Hence, besides the startup problems, we also get a two-step ahead predictor using the standard architecture.

In order to avoid the problems with the conventional Williams&Zipser update scheme we use a layered updating scheme inspired by traditional feedforward nets, in which we distinguish between hidden layer units and the output unit. At time t, the hidden units work from the input vector $\mathbf{z}^h(t)$

$$z_k^h(t) = \begin{cases} x_k(t-1) & , \quad k \in I \\ y_k^h(t-1) & , \quad k \in U \\ y^o(t-1) & , \quad k = O \end{cases}$$

where I denotes the input indices, U denotes the hidden layer units and O the output unit. Further, we use superscripts h and o to distinguish between hidden unit and output units. The activation of the hidden units is calculated according to

$$y_k^h(t) = f_k^h[s_k^h(t)] = f_k^h\left[\sum_{j \in I \cup U \cup O} w_{kj} z_j^h(t)\right] \quad , \quad k \in U \tag{2}$$

The hidden unit outputs are forwarded to the output unit, which then sees the input vector $\mathbf{z}_k^o(t)$

$$z_k^o(t) = \begin{cases} y_k^h(t) & , \quad k \in U \\ y^o(t-1) & , \quad k = O \end{cases}$$

and is updated according to

$$y^o(t) = f^o[s^o(t)] = f^o\left[\sum_{j \in U \cup O} w_{oj} z_j^o(t)\right] \tag{3}$$

The cost function is defined as $C = E + \mathbf{w}^T\mathbf{R}\mathbf{w}$. $\mathbf{R}$ is a regularization matrix, $\mathbf{w}$ is the concatenated set of parameters, and the sum of squared errors is

$$E = \frac{1}{2}\sum_{t=1}^{T}[e(t)]^2 \quad , \quad e(t) = x(t) - y^o(t), \tag{4}$$

where T is the size of the training set series. RTRL is based on gradient descent in the cost function, here we investigate accelerated training using Newton methods. For that we need to compute first and second derivatives of the cost function. The essential difficulty is to determine derivatives of the sum of squared errors:

$$\frac{\partial E}{\partial w_{ij}} = -\sum_{t=1}^{T} e(t)\frac{\partial y^o(t)}{\partial w_{ij}} \tag{5}$$

The derivative of the output unit is computed as

$$\frac{\partial y^o(t)}{\partial w_{ij}} = \frac{\partial f^o[s^o(t)]}{\partial s^o(t)} \cdot \frac{\partial s^o(t)}{\partial w_{ij}} \tag{6}$$

where

$$\frac{\partial s^o(t)}{\partial w_{ij}} = \delta_{oi} z_j^o(t) + \sum_{j' \in U} w_{oj'} \frac{\partial y_{j'}^h(t)}{\partial w_{ij}} + w_{oo} \frac{\partial y^o(t-1)}{\partial w_{ij}} \tag{7}$$

where δ_{jk} is the Kronecker delta. This expression contains the derivative of the hidden units

$$\frac{\partial y_k^h(t)}{\partial w_{ij}} = \frac{\partial f_k^h[s_k^h(t)]}{\partial s_k^h(t)} \cdot \frac{\partial s_k^h(t)}{\partial w_{ij}} \quad , \quad k \in U \tag{8}$$

where

$$\frac{\partial s_k^h(t)}{\partial w_{ij}} = \delta_{ki} z_j^h(t) + \sum_{j' \in U} w_{kj'} \frac{\partial y_{j'}^h(t-1)}{\partial w_{ij}} + w_{ko} \frac{\partial y^o(t-1)}{\partial w_{ij}} \tag{9}$$

