Användningen av neurala nätverk för tonigenkänning

Niklas Gustin, Victor Sellstedt, Björn Thorén, Kristoffer Åhgren
Handledare: *Lars Gråsjö*

2016-01-29

Abstract

Research of neural networks is essential for the continued development of modern artificial intelligence and deep learning structures. Neural networks are very useful tools to identify patterns, such as face recognition, that might be simple for humans to recognize but difficult for computers. With the seemingly endless possibilities for neural networks it raises the question of what is required for a neural network to be able to differentiate between different musical tones. By creating our own neural network and using generated data we studied the results generated by networks distinguishable by number of layers and nodes when they were given real, recorded tones. The results showed that differentiating between all of the tones of a smaller piano is a too complicated task for a three layer network to complete. By adding layers and nodes while reducing the number of tones to distinguish, we managed to create networks that were more sucessfull. Further research must be made, but this implies that by adding layers and nodes it might be possible to identify all of the tones, but at the cost of a longer training time.

Innehåll

1 FÖRKORTNINGAR OCH BEGREPP

1. NN - Neurala nätverk
2. AI - Artificiell intelligens
3. BP - Backpropagation

2 INLEDNING

2.1 Bakgrund

Det är vida känt att datorer är väldigt användbara inom ett flertal områden som t.ex. industri, kommunikation och informationslagring. Orsaken till detta är att datorer har möjligheter att göra miljontals beräkningar varje sekund vilket gör dem överlägsna människan när det kommer till att producera önskade resultat, så länge det bara rör sig om att göra beräkningar. Andra saker som t.ex. känna igen ett ansikte eller koppla en bild på ett djur till dess namn är saker som är triviala för de flesta barn men är nästintill omöjligt för de flesta datorer. Nyckeln till detta är människans möjlighet att tänka, att vi har en hjärna. Hundratusentals neuroner som avfyras i hjärnan varje sekund gör det möjligt för oss att tänka och lära. Under 1900-talet började man fundera över datorer som kunde fungera som hjärnor genom att härma hjärnans uppbyggnad. Man skulle skapa s.k. Neurala Nätverk (NN) som var grafer där varje nod representerade en neuron och kanterna mellan dem representerade synapserna. 1943 utvecklade Warren McCulloch och Walter Pitts en modell för NN som kallades för tröskel-logik som lade grunden för forskning om NN att dela upp sig mellan de biologiska processerna i hjärnan och hur man kunde använda NN för artificiell intelligens (AI).[1] De möjligheter som forskning om NN skapar är nästintill outtömliga eftersom att NN ligger i grunden för utveckling av AI och är användbara när det kommer till olika områden av mönsterigenkänning.

2.2 Teori

Som det tidigare nämnts ligger grunden för NN i att efterlikna hjärnans uppbyggnad och funktion. Genom att kombinera det med annan viktig teori för ljudigenkänning och tidskomplexitet bör man kunna konstruera ett NN som kan tränas till att känna igen toner.

2.2.1 Neuroner

Det mest grundläggande med hjärnan är dess neuroner. Neuronerna i hjärnan fungerar genom att de tar emot elektriska signaler ifrån receptorer eller andra neuroner för att i sin tur avfyra elektriska impulser vidare till andra neuroner. Detta är det som gör det möjligt för oss att tänka och det finns några olika modeller för att simulera hjärnas neuroner. De olika modellerna som vi kommer ta upp bygger till stor del på tröskel-logik, det binära system som McCulloch och Pitts arbetade med. Kortfattat bygger tröskellogiken på att man har en neuron som representeras av en nod i en graf får en viss indata $\{x_1, x_2, x_3 ... x_n\}$ som man summerar och jämför med en viss tröskel θ. Om $\sum_{i=1}^{n} x_i \geq \theta$ "avfyrar" och skickar vidare 1 som utdata. Annars om $\sum_{i=1}^{n} x_i < \theta$ "avfyrar" den inte och skickar vidare 0.[2] Detta system fungerar lite som hjärnans exitatorer (indata) och inhibitorer (tröskeln) som styr om en neuron kommer avfyras. Med denna grundläggande princip går vi vidare till två olika modeller på noder i NN.

