NTNU
Det skapende universitet

Anvendelse av biologisk inspirerte metoder i musikk

Per Kåre Hollund Otteren

Master i datateknikk
Oppgaven levert: Juni 2006
Hovedveileder: Gunnar Tufte, IDI
Medveileder(e): Øyvind Brandtsegg, Musikkkonservatoriet

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Oppgavetekst

Dette prosjektet er relatert til musiker Øyvind Brandtseggs arbeid. Ideen med å anvende metoder fra kunstig intelligens er for å gjøre det mulig å anvende sanntids komponering for live fremføring. Sanntid er viktig da systemet skal benytte improvisasjon basert på interaksjon med musikeren. Prosjektet består av to deler. Først, finn ut hva andre gjør innen område med å anvende KI metoder for komposisjon med fokus på sanntid. Deretter forsøk å anvende metoder funnet i samarbeid med Øyvind.

Oppgaven gitt: 20. januar 2006
Hovedveileder: Gunnar Tufte, IDI

Per Kåre Otteren

Anvendelse av biologisk inspirerte metoder i musikk

Institutt for Datateknikk og Informasjonsvitenskap
Norsk Teknisk Naturvitenskapelige Universitet
7491 Trondheim, Norge

Sivilingeniør Datateknikk, Masteroppgave
Våren 2006

Veiledere Gunnar Tufte og Øyvind Brandsegg

Sammendrag

Sammendrag

Det eksisterer mange kjente metoder for å etterligne menneskelig tenkegang eller utøve kunstig intelligens. Et interessant anvendelsesområde for slike metoder er i musikken, siden dette er en kunstform uten klare regler eller absolutte sannheter. Uten regler og sannheter er det heller ikke en korrekt måte å lage musikk på.

Arbeidet beskrevet i denne rapporten går ut på å bygge et system for å tolke musikalsk input fra en musikkutøver. Til dette blir det forsøkt benyttet nevrale nettverk. Et nevralt nettverk kan utføre denne oppgaven ved å trenes opp til å gjenkjenne hvilke akkorder som blir spilt. Systemet blir testet ved å koble det sammen med programmet *Happysound* beskrevet i [22]. Dette er et program som benytter fuzzy logikk for å genererer musikalske soloer. Ved å koble et nevralt nettverk til dette fåes et system som tar inn akkompagnement fra en medspiller og spiller av soloer som passer til dette. Resultatene som presenteres viser at nevrale nettverk er en god metode for musikalske formål men at det legges begrensninger av hvor mange forskjellige akkordrepresentasjoner ett enkelt nevralt nettverk klarer å kjenne igjen. Til slutt fremlegges en utvidelse for å overkomme disse begrensningene.

Oppgaveteksten

Oppgaveteksten til dette arbeidet er som følger:

> Dette prosjektet er relatert til musiker Øyvind Brandtseggs arbeid. Ideen med å anvende metoder fra kunstig intelligens er for å gjøre det mulig å anvende sanntids komponering for live fremføring. Sanntid er viktig da systemet skal benytte improvisasjon basert på interaksjon med musikeren. Prosjektet består av to deler. Først, finn ut hva andre gjør innen område med å anvende KI metoder for komposisjon med fokus på sanntid. Deretter forsøk å anvende metoder funnet i samarbeid med Øyvind Brandtsegg.

Forord

> Masteroppgaven i sivilingeniørstudiet har et omfang på 30 studiepoeng og gjennomføres i studiets 10. og siste semester. Generelt kan det sies at formen på en masteroppgave ikke er ulik formen på prosjektdelen av fordypningsemnet i 9. semester. Den største forskjellen vil være oppgavens omfang, samt at det forventes at studenten har nådd enda et trinn videre i faglig modenhet i 10. semester, sammenliknet med det foregående.

I denne rapporten beskrives arbeidet som er gjort i forbindelse med Masteroppgaven ved sivilingeniørstudiet i datateknikk ved NTNU.

Arbeidet beskrevet i rapporten er en videreføring av Prosjektarbeid høsten 2005. Her blir det nevnt noen forslag til fremtidige arbeider, og det er et av disse som er forsøkt gjennomført. Arbeidet har vært inspirerende både teknisk og musikalsk, og jeg har lært mye.

Jeg vil rette en stor takk til min veiledere Gunnar Tufte, og Øyvind Brandtsegg som har hjulpet meg med arbeidet.

Per Kåre Otteren
12. juni 2006

Innhold

Kapittel 1

Introduksjon

Å anvende biologisk inspirerte metoder innen musikalske arenaer har vært gjort i mange sammenhenger. Det er benyttet evolusjonære metoder som for eksempel *genetiske algoritmer* for å generere både enkle jazz soloer [4] og hele komposisjoner [13][15]. *Fuzzy logikk* er benyttet blant annet for å forbedre omgjøring fra presis definert Midi styrke til musikalsk dynamikk [8]. Også *nevrale nettverk* er benyttet i flere anledninger til for eksempel å gjenkjenne mønstre i musikk som kan hjelpe en til å klassifisere enten instrument eller sjanger [7][26], eller for å registrere om det er en akkord eller en enkelt tone som spilles av[18].

At disse metodene er inspirert fra biologien betyr at de har hentet sin inspirasjon fra naturlige (og biologiske) fenomen. De prøver å etterligne intelligensen som finnes hos mennesket. Nevrale nettverk henter sin inspirasjon fra hjernen, og prøver å etterligne måten den er koblet sammen på i håp om å kunne oppnå samme evne til gjenkjenning og opplæring. Evolusjonære metoder henter inspirasjonen fra selve utviklingen av livet på jorden og den darwinistiske læren om utvelgelse og reproduksjon.

Også innen emnet *musikk transkribering*, altså konvertering fra musikk til tekstlig beskriving (noter), er det ved flere anledninger benyttet nevrale nettverk i sammenheng med andre metoder for å gjenkjenne toner. De vanligste anvendelsene er for å detektere polyfoni [16], eller for å kjenne igjen en enkelt tone (benytter da et eget nettverk for hver tone som skal detekteres) [18].

Mitt arbeide går ut på å benytte et nevralt nettverk for å oversette input fra menneskelig akkompagnement til en program som genererer musikalske soloer. Det nevrale nettverkets oppgave er å kjenne igjen hvilken toneart det spilles i slik at programmet kan generere soloer som passer til akkompagnementet.

Rapporten gir i kapittel 2 bakgrunnsteori om berørte emner i prosjektet. Det forklares først teori om musikk og hvilke av musikkens egenskaper som best kan benyttes til ønskede formål, og videre nevnes en del eksisterende arbeid innen musikk transkribering. Her nevnes både transkribering av monofone lyder som har mange gode eksisterende metoder, og det mer krevende feltet med transkribering av polyfoniske lyder. Videre gies en innføring i nevrale nettverk og dets anvendelser innen musikk.

Videre i kapittel 3 og 4 gies en oversikt over det utviklede systemet, og det beskrives

de valg og implementasjoner som er gjort for å oppnå ønskede resultater. Kapittel 5 kan vise til gode resultater innen en del områder og disse diskuteres så i kapittel 6 med tanke på nytteverdi og videre arbeid.

Kapittel 2

Bakgrunn

2.1 Musikkteori

Lyd er i utgangspunktet et analogt fenomen og for å kunne behandle dette digitalt er det en del egenskaper man bør være klar over. Jeg vil gi en kort innføring i hvilke egenskaper en lyd har og på hvilke områder en lyd skiller seg fra en annen.

2.1.1 Frekvens,amplitude,klang

Det er tre grunnleggende parametere som bestemmer hvordan en lyd oppfattes av øret: Amplityde, frekvens og klangfarge[30].

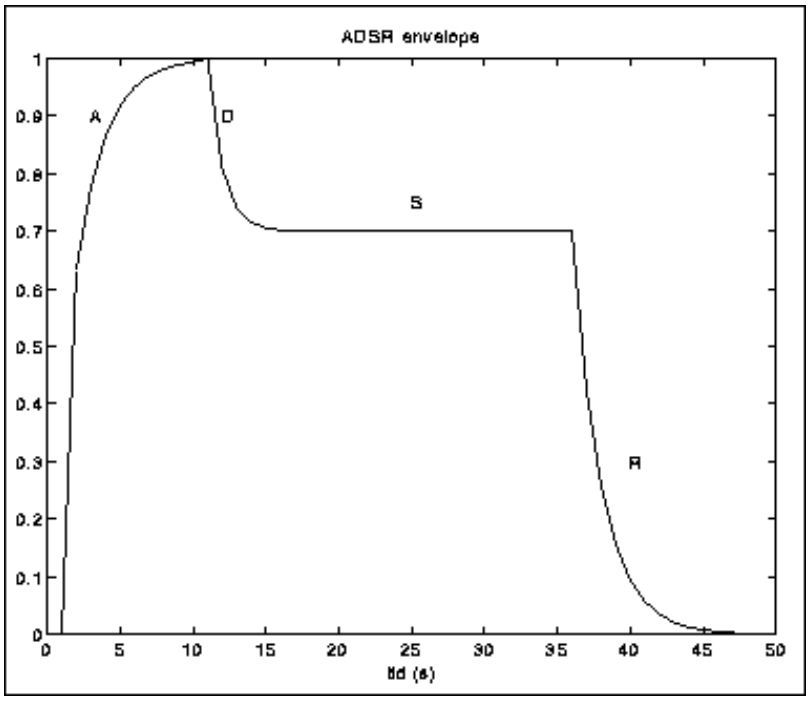
Amplituden er det vi oppfatter som lydstyrke. Større amplitude gir kraftigere lyd. Men det er slik at øret oppfatter lydstyrke logaritmisk så sammenhengen mellom lydstyrke og amplitude er ikke lineær. 3 dB økning i amplitude gir dobling i opplevd lydstyrke[1]. En viktig konsekvens for datamusikere er at man ikke kan øke amplituden lineært for å få en naturlig cresendo. Man må isteden benytte en eksponentiell funksjon.

Frekvensen angir den opplevde tonehøyden. For at vi skal kunne oppleve en tonehøyde på en lyd må lydbølgene ha en periodisitet. Denne perioden angir lydens frekvens. Denne angir antall gjentakelser per sekund og måles i Hertz. Heller ikke sammenhengen mellom frekvens og oppfattet tonehøyde er lineær. En dobling i frekvens oppleves som et tonesprang på en oktav, eller 12 halvtoner. Altså blir det lengre avstand mellom halvtonene jo høyere opp i registeret man spiller. For å gå opp et halvtonetrinn må man altså multiplisere frekvensen med faktoren $\sqrt[12]{2}$. Dersom lydens kurveform ikke er periodisk vil den enten oppfattes som en dissonant akkord, eller som støy uten noe bestemt tonehøyde dersom kurven er "bare rot".

[1] Hvis endringen er 1 dB, er størrelsen endret med en faktor $10^{1/10}$, eller ca. 1,26. ($1,26^3$=2)

Klangfargen bestemmes av lydkurvens form. Denne fører til at to lyder med samme amplitude og frekvens likevel kan høres forskjellig ut. Matematikeren Fourier viste at enhver repeterende kurveform kan genereres ved å addere et antall sinusoide deltoner som alle er et heltallsmultiplum av grunnfrekvensen[30]. Slik er det også i musikken, at forskjellige instrumenter gir lyden forskjellig klangfarge ved at de harmoniske overtonene blir ulikt representert av ulike instrumenter. En harmonisk overtone er gitt ved et heltallsmultiplum av frekvensen til grunntonen(1. harmoniske).

Elle disse egenskapene har en relatert psykoakkustisk egenskap hos oss mennesker[5]. Amplituden oppleves som lydstyrke, frekvens oppleves som tonehøyde og kurveformen oppleves altså som klang. I tillegg kan det være en fjerde parameter, nemlig lokasjon på lydkilden, som vil være med å farge vår lytteopplevelse.



Figur 2.1: En naturlig envelope ADSR

2.1.2 Envelope

Alle analoge musikkinstrumenter fungerer slik at amplituden vil forandre seg i løpet av hver tone. Når man slår an en tone på f.eks en gitar vil den først få en rask økning i amplituden og så dø ut langsomt. Dette kalles envelopen til en tone. En envelope deles gjerne inn i fire deler; Attack, Decay, Sustain, og Release (figur 2.1). Disse 4 delene forteller om tonens amplitudevariasjon. Amplituden for en tone når sin høyeste

verdi i attack perioden, før den synker litt i decay perioden og holder seg gjennom sustainperioden, og dør deretter ut i release perioden.

Gitarens envelope

For å finne et godt bilde av en lyd er det mest gunstig å finne sustain perioden i ADSR envelopen. Dette er tiden like etter anslaget når tonene holder et konstant nivå før de fader ut. Lyder fra en gitar får ikke en like fin envelope som den vist i figur 2.1 men den består stort sett av et raskt anslag og så en konstant release etter det som vi kan se fra bølgeformen i figur 2.2.

Figur 2.2: Envelope for gitar

2.1.3 Sampling

Lyd er en analog signaltype så for å behandle den i en datamaskin trenger vi å omforme den til digital form. Dette gjøres ved sampling. Et analogt signal er både tidskontinuert og amplitudekontinuert. Det vil si at signalet kan endre seg kontinuerlig over både tid og amplitude. Signalet kan ha enhver amplitudeverdi innenfor intervallet som bestemmes av systemets dynamiske område[30]. For å benytte dette i en datamaskin trenger vi å diskretisere signalet. Til dette må vi bestemme oss for en samplingsrate og en kvantifiseringsbredde. Samplingsraten sier hvor ofte vi skal måle signalets verdi, og kvantifiseringsbredden bestemmer med hvor stor nøyaktighet vi kan representere den. På en vanlig CD benyttes 44.1 KHz samplingrate og 16 bits nøyaktighet.

En grunnleggende regel i samplingsteorien er at samplingsfrekvensen må være mer enn dobbelt så høy som den høyeste frekvenskomponenten vi ønsker å kunne registrere. Dette kommer av Shannon og Nyquist's samplingsteorem [27][21]. Det vil si at en cd med 44.1 Khz samplingsrate kan registrere frekvenser opp til 22 Khz. Verdier med høyere frekvens enn dette vil føre til støy fordi de ikke blir fullstendig fanget opp, så derfor benyttes et lavpass filter for å filtrere vekk høyere frekvenser enn det man kan registrere før man foretar samplingen. Dersom dette ikke gjøres vil man oppleve det som kalles *folding*.

Dette er at frekvenskomponenter som ligger over halve samplingsfrekvensen blir speilet nedover i spekteret og danner falske frekvenser i resultatet.