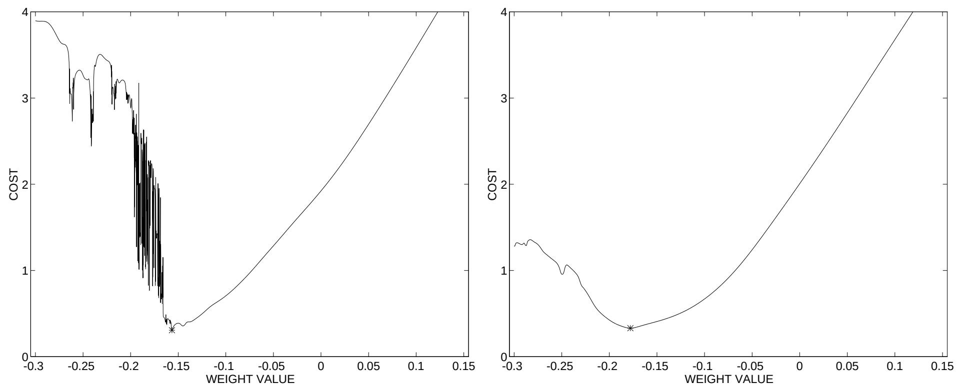


Figure 1: Cost function dependence of a weight connecting two hidden units for the sunspot benchmark series. Left panel: Cost function with small weight decay, the (local) optimum chosen is marked by an asterix. Right panel: The same slice through the cost function but here retrained with higher weight decay.

The complexity of the training problem for the recurrent net using RTRL is demonstrated in figure 1. The important role of weight decay (we have used a simple weight decay $\mathbf{R} = \alpha\mathbf{1}$) in controlling the complexity of the cost function is evident in the right panel of figure 1. The example studied is the sunspot benchmark problem (see e.g. (Weigend et al., 90) for a definition). First, we trained a network with the small weight decay and recorded the left panel result. Secondly, the network was retrained with increased weight decay and the particular weight connecting two hidden units was varied to produce the right panel result. In both cases all other weights remained fixed at their optimal values for the given weight decay. In addition to the complexity visible in these one-parameter slices of the cost function, the cost function is highly anisotropic in weight space and consequently the network Hessian is ill-conditioned. Hence, gradient descent is hampered by slow convergence.

2 SECOND ORDER PROPERTIES OF THE COST FUNCTION

To improve training by use of Newton methods and for use in OBS-pruning we compute the second derivative of the error functional:

$$\frac{\partial^2 E}{\partial w_{ij}\partial w_{pq}} = -\sum_{t=1}^{T}\left[e(t)\frac{\partial^2 y^o(t)}{\partial w_{ij}\partial w_{pq}} - \frac{\partial y^o(t)}{\partial w_{ij}}\cdot\frac{\partial y^o(t)}{\partial w_{pq}}\right] \tag{10}$$

The second derivative of the output is

$$\frac{\partial^2 y^o(t)}{\partial w_{ij}\partial w_{pq}} = \frac{\partial^2 f^o[s^o(t)]}{\partial s^o(t)^2}\cdot\frac{\partial s^o(t)}{\partial w_{ij}}\cdot\frac{\partial s^o(t)}{\partial w_{pq}} + \frac{\partial f^o[s^o(t)]}{\partial s^o(t)}\cdot\frac{\partial^2 s^o(t)}{\partial w_{ij}\partial w_{pq}} \tag{11}$$

with

$$\frac{\partial^2 s^o(t)}{\partial w_{ij}\partial w_{pq}} = \delta_{oi}\frac{\partial z_j^o(t)}{\partial w_{pq}} + \sum_{j'\in U} w_{oj'}\frac{\partial^2 y_{j'}^h(t)}{\partial w_{ij}\partial w_{pq}} + w_{oo}\frac{\partial^2 y^o(t-1)}{\partial w_{ij}\partial w_{pq}} + \delta_{op}\frac{\partial z_q^o(t)}{\partial w_{ij}} \tag{12}$$