Perceptroner
1957 kom Frank Rosenblatt[3] på en algoritm för en modell av en endaste neuron som han kallade för perceptron. Perceptronen består av tre delar: indata, processor och utdata.[4] Perceptronen kan ta emot hur stort antal indata som helst men den skickar bara ut ett enda resultat som är antingen 1 eller 0. Det här påminner väldigt mycket av tröskel-logiken men Rosenblatt införde en till egenskap för att beräkna

[1] Wikipedia. *Artificial neural network*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.
[2] R. Rojas. *Threshold Logic*. Okt. 1996. URL: http://cgm.cs.mcgill.ca/~godfried/teaching/dm-reading-assignments/Threshold-Logic.pdf.
[3] Wikipedia. *Perceptron*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Perceptron.
[4] Daniel Shiffman. *The nature of code*. 2012. Kap. 10.

utdatan, han införde vikter. Det betyder att om vi igen har indatan $\{x_1, x_2, x_3 ... x_n\}$ och tröskeln θ fungerar perceptronen enligt:[5]

$$output = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i x_i \leq \theta \\ 1 & \text{if } \sum_{i=1}^{n} w_i x_i > \theta \end{cases}$$

Det här ger många olika resultat beroende på vikterna men det finns ändå stora begränsningar. Nodens funktion kan beskrivas som en trappstegsfunktion vilket innebär att en liten förändring av vikterna kan antingen ge nästan exakt samma resultat eller så kan den ge ett helt annat resultat. Ett annat problem är att det blir inte så avancerade resultat om man endast använder en enda perceptron, det problemet löser vi enkelt genom att vi skapar ett nätverk som består av flera "lager" med perceptroner där det första lagret får indatan och de kommande lagren får det förra lagrets utdata. Med hjälp av många perceptroner som arbetar parallellt och baserar sina resultat på resultaten från det förra lagret kan man nu få mycket mer exakta resultat.

Man kan föreställa sig det såhär, tänk dig att vi har ett företag som förvaltar aktier och har arbetare, chefer och en VD. Arbetarna ska varje månad skriva en sammanställning till sin chef om vilka aktier de tycker man ska köpa. Cheferna ska i sin tur skriva en sammanställning till VD:n som delvis är beroende av hur bra cheferna tycker deras arbetares förslag är. VD:n drar sedan sin slutsats om vilka aktier som ska köpas baserat på vad hans chefer rekommenderade och köper dessa aktier. VD:ns val är baserat på en bredd av information som har blivit säkrare och säkrare med varje steg man klättrar uppåt i heirarkin på företaget. Med tiden blir företaget bättre eftersom att VD:n lär sig vilka chefer som har bra omdömen och cheferna lär sig vilka arbetare som har bra omdömen vilket gör att företaget för varje gång man köper aktier blir bättre. Det här är ett sätt att försöka visualisera ett flerlagers NN och kan vara till hjälp att förstå varför det är bättre med fler lager än bara ett. Men bara för att fler lager är bättre än en perceptron betyder inte det att man behöver ha hur många lager som helst, det räcker för det mesta med tre lager.
För att återgå till NN i sig. Om man nu varierar vikterna mellan perceptronerna kan man få många olika utdata utgående ifrån samma indata vilket innebär att det finns en större möjlighet än tidigare att skapa en inlärningsalgoritm.

Sigmoidneuroner
En av förutsättningarna för att få en effektiv inlärningsalgoritm är att man kan göra små förändringar eftersom att alla stora förändringar påverkar utdatan kraftigt. Som tidigare nämdes kan perceptroneras aktiveringsfunktion eller process beskrivas av en trappstegsfunktion som går mellan 0 och 1. Denna trappstegsfunktion gör att om man ändrar en vikt med väldigt lite kan det vara skillnaden mellan att perceptronen skickar 1 eller 0[6]vilket gör att en inlärningsalgoritm kan vara väldigt osäker. För att åtgärda detta problem skapas en s.k. sigmoidneuron. Sigmoidneuronen baseras på sigmoid funktionen $\sigma = \frac{1}{1+e^{-z}}$ och kan istället för att ge utdatan 0 eller 1 ge utdata *mellan* 0 och 1. Sigomoid funktionens variabel z representerar $\sum_{i=1}^{n} w_i x_i - \theta$ och nu används tröskeln som en tröskel för summan av alla vikter och indata (kan också beskrivas som skalärprodukten av vikterna och indatan) men kommer fylla samma funktion. Det innebär att aktiveringsfunktionen för sigmoidneuronen kan beskrivas som:

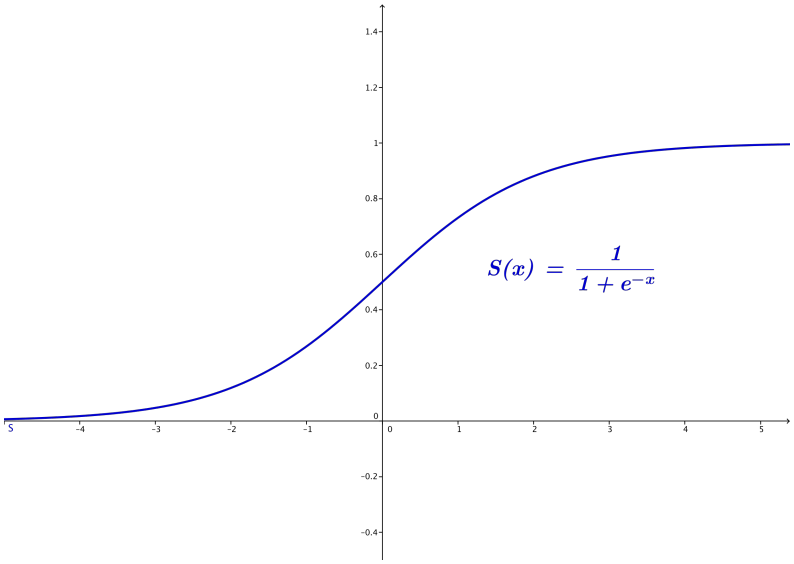
$$\sigma = \frac{1}{1+e^{-(\sum_{i=1}^{n} w_i x_i - \theta)}}$$

Sigmoidneuronen påminner mycket om perceptronen trots det att det är så stor skillnad mellan sigmoid funktionen och trappstegsfunktionen. T.ex. om $z \to -\infty$ (eller går mot ett litet tal) så kommer $\sigma(z) \to 0$ och på liknande sätt om $z \to \infty$ så $\sigma(z) \to 1$. Men dess fördel är att däremellan sker förändingarna stegvis. En fördel med sigmoidneuronen är att sigmoid funktionen till skillnad från trappstegsfunktioner är deriverbar vilket ger ännu fler möjligheter med inlärningsalgoritmer.

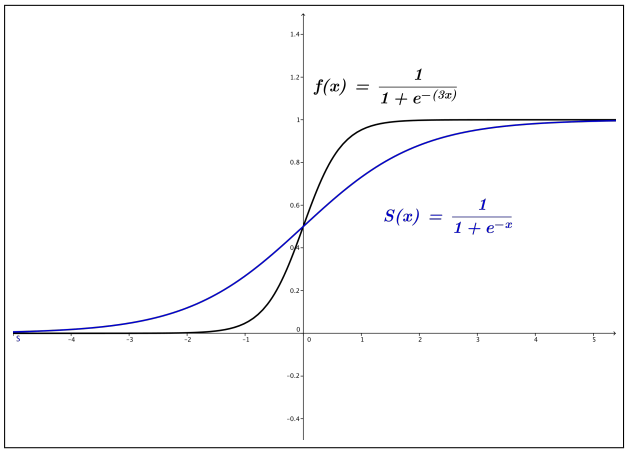
[5]Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.
[6]Ibid.