2.1.4 Fouriertransform og FFT

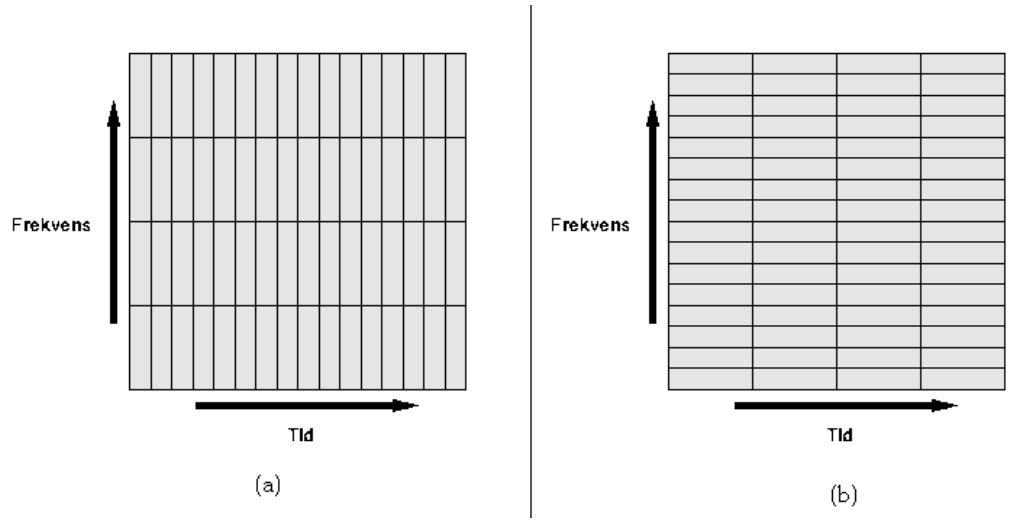
Analyse av lyd kan gjøres enten i tidsdomenet eller i frekvensdomenet. I tidsdomenet ser man på samplingsverdiene direkte, altså lydkurven som består av amplitudeverdier over tid. I frekvensdomenet ser man på signalstyrken til enhver frekvens i et gitt tidsrom (korttids fouriertransform), evt som et gjennomsnitt over lengre tid. For å overføre en samplet lyd til frekvensdomenet benytter vi fouriertransformasjon. Dette er egentlig en veldig tung regneoperasjon, men det har kommet flere veldig effektive algoritmer for å løse dette. Den vanligste er Fast Fourier Transform eller FFT. Når man benytter korttids FFT må man velge hvor mange punkter man vil ha i FFT'en. Det vil si hvor mange samples som skal analyseres. Dette antallet får betydning for analysens oppløsning i både tid og frekvens. Når man benytter en FFT med N samples deler man frekvensspekteret til lyden inn i N/2 bånd, og får en verdi for hver av disse. Dette fører til at jo flere punkter vi har i FFT'en, jo større oppløsning får vi i frekvensdomenet. Med en samplingfrekvens på 44.1 kHz, vil N=512 gi en oppløsning på 86 Hz, mens N=1024 vil gi en oppløsning på 43 Hz. Men når vi øker N må vi analysere stadig lengre lydbiter, og det gjør at vi får en dårligere oppløsning i tid. 44.1 kHz og N=512 vil gi 86 spektre i sekundet (hver lydbit blir ca 12 ms lang), mens N=1024 vil gi 43 spektre i sekundet, så hver lydbit blir ca 23 ms lang. Vi må altså gjøre en avveining mellom presisjon i tid eller frekvens. I figur 2.3 er det vist hvordan vi kan dele opp frekvens-tid-planet på ulike måter ved å justere N. Vi kan ikke plassere en hendelse i tid og frekvens med større nøyaktighet enn en boks. Disse boksene har alltid samme areal, slik at hvis vi øker nøyaktigheten i frekvens (boksene blir lavere) minsker nøyaktigheten i tid (boksene blir bredere)

Oftest er det ønskelig å sette N så liten som mulig for å få best temporal oppløsning, men ikke så liten at frekvensbåndene gaper over mer enn en deltone i lyden.

2.2 Musikk transkribering

Å lytte til musikk og skrive ned notasjonen for denne kalles transkribering av musikk. Å kunne gjøre dette automatisk har vært en interesse for både musikere og datateknikere i over 35 år.[19].

Transkribering av musikk er prosessen med å trekke ut en symbolsk representasjon for den musikalske informasjonen som ligger i en lydsekvens[24]. Dette kan så benyttes for å spille av det samme musikkstykket igjen. Men selv om man for å kunne gjengi et musikkstykket eksakt slik som originalen trenger å detektere/representere både amplituden, frekvensen og klangfargen (og kanskje også retning på lyden), så vil man i de fleste tilfeller være fornøyd med å kun representere en av dem. Nemlig grunnfrekvensen til de spilte tonene. Man vil da ikke få en eksakt kopi av stykket med identiske psykoakustisk egenskaper, men man vil få en representasjon som kan benyttes til å fremføre det samme stykket på nytt. Denne representasjonen vil i de fleste tilfeller være som noter.



Figur 2.3: FFT inndeling

2.2.1 Monofonisk transkripsjon

Monofonisk transkripsjon, også kalt *pitch tracking*, betegner transkripsjon av musikk der kun en tone spilles av gangen. Dette er et modent felt og kan gjøres med ganske god nøyaktighet og flere gode algoritmer finnes. Dette kan gjøres enten i tidsdomenet med for eksempel teknikker basert på autokorrrelasjon eller nullpunkts-kryssing. Eller i frekvensdomenet ved bruk av teknikker basert på Fouriertransformasjon eller cepstrum[6].

2.2.2 Polyfonisk transkripsjon

Polyfonisk transkripsjon er mye vanskeligere både grunnet delte overtoner og usikkerhet om hva som er grunntoner eller harmonier. Men det er gjort mange forsøk på dette med mer eller mindre suksess. På midten av 1970-tallet laget Moorer et system for transkribering av duetter[20]. Systemet hans var begrenset til musikk med kun to instrumenter med forskjellig klang og frekvensområde. Systemet var også begrenset i at det ikke klarte å finne oktaver, eller andre intervaller der den ene tones grunnfrekvens sammenfalt med den andres overtoner. Senere har Hawley beskrevet et system som benytter en differsensiell spektrum analyse for å transkribere pianostykker[12]. Dette var suksessfullt innenfor et begrenset område. Blant annet fungerte det fordi piano toner ikke modulerer i tonehøyde. Dette systemet viser at man kan oppnå relativt gode resultater ved å begrense rekkevidden av de signaler som skal behandles og spesielle karakteristikker ved disse. Men dette fører igjen til at systemene ikke vil kunne fungere for generelle tilfeller. Først på slutten av 1990-tallet har det kommet systemer som er i stand til å transkribere mer en to samtidige toner. Og selv dette er kun i begrenset omfang for generelle tilfeller[19][16]. Disse systemene benytter et såkalt *blackboard system*, som er en relativt kompleks problemløsningsmodell som baserer seg på flere kilder med kunnskap som alle poster en delvis løsning til et problem på en tavle(blackboard)[9]. Etter hvert som det kommer flere delvise løsninger vil kunnskapskildene komme med nye løsninger

basert på disse helt til en eventuell fullstendig løsning er funnet.

Slike kunnskapskilder kan i denne sammenhengen være små systemer som for eksempel:

- bestemmer om det er en eller flere toner som spilles samtidig
- finner topper i frekvensspekteret
- finner korrelasjoner
- detekterer oktaver
- finner forslag til noter
- fjerner tydelige uriktige noteforslag

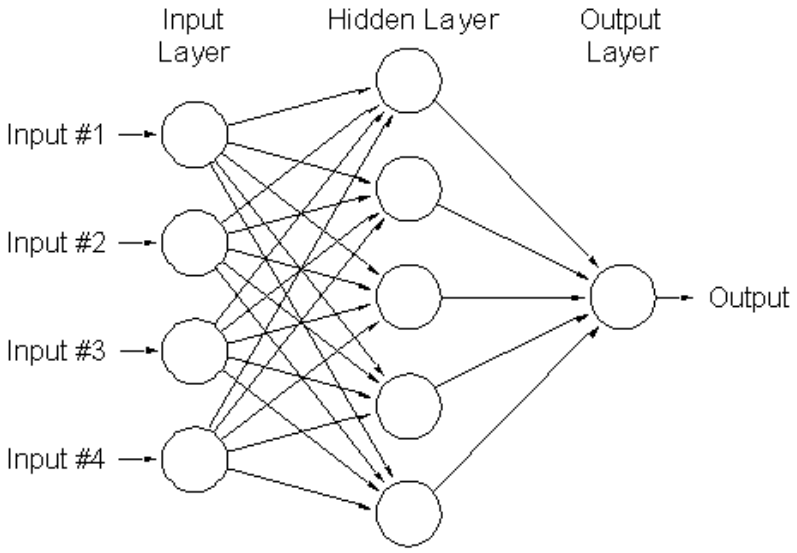
I 2000 beskriver Klapuri,Virtanen og Holm et system for robust multipitch estimering som baserer seg på iterativ estimering og separering av de mest fremtredende stemmene i en lydbit[2]. Dette systemet gir bedre resultater enn gjennomsnittet fra 10 trente musikere, og fungerer tilfredsstillende på opp til 6 samtidige lyder.

2.3 Nevrale Nettverk

Det finnes ulike definisjoner på nevrale nettverk, men de fleste vil si seg enig i at det inneholder et nettverk av prosesseringselementer som kan fremvise kompleks global oppførsel med kun enkel lokal prosessering. Dette vil si at det kun utføres veldig enkle operasjoner i hver node i nettverket, men alle disse tilsammen kan benyttes til å løse komplekse oppgaver. Inspirasjonen til denne teknikken kommer fra hjernens sentrale nervesystem og nevronene som utgjør systemets viktigste prosesseringselementer. Et nevralt nettverk består i så måte av noder med en enkel aktiveringsfunksjon som er koblet sammen til et nettverk. Et slikt nettverk er vist i figur 2.4. Dette består av 4 input noder som utfra en aktiveringsfunksjon bestemmer om koblingen frem til alle nodene i neste lag skal aktiveres eller ikke. Nodene i lag 2 benytter så sin aktiveringsfunksjon sammen med inndataen fra de foregående nodene til å aktivere sin link til den siste noden. Denne gir så ut en verdi ut fra sin funksjon og sine inndata.

Funksjonen i hver node bestemmer hvilket signal noden skal gi ut basert på signalene den får inn. Vanlige benyttede funksjoner er sigmoid, eller hyperbolsk tangent. Disse gir ut en verdi mellom 0 og 1 eller -1 og 1 og denne kan så aktivere nodens kobling eller ikke basert på en enkel terskel. Figur 2.5 viser et nevron med n inputs med verdi x_n og vekt w_n. Disse verdiene blir så summert og dersom verdien er over terskelen vil nevronet gi ut en verdi y til alle noder videre fremover i nettverket.

Et viktig poeng med et slikt nettverk er at det skal være adaptivt. Det vil si det skal tilpasse seg, eller lære hvordan det skal fungere. Dette skjer for eksempel ved at funksjonen i nodene endrer terskelen for når de skal gi ut et signal eller ved at nettverket endrer vekting på hvor mye forskjellige sammenkoblinger har å si for resultatet. På denne måten kan man trene opp nettverket til å kjenne igjen spesielle mønstre. Og det er



Figur 2.4: Et feed forward Nevralt Nettverk

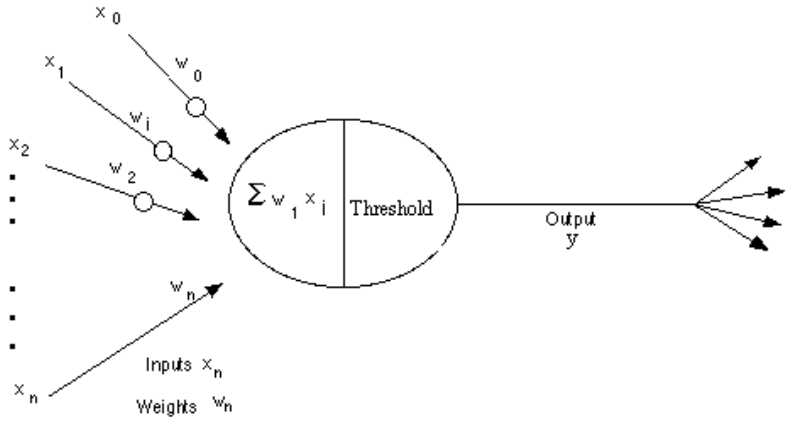
nettopp dette som er den store fordelen med nevrale nettverk sammenlignet med andre beregningsmodeller.

Nevrale nettverk er spesielt godt egnet i situasjoner der vi

- Ikke kan formulere en algoritmisk løsning
- Kan finne/lage mange eksempler på ønsket oppførsel
- Trenger å finne en struktur fra eksisterende data

2.3.1 Feed Forward, Back Propagation Neural Network

Det finnes mange forskjellige typer implementasjoner av nevrale nettverk[28]. Nettverket jeg har benyttet i min oppgave er av den enkleste typen. Nemlig et *feed-forward* nevralt nettverk[29]. I et slikt nettverk går informasjonen kun fremover i nettverket, så det er ingen sykler eller løkker i nettverket. Nettverket i figur 2.4 er et slikt nettverk, med 4 input noder, 5 skjulte noder og 1 utput node. Det kan benyttes flere lag med skjulte noder mellom input og utput lagene. Feedforward nettverk som dette kalles også *multilayer perceptrons* (MLP), da de består av flere lag med noder(perceptroner).



Figur 2.5: Et nevron

Det *Universelle Aproksimersingsteoremet* sier at enhver kontinuerlig funksjon som mapper et intervall av reelle tall til et ut-intervall av reelle tall kan tilnærmes tilstrekkelig med et MLP med kun ett skjult lag[17]. Dette gjelder dog kun for en begrenset klasse aktiveringsfunksjoner, for eksempel sigmoid funksjonen.

Den mest benyttede opplæringsteknikken for slike fler-lags nettverk er *back propagation*, også kalt *generalised delta rule*[11]. Denne fungere ved at utverdiene fra nettverket blir sammenlignet med de korrekte svarene for en gitt input, og man får en feilverdi som så mates tilbake igjennom nettverket (tilbake propagering). På bakgrunn av denne feilverdien justerer nettverket vektene på hver kobling slik at feilen blir lavest mulig. Etter å ha gjentatt dette for en stor mengde treningssykler vil man kunne redusere feilen nok til at nettverket kommer med det riktige svaret for en gitt mengde testverdier. Det som gjerne er ønskelig er at nettverket skal kunne generalisere ut fra det gitte treningssettet. Man ønsker ikke at nettverket kun skal godta identiske innverdier, men at det skal kunne kjenne igjen mønstre i de verdiene man trener med. Derfor er det viktig å ikke overtrene nettverket, slik at det har spesialisert seg på helt korrekte verdier.