This expression contains the second derivative of the hidden unit outputs

$$\frac{\partial^2 y_k^h(t)}{\partial w_{ij}\partial w_{pq}} = \frac{\partial^2 f_k^h[s_k^h(t)]}{\partial s_k^h(t)^2}\cdot\frac{\partial s_k^h(t)}{\partial w_{ij}}\cdot\frac{\partial s_k^h(t)}{\partial w_{pq}} + \frac{\partial f_k^h[s_k^h(t)]}{\partial s_k^h(t)}\cdot\frac{\partial^2 s_k^h(t)}{\partial w_{ij}\partial w_{pq}} \tag{13}$$

with

$$\frac{\partial^2 s_k^h(t)}{\partial w_{ij}\partial w_{pq}} = \delta_{ki}\frac{\partial z_j^h(t)}{\partial w_{pq}} + \sum_{j'\in U} w_{kj'}\frac{\partial^2 y_{j'}^h(t-1)}{\partial w_{ij}\partial w_{pq}} + w_{ko}\frac{\partial^2 y^o(t-1)}{\partial w_{ij}\partial w_{pq}} + \delta_{kp}\frac{\partial z_q^h(t)}{\partial w_{ij}} \tag{14}$$

Recursion in the five index quantity (14) imposes a significant computational burden; in fact the first term of the Hessian in (10), involving the second derivative, is often neglected for computational convenience (LeCun et al., 90). Here we start by analyzing the significance of this term during training. We train a layered architecture to predict the sunspot benchmark problem. In figure 2 the ratio between the largest eigenvalue of the second derivative term in (10) and the largest eigenvalue of the full Hessian is shown. The ratio is presented for two different magnitudes of weight decay. In line with our observations above the second order properties of the "ironed" cost function are manageable, and we can simplify the Hessian calculation by neglecting the second derivative term in (10), i.e., apply the Gauss-Newton approximation.

3 PRUNING BY THE OPTIMAL BRAIN SURGEON

Pruning of recurrent networks has been pursued by (Giles and Omlin, 94) using a heuristic pruning technique, and significant improvement in generalization for a sequence recognition problem was demonstrated. Two pruning schemes are based on systematic estimation of weight *saliency*: the Optimal Brain Damage (OBD) scheme of (LeCun et al., 90) and OBS by (Hassibi et al., 93). OBD is based on the diagonal approximation of the Hessian and is very robust for forecasting (Svarer et al., 93). If an estimate of the full Hessian is available OBS can be used

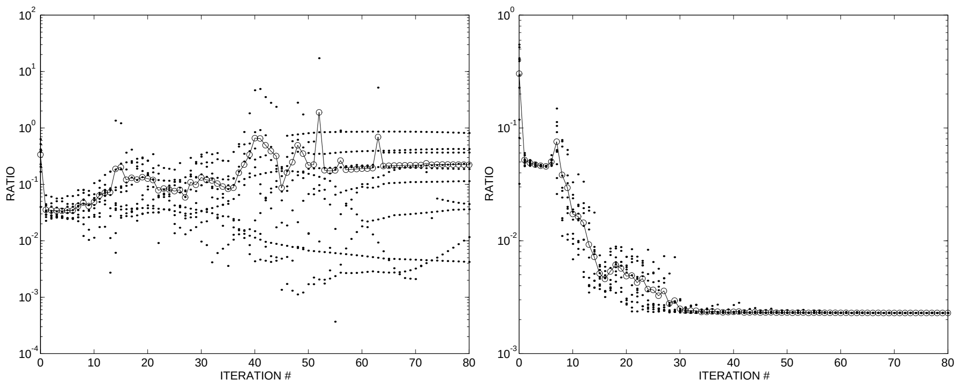


Figure 2: Ratio between the largest magnitude eigenvalue of the second derivative term of the Hessian (c.f. equation (10)) and the largest magnitude eigenvalue of the complete Hessian as they appeared during ten training sessions. The connected circles represent the average ratio. Left panel: Training with small weight decay. Right panel: Training with a high weight decay.