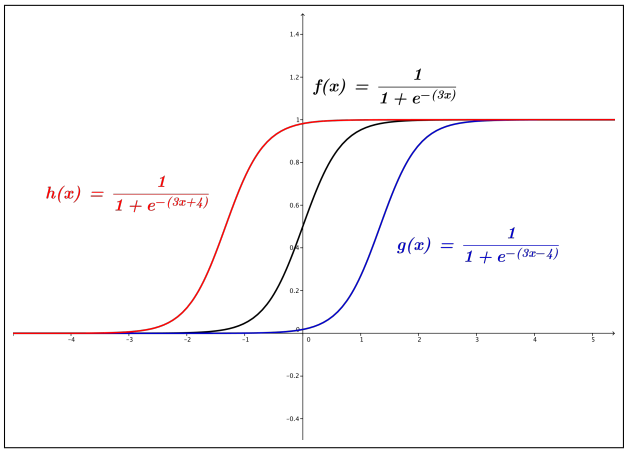
Biasnoder

Som nämt i föregående stycke kan man ha en aktivationsfunktion (för sigmoidneuroner sigmoidfunktionen) som gör en viktad summering av alla indata $\sum_{i=1}^{n} w_i x_i$ och jämför den med en tröskel θ. Denna tröskel skulle också kunna beskrivas med en annan typ av nod som inte tar emot indata och alltid returnerar 1. Därefter kan man ge den en egen vikt som motsvarar θ. Denna modell gör det lättare att förändra neuronernas trösklar eftersom att de nu är noder i NN och kommer följa samma regler som resten av nätverket med undantag för att de inte har några indata. Det gör att det blir enklare att programmera nätverket då det är lättare att lägga till en vikt för en nod jämfört med att skapa en särskild uppdateringsalgoritm för tröskeln eller biasen. Mer om själva uppdateringen av vikterna ligger under inlärning och backpropagation. För att vidare gå in på biasnodens funktion behöver vi analysera sigmoidfunktionen kort. Om vi tänker oss sigmoidfunktionen som en funktion av en variabel, $\sigma(x)$, så kan den tänkas se ut så här:



Om vi nu istället föreställer oss ett neuron som endast tar emot en indata och som har en bias som är noll kommer summan av indata och bias vara $\sum_{i=1}^{n} w_i x_i - 0 = wx$ och aktiveringsfunktionen vara $\sigma(wx)$. Om vi nu ändrar på vikten påverkar det funktionens lutning när den går från 0 till 1 men förändrar inte dess läge i x-led som man kan se i det vänstra exemplet. Om vi nu håller vikten w konstant och ändrar biasnodens värde är lutningen konstant men funktionen förflyttar sig i x-led som man kan se i det högra exemplet.





NN kan alltså använda biasnodens konstanta utdata för att justera sitt läge längs en axel. Biasnoder i samspel med flera lager innebär att man kan skapa mycket avancerade funktioner.

2.2.2 Inlärningsalgoritmen

Det finns olika sätt att angripa problemet med att få NN att lära sig och bli bättre. Det finns ett antal olika inlärningsalgoritmer men den vanligaste är backpropagation med hjälp av gradientsökning.

Backpropagation
Backpropagation (BP) är en förkortning av *"Backward propagation of errors"*[7] som uppkom under 1970 talet men betydelsen av algoritmen var inte helt förstådd tills 1986 när David Rumelhart och Ronald Williams skrev en artikel angående dess effektivitet för många NN.[8] BP användersig av det som man kallar för övervakad inlärning vilket innebär att man tillhandhåller NN ett facit för sin träningsdata så den kan jämföra med det resultat som den själv fått. BP är en vanlig inlärningsmetod för NN och används tillsammans med en optimiserings metod som t.ex. Gradientsökning. Gradientsökning är en optimiseringsalgoritm som söker att hitta en minpunkt för en funktion genom att gå ett antal steg som är negativt proportionerligt mot derivatan av en funktion.[9] I fallet med BP resulterar det i att man söker att minimera errorfunktionen mellan NN:s resultat och facit för inlärningsdata. BP algoritmen fungerar genom att man först ger NN indatan som den får bearbeta och producera ett resultat. Sedan beskriver man en errorfunktion baserad på det önskade resultatet och det faktiska resultatet t.ex. funktionen
Mean Square Error ($MSE = \frac{(y-t)^2}{2}$).
Därefter gås hela nätverket igenom baklänges och ändrar vikterna enligt principen för gradientsökning genom att minska vikterna och biasnoderna med en inlärningsfaktor multiplicerat med derivatan för errorfunktionen vid den noden. Inlärningsfaktorn kan vi alltså använda för att beskriva hur stora ändringar vi ska göra varje gång vilket motsvarar hur snabbt nätverket lär sig. En mer fördjupad matematisk förklaring till hur och varför den funkar finns med som bilaga A. Inlärningsfaktorn α är en konstant faktor sådan att gradient descent inte skall hoppa över minima på grund av för stora hopp, då gradientsökningens princip är för α går mot 0.

2.2.3 Nätverkets storlek

Nätverkets storlek kommer att vara beroende av flera olika faktorer. De faktorer som väger tyngst för storleken är komplexiteten hos BP samt antalet lager i nätverket.