Forløpet til en opplæring ved backpropagation vil være som følger:

1. Mat nettverket med et treningssett og la nettverket beregne en verdi på sine utput noder
2. Beregn feilen e_k ved
$$e_k = d_k - y_k \tag{2.1}$$
for hver utnode k, der d_k er ønsket utverdi og y_k er beregnet verdi fra 1.
3. Beregn så en justeringsverdi δ_k for hver node ved
$$\delta_k = e_k g'(y_k) \tag{2.2}$$
som benyttes for å justere vektene på koblingene mellom nodene. $g'()$ er her den deriverte av aktiveringsfunksjonen for noden.

4. Vi kan nå beregne δ_j for det foregående laget. Dette gjøres ved

$$\delta_j = ng'(y_j) \sum_{k=0}^{K} \delta_k w_{jk} \tag{2.3}$$

   der K er antallet noder i dette laget, n er opplæringsrate parameteren (som avgjør hvor mye vekten skal justeres for hvert pass) og w_{jk} er vekten på koblingen mellom node j og k

5. Fra disse δ verdiene kan vi nå finne endringen for hver kobling av

$$\Delta w_{jk} = \delta_j y_k \tag{2.4}$$

   som benyttes til å justere vektene ved

$$w_{jk} = w_{jk} + \Delta w_{jk} \tag{2.5}$$

6. Disse punktene kjøres så til man når en ønsket nedre verdi for den totale feilen, eller et annet stop kriterium.

Det første steget i denne fremgangsmåten er et *forward pass* da informasjonen her flyter fremover i nettverket, mens stegene 2-5 kalles *backward pass* da man propagerer bakover i nettverket for å oppdatere nodene. Formelen benyttet i eq2.3 er bare en av flere mulige. En annen ofte benyttet (og litt mer avansert) er *gradient descent*[3] som ikke benytter en opplæringsrate, men isteden et sett av mer avanserte parametere for å gjøre en kvalifisert gjetning på hvor mye vekten skal justeres.

2.3.2 Anvendelser innen musikk

Evnen nevrale nettverk har til å gjenkjenne mønster gjør det godt egnet til flere oppgaver innen musikk og lydbehandling. Nicholson, Takahashi og Nakatsu benytter nevrale nettverk til å kjenne igjen følelser i tale[14], og Scott benytter slike ettverk for å klassifisere musikalsk sjanger[26]. Eronen beskriver et system for å klassifisere instrumenter[10].

Enda mer interessant for mitt arbeide er Marolt sitt SONIC system, som benytter nevrale nettverk for å transkribere piano musikk[18], og Pertusa og Inesta sitt forsøk på et generelt musikk transkriberings system basert på nevrale nettverk og spektral mønster identifikasjon[23]. Marolt benytter i sitt SONIC 76 adaptive nevrale nettverk som hver er trent opp til å kjenne igjen kun en tone. På denne måten dekkes alle toner fra A1 til C8. Dette har gitt ganske gode resultater for piano musikk. Pertusa og Inesta benytter noe av den samme teknikken, der de deler opp frekvensspekteret i 94 spektralbånd i en logaritmisk skala av halvtoner som strekker seg fra G#0 til F8 (50-10600 Hz). Det er så den totale energien i hvert av disse båndene som gies inn til nettverkets 94 inputnoder. Nettverket har også 94 utverdier som angir hvilke toner som er representert i musikkbiten. Dette nettverket gir gode resultater når nettverket blir trent opp med samme instrument som det trenes med, men resultatene degraderer signifikant for instrumenter med ulik klang.

Kapittel 3

Nettverket

3.1 Feed forward, backpropagation

Jeg valgte å benytte meg av et nevralt nettverk av typen *feed forward* med ett skjult lag fordi dette er den mest benyttede typen og det finnes gode implementasjoner i de fleste programmeringsspråk. Her flyter informasjonen kun fremover og det er tilstrekkelig for min applikasjon. Videre valgte jeg å benytte backpropagation for opplæring siden denne har vist seg å kunne gi rask konvergering til et tilfredsstillende minimum av feil. Igjen er også dette vanligvis foretrukne opplæringsmetoden for feed forward nettverk.

Jeg benytter et Feed Forward, Back Propagation Nevralt Nettverk, som er implementert i python av Neil Schemenauer og finnes i det offentlige domenet på www.python.org [25].

3.2 Inndata

Inndataen til nettverket er musikksignaler, og her er det valgt å benytte frekvensdomenet av musikken da det er nettopp frekvensen til musikken som bestemmer hvilken toneart det spilles i og dermed er interessant for oppgaven. For denne spektrale analysen av musikken benyttes et bibliotek laget av KTH i Sverige, avdeling for Tale, Musikk og Hørsel, kalt The Snack Sound Toolkit"[1]. Dette er et bibliotek til Tcl/tk og Python. For å begrense datamengden bestemte jeg meg for å kun benytte 16Khz samplingsfrekvens. Dette fører til at jeg kun får samplet lyder i området 0-8000 Hz, men dette er tilstrekkelig for å få et brukbart lydbilde og jeg får med minst 3 overtoner for selv de lyseste tonene man kan spille på en gitar (4-6 overtoner for de vanligste notene). Det behøves heller ikke bedre kvalitet med tanke på at det som taes opp ikke skal benyttes videre til annet en å bestemme parametere.

3.3 Utdata

Utdataene fra nettverket er en akkordrepresentasjon som videre skal benyttes for å sette parametere i programmet Happysound. Nettverket er begrenset oppad til å kunne lære

inn max 16 akkorder. Dette gies av 4 utnoder som tilsammen gir en 4 bits verdi ut.

3.4 Tilpasning

Etter den innledende analysen var altså forutsetningene slik:

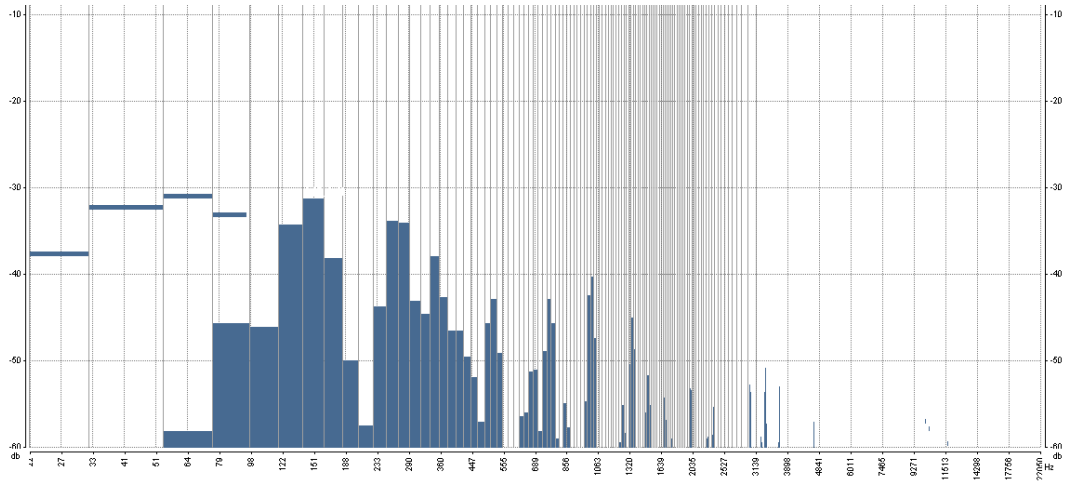
> Et feed forward nevralt nettverk med backpropagation opplæring som basert på x inputnoder og x noder i ett skjult lag skal gi ut en 4 bits verdi som representerer en akkord. Innverdiene skal komme fra en FFT analyse av en lydbit.

Så utfordringene ligger i å tilpasse dette riktig. Hvor mange punkter som benyttes i FFT analysen vil gi føringer for både antall innverdier til nettverket og oppløsningen på frekvensspekteret. Det vil også være vesentlig å bestemme når i lyden analysen skal taes. Skal den gjøres kontinuerlig i små biter, et gjennomsnitt over en lengre periode, eller bør man lete etter anslaget til akkorden. Videre må det vurderes om hele frekvensspekteret trenger å være med i analysen eller om det holder med for eksempel de nederste 3 Khz.

3.5 Analyse

I starten av arbeidet ble det gjort mange forskjellige analyser for å prøve å finne de beste løsningene på de nevnte tilpasningene. Ikke alle analysene gav nyttige resultater. En innledende analyse gikk ut på å finne hvor i frekvensspekteret mesteparten av informasjonen lå. Figur 3.1 viser et øyeblikksbilde av frekvensspekteret til en gitar akkord. Her kan vi se at utslagene er klart størst innen gitarens mest brukte toner som strekker seg fra den laveste E tonen på 82 Hz til 5 bånd på den lyse E strengen som ligger på 440Hz. Videre analyse av spekteret viste at verdier over 1000Hz sjelden fikk høyere amplitude enn -50 dB, og de fleste overtonene forsvant veldig raskt etter akkordens attack.

Selv om de største utslagene kom innenfor et veldig begrenset område av frekvensspekteret viste det seg at utslagene også var veldig like for de forskjellige akkordene. Alle akkordene inneholder mange av de samme tonene, og det var ofte det samme frekvensbåndet som fikk høyest verdi hos flere ulike akkorder. Derfor er det viktig å ta med større deler av spekteret for å finne forskjellene blant akkordene. En videre analyse som da ble gjort var å finne ut når det var størst differanse i datasettene som ble generert av FFT analysen ved forskjellige verdier. Vedlegg A.1 viser et resultat fra en slik analyse. Det som ble gjort var å utføre FFT analyse på tre forskjellige akkorder, en A dur og to forskjellige G dur akkorder. Dette ble gjort for FFT verdier mellom 8 og 4096 (kun 2'er potenser). Deretter ble det regnet ut differansen i verdiene mellom de to forskjellige akkordene(A-G) og mellom de to like akkordene (G-G). For hver FFT verdi ble dette regnet ut først med alle utverdiene, deretter halvparten (nederste del), deretter en fjerdedel, og så videre. Det ble alltid benyttet den nedre delen. Ønsket resultat fra dette var å finne ut om det var noen inndelinger som gav større forskjell enn andre og om det behøvdes å benytte hele frekvensspekteret (0-8Khz) eller om det hold med bare en del av det. Denne analysen gav ingen klare resultater men den gav en pekepinn om at det



Figur 3.1: Analyse av frekvensspekter for akkord spilt på gitar

kanskje ville være tilstrekkelig å benytte nedre halvdel(0-4Khz) eller fjerdedel(0-2Khz) av frekvensspekteret. Figur 3.2 viser et utdrag fra testen. Resultatet er fra FFT lengde 512 og 1024, hvor vi ser at 0-4 gir generelt mindre differanse mellom 2 akkorder, men vi får en større forholdsvis differanse (Tot=A-G*100/G-G).

```
FFT: 512
Alle     256 verdier (0-8Khz): A-G: 8.4 , G-G: 7.6 Tot: 109.8
Nederste 128 verdier (0-4Khz): A-G: 7.3 , G-G: 6.4 Tot: 113.3
FFT: 1024
Alle     512 verdier (0-8Khz): A-G: 8.2 , G-G: 7.6 Tot: 108.8
Nederste 256 verdier (0-4Khz): A-G: 7.1 , G-G: 6.2 Tot: 113.1
Nederste 128 verdier (0-2Khz): A-G: 6.8 , G-G: 5.9 Tot: 114.6
```

Figur 3.2: Differanse ved forskjellige FFT verdier

Det ble også gjort noen tester og analyser for å finne den beste måten å registrere en akkord på. Siden en gitarlyd ikke har noen tydelig sustain periode ble det naturlig å prøve å finne toppen av attacken til akkorden og benytte tiden rett etter dette for å hente ut frekvensspekteret. Denne toppen kan man finne i en lydfil som inneholder en enkelt akkord ved å lete etter posisjonen med høyest amplitude. Ved kontinuerlig avspilling blir dette vanskeligere da man ikke lenger vet når man skal lete etter en topp. Metoden som fikk best resultater her var å lete etter en akkord ved gitte intervaller. Da ble det søkt igjennom siste avspilte intervall og den høyeste amplitudeverdien der ble benyttet som topp. På denne måten kan man risikere å miste noen akkorder dersom flere akkorder faller innenfor samme intervall, men ved å benytte små nok intervaller vil en likevel få god kontinuitet i akkordrekken.

Kapittel 4

Systembeskrivelse

4.1 Implementasjon

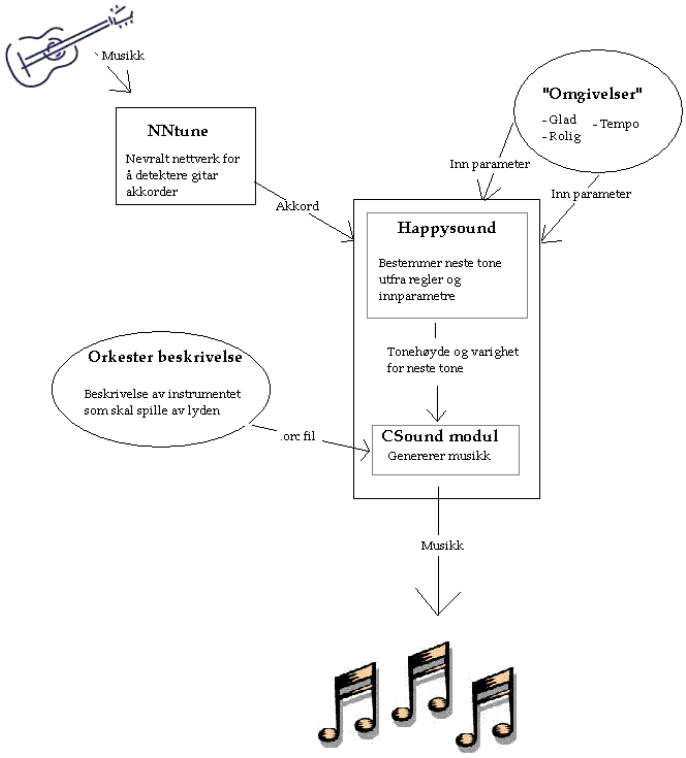
Ut fra testene og analysene beskrevet i foregående kapittel kom jeg frem til en del parametere som fungerte tilfredsstillende. I programmet som ble utviklet er det mulig å endre de fleste av disse parametrene men det er satt som standard å benytte en FFT lengde på 512 samples og benytte verdi nummer 2-128 av de 256 som kommer ut fra denne analysen. Dette tilsvarer verdier innenfor 80-4000 Hz. Det er i tillegg valgt å nulle ut (sette til -100) frekvensbånd med amplitudeverdier under -60dB. Frekvensanalysen gjøres hvert 400ms (kan endres) og benytter seg av at gjennomsnitt over 200ms. Hvor i den 400ms lange lydbiten analysen skal gjøres bestemmes utfra maxverdien innenfor intervallet. Dette settes som startposisjon. Ved denne metoden får man minst 200ms forsinkelse på bestemmelse av akkorden siden bestemmelsen gjøres over et intervall på 200ms, og dette kan i verste fall utgjøre at det spilles en 4.dels tone i feil toneart ved akkordskifte dersom tempoet er 120 slag i minuttet.