for estimation of saliencies incorporating *linear* retraining. In (Hansen and With Pedersen, 94) OBS was generalized to incorporate weight decays; we use these modifications in our experiments. Note that OBS in its standard form only allows for one weight to be eliminated at a time. The result of a pruning session is a nested family of networks. In order to select the optimal network within the family it was suggested in (Svarer et al., 93.) to use the estimated test error. In particular we use Akaike's Final Prediction Error (Akaike, 69) to estimate the network test error $\widehat{E}_{\text{test}} = ((T+N)/(T-N)) \cdot 2E/T$ [1], and N is the number of parameters in the network. In figure 3 we show the results of such a pruning session on the sunspot data starting from a (4-4-1) network architecture. The recurrent network was trained using a *damped* Gauss-Newton scheme. Note that the training error increases as weights are eliminated, while the test error and the estimated test error both pass through shallow minima showing that generalization is slightly improved by pruning. In fact, by retraining the optimal architecture with reduced weight decay both training and test errors are decreased in line with the observations in (Svarer et al., 93). It is interesting to observe that the network, though starting with access to a lag-space of four delay units, has lost three of the delayed inputs; hence, rely solely on its internal memory, as seen in the right panel of figure 3. To further illustrate the memory properties of the optimal network, we show in figure 4 the network response to a unit impulse. It is interesting that the response of the network extends for approximately 12 time steps corresponding to the "period" of the sunspot series.

[1] The use of Akaike's estimate is not well justified for a feedback net, test error estimates for feedback models is a topic of current research.

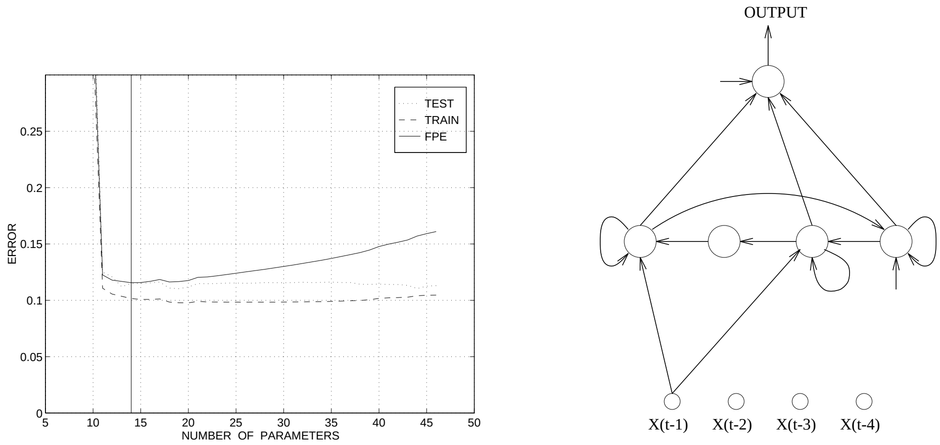


Figure 3: Left panel: OBS pruning of a (4-4-1) recurrent network trained on sunspot benchmark. Development of training error, test error, and Akaike estimated test error (FPE). Right panel: Architecture of the FPE-optimal network. Note that the network discards the available lag space and solely predicts from internal memory.