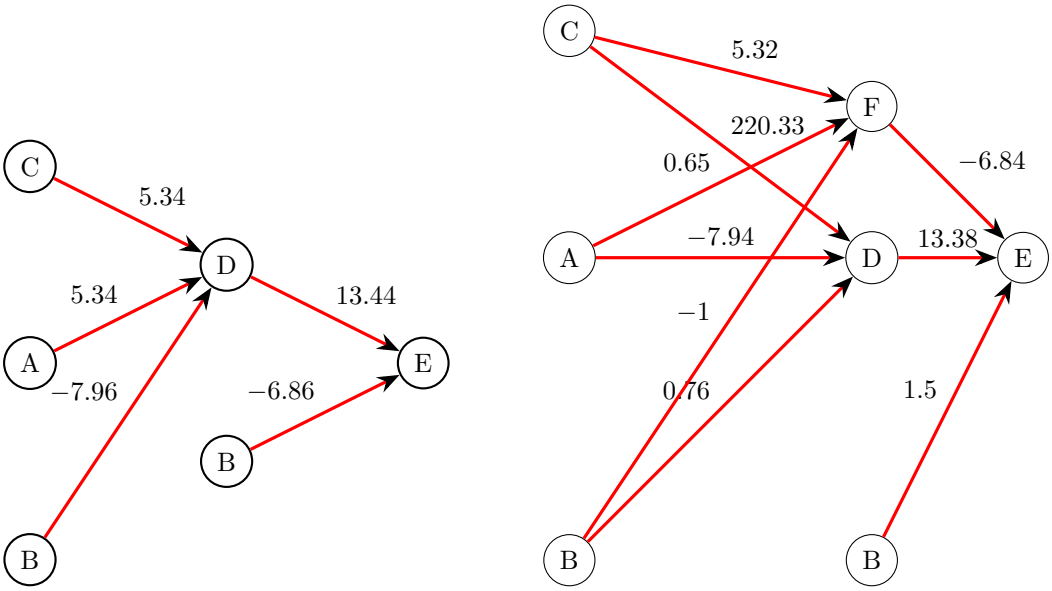
Antal lager
Antalet lager som ska användas för ett nätverk är mycket svårt att bedömma då det beror på många faktorer som antalet indata, om funktionen är sammanhängande, vilken aktiveringsfunktion som används i neuronera os.v. När antalet lager i ett nätverk ökar kan nogrannheten öka men det finns ingen försäkring på att det kommer göra det. Det finns t.ex. två olika modeller för antalet lager. NN med få lager kallas för ytliga nätverk och de NN som har flera dolda lager mellan indatalagret och utdatalagret kallas för djupa nätverk.[10] De ytliga nätverken är lättare att programmera och arbeta med eftersom att algoritmen för inlärning blir lättare tack vare att de består av ett fåtal dolda lager (1-2). De kan approximera de flesta funktioner[11] men nackdelen är att det kan vara svårare för ytliga nätverk att vara generaliserade eftersom att de kan kräva stora mängder träningsset och noder i de dolda lagren. För vissa funktioner fungerar de lika bra som de djupa nätverken som är svårare att implementera[12] vilket innebär att ytliga nätverk fortfarande är mycket användbara.

Inlärning
Den algoritm för backpropagation som kommer att beskrivas är anpassad för ett nätverk med 3 lager

[7] Wikipedia. *Backpropagation*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Backpropagation.
[8] Nielsen, *Neural Networks and Deep Learning*.
[9] Wikipedia. *Gradient descent*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Gradient_descent.
[10] Wikipedia. *Deep Learning*. Jan. 2016. URL: https://en.wikipedia.org/wiki/Deep_learning.
[11] Yoshua Bengio och Yann LeCun. "Scaling Learning Algorithms towards AI". I: *Large-Scale Kernel Machines* (aug. 2007), s. 3,25.
[12] Ibid.

(indatalager, dolt lager och utdatalager) eftersom att det räcker med ett dolt lager för de flesta NN.[13][14] Algoritmen är beskriven i pseudokod i bilaga D och kommer att ha en tidskomplexitet på $\mathcal{O}(|I \cdot D + U \cdot D|) = \mathcal{O}(|W|)$ där $|W|$ betecknar storleken av mängden av alla vikter och I, D, U är indata, dolda och utdatalagret för nätverket enligt bilaga E som baseras på principen om gradient descent.[15]