Standardverdiene blir altså:

- Lydene taes opp med en samplerate på 16000 samples som gir et frekvensspekter på 0-8000 Hz
- FFT lengde på 512 samples. FFT vindu på 256 samples
- Fra FFT analysen får jeg ut 256 frekvensbånd med tilhørende amplitude. Dette betyr en oppdeling på 8000/256=31,25Hz.
- Bånd nr 2-128, dvs frekvenser fra 60Hz til 4Khz, benyttes som input til nettverket. I tillegg settes alle verdier under -60 dB til -100 dB.
- FFT analyse over 200ms, resultatet er et gjennomsnitt av dette
- FFT analysen starter på det sterkeste punktet i lydenbiten (høyest amplitude).

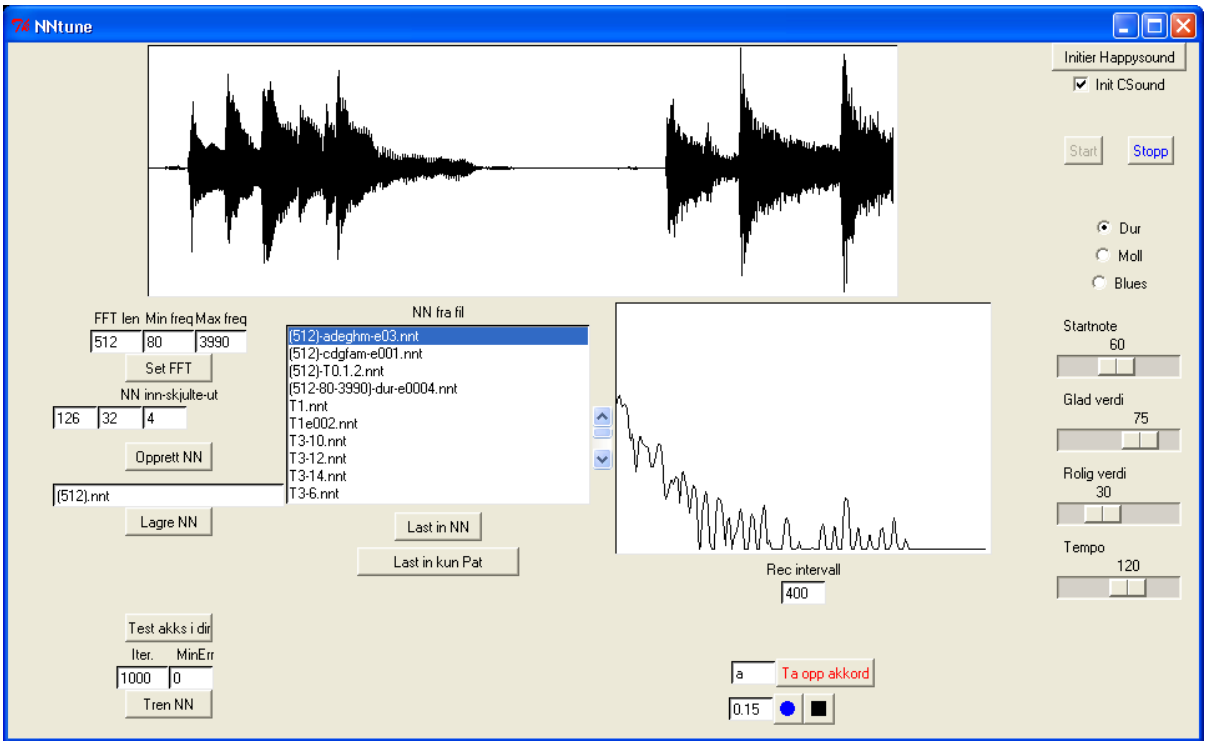
4.2 Tuned Happysound

Ved å koble sammen systemet beskrevet i denne rapporten med systemet som ble utviklet i forbindelse med Prosjektarbeid høsten 2005 er resultatet blitt programmet *Tuned Happysound*. Dette er et program som tar inn input fra en person ved at denne spiller på en gitar som er koblet til en pc, og denne innputten bestemmer hvilken toneart programmet skal generere sine soloer i. Figur 4.1 viser en modell over systemet. Systemet består hovedsaklig av 2 moduler (markert som firkantede bokser) som begge tar inn parametere utenfra for å bestemme sine resultater. Modulen *Happysound* er beskrevet i [22] og tar inn parametere som benyttes av et regelverk basert på fuzzy logic for å generere toner. I tillegg gies det inn informasjon om hvordan lyden som kommer ut skal høres ut ved hjelp av en orkesterdefinisjon. Den nye parameteren som er kommet til etter beskrivelsen i [22] er en akkordparameter. Denne bestemmer hvilken toneart programmet skal generere toner i. Denne parameteren kommer fra den andre modulen kalt *NNtune*. Denne modulen får igjen sin input fra et instrument som er koblet til systemet og gir inn musikksignaler.



Figur 4.1: En oversikt over systemet

Brukergrensesnittet til programmet er vist i figur 4.2. Det man ser her er et kontrollpanel for å sette parametrene for nettverket og innparametrene for Happysound. Det er også visninger for lydens bølgeform og frekvensspekter.



Figur 4.2: Tuned Happysound

Når man skal opprette et nettverk, bestemmer man hvor mange inn-noder man ønsker til venstre i grensesninttet. Dette gjøres indirekte ved at man bestemmer parametrene for FFT analysen, som dermed gir et bestemt antall verdier ut. Disse verdiene går på hver sin inngang i nettverket. Som standard er det benyttet en FFT lengde på 512 samples som gir ut 256 verdier. Disse verdiene representerer hele frekvensspekteret fra 0-8000 Hz (gitt av samplingsfrekvensen som er satt til 16Khz). Man kan så velge å benytte bare deler av dette, og her er standard å benytte fra 80 til 4000 Hz. Dette fører da til at man får 126 verdier (nr. 2-128 av de 256) som gies til nettverket. Man kan videre bestemme hvor mange skjulte noder man ønsker og deretter opprette nettverket ved å trykke på "Opprett NN" knappen.

Neste steg i prosessen er å opprette treningsmønstre for nettverket. Dette gjøres ved å ta opp lydstykker og registrere fftverdiene fra dem. Når man trykker på knappen "Ta opp akkord" starter opptakningen og for hvert 400ms (eller den angitte verdien i feltet "Rec intervall") gjøres en fftanalyse av lydbiten og det registreres ett sett på N verdier (N gitt av parameterne som er satt for nettverket. I dette eksemplet 126) som får navn gitt av feltet ved siden av knappen. Hvert opptak varer i 10 sekunder, som skulle gi ca

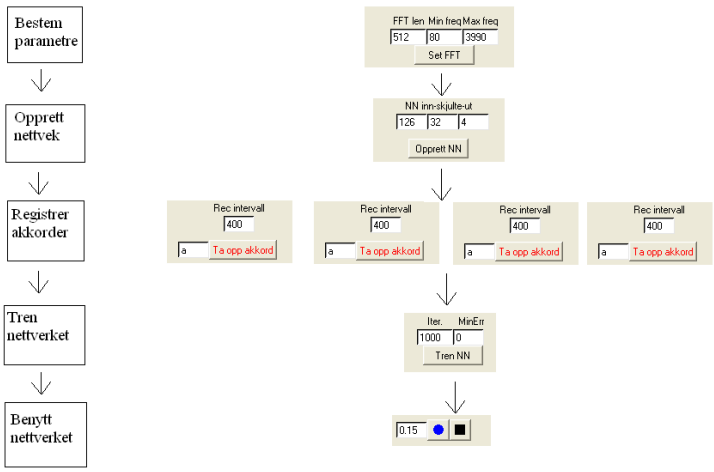
25 sett med verdier. En verdi registreres kun dersom signalet i aktuelle lydbit har en amplitudeverdi på over 5000. Man har nå ca 25 treningsverdier for èn akkord. Dette gjentar man så for så mange forskjellige akkorder man ønsker å trene opp nettverket med.

Videre blir neste steg å trene opp nettverket. Dette gjøres ved å trykke på knappen "Tren NN". Da utføres det trening ved backpropagation i antall iterasjone gitt i feltet over. For å trene opp 5 akkorder til brukbart nivå trengs kanskje 500 iterasjoner, men dette kan variere veldig. Man kan også sette en "MinErr" verdi som vil føre til at opptreningen stanser når den totale feilen er kommet under denne verdien. Et brukbart resultat bør ha en total feil på under 0,1.

Når nettverket er trent opp kan det benyttes til å registrere akkorder "live". Dette gjøres ved å trykke på den blå record knappen. Det gjøres da en fftanalyse i hvert recintervall (kan være forskjellig fra ved opptrening) og nettverket gir ut en antatt akkord for dette signaler dersom den kommer innenfor en gitt terskel av sikkerhet. Denne terskelen kan endres i feltet ved siden av record knappen. Standard er 0.15 som betyr at det er mindre enn 0,15 i differanse mellom utverdiene fra nettverket og bitkoden til akkorden ([0, 0.9, 1, 0.05] -> [0,1,1,0] med 0,15 i differranse).

For å benytte nettverket til å styre musikk generert av Happysound må dette initieres ved å trykke på knappen "Initier Happysound". Da startes Happysound modulen og når man trykker på "start" begynner den å generere musikk. Denne generer så musikk utfra sine regler og parametere, og resultater fra det nevrale nettverket vil da bli en innparameter til dette som alle andre. Dersom ikke Happysound er initiert vil akkordene kun skrives ut fortløpende, og modulen NNTune benyttes uavhengig av Happysound

Figur 4.3 viser flyten i programmet.



Figur 4.3: Programflyt

Kapittel 5

Resultater

5.1 Resultat av gjennomkjøring

5.1.1 Opplæring av 5 akkorder

Første gjennomkjøring var å trene opp nettverket til å kjenne igjen 5 forskjellige akkorder på kort tid (mindre enn 10 minutter). Det ble benyttet standardverdier for FFT og NN. Altså FFT lengde på 512 og nettverk med 126 innganger. Frekvensbånd fra 80-4000Hz. Etter opprettelse av nettverket ble det trent opp med akkordene C,D,G,F og Am. Registreringsintervallet er satt til 400ms så det resulterte i totalt 119 treningsmønstre fordelt på de 5 akkordene. Deretter ble nettverket trent opp med 300 iterasjoner, og det endte med en total feil på 0,02. Opptreningen tok 7 og et halvt minutt. Progresjonen er vist i figur 5.1. Error verdien som skrives ut er den totale feilen fra alle nodene etter en opptreningssykel. Sum av alle e_k fra ligning 2.1 i kapittel 2.3.1.

```
10:39:54 : Trener pat1... Lengde: 119
300 iterasjoner
0   error 47.758261
30  error 3.397538
60  error 1.119274
90  error 0.465581
120 error 0.244741
150 error 0.130809
180 error 0.070691
210 error 0.044509
240 error 0.033405
270 error 0.027331
10:47:23 : Ferdig. Error 0.023483
```

Figur 5.1: Progresjon ved opptrening av nettverk. Listen viser total feil ved hver 30. iterasjon.

Deretter ble nettverket testet. Dette ble gjort ved å spille akkordrekken i figur 5.2 på

gitaren og registrere resultatene.

```
|G       |D       |G       |D       |
|C   G   |C   G   |Am      |F       |
|G       |G       |
```

Figur 5.2: Akkorder spilt av ved testing

Tabell 5.1 viser resultatene fra 5 forskjellige gjennomspillinger. Programmet skriver ut en ny akkord hver gang den registrerer en verdi med feil mindre enn 0.15 og som er forskjellig fra forrige. Tabellen viser alle akkorder som ble skrevet ut og totalfeilen de ble registrert med. Uriktige akkorder vises i kursiv. Vi ser i tabellen at nettverket viser gangske gode resultater, men at det forekommer feil. Den vanligste feilen er mellom c og am, og dette er akkorder som er veldig like i virkeligheten. Der det er registrert feil akkord ble riktig akkord registrert kort tid etterpå i nesten alle tilfeller.

Akkord	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**
G	g 0.02	g 0.02	g 0.04	g 0.02	g 0.05
D	d 0.12	d 0.0	d 0.02	d 0.13	d 0.03
G	g 0.03	g 0.03	g 0.12	g 0.02	g 0.08
D	d 0.05	d 0.0	d 0.01	d 0.0	d 0.01
C	*am 0.15*	c 0.01	c 0.01	-	c 0.02 *am 0.04*
G	g 0.03	g 0.11	g 0.03	g 0.03	g 0.05
C	c 0.01	c 0.05	c 0.01	*am 0.01*	c 0.04 *am 0.1*
G	g 0.02	g 0.03	g 0.03	g 0.02	g 0.03
Am	*c 0.06* am 0.01	am 0.11	am 0.1 *c 0.05* am 0.01	am 0.03	am 0.03
F	f 0.05	*d 0.15* f 0.03	f 0.01	f 0.01	f 0.05
G	g 0.14 *d 0.01* g 0.1	g 0.03	g 0.09 *d 0.11* g 0.02	*d 0.04* *f 0.1* g 0.02	*d 0.14* *f 0.0* g 0.06

Tabell 5.1: Testresultater

En fullstendig utskrift av en gjennomkjøring med toner avspilt av Happysound finnes i vedlegg A.2

5.1.2 Opplæring av flere akkorder

Som vi ser av resultatene i kapittel 5.1.1 viser nettverket gode resultater ved opptrening av få akkorder. Dette går både raskt og med god nøyaktighet. Men det viser seg å være mer utfordrende med en gang nettverket skal lære seg flere forskjellige akkorder. Tabell

5.2 viser opptreningsresultater fra opptrening av 6 til 14 forskjellige akkorder. Her ser vi at ved 6 og 8 akkorder klarer nettverket å finne mønsteret i treningssettet. I tilfellet med 6 akkorder får den veldig gode resultater etter 300 iterasjoner, mens for 8 akkorder vil den trenge noen iterasjoner til for å få tilfredsstillende totalfeil. Men ved 10 akkorder og over ser vi at nettverket aldri klarer å finne mønsteret slik at den totale feilen hopper opp og ned, og ikke ser ut til å konvergere mot et minimum. Disse vil sannsynligvis aldri få en tilfredsstillende totalfeil uansett hvor mange iterasjoner det trenes.