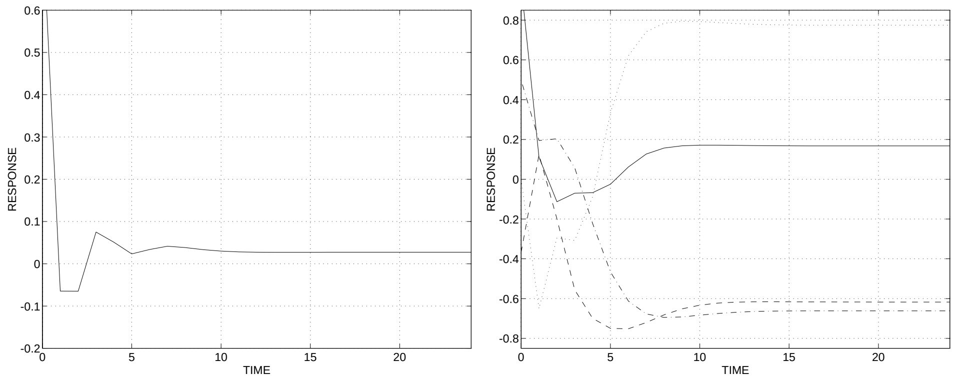


Figure 4: Left panel: Output of the pruned network after a unit impulse input at $t = 0$. The internal memory is about 12 time units long which is, in fact, roughly the period of the sunspot series. Right panel: Activity of the four hidden units in the pruned network after a unit impulse at time $t = 0$.

4 CONCLUSION

A layered recurrent architecture, which has a feedforward net as a special case, has been investigated. A scheme for recursive estimation of the Hessian of the fully recurrent neural net is devised. It's been shown that weight decay plays a decisive role when adapting recurrent networks. Further, it is shown that the second order information may be used to train and prune a recurrent network and in this process the network may discard the available lag space. The network builds an efficient internal memory extending beyond the lag space that was originally available.

Acknowledgments

We thank Jan Larsen, Sara Solla, and Claus Svarer for useful discussions, and Lee Giles for providing us with a preprint of (Giles and Omlin, 94). We thank the anonymous reviewers for valuable comments on the manuscript. This research is supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT).

References

H. Akaike: *Fitting Autoregressive Models for Prediction.* Ann. Inst. Stat. Mat. **21**, 243-247, (1969).

Y. Le Cun, J.S. Denker, and S.A. Solla: *Optimal Brain Damage.* In Advances in Neural Information Processing Systems 2, (Ed. D.S. Touretzsky) Morgan Kaufmann, 598-605, (1990).

C.L. Giles and C.W. Omlin: *Pruning of Recurrent Neural Networks for Improved Generalization Performance.* IEEE Transactions on Neural Networks, to appear. Preprint NEC Research Institute (1994).

L.K. Hansen and M. With Pedersen: *Controlled Growth of Cascade Correlation Nets*, International Conference on Artificial Neural Networks ICANN'94 Sorrento. (Eds. M. Marinaro and P.G. Morasso) Springer, 797-801, (1994).

B. Hassibi, D. G. Stork, and G. J. Wolff, *Optimal Brain Surgeon and General Network Pruning,* in Proceedings of the 1993 IEEE International Conference on Neural Networks, San Francisco (Eds. E.H. Ruspini et al.) IEEE, 293-299 (1993).

C. Svarer, L.K. Hansen, and J. Larsen: *On Design and Evaluation of Tapped Delay Line Networks,* In Proceedings of the 1993 IEEE International Conference on Neural Networks, San Francisco, (Eds. E.H. Ruspini et al.) 46-51, (1993).

A.S. Weigend, B.A. Huberman, and D.E. Rumelhart: *Predicting the future: A Connectionist Approach.* Int. J. of Neural Systems **3**, 193-209 (1990).

A.S. Weigend and N.A. Gershenfeld, Eds.: *Times Series Prediction: Forecasting the Future and Understanding the Past.* Redwood City, CA: Addison-Wesley (1994).

R.J. Williams and D. Zipser: *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks*, Neural Computation **1**, 270-280, (1989).

Litteraturfortegnelse

[Barton 90] Barton, S. A. : *A Matrix Method for Optimizing a Neural Network.* I "Neural Computation", vol. 3, 450–459 (1990).

[Catfolis 93] Catfolis, T. : *A Method for Improving the Real Time Recurrent Learning Algorithm.* I "Neural Networks", vol. 6, 807–821 (1993).

[Dennis 83] Dennis, J. E. & Schnabel, R. B. : *Numerical methods for unconstrained optimization and non-linear equations.* Prentice-Hall series in computational mathematics (1983).