Antal noder
När det kommer till antalet noder i varje lager finns det några som har försökt skapa tumregler för hur många noder som ska användas F. Sanningen är att det är en otroligt komplicerad uppgift att komma fram till hur många noder som ska användas eftersom att det är nästintill individuellt för varje NN. Exempelvis har vi här ett nätverk som beräknar AND (vänster) och ett nätverk som beräknar XOR (höger). AND returnerar 1 om man ger den $[1, 1]$ som input och 0 om man ger den $[1, 0], [0, 1], [0, 0]$. XOR returnerar 1 om den får $[1, 0] eller [0, 1]$ och 0 annars. Båda dessa funktioner kan verka någorlunda simpla men skillnaden är att XOR inte går att träna om det inte har 2 noder i det dolda lagret medan AND kan tränas med endast 1 nod i det dolda lagret. Eftersom att det är så svårt att välja antalet noder



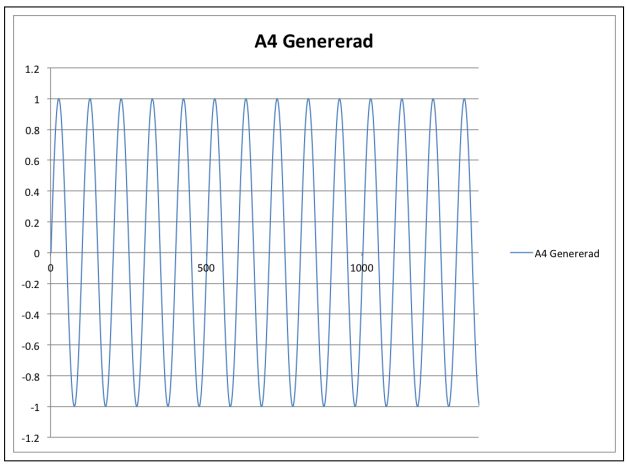
kan man dock använda sig av vissa tumregler för att påbörja testerna och sedan optimera vid behov. T.ex. kan man använda en tumregel som säger att antalet noder i det dolda lagret ska vara medelvärdet på indatalagret och utdatalagret[16] och anpassa därefter när det behövs enligt F.
Både AND och XOR har väl definierad indata. Vårt ljudproblem blir mer abstrakt och mindre väldefinierat indata, men precis som i fallet med de logiska kretsarna så har vi utdata som ska ges av heltal 1 eller 0. För att jämföra logiska kretsar och vårt problem så använder de logska kretsarna väldigt klara diskreta indata i de olika fallen: $0, 0,\ 1, 0,\ 0, 1,\ 1, 1$. Vårt fall har en betydligt mer mjuk, kontinuerlig kurva. Dessutom så ska nätverket inte bara känna igen någon form av lutning (som AND och XOR kan föreklas till), vårt ska känna av en periodtid, eller frekvens.

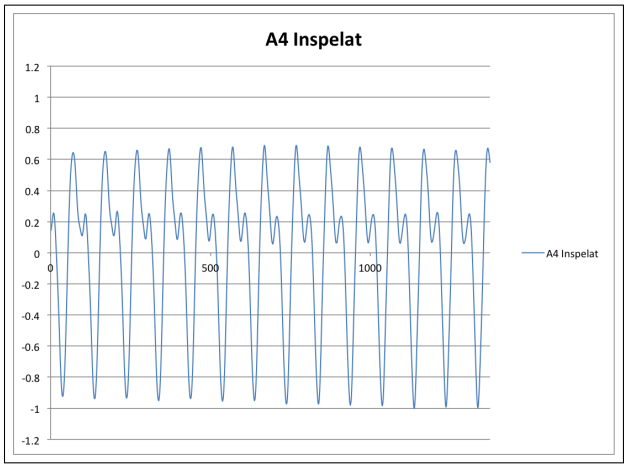
2.2.4 Data

För att vi ska kunna träna nätverket måste vi ge den indata och korrekt utdata som kan användas i errorfunktionen. Vi generar en mängd med testdata av *"rena toner"* som genereras enligt $y = \sin(f_{not} *$

[13] Douglas Y'barbo. *Stack Exchange Forum, How to choose the number of hidden layers and nodes in a feedforward neural network.* Not: baserad på Bilaga C. Aug. 2010. URL: http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.
[14] Bengio och LeCun, "Scaling Learning Algorithms towards AI".
[15] Wikipedia, *Gradient descent.*
[16] Y'barbo, *Stack Exchange Forum, How to choose the number of hidden layers and nodes in a feedforward neural network.*