	6 Akk	8 Akk	10 Akk	12 Akk	14 Akk
Lengde	141	189	238	285	333
it.nr					
0	35.590449	52.270441	60.221248	71.607093	86.182235
30	4.498658	11.920479	31.906270	44.374671	63.424750
60	1.966272	18.019484	27.355990	43.859910	69.628634
90	0.628701	15.982876	25.974381	48.392304	70.973820
120	0.135111	10.336219	25.744045	42.564610	119.254537
150	0.041868	5.150631	20.880262	56.821869	55.743967
180	0.023691	3.942091	19.555133	41.877052	41.645140
210	0.017087	1.925537	56.023565	35.081048	49.951063
240	0.013491	1.147984	52.342396	60.118295	72.474039
270	0.011194	0.777246	22.314771	48.228846	53.214921
300	0.009637	0.654238	24.376111	60.164709	49.568451
tid	9:22	12:31	16:56	19:00	22:54

Tabell 5.2: Progresjon ved opptrening av 6-14 akkorder. Lengde angir totalt antall akkorder i treningssettet og tiden er opptreningstiden i minutter

Det ble også gjort forsøk på å trene opp 14 akkorder ved å endre de andre parametrene. Men endring i størrelse på nettverket og FFT lengde utgjorde ingen forskjell på resultatene. Det som derimot utgjorde en forskjell var å kontrollere registreringen av opptreningsakkorder. Det ble gjort et forsøk på å spille inn en og en akkord til hver sin lydfil og så registrere dem ved å finne makspunktet i hver lydfil. På denne måten er man sikker på å finne startpunktet til akkorden og ikke risikerer å havne midt inne i en akkord som kan være tilfellet med liveregistreringen i det beskrevne programmet. På denne måten klarte nettverket å lære alle de 14 registrerte akkordene tilfredsstillende (totalfeil på 0.07 etter 300 iterasjoner) utfra et treningssett på 146 akkorder. Men nå er nettverket trent opp på "perfekte" akkorder og er dermed mindre tolerant for avvik som kommer ved live innspilling. Dette viste seg ikke å være brukbart til avspilling. Figur 5.3 viser resultater fra samme test som i figur 5.1. Men her er det altså ikke mulig å finne igjen akkordrekkefølgen. Med bakgrunn i dette ble det så gjort forsøk på å kontrollere registreringen av akkorder på lignende måte ved liveregistrering. Ved å sette rec intervallet til 5000ms kunne det spilles inn 2 akkorder i hvert 10 sekunders intervall og dermed være garantert at det kommer en hel akkord innenfor hvert rec intervall. Ved å gjøre dette fikk nettverket et treningssett på 28 akkorder som skulle gi et resultatsett på 14 akkorder. Ved 300 iterasjoner kom nettverket ned i en total feil på 1.05, men her flatet det ut og selv ved 700 iterasjoner til kom det aldri under 1.0. Dermed klarte det heller ikke å prestere godt nok ved avspilling.

16:14:33 start live
gm 0.03
d 0.07
em 0.14
null 0.08
d 0.05
a 0.07
g 0.11
c 0.14
f 0.02
c 0.12
dm 0.08
am 0.07
g 0.14
null 0.13
g 0.08
h 0.07
cm 0.13
d 0.01
c 0.04
gm 0.11
c 0.0
am 0.02
c 0.04
f 0.11
gm 0.1
g 0.03
16:15:41 stop

Figur 5.3: Testresultater

5.2 Analyse

Vi kan se fra resultatene i kapittel 5.1.1 at nettverket fungerer veldig bra til å lære seg 5 akkorder. Dette kan gjøres raskt og med god nøyaktighet. Videre ser vi i kapittel 5.1.2 at dette fungerer bra også for 6 og 8 akkorder. Men når resultatsettet blir større en 8 ser vi at nettverket får problemer med å finne mønsteret i treningssettet, og det klarer aldri å konvergere mot et bestemt resultat. Ved liveopptak av akkorder blir det for mange akkorder med for lite variasjon til at nettverket klarer å skille dem fra hverandre. Disse problemene kan komme av nettop den egenskapen til et nevralt nettverk som utnyttes i dette arbeidet. Nemlig å gjenkjenne mønstre. Det viser seg at når et nettverk får for mange forskjellige mønstre å kjenne igjen vil resultatene degradere. Dette er nok også årsaken til at det i de fleste anvendelser nevnt i kapittel 2.3.2 benyttes ett nettverk for hvert mønster som skal gjenkjennes. I dette arbeidets sammenheng vil det tilsvare å ha ett nettverk for hver akkord. Hvert nettverk vil da trenes opp til å kjenne igjen kun 1

akkord og vil så kunne gi et svar på om denne akkorden eksisterer eller ikke (med en tilhørende presisjon) ved en live fremføring.

Kapittel 6

Oppsummering

6.1 Fremtidig arbeid

En naturlig utvidelse av mitt arbeide er å benytte ett nevralt nettverk for hver akkord. Dette vil gi bedre resultater for større resultatsett (kan lære mange flere akkorder). Hvert nettverk trenes opp til å kjenne igjen en enkelt akkord og resultatet fra hvert nettverk blir om akkorden er tilstede eller ikke. Dette vil utnytte nevrale nettverks egenskap til å gjenkjenne mønstre på en mye bedre måte, da et nettverk nå vil ha kun et mønster å forholde seg til. Jeg vil anta at dette vil gi betraktelig bedre resultater.

6.2 Konklusjon

Arbeidet beskrevet i rapporten har gitt gode resultater for opptrening av opptil 8 ulike akkorder. Resultatene fra dette er gode nok til å kunne styre programmet Happysound beskrevet i [22]. Det vil si at man kan benytte nevrale nettverk for å gjenkjenne akkorder slik at Happysound genererer toner i riktig toneart. Jeg vil av dette konkludere med at nevrale nettverk er en god metode til dette formål med en begrensning i hvor mange akkorder som kan læres. Begrensingen kan trolig reduseres ved å utvide systemet til å inneholde ett nettverk for hver akkord som skal gjenkjennes.

Musikk er et veldig komplekst område og det er ikke en eksakt vitenskap. Derfor er biologisk inspirerte metoder meget nyttig i denne sammenheng ved at de kan håndtere slike utfordringer uten å ha en bestemt algoritmisk løsning.

Bibliografi

[1] The snack sound toolkit. [http://www.speech.kth.se/snack/]. KTH, Sverige, avdeling for Tale, Musikk og Hørsel.

[2] Jan-Markus Holm Anssi Klapuri, Tuomas Virtanen. Robust multipitch estimation for the analysis and manipulation of polyphonic musical signals. 2000. In Proc. COST-G6 Conference on Digital Audio Effects, December 7-9.

[3] P. Baldi. Gradient descent learning algorithm overview: a general dynamicalsystems perspective.

[4] J. Biles. Genjam: A genetic algorithm for generating jazz solos, 1994. Biles, J. A. (1994). GenJam: A genetic algorithm for generating jazz solos. In ICMC Proceedings 1994. The Computer Music Association.

[5] A. Bregman. Auditory scene analysis. 1990.

[6] Judith C. Brown and Miller S. Puckett. A high resolution fundamental frequency determination based on phase changes of the fourier transform. *J Acoust. Soc Am 94*, pages 662–667, 1993.

[7] A. Cemgil and F. Gurgen. Classification of musical instrument sounds using neural networks. In *Proc. of SIU97*, 1997.

[8] Peter Elsea. Musical applications of fuzzy logic.

[9] R.S. Engelmore and A.J. Morgan. Blackboard systems. 1988.

[10] Antti Eronen and Anssi Klapuri. Musical instrument recognition using cepstral coefficienta and temporal features.

[11] Kevin Gurney. *An Introduction to Neural Networks*. London: Routledge, 1997.

[12] Michael Hawley. Structure out of sound. 1993. Ph. D Thesis.

[13] A. Horner and D. E. Goldberg. Genetic algorithms and computerassisted music composition., 1995. Technical report, University of Illinois.

[14] K. Takahashi J. Nicholson and R. Nakatsu. Emotion recognition in speech using neural networks. 2000.

[15] B. Jacob. Composing with genetic algorithms, 1995. In Proceedings of the 1995 International Computer Music Conference, Banff, Alberta.

[16] Giuliano Monti Juan P. Bello and Mark Sandler. Techniques for automatic music transcription. 2000.

[17] M. Stinchcombe K. Hornik and H. White. Multilayer feedforward neural networks are universal approximators. 1989.

[18] Matija Marolt. Sonic: Transcription of polyphonic piano music with neural networks. 2001.

[19] Keith D. Martin. Automatic transcription of simple polyphonic music:robust front end processing. *MIT Media Lab Perceptual Computing Section Technical Report no. 399*, 1996.

[20] James A. Moorer. On the segmentation and analysis of continuous musical sound by digital computer. 1975. Ph. D Thesis.

[21] Harry Nyquist. Certain topics in telegraph transmission theory. *Trans. AIEE, vol. 47*, pages 617–644, 1928.

[22] Per Kåre Otteren. Anvendelse av biologisk inspirerte metoder for improvisasjon i musikk. 2005.

[23] Antonio Pertusa and Josè M. Inesta. Polyphonic music transcription through dynamic networks and spectral patterns identification. 2003.

[24] Martin Scheirer. Extracting expressive performance information from recorded music. 1995. Master Thesis, MIT.

[25] Neil Schemenauer. A feed forward, back propagation neural network implemented in python. [http://www.python.org]. bpnn.py.

[26] Paul Scott. Music classification using neural networks.

[27] Claude E Shannon. Communication in the presence of noise. *Proc. Institute of Radio Engineers, vol. 37, no.1*, pages 10–21, 1949.

[28] Christos Stergiou and Dimitrios Siganos. Neural nerworks. 1996.

[29] Kvasnicka V. Svozil D. and Pospichal J. Introduction to multi-layer feed-forward neural networks.

[30] Øyvind Hammer. Digital lydbehandling. 1997.

Tillegg A

Resultater

A.1 Differanse analyse

```
Benytter  A.wav, G.wav og G2.wav
som har 16Khz sample rate, mono og 256 kbps


Kjører med forskjellige fftlengder og forskjellig oppdeling innen hver lengde
Prøver å finne ut hvilke verdier som gir størst differanse mellom A og G samtidig
som differansen mellom G og G2 er liten.


FFT N gir N/2 verdier

For hver FFT verdi prøver jeg å benytte først alle verdier,
deretter bare den nederste halvparten, deretter nederste fjerdedel osv.

Ønsket resultat er høy verdi på A-G, lav verdi på G-G og høy Tot.
Tot = A-G*100/G-G

FFT: 8
Alle    4 verdier: A-G: 1.8 , G-G: 4.4  Tot: 41.4
Nederste  2 verdier: A-G: 1.6 , G-G: 4.6  Tot: 34.7
FFT: 16
Alle   8 verdier: A-G: 4.7 , G-G: 6.5  Tot: 72.2
Nederste  4 verdier: A-G: 5.4 , G-G: 6.1  Tot: 87.8
Nederste  2 verdier: A-G: 3.0 , G-G: 5.6  Tot: 52.5
FFT: 32
Alle   16 verdier: A-G: 7.5 , G-G: 6.0  Tot: 125.2
Nederste  8 verdier: A-G: 4.6 , G-G: 6.6  Tot: 69.8
Nederste  4 verdier: A-G: 3.8 , G-G: 9.6  Tot: 40.1
Nederste  2 verdier: A-G: 2.4 , G-G: 7.8  Tot: 30.2
FFT: 64
Alle   32 verdier: A-G: 5.6 , G-G: 6.1  Tot: 92.2
Nederste  16 verdier: A-G: 5.3 , G-G: 4.2  Tot: 126.5
Nederste  8 verdier: A-G: 6.1 , G-G: 4.9  Tot: 123.7
Nederste  4 verdier: A-G: 5.2 , G-G: 4.2  Tot: 123.7
Nederste  2 verdier: A-G: 7.7 , G-G: 6.6  Tot: 116.5
FFT: 128
```

```
Alle   64 verdier: A-G: 6.6 , G-G: 6.7  Tot: 99.2
Nederste  32 verdier: A-G: 6.4 , G-G: 6.3  Tot: 102.1
Nederste  16 verdier: A-G: 7.6 , G-G: 6.8  Tot: 110.6
Nederste  8 verdier: A-G: 7.6 , G-G: 6.8  Tot: 112.0
Nederste  4 verdier: A-G: 10.1 , G-G: 5.7  Tot: 175.6
Nederste  2 verdier: A-G: 10.5 , G-G: 5.4  Tot: 193.8
FFT: 256
Alle   128 verdier: A-G: 8.4 , G-G: 7.5  Tot: 111.8
Nederste  64 verdier: A-G: 7.1 , G-G: 6.5  Tot: 109.8
Nederste  32 verdier: A-G: 7.3 , G-G: 6.0  Tot: 121.5
Nederste  16 verdier: A-G: 7.1 , G-G: 6.1  Tot: 117.0
Nederste  8 verdier: A-G: 6.9 , G-G: 5.9  Tot: 117.3
Nederste  4 verdier: A-G: 7.2 , G-G: 5.3  Tot: 135.6
Nederste  2 verdier: A-G: 8.8 , G-G: 5.1  Tot: 171.6
FFT: 512
Alle   256 verdier: A-G: 8.4 , G-G: 7.6  Tot: 109.8
Nederste  128 verdier: A-G: 7.3 , G-G: 6.4  Tot: 113.3
Nederste  64 verdier: A-G: 6.9 , G-G: 6.1  Tot: 113.9
Nederste  32 verdier: A-G: 5.7 , G-G: 5.6  Tot: 102.1
Nederste  16 verdier: A-G: 5.0 , G-G: 5.0  Tot: 100.3
Nederste  8 verdier: A-G: 4.8 , G-G: 4.1  Tot: 116.1
Nederste  4 verdier: A-G: 4.9 , G-G: 4.0  Tot: 121.3
FFT: 1024
Alle   512 verdier: A-G: 8.2 , G-G: 7.6  Tot: 108.8
Nederste  256 verdier: A-G: 7.1 , G-G: 6.2  Tot: 113.1
Nederste  128 verdier: A-G: 6.8 , G-G: 5.9  Tot: 114.6
Nederste  64 verdier: A-G: 5.5 , G-G: 5.4  Tot: 101.7
Nederste  32 verdier: A-G: 4.7 , G-G: 5.0  Tot: 92.8
Nederste  16 verdier: A-G: 3.8 , G-G: 4.2  Tot: 90.8
Nederste  8 verdier: A-G: 3.1 , G-G: 4.3  Tot: 71.5
FFT: 2048
Alle   1024 verdier: A-G: 8.2 , G-G: 7.5  Tot: 108.9
Nederste  512 verdier: A-G: 7.0 , G-G: 6.3  Tot: 111.7
Nederste  256 verdier: A-G: 6.8 , G-G: 6.0  Tot: 113.2
Nederste  128 verdier: A-G: 5.4 , G-G: 5.5  Tot: 98.3
Nederste  64 verdier: A-G: 4.6 , G-G: 5.2  Tot: 89.3
Nederste  32 verdier: A-G: 3.6 , G-G: 4.2  Tot: 87.1
Nederste  16 verdier: A-G: 2.8 , G-G: 4.2  Tot: 65.3
FFT: 4096
Alle   2048 verdier: A-G: 8.2 , G-G: 7.5  Tot: 108.9
Nederste  1024 verdier: A-G: 7.0 , G-G: 6.2  Tot: 111.7
Nederste  512 verdier: A-G: 6.7 , G-G: 6.0  Tot: 113.0
Nederste  256 verdier: A-G: 5.4 , G-G: 5.5  Tot: 97.8
Nederste  128 verdier: A-G: 4.6 , G-G: 5.1  Tot: 88.9
Nederste  64 verdier: A-G: 3.6 , G-G: 4.1  Tot: 86.4
Nederste  32 verdier: A-G: 2.6 , G-G: 4.1  Tot: 63.5
```