[Doya 93] Doya, K. : *Universality of Fully-Connected Recurrent Neural Networks.* Sendt til IEEE Transactions on Neural Networks. Tilgængelig via `ftp` fra Neuroprose : `doya.universality.ps.Z` (1993).

[Fahlman 90] Fahlman, S. E. & Lebiere, C. : *The Cascade-Correlation Learning Architecture.* I NIPS 2, Ed. D. S. Touretzsky, 524–532. San Mateo, Morgan Kaufmann (1990).

[Fahlman 91] Fahlman, S. E. : *The Recurrent Cascade-Correlation Architecture.* I NIPS 3, Eds. D. S. Touretzsky et al. , 190–196. San Mateo, Morgan Kaufmann (1991).

[Gabel 80] Gabel, R. A. & Roberts, R. A. : *Signals and Linear Systems, 2. ed.* John Wiley and Sons (1980).

[Giles 92] Giles, C. L. , Miller, C. B. et al. :*Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks.* I "Neural Computation", vol. 4, 393–405 (1992).

[Giles 94] Giles, C. L. , Omlin, C. W. : *Pruning of Recurrent Neural Networks for Improved Generalization Per-*

formance. Kommer i "IEEE Transactions on Neural Networks". Preprint fra NEC Research Institute (1994).

[Gorodkin 93] Gorodkin, J. , Hansen, L. , Krogh, A: , Svarer, C. & Winther, O. : *A Quantitative Study of Pruning By Optimal Brain Damage.* I "Int. J. of Neural Systems" **4**, 159–171 (1993).

[Guignot 94] Guignot, J. , Gallinari, P. : *Recurrent Neural Networks with Delays.* I "Proceedings of ICANN'94 International Conference on Neural Networks", Sorrento, Italy, 1994. Eds. M. Marinaro & P. G. Morasso, 389–392 (1994).

[Hansen 94a] Hansen, L. K. , Rasmussen, C. E. , Svarer, C. & Larsen, J. : *Adaptive regularization.* Accepteret til IEEE Workshop on Neural Networks for Signal Processing, Greece 1994. Elektronisk Institut (1994).

[Hansen 94b] Hansen, L. K. : *Upublicerede håndskrevne noter.* Elektronisk Institut, DTU (1994).

[Hansen&With 94] Hansen, L. K. & Pedersen, M. W. : *Controlled Growth of Cascade Correlation Nets.* I "Proceedings of ICANN'94 International Conference on Neural Networks", Sorrento, Italy, 1994. Eds. M. Marinaro & P. G. Morasso, 797–800 (1994).

[Hassibi 93] Hassibi, B. , Stork, D. G. & Wolff, G. J. : *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon.* I NIPS 5, Eds. S. J. Hanson, et al., 164. San Mateo, Morgan Kaufmann (1993).

[Haykin 94] Haykin, S. : *Neural Networks – A comprehensive Foundation.* Macmillan College Publishing Company (1994).

[Hertz 91] Hertz, J. , Krogh, A. , & Palmer, R. G. : *Introduction to the Theory of Neural Computation.* Addison-Wesley, New York (1991).

[Høholdt 89] Høholdt, T. , Nielsen, F. et al. : *Noter til 0122 lineære systemer.* Matematisk Institut, DTU (1989).

[Kohavi 78] Kohavi, Z. : *Switching and Finite State Automata Theory, 2. ed.* Tata McGraw-Hill Publishing Company (1978).

[LeCun 90a] Le Cun, Y. , Boser, B. , Denker, J. S. , Henderson, D. , Howard, R. E. , Hubbard, W. & Jackel, L. D. : *Handwritten Digit Recognition with a BackPropagation Network*. I NIPS 2, Ed. D. S. Touretzsky, 396–404. San Mateo, Morgan Kaufman (1990).

[LeCun 90b] Le Cun, Y. , Denker, J. S. & Solla, S. A. : *Optimal Brain Damage*. I NIPS 2, Ed. D. S. Touretzsky, 598–605. San Mateo, Morgan Kaufmann (1990).