$2\pi t$).[17]
f_{not} är frekvensen för respektive not av de 87 tonerna A0 t.om. B7 och t är tiden i sekunder. C8 som är den högsta frekvensen svänger såpass fort så att det blir svårt att skapa bra genererad data. Dessutom dör tonen ut fort på pianot vilket gör att det finns stora risker att det blir oanvändbar mätdata.
Ljud är tryckvågor. Dessa går naturligtvis inte att spara i rå form på en dator. Detta då en dator enbart kan spara binära värden och inte fysiska vågor. Därför måste man för att kunna arbeta med ljudvågor på en dator representera de i binär form. Det finns en hel del sätt att göra detta. Den vanligaste metoden är att spara det som en mängd data punkter som beskriver kompressionen hos ljudvågen. För att spela in ljud måste man interagera med en mikrofon. Vi bestämde oss för att använda SFML. SFML är ett gränssnitt mellan program och datorns auditiva samt grafiska hårdvara. I projektet är vi främst intresserade av ljudens frekvens eftersom man inom musiken kategoriserar ljud som toner efter deras frekvens. Ett fenomen som uppkommer när man spelar olika instrument är att man får svävningar vilket gör att inspelade toner får en annorlunda form gentemot rena toner. Svävningar är när två toner med likartad frekvens spelas samtidigt så uppfattas de av örat som en och samma mellanliggande ton.[18] Detta kan förklaras enkelt med hjälp av matematik: Sin(a + b) + Sin(a - b) = 2 * Sin(a) * Cos(b). Då ser vi även att vågen kommer gå upp och ner i amplitud över ett större intervall, ty allt multipliceras med Cos(b).
Eftersom att pianot som ett musikinstrument är byggt så att det låter annorlunda jämfört med en t.ex. stämgaffel vars ljudvåg är mer snarlik en perfekt sinusvåg innebär det att vår genererade data kommer att skilja sig ifrån den inspelade. Detta kan bättre illustreras med de två graferna nedan som föreställer den genererade tonen och den inspelade tonen av A4 440Hz.





Inlärningsdata är inspelad i ett tyst rum med minimalt med bakgrundsbrus på ett piano. Det finns ett antal sätt att representera ljud på en dator. Den vanligaste formen är att spara det som en mängd datapunkter som beskriver amplituden på ljudvågen. För att spela in ljud måste man interagera med en mikrofon. Vi bestämmde oss för att använda SFML. SFML är ett gränssnitt mellan program och datorns auditiva samt grafiska hårdvara.
Vi ville minimera antalet faktorer nätverket måste handskas med. Vi normaliserade ljudvågens amplitud så att alla toppar och dalar alltid låg mellan -1 och 1. Datan som spelas in delas upp i mindre segment, i varje av dessa segment väljs värdet som ligger längst ifrån x-axeln och dividerar alla andra värden med dess absolutbelopp och detta ger en kurva som förhoppningsvis ger högre vikt på frekvensen än det annars skulle gjort. Detta eftersom att när man spelar in tonerna är det svårt att spela in alla med en konstant ljudstyrka. t.ex. avtar ljudstyrkan allt eftersom att tiden går. Eftersom att NN tar emot flyttal eller decimaltal innebär det att det skulle kunna resultera i att NN returnerar olika toner om man spelar samma ton med olika ljudstyrka eftersom att högre ljudstyrka ger högre amplitud vilket ger ett högre indatatal åt det NN.

[17] Jonathan Rogness. *Trigonometry in Nature*. Email: rogness@math.umn.edu. URL: http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.
[18] Wikipedia. *Svävning*. [Online; hämtad 29-januari-2016]. 2015. URL: //sv.wikipedia.org/w/index.php?title=Sv%C3%A4vning&oldid=32054228.

2.3 Syfte och frågeställningar

Syftet med vår undersökning är att lära oss mer om hur NN fungerar och hur man kan anpassa dem för olika uppgifter. Vi vill också undersöka hur effektivt nätverket är eftersom att en av de stora begränsningarna med NN var att de krävde mycket processorkraft vilket gjorde att forskningen stannade upp ett tag runt 1970-talet tills det att man kunde tillverka datorer som kunde utföra fler processer per sekund än tidigare. Vi är också intresserade av hur bra vårt nätverk blir på att gissa rätt (samt hur liten vi kan få errorfunktionen att bli för alla noder), hur lång tid det tar för det att lära sig och vad det har för begränsningar.