A.2 Test av opptrent nettverk

```
16:06:06 : Start
setter tempo til 120
setter rolig til 30
setter glad til 75
setter noten til 60
Lastet inn nettverk fra T1e002.nnt
```

```
Nettverk: 126 32 4 exI [512, 2, 128]
Initierer Happy
16:06:21 start live rec int 400
-32 g 0.02 --- dur 67
0 Spiller noten  69 lengde 0.25
4 Spiller noten  72 lengde 0.25
8 Spiller noten  74 lengde 0.25
12 Spiller noten  76 lengde 0.25
16 Spiller noten  79 lengde 0.25
20 Spiller noten  76 lengde 0.25
24 Spiller noten  79 lengde 0.25
28 Spiller noten  81 lengde 0.25
32 Spiller noten  84 lengde 0.25
36 Spiller noten  83 lengde 0.25
40 Spiller noten  86 lengde 0.5
48 Spiller noten  83 lengde 0.5
56 Spiller noten  81 lengde 0.5
64 Spiller noten  78 lengde 0.5
69 d 0.12 --- dur 62
72 Spiller noten  61 lengde 0.5
80 Spiller noten  103 lengde 0.5
88 Spiller noten  86 lengde 0.5
96 Spiller noten  83 lengde 0.25
100 Spiller noten  86 lengde 0.25
102 g 0.03 --- dur 67
104 Spiller noten  69 lengde 0.25
108 Spiller noten  72 lengde 0.25
112 Spiller noten  74 lengde 0.25
116 Spiller noten  76 lengde 0.25
120 Spiller noten  79 lengde 0.25
124 Spiller noten  76 lengde 0.25
128 Spiller noten  79 lengde 0.25
132 Spiller noten  81 lengde 0.25
136 Spiller noten  84 lengde 0.25
137 d 0.05 --- dur 62
140 Spiller noten  64 lengde 0.25
144 Spiller noten  62 lengde 0.5
152 Spiller noten  61 lengde 0.5
160 Spiller noten  62 lengde 0.5
168 Spiller noten  61 lengde 0.5
176 Spiller noten  59 lengde 0.5
177 am 0.15 --- moll 69
182 g 0.03 --- dur 67
184 Spiller noten  66 lengde 0.5
192 Spiller noten  64 lengde 0.25
196 Spiller noten  67 lengde 0.25
200 Spiller noten  69 lengde 0.25
204 c 0.01 --- dur 60
204 Spiller noten  62 lengde 0.25
208 Spiller noten  65 lengde 0.25
212 Spiller noten  67 lengde 0.25
216 g 0.02 --- dur 67
216 Spiller noten  69 lengde 0.25
220 Spiller noten  72 lengde 0.25
224 Spiller noten  69 lengde 0.25
228 Spiller noten  72 lengde 0.25
```

```
232 Spiller noten  74 lengde 0.25
236 Spiller noten  76 lengde 0.25
240 Spiller noten  72 lengde 0.5
248 c 0.06 --- dur 60
248 Spiller noten  59 lengde 0.5
253 am 0.01 --- moll 69
256 Spiller noten  68 lengde 0.5
264 Spiller noten  65 lengde 0.5
272 Spiller noten  64 lengde 0.5
276 f 0.05 --- dur 65
280 Spiller noten  64 lengde 0.5
288 Spiller noten  55 lengde 0.25
292 Spiller noten  58 lengde 0.25
296 Spiller noten  60 lengde 0.25
300 Spiller noten  62 lengde 0.25
304 Spiller noten  65 lengde 0.25
308 Spiller noten  62 lengde 0.25
310 g 0.14 --- dur 67
312 Spiller noten  69 lengde 0.25
314 d 0.01 --- dur 62
316 Spiller noten  64 lengde 0.25
319 g 0.1 --- dur 67
320 Spiller noten  69 lengde 0.25
324 Spiller noten  72 lengde 0.25
328 Spiller noten  74 lengde 0.25
332 Spiller noten  76 lengde 0.25
335 d 0.0 --- dur 62
336 Spiller noten  91 lengde 0.5
340 g 0.14 --- dur 67
344 Spiller noten  88 lengde 0.5
352 Spiller noten  86 lengde 0.5
360 Spiller noten  83 lengde 0.5
368 Spiller noten  81 lengde 0.5
376 Spiller noten  84 lengde 0.5
384 Spiller noten  81 lengde 0.25
16:07:00 stop


2 Runder rec int 400

16:08:26 start live
-32 g 0.02 --- dur 67
-32 d 0.13 --- dur 62
-32 g 0.04 --- dur 67
-32 d 0.02 --- dur 62
-32 g 0.12 --- dur 67
-32 d 0.01 --- dur 62
-32 c 0.01 --- dur 60
-32 g 0.03 --- dur 67
-32 c 0.01 --- dur 60
-32 g 0.03 --- dur 67
-32 am 0.1 --- moll 69
-32 c 0.05 --- dur 60
-32 am 0.01 --- moll 69
-32 f 0.01 --- dur 65
-32 g 0.09 --- dur 67
```

```
-32 d 0.11 --- dur 62
-32 g 0.02 --- dur 67
-32 d 0.0 --- dur 62
-32 g 0.03 --- dur 67
-32 d 0.0 --- dur 62
-32 c 0.01 --- dur 60
-32 g 0.11 --- dur 67
-32 c 0.05 --- dur 60
-32 g 0.03 --- dur 67
-32 am 0.11 --- moll 69
-32 d 0.15 --- dur 62
-32 f 0.03 --- dur 65
-32 g 0.03 --- dur 67
16:09:27 stop


2 runder rec int 200

16:12:47 start live
0 g 0.02 --- dur 67
38 d 0.13 --- dur 62
71 g 0.02 --- dur 67
104 d 0.0 --- dur 62
144 g 0.03 --- dur 67
169 am 0.01 --- moll 69
174 g 0.02 --- dur 67
189 am 0.03 --- moll 69
226 f 0.01 --- dur 65
231 d 0.04 --- dur 62
247 f 0.1 --- dur 65
254 g 0.02 --- dur 67
288 d 0.08 --- dur 62
294 g 0.05 --- dur 67
347 d 0.03 --- dur 62
379 g 0.08 --- dur 67
412 d 0.01 --- dur 62
446 c 0.02 --- dur 60
455 am 0.04 --- moll 69
461 g 0.05 --- dur 67
479 c 0.04 --- dur 60
487 am 0.1 --- moll 69
491 g 0.03 --- dur 67
508 am 0.03 --- moll 69
541 f 0.05 --- dur 65
553 d 0.14 --- dur 62
555 f 0.0 --- dur 65
582 g 0.06 --- dur 67
16:13:41 stop
Stopper happy
RECEIVED CppSound::stop...
------------------------Avspilte noter--------------------------------
[60, 60, 69, 72, 74, 76, 79, 76, 79, 81, 84, 83, 64, 62, 64, 62, 66, 64, 62, 60,
 62, 64, 67, 66, 69, 71, 74, 73, 76, 79, 81, 66, 64, 62, 64, 66, 64, 68, 71, 74,
 77, 74, 77, 80, 83, 86, 110, 64, 67, 66, 64, 66, 64, 62, 60, 59, 66, 69, 72, 71
, 74, 76, 79, 78, 81, 84, 86, 83, 81, 61, 90, 88, 85, 66, 69, 72, 74, 76, 79, 76
, 79, 81, 83, 86, 83, 81, 78, 59, 88, 66, 64, 59, 62, 72, 69, 72, 74, 76, 72, 74
```

```
, 77, 80, 83, 80, 77, 64, 81, 64, 62, 60, 62, 69, 72, 71, 74, 76, 79, 78, 81, 84
, 86, 83, 81]
Verdier: note 81 tempo 120 glad 25 rolig 30
```

Tillegg B

Kildekode

B.1 NNTuned Happysound.py

```
# -*- coding: latin-1 -*-

from Tkinter import *
from NNFrame import NNFrame
from tkSnack import *
import CsoundVST    #Oppretter et globalt objekt 'csound'




if __name__ == '__main__' :
    global csound
    root = Tk()
    initializeSnack(root)
    NNFrame(root,csound)
```