[Lindsey 92] Lindsey, R. L. , Ruck, D. W. , Rogers, S. K. & Kabrisky, M. : *Function Prediction Using Recurrent Neural Networks*. I Proceedings of the SPIE, vol. 1710, 438–448 (1992).

[Ljung 87] Ljung, L. : *System Identification: Theory for the User*. Prentice Hall, Information and System Sciences series (1987).

[Madsen 91a] Madsen, K. : *Hæfte 38: Minimering af kvadratsummer*. Numerisk Instutut, DTU (1991).

[Madsen 91b] Madsen, K. : *Hæfte 40: Optimering under bibetingelser*. Numerisk Instutut, DTU (1991).

[MATLAB] *MATLAB Reference Guide*. The MathWorks Inc. , Natick MA (1992).

[Nielsen 72] Nielsen, H. B. & Madsen, K. : *Hæfte 23: Løsning af lineære ligningssystemer*. Numerisk Institut, DTU (1972).

[Nielsen 91] Nielsen, H. B. & Madsen, K. : *Hæfte 52: Talrepræsentation og afrundingsfejl*. Numerisk Institut, DTU (1991).

[Nielsen 94] Nielsen, H. B. : *Personlig samtale*. Instituttet for Matematisk Modellering (1994).

[MWP 93] Pedersen, M. W. & Bundesen, J. : *Cascade-Correlation & Recurrent Cascace-Correlation*. Individuelt kursusarbejde ved Elektronisk Institut, DTU (1993).

[MWP 94] Pedersen, M. W. : *Optimal Brain Surgeon implementeret på Cascade-Correlation*. Individuelt kursusarbejde ved Elektronisk Institut, DTU (1994).

[Press 92] Press, W. H. et al. : *Numerical Recipes in C, 2. ed.* Cambridge University Press, Cambridge (1992).

[Rasmussen 93] Rasmussen, C. E. : *Generalization in Neural Networks.* Projektarbejde ved Elektronisk Institut, DTU (1993).

[Saarinen 93] Saarinen, S. , Bramley, R. & Cybenko, G. : *Ill-conditioning in neural network training problems.* I "SIAM Journal of scientific computing", vol. 14, 693–714 (1993).

[Schmidhuber 92] Schmidhuber, J. : *A Fixed Size Storage $O(n^3)$ Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks.* I "Neural Computation" **4**, 243–248 (1992).

[Svarer 93] Svarer, C. , Hansen, L. K. & Larsen, J. : *On Design and Evaluation of Tapped-Delay Neural Network Architectures.* The 1993 IEEE Int. Conference on Neural Networks, San Francisco. Eds. H. R. Berenji et al., 45–51, (1993).

[Thodberg 91] Thodberg, H. H. : *Improving Generalization of Neural Networks Through Pruning.* I "Int. J. of Neural Systems" **1**, 317-326 (1991).

[Tingleff 93] Tingleff, O. et al. : *Nye noter i optimeringsalgoritmer.* Numerisk Institut, DTU (1993).

[Tsonis 94] Tsonis, A. A. : *Chaos From Theory to Applications.* Plenum Press, New York (1992).

[Weigend 90] Weigend, A. S. , Huberman, B. A. , Rumelhart, D. E. : *Predicting the Future: A Connectionist Approach.* I "Int. J. of Neural Systems", vol. 3, 193-209 (1990).

[Weigend 94] Weigend, A. & Gershenfeld, N. A. (eds) : *Time Series Prediction: Forecasting the Future and Understanding the Past.* Vol. 15, Santa Fe Institure Studies in the Sciences of Complexity. Addison-Wesley (1994).

[Widrow 85] Widrow, B. & Stearns, S. D. : *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice Hall (1985).

[Williams 89] Williams, R. J. & Zipser, D. : *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.* I "Neural Computation" **1**, 270–280 (1989).