B.2 NNFrame.py

```
# -*- coding: latin-1 -*-

from time import localtime,strftime
from tkSnack import *
from Tkinter import *
from NNdef  import *

class NNFrame:
    """ Rammen som inneholder alle de grafiske elementene """
    def __init__(self, master,csound):
     """ Oppretter applikasjonsvinduet og setter alle elementer"""
        print strftime("%H:%M:%S", localtime()),': Start'
        self.csound = csound
        self.musmak = Musikkmaker.Musikkmaker(self.csound)
        self.fs = 16000
        self.id = 0
        self.wait = 400 # tid mellom kjøringer av rec eller draw
        self.p0 = 0
        self.sLength = 0
        self.lastet = False
        self.fftl = 512
        self.fqmin = 2
        self.fqmax = 128
        self.nnlen = 126
        self.forrigedur = [0,0,0,0]
        self.sound = Sound()
        master.geometry('800x600')
        self.toplevel = master
        self.toplevel.title('NNtune')
        self.toplevel.iconname('NNtune')


        # Knapper i bunn
        self.bottomFrame = Frame(self.toplevel)
        self.bottomFrame.pack(side=BOTTOM, pady=5, padx=5, fill=X )


        # Høyre Frame
        self.hFrame = Frame(self.toplevel)
        self.hFrame.pack(side=RIGHT,fill=BOTH,padx=10)
        self.initHappyBtn = Button(self.hFrame, text='Initier Happysound', command=self.initHappy)
        self.initHappyBtn.pack(side='top',fill=X)
        self.laglyd = IntVar()
        cb = Checkbutton(self.hFrame, text="Init CSound", variable=self.laglyd)
        cb.pack()
        cb.select()
        mmFrame = Frame(self.hFrame)
        mmFrame.pack(side=TOP,fill=X,pady=30)
        self.startHappyBtn = Button(mmFrame, text='Start', fg='red', command=self.startHappy, state=DISABLED)
        self.startHappyBtn.pack(side=LEFT,padx=10)
        self.stopHappyBtn = Button(mmFrame, text='Stopp', fg='blue', command=self.stoppHappy, state=NORMAL)
        self.stopHappyBtn.pack(side=RIGHT,padx=10)

        durvFrame = Frame(self.hFrame)
```

```
durvFrame.pack(side=TOP,fill=X,pady=10)
self.durValg = IntVar()
r1 = Radiobutton(durvFrame, text="Dur", variable=self.durValg, value=1, command=self.seldur)
r1.pack(side=TOP)
Radiobutton(durvFrame, text="Moll", variable=self.durValg, value=2, command=self.seldur).pack(side=TOP)
Radiobutton(durvFrame, text="Blues", variable=self.durValg, value=3, command=self.seldur).pack(side=TOP)
r1.select()
self.s1 = Scale(self.hFrame,label="Startnote",orient=HORIZONTAL,command=self.selStartN, to=127)
self.s1.pack()
self.s1.set(60)
self.s2 = Scale(self.hFrame,label="Glad verdi",orient=HORIZONTAL,command=self.selStartG)
self.s2.pack()
self.s2.set(75)
self.s3=Scale(self.hFrame,label="Rolig verdi",orient=HORIZONTAL,command=self.selStartR)
self.s3.pack()
self.s3.set(30)
self.s4=Scale(self.hFrame,label="Tempo",orient=HORIZONTAL,command=self.selStartT, to=200,resolution=10)
self.s4.pack()
self.s4.set(120)



# Topp Frame
self.topFrame = Frame(self.toplevel)
self.topFrame.pack(side=TOP, padx=5, fill=X)
self.topFrame.pack()
self.c  = SnackCanvas(self.topFrame, height=200, width=600, bg='white', relief='sunken', bd=1)
self.cid = self.c.create_waveform (0, 0, sound=self.sound, height=200, width=600)
self.c.pack ()



#Middle Frame
self.middleFrame = Frame(self.toplevel)
self.middleFrame.pack(fill=Y,expand=YES)
self.mlFrame = Frame(self.middleFrame)
self.mlFrame.pack(side=LEFT,fill=BOTH)
self.mltFrame = Frame(self.mlFrame)
self.mltFrame.pack(side=TOP,pady=5,fill=BOTH)
self.mlcFrame = Frame(self.mlFrame)
self.mlcFrame.pack(pady=5,fill=BOTH)
self.mlbFrame = Frame(self.mlFrame)
self.mlbFrame.pack(pady=5,fill=BOTH)
self.mlb2Frame = Frame(self.mlFrame)
self.mlb2Frame.pack(side=BOTTOM,pady=5)
self.mrFrame = Frame(self.middleFrame)
self.mrFrame.pack(side=RIGHT,fill=BOTH)

#MR
self.c2 = SnackCanvas(self.mrFrame, height=200, width=300, bg='white', relief='sunken', bd=1)
self.cid2 = self.c2.create_section    (0, 0, sound=self.sound, height=200, width=300)
self.c2.pack()
bbar = Frame(self.mrFrame)
bbar.pack(side='bottom')
self.tsEntry = Entry(bbar,background = 'white',width=5)
self.tsEntry.insert(INSERT, "0.15")
```

```
self.tsEntry.pack(side=LEFT)
Button(bbar, bitmap='snackRecord', fg='blue', command=self.startLive).pack(side='left')
Button(bbar, bitmap='snackStop', command=self.stop).pack(side='left')
bbar2 = Frame(self.mrFrame)
bbar2.pack(side='bottom',pady=5)
self.akkEntry = Entry(bbar2,background = 'white',width=5)
self.akkEntry.insert(INSERT, "a")
self.akkEntry.pack(side=LEFT)
Button(bbar2, text='Ta opp akkord', fg='red', command=self.startRec).pack(side='left')
self.wEntry = Entry(self.mrFrame,background = 'white',width=5)
self.wEntry.insert(INSERT, "400")
Label(self.mrFrame,text="Rec intervall").pack(side=TOP)
self.wEntry.pack(side=TOP)

#HML
ffbar = Frame(self.mltFrame)
ffbar.pack(side=TOP)
ffbar2 = Frame(self.mltFrame)
ffbar2.pack(side=TOP)
Label(ffbar,text="FFT len", width=6).pack(side=LEFT)
Label(ffbar,text="Min freq", width=6).pack(side=LEFT)
Label(ffbar,text="Max freq", width=6).pack(side=LEFT)
self.FFtEntry = Entry(ffbar2,background = 'white', width=6)
self.FFtEntry.insert(INSERT, "512")
self.FFtEntry.pack(side=LEFT)
self.fqminEntry = Entry(ffbar2,background = 'white', width=6)
self.fqminEntry.insert(INSERT, "80")
self.fqminEntry.pack(side=LEFT)
self.fqmaxEntry = Entry(ffbar2,background = 'white', width=6)
self.fqmaxEntry.insert(INSERT, "3990")
self.fqmaxEntry.pack(side=LEFT)
Button(self.mltFrame, text='Set FFT',     command=self.setFFT, width=10).pack()
Label(self.mltFrame,text="NN inn-skjulte-ut").pack(side=TOP)
self.NNiEntry = Entry(self.mltFrame,background = 'white',width=5)
self.NNiEntry.insert(INSERT, "126")
self.NNiEntry.pack(side=LEFT)
self.NNhEntry = Entry(self.mltFrame,background = 'white',width=5)
self.NNhEntry.insert(INSERT, "32")
self.NNhEntry.pack(side=LEFT)
self.NNuEntry = Entry(self.mltFrame,background = 'white',width=5)
self.NNuEntry.insert(INSERT, "4")
self.NNuEntry.pack(side=LEFT)
Button(self.mlcFrame, text='Opprett NN',     command=self.opprettNN, width=10).pack()

self.fileFrame = Frame(self.middleFrame)
self.fileFrame.pack(side=TOP, fill=BOTH)
self.filesBar = Scrollbar(self.fileFrame)
self.filesBar.pack(side=RIGHT)
self.fileList = Listbox(self.fileFrame,background = 'white',exportselection=0,height=10,width=40,
             yscrollcommand=(self.filesBar, 'set'))
self.filesBar.configure(command = self.fileList.yview)
Label(self.fileFrame,text="NN fra fil").pack(side='top')
self.fileList.pack(side=TOP)
for item in finnSaved():
    self.fileList.insert(END, item)
self.fileList.select_set(0)
```

```
    Button(self.fileFrame,  text='Last in NN',  command=self.lastInn,   width=10).pack( side=TOP, pady=5)
    Button(self.fileFrame,  text='Last in kun Pat',  command=self.lastInnPat, width=20).pack( side=TOP, pady=0)

    self.NNsaveEntry = Entry(self.mlbFrame,background = 'white',width=30)
    self.NNsaveEntry.insert(INSERT, "(512).nnt")
    self.NNsaveEntry.pack()
    Button(self.mlbFrame,  text='Lagre NN',    command=self.saveData,  width=10).pack( side=BOTTOM)

    Button(self.mlb2Frame,  text='Test akks i dir',    command=self.testDir,  width=10).pack( side=TOP)


    self.labelFrame = Frame(self.mlb2Frame)
    self.labelFrame.pack(side=TOP, fill=X)
    self.entFrame = Frame(self.mlb2Frame)
    self.entFrame.pack(side=TOP, fill=X)
    Label(self.labelFrame,text="Iter.", width=6).pack(side=LEFT)
    self.itEntry = Entry(self.entFrame,background = 'white', width=6)
    self.itEntry.insert(INSERT, "1000")
    self.itEntry.pack(side=LEFT)
    Label(self.labelFrame,text="MinErr", width=6).pack(side=LEFT)
    self.minerrEntry = Entry(self.entFrame,background = 'white',width=5)
    self.minerrEntry .insert(INSERT, "0")
    self.minerrEntry .pack(side=LEFT)
    Button(self.mlb2Frame, text='Tren NN',    command=self.tren,  width=10).pack()


    #Start applikasjonen
    self.csKontroll = False
    self.toplevel.mainloop()
    self.musmak.avslutt = True
    if self.csKontroll:
        self.csKontroll.avslutt()
    print strftime("%H:%M:%S", localtime()),': Avsluttet Happysound'


def initHappy(self):
    print 'Initierer Happy'
    self.c.delete(self.cid)
    self.c2.delete(self.cid2)
    self.initHappyBtn.config(state = DISABLED)
    self.startHappyBtn.config(state = NORMAL)
    self.csKontroll = CsoundKontroll.CsoundKontroll(self.csound,self.laglyd.get())
    self.musmak.spill = self.laglyd.get()
    self.musmak.start()


def startHappy(self):
    print 'Starter happy'
    self.startHappyBtn.config(state = DISABLED)
    self.musmak.lagmusikk = True

def stoppHappy(self):
    print 'Stopper happy'
    self.startHappyBtn.config(state = NORMAL)
    self.musmak.stopp()
    self.musmak.nullstill()
```

```
    self.musmak.initval()
    self.s1.set(60)
    self.s2.set(75)
    self.s3.set(30)
    self.s4.set(120)

def seldur(self):
    if self.durValg.get()==1:
        self.musmak.skalaibruk = self.musmak.dur
        print 'Setter skala til dur'
    elif self.durValg.get()==2:
        self.musmak.skalaibruk = self.musmak.moll
        print 'Setter skala til moll'
    elif self.durValg.get()==3:
        self.musmak.skalaibruk = self.musmak.blues
        print 'Setter skala til blues'


def selStartN(self,val):
    self.musmak.noten = int(val)
    print 'setter noten til', val

def selStartG(self,val):
    self.musmak.glad = int(val)
    print 'setter glad til', val

def selStartR(self,val):
    self.musmak.rolig = int(val)
    print 'setter rolig til', val

def selStartT(self,val):
    self.musmak.tempo = int(val)
    print 'setter tempo til', val

def tren(self):
    tren(self.nn,it=int(self.itEntry.get()),minerr=float(self.minerrEntry.get()))

def opprettNN(self):
    self.nn = opprett(ni=int(self.NNiEntry.get()),nh=int(self.NNhEntry.get()), nu=int(self.NNuEntry.get()))
    self.lastet = True

def testDir(self):
    finnDir(self.nn,'akks',fft=self.fftl,fqs=self.fqmin,fqe=self.fqmax)

def setFFT(self):
    self.fftl = int(self.FFtEntry.get())
    hzprb = (self.fs/2)/(self.fftl/2)
    self.fqmin = int(self.fqminEntry.get())/hzprb
    self.fqmax = int(self.fqmaxEntry.get())/hzprb
    self.nnlen = self.fqmax-self.fqmin
    self.NNiEntry.delete(0,END)
    self.NNiEntry.insert(INSERT, self.nnlen)
    print 'setter fftlen=',self.fftl,'og nnlen=',self.nnlen
    print self.fqmin, self.fqmax

def lastInn(self):
```

```
 """Laste inn nettverk fra fil"""
    self.nn, exI = restoreData(fil=self.fileList.get(self.fileList.curselection()[0]))
    if exI != 0:
        self.fftl = exI[0]
        self.FFtEntry.delete(0,END)
        self.FFtEntry.insert(INSERT, self.fftl)
        hzprb = (self.fs/2)/(self.fftl/2)
        self.fqmin = exI[1]
        self.fqminEntry.delete(0,END)
        self.fqminEntry.insert(INSERT, self.fqmin*hzprb)
        self.fqmax = exI[2]
        self.fqmaxEntry.delete(0,END)
        self.fqmaxEntry.insert(INSERT, self.fqmax*hzprb)

    self.lastet = True
    self.nnlen = self.nn.ni-1
    self.NNiEntry.delete(0,END)
    self.NNiEntry.insert(INSERT, self.nnlen)
    print 'Nettverk:',self.nn.ni-1,self.nn.nh,self.nn.no,'exI',exI

def lastInnPat(self):
    self.pat1 = restoreDataPat(fil=self.fileList.get(self.fileList.curselection()[0]))

def saveData(self):
    saveData(self.nn,fil=self.NNsaveEntry.get(),exI=[self.fftl,self.fqmin,self.fqmax])

def stop(self):
    print strftime("%H:%M:%S", localtime()), 'stop'
    self.sound.stop()
    self.toplevel.after_cancel(self.id)
    if self.csKontroll:
        self.musmak.lagmusikk = False

def startLive(self):
 """ starter opp draw metoden"""
    print strftime("%H:%M:%S", localtime()), 'start live'
    self.wait = int(self.wEntry.get())
    self.p0 = 0
    self.sLength = 0
    if not(self.lastet):
        print 'Ingen nettverk lastet'
        return
    self.sound.record()
    self.id = self.toplevel.after(self.wait,self.draw)

def startRec(self):
 """Starter opp rec metoden"""
    self.wait = int(self.wEntry.get())
    print strftime("%H:%M:%S", localtime()), 'start rec, int:',self.wait
    self.p0 = 0
    self.sLength = 0
    self.sound.record()
    self.id = self.toplevel.after(self.wait,self.rec)

def rec(self):
 """ Metode for å registrere akkorder i treningssettet
```

```
 Metoden kjører i 10 sekunder og registrerer ny akkord
 ved hvert rec-intervall"""
    spec = self.getspec()
    if (spec != -1):
        pat1.append([ spec,akklist[self.akkEntry.get()] ])
        print 'la til en akk i pat1:', len(pat1),self.akkEntry.get()
    if (self.sound.length(unit='sec') > 10) :
        self.sound.stop()
    else:
        self.id = self.toplevel.after(self.wait,self.rec)

def draw(self):
    """ Metoden som viser hvilken akkord som blir spilt.
    Henter specter fra avspilt akkord via
    getspec metoden og tester disse verdiene i nettverket.
    Metoden gjentar seg selv etter self.wait millisekund  """

    self.id = self.toplevel.after(self.wait,self.draw)
    spec = self.getspec()
    terskel = self.tsEntry.get()
    if (spec != -1):
        erval = self.nn.finnValE(spec)
        #if erval[1]<0.15 and erval[0] != self.forrigedur:
        if erval[1]<terskel and erval[0] != self.forrigedur:
            akk = finnAkk(erval[0])
            note = finnMidiN(akk)
            dur = finnDur(akk)
            print self.musmak.spilltick, akk,round(erval[1],2),"---",dur,note
            if self.csKontroll:
                self.musmak.settNote(note,dur)
            self.forrigedur = erval[0]

def finnmax(self):
    """Returnerer posisjonen til samplet med høyest verdi
    Metoden leter i siste avspilte intervall"""
    self.p0 = self.sLength
    self.sLength = self.sound.length() - int(0.2*self.fs) -1
    if self.sLength<0:
        self.sLength = 0
    #print 'slen  ',int(self.sLength*1000/self.fs)
    max = self.sound.max(start=self.p0, end=self.sLength)
    if(max<5000):
        return -1
    for index in range(0,self.sLength-self.p0):
        if self.sound.sample(index+self.p0) == max:
            return index+self.p0
    return 0

def getspec(self):
 """ Returnerer powerspectrum fra siste lydbit
 Startpunkt bestemmes av finnmax metoden og lengden er 0.2 sekunder
 Spekteret normaliseres og lave punkt (under -60) nulles ut"""
    pos = self.finnmax()
    if pos==-1:
        return -1
    if self.fftl>=256:
```

```
    wlen=256
else:
    wlen=fft
end= pos + int(0.2*self.fs)
spec=[]
spec += self.sound.dBPowerSpectrum(start=pos,end=end,fftlen=self.fftl,winlen=wlen)[self.fqmin:self.fqmax]
for n in range(self.fqmax-self.fqmin):
    if spec[n] <-60:
        spec[n] = -100
return normalize(spec)
```

B.3 NNdef.py

```
# -*- coding: latin-1 -*-

import bpnn
import cPickle
from time import localtime,strftime
from os import listdir
from tkSnack import *

import Musikkmaker
import CsoundKontroll


akklist = { "null":[0,0,0,0],"d":[0,0,0,1],"e":[0,0,1,0],"f":[0,0,1,1],"g":[0,1,0,0],"a":[0,1,0,1],"h":[0,1,1,0]
        "cm":[0,1,1,1],"dm":[1,0,0,0],"em":[1,0,0,1],"fm":[1,0,1,0],"gm":[1,0,1,1],"am":[1,1,0,0],"hm":[1,1,0,1]
        "c":[1,1,1,0],"d7":[1,1,1,1]    }
midinotelist = { "null":60,"d":62,"e":64,"f":65,"g":67,"a":69,"h":71,
        "cm":60,"dm":62,"em":64,"fm":65,"gm":67,"am":69,"hm":71,
        "c":60,"d7":62    }
durlist = { "null":"dur","d":"dur","e":"dur","f":"dur","g":"dur","a":"dur","h":"dur",
        "cm":"moll","dm":"moll","em":"moll","fm":"moll","gm":"moll","am":"moll","hm":"moll",
        "c":"dur","d7":"blues"    }
soundsinit = False
pat1 = []

def normalize(list):
    """ Normaliserer listen til å inneholde verdier mellom 0 og 1
        Regner med at den bare får bare inn negative verdier!!!
    """
    outlist = []
    if(abs(max(list))>abs(min(list))):
        print 'FEIL!!! - Fant en positiv verdi'
        oldMax = float(max(list))
    else:
        oldMax = float(abs(min(list)))
    for item in list:
        item = (item/oldMax)*-1
        outlist.append(item)
    return outlist

def finnAkk(val):
    """ Finner en akkord for gitt verdi """
    for key in akklist:
        if akklist[key]==val:
            return key
    return "fant ingen passende akkord"

def finnDur(val):
    """ Finner dur/moll for gitt verdi """
    return durlist[val]

def finnMidiN(val):
    """ Finner midinummer for gitt verdi """
    return midinotelist[val]
```

```
def finnDir(nn,dir,fft=512,fqs=0,fqe=128):
    """Tester alle akkorder i en mappe"""
    for file in listdir(dir):
        val = getAkkVals(dir+'/'+file,fft=fft,fqs=fqs,fqe=fqe)
        print file,':',finnAkk(nn.finnVal(val)),
        print '     ',
        nn.finnValError(val)

def getAkkVals(fil,fft=512,fqs=0, fqe=128):
    """Returnerer en tuple med verdier fra PowerSpectrum"""
    ns = Sound(load=fil)
    if fft>=256:
        wlen=256
    else:
        wlen=fft
    start=findmax(ns)
    if (ns.length()-start)<fft:
        start=ns.length()-fft
    end=start + int(0.2*ns.cget('frequency'))
    if end>ns.length():
        end = ns.length()-1
    spec = []
    spec += ns.dBPowerSpectrum(start=start,end=end,fftlength=fft,windowlength=wlen)[fqs:fqe]
    for n in range(len(spec)):
        if spec[n] <-60:
            spec[n] = -100
    return normalize(spec)

def findmax(ns):
    """Returnerer posisjonen til samplet med høyest verdi"""
    lengde = ns.length()
    max = ns.max()
    #print 'max:',max
    for index in range(0,lengde):
        if ns.sample(index) == max:
            return index
    return 0

def opprett(ni=128,nh=24, nu=4):
    """ Oppretter et nevralt nettverk med ni inn-noder, nh skjulte noder og nu ut-noder"""
    print strftime("%H:%M:%S", localtime()),': Oppretter nettverk (',ni,nh,nu,')'
    nn = bpnn.NN(ni, nh, nu)
    return nn

def tren(nn,it=1000,minerr=0.0):
    """ Trener opp nettverket. Stopper etter gitt antall iterasjoner, eller når gitt
        minimumserror er oppnådd
    """
    if(len(pat1)<1):
        print 'Ingen opptreningsverdier i pat1'
        return
    print strftime("%H:%M:%S", localtime()),': Trener pat1... Lengde:',len(pat1)
    print it,'iterasjoner'
    nn.train(pat1,iterations=it,stoperr=minerr)
    print strftime("%H:%M:%S", localtime()),': Ferdig med trening'
```

```
def saveData(nn,fil='nntune.nnt',exI = []):
    """lagrer data til fil"""
    file = open(fil, 'w')
    data = {'nn' : nn,'pat1' : pat1, 'exI' : exI}
    cPickle.dump( data, file)
    file.close()
    print 'Lagret nettverk til', fil

def restoreData(fil='nntune.nnt'):
    """henter data fra fil"""
    file = open(fil, 'r')
    data = cPickle.load(file)
    file.close()
    #global nn,pat1
    nn = data[ 'nn']
    pat1 = data[ 'pat1']
    print 'Lastet inn nettverk fra',fil
    if data.has_key('exI'):
        return nn,data['exI']
    else:
        return nn,0

def restoreDataPat(fil='nntune.nnt'):
    """henter data fra fil"""
    file = open(fil, 'r')
    data = cPickle.load(file)
    file.close()
    global pat1
    pat1 += data[ 'pat1']
    print 'La til pat fra',fil
    print len(pat1), 'patterns'
    return pat1


def finnSaved():
    """Returnerer en liste over alle lagrede nettverk i gjeldende mappe"""
    outlist = []
    for file in listdir('./'):
        if file[-3:len(file)] == 'nnt':
            outlist.append(file)
    return outlist
```

B.4 bpnn.py

```
# Back-Propagation Neural Networks
#
# Written in Python.  See http://www.python.org/
# Placed in the public domain.
# Neil Schemenauer <nas@arctrix.com>

import math
import random
import string

random.seed(0)

# calculate a random number where:  a <= rand < b
def rand(a, b):
    return (b-a)*random.random() + a

# Make a matrix (we could use NumPy to speed this up)
def makeMatrix(I, J, fill=0.0):
    m = []
    for i in range(I):
        m.append([fill]*J)
    return m

# our sigmoid function, tanh is a little nicer than the standard 1/(1+e^-x)
# tanh gir verdier fra -1 til 1, sigmoid gir fra 0 til 1 som passer mye bedre i mitt tilfelle
def sigmoid(x):
    return 1/(1+math.exp(-x))
    #return math.tanh(x)

# derivative of our sigmoid function
def dsigmoid(y):
    return y * (1 - y)
    #return 1.0-y*y

#sigmoid <- function (x)  1 / (1 + exp(-x))
#dsigmoid <- function (x)  x * (1 - x)



class NN:
    def __init__(self, ni, nh, no):
        # number of input, hidden, and output nodes
        self.ni = ni + 1 # +1 for bias node
        self.nh = nh
        self.no = no

        # activations for nodes
        self.ai = [1.0]*self.ni
        self.ah = [1.0]*self.nh
        self.ao = [1.0]*self.no

        # create weights
        self.wi = makeMatrix(self.ni, self.nh)
        self.wo = makeMatrix(self.nh, self.no)
```

```
    # set them to random vaules
    for i in range(self.ni):
        for j in range(self.nh):
            self.wi[i][j] = rand(-2.0, 2.0)
    for j in range(self.nh):
        for k in range(self.no):
            self.wo[j][k] = rand(-2.0, 2.0)

    # last change in weights for momentum
    self.ci = makeMatrix(self.ni, self.nh)
    self.co = makeMatrix(self.nh, self.no)
    print 'Opprettet nettverk (',ni,nh,no,')'

def update(self, inputs):
    if len(inputs) != self.ni-1:
        raise ValueError, 'wrong number of inputs'

    # input activations
    for i in range(self.ni-1):
        #self.ai[i] = sigmoid(inputs[i])
        self.ai[i] = inputs[i]

    # hidden activations
    for j in range(self.nh):
        sum = 0.0
        for i in range(self.ni):
            sum = sum + self.ai[i] * self.wi[i][j]
        self.ah[j] = sigmoid(sum)

    # output activations
    for k in range(self.no):
        sum = 0.0
        for j in range(self.nh):
            sum = sum + self.ah[j] * self.wo[j][k]
        self.ao[k] = sigmoid(sum)

    return self.ao[:]


def backPropagate(self, targets, N, M):
    if len(targets) != self.no:
        raise ValueError, 'wrong number of target values'

    # calculate error terms for output
    output_deltas = [0.0] * self.no
    for k in range(self.no):
        error = targets[k]-self.ao[k]
        output_deltas[k] = dsigmoid(self.ao[k]) * error

    # calculate error terms for hidden
    hidden_deltas = [0.0] * self.nh
    for j in range(self.nh):
        error = 0.0
        for k in range(self.no):
            error = error + output_deltas[k]*self.wo[j][k]
        hidden_deltas[j] = dsigmoid(self.ah[j]) * error
```

```
    # update output weights
    for j in range(self.nh):
        for k in range(self.no):
            change = output_deltas[k]*self.ah[j]
            self.wo[j][k] = self.wo[j][k] + N*change + M*self.co[j][k]
            self.co[j][k] = change
            #print N*change, M*self.co[j][k]

    # update input weights
    for i in range(self.ni):
        for j in range(self.nh):
            change = hidden_deltas[j]*self.ai[i]
            self.wi[i][j] = self.wi[i][j] + N*change + M*self.ci[i][j]
            self.ci[i][j] = change

    # calculate error
    error = 0.0
    for k in range(len(targets)):
        error = error + 0.5*(targets[k]-self.ao[k])**2
    return error


def test(self, patterns):
    for p in patterns:
        print p[0], '->', self.update(p[0])

def weights(self):
    print 'Input weights:'
    for i in range(self.ni):
        print self.wi[i]
    print
    print 'Output weights:'
    for j in range(self.nh):
        print self.wo[j]

def train(self, patterns, iterations=1000, N=0.5, M=0.1, stoperr=0.0):
    """Trener opp nettverket."""
    # N: learning rate
    # M: momentum factor
    minerr = 100
    minerri = 0
    for i in xrange(iterations):
        error = 0.0
        for p in patterns:
            inputs = p[0]
            targets = p[1]
            self.update(inputs)
            error = error + self.backPropagate(targets, N, M)
        if i % (int(iterations/10)) == 0:
            print i,'error %-14f' % error, 'min:',minerri, minerr
        if error<minerr:
            minerr=error
            minerri=i
        if i==(iterations-1):
            print 'Ferdig. Error %-14f' % error
```

```
            print 'Min error:',minerr,'index:',minerri
        if error<stoperr:
            print 'Ferdig ved < stoperr. Error %-14f' % error
            print 'Min error:',minerr,'index:',minerri
            break

def finnMinErrIt(self, patterns, maxit=1000):
    """Minner beste antall iterasjoner for treningssett."""
    iterations=maxit
    N=0.5
    M=0.1
    print 'Leter etter index for minste error'
    minerr = 100
    minerri = 0
    for i in xrange(iterations):
        error = 0.0
        for p in patterns:
            inputs = p[0]
            targets = p[1]
            self.update(inputs)
            error = error + self.backPropagate(targets, N, M)
        if i % 100 == 0:
            print i,'error %-14f' % error
        if error<minerr:
            minerr=error
            minerri=i
        if i==(iterations-1):
            print 'Ferdig. Error %-14f' % error
            print 'Min error:',minerr,'index:',minerri
    return minerri

def finn(self, patterns):
    for p in patterns:
        ret = self.update(p[0])
        #print p[0], '->', int(round(ret[0])),int(round(ret[1])),int(round(ret[2]))
        print '(',p[1],')->', round(ret[0],2),round(ret[1],2),round(ret[2],2),round(ret[3],2)

def finnVal(self, pattern):
    ret = self.update(pattern)
    return [int(round(ret[0],0)), int(round(ret[1],0)), int(round(ret[2],0)), int(round(ret[3],0))]

def finnValError(self, pattern):
    ret = self.update(pattern)
    e=0
    for r in ret:
        if r>0.5:
            e+=1-r
        else:
            e+=r
    print round(ret[0],1), round(ret[1],1), round(ret[2],1), round(ret[3],1), 'Total error:', round(e,2)

def finnValE(self, pattern):
    ret = self.update(pattern)
    e=0
    for r in ret:
        if r>0.5:
```

```
                e+=1-r
            else:
                e+=r
        return [[int(round(ret[0],0)), int(round(ret[1],0)), int(round(ret[2],0)), int(round(ret[3],0))],e]

def test():
    ''' Akkorder (1-7:dur,:8-14:moll, 0&15:ingen akk)
    1-C 2-D 3-E 4-F 5-G 6-A 7-H


    '''
    pat = [
        # 1 2 3 4 5 6 7 8 9 101112
        [[0,0,0,0,0,0,0,0,0,0,0,0], [0,0,0,0]],
        [[1,0,0,0,1,0,0,1,0,0,0,0], [0,0,0,1]], # C
        [[0,0,1,0,0,0,1,0,0,1,0,0], [0,0,1,0]], # D
        [[0,0,0,0,1,0,0,0,1,0,0,1], [0,0,1,1]], # E
        [[1,0,0,0,0,1,0,0,0,1,0,0], [0,1,0,0]], # F
        [[0,0,1,0,0,0,0,1,0,0,0,1], [0,1,0,1]], # G
        [[0,1,0,0,1,0,0,0,0,1,0,0], [0,1,1,0]], # A
        [[0,0,0,1,0,0,1,0,0,0,0,1], [0,1,1,1]], # H
        [[1,0,1,1,1,1,0,1,0,1,0,0], [0,0,0,0]], # --
        [[1,0,1,0,1,0,0,1,1,1,0,0], [0,0,0,0]], # --
        [[1,0,1,1,1,0,1,1,0,1,0,0], [0,0,0,0]], # --
        [[1,0,1,0,1,0,1,0,1,1,0,0], [0,0,0,0]], # --
        [[1,0,1,1,1,0,0,1,1,1,0,0], [0,0,0,0]], # --
        [[1,0,1,0,1,0,0,1,1,1,1,0], [0,0,0,0]], # --
        [[1,0,1,1,1,0,0,1,0,1,0,0], [0,0,0,0]], # --
        [[1,0,1,0,1,0,0,1,1,1,1,1], [0,0,0,0]], # --
        [[1,1,1,1,1,0,0,1,0,1,0,0], [0,0,0,0]], # --
        [[1,0,0,1,1,0,0,1,1,1,0,0], [0,0,0,0]], # --
        [[1,0,1,1,0,0,0,0,0,0,0,0], [0,0,0,0]], # --
        [[1,0,1,0,1,0,0,1,1,1,0,0], [0,0,0,0]], # --
        [[1,1,0,0,0,0,0,1,1,1,0,0], [0,0,0,0]], # --
        [[0,0,1,1,1,0,0,0,0,0,0,0], [0,0,0,0]], # --
    ]

    testpat = [
        [[0,0,0,0,0,0,0,0,0,0,0,0], [0,0,0,0]],
        [[1,0,0,0,1,0,0,1,0,1,0,0], [0,0,0,1]], # C
        [[0,0,1,0,0,0,1,0,0,1,0,0], [0,0,1,0]], # D
        [[0,0,0,0,1,0,0,0,1,0,0,1], [0,0,1,1]], # E
        [[1,0,0,0,0,1,0,0,0,1,0,0], [0,1,0,0]], # F
        [[0,0,1,0,0,0,0,1,0,0,0,1], [0,1,0,1]], # G
        [[1,1,1,1,1,0,0,1,0,1,0,0], [0,0,0,0]], # --
        [[1,0,0,1,1,0,0,1,1,1,1,0], [0,0,0,0]], # --
        [[1,0,1,1,0,0,0,0,0,0,0,0], [0,0,0,0]], # --
        [[1,0,1,0,1,0,0,0,0,1,0,0], [0,0,0,0]], # --
        [[0,1,0,0,0,0,0,1,0,0,0,0], [0,0,0,0]], # --
        [[0,0,1,0,1,0,0,0,0,0,1,1], [0,0,0,0]], # --
    ]

    # create a network with two input, two hidden, and one output nodes
    n = NN(12, 4, 4)
    # train it with some patterns
    n.train(pat)
```

```
    # test it
    #n.finn(testpat)
    print n.finnVal([1,1,1,1,1,1,0,0,0,0,1,1])



if __name__ == '__main__':
    test()
```