2.4 Frågeställningar

1. Hur kan man anpassa ett neuralt nätverk för att känna igen toner?
2. Hur träffsäkert blir nätverket?
3. Vad är det för tidskomplexitet på nätverket? (körning, inlärning)

2.5 Avgränsningar

Från början var arbetet tänkt att omfatta röstigenkänning men vi gjorde därefter om det till ljudigenkänning. Då det kändes för brett och ospecificerat gjorde vi det till ton-igenkänning. Inom ton-igenkänning vill vi inkludera tonerna för främst piano och hur nätverkets påverkas av att pianots ljudvåg har andra vågor som är superpositionerade på tonen.
På grund av tidsbrist har vi behövt strukturera om och testa ett nätverk som bara undersöker om den kan lära sig att se skillnad på A4 och C4 när det består av 3 lager och 4 lager och ett nätverk som undersöker om den kan särskilja 12 toner (C4-B4).

3 METOD

3.1 Förklaring av Nätverket

Nätverket består av en lagerstruktur. Lagren samlas i något som kallas ett lagerkluster, lagren innehåller noder. I varje lager finns det en så kallad bias-nod som alltid ger utdata 1. Varje nod (förutom bias-noden) innehåller en kant till alla noder i det tidigare lagret (inklusive bias-noden). Det första lagret representeras endast av en lista av utdata som går med viktning till det första dolda lagret.

3.1.1 Kodstruktur

Koden består av ett klusterobjekt som innehåller en indatalista (som beskrivet tidigare) och en lista av lager. Varje lager innehåller ett antal noder som innehåller (som beskrivet tidigare) alla vikter till det tidigare lagret.

3.2 Nätverken

Enligt teorin kommer antalet beräkningar förhålla sig kvadratiskt gentemot antalet noder i de olika lagren. Ljudet har spelats in med en frekvens 44000 Hz. Eftersom varje inläst värde ska representeras av en indatanod innebär det att antalet beräkningar förhåller sig kvadratiskt i förhållande till den inspelade tiden. Nätverket hinner inte bearbeta värdena från en sekunds inspelning på en sekund om alla värden beräknas som ett intervall. Istället styckas de inspelade värdena för en sekund upp i 32 mindre intervall av längd:

$$\frac{44000}{32} = 1375$$

Även om nätverket kommer att behöva bearbeta 32 intervall av denna längd per sekund så kommer det gå snabbare att arbeta igenom de 32 mindre segmenten. Detta då antalet beräkningar var kvadratiskt

proportionel mot antalet värdeni intervallet, alltså kommer $\frac{1}{32}$ antalet värden beräknas på $(\frac{1}{32})^2$ av tiden vilket i sin tur innebär att det går 32 gånger så snabbt om man delar upp det i än mindre segment. Så en legitim följdfråga är varför vi inte drar ner lägden av segmenten till ett. Detta är på grund av att med färre beräkningar tillkommer att lite data inte räknas med. Så i detta fall innebär det att vi inte riktigt kan beräkna toner av lägre frekvens då vi inte hinner mäta upp ens en halv period. BP valdes som inlärningsalgoritm för alla nätverk eftersom att den är relativt enkel och fungerade tillräckligt bra för upplärning av AND och XOR nätverken. Inlärningen kan bromsas om vi hamnar i ett lokalt minimum[19] men det är en risk som är svår att undvika oberoende av val av inlärningsalgoritm. Den algoritm som kommer användas för BP är beskriven i pseudokod i bilaga D. tonera dock att den bara är beskriven för nätverken med 3 lager. Vi bygger upp nätverket och sparar det så att det går att återskapa ifrån det vi hade tidigare.

3.2.1 Nätverk 1

Nätverket bestod av 87 utdatanoder vilka representerade de 88 tonerna av pianot om man utelämnar C8. Det dolda lagret i det första nätverket har ungefär medelvärdet av utdata och indatalagrens noder (avrundat uppåt) alltså:

$$\frac{1375 + 87}{2} = 731$$

Men för att ligga över medel använde vi 732 noder. Nätverket tränades med en slumpad lista av genererade toner, ListN, som innehåller 10^5 fall av 30 sekvenser för de 87 olika tonerna som har delats upp i 31 mindre segment. Nätverket tränades med en inlärningsfaktor på 10,1,0.5,0.1.

3.2.2 Nätverk 2

Efter det att Nätverk 1 misslyckades med att returnera korrekta svar ökas antalet nätverk i ett försök att anpassa nätverket. Det dolda lagret i det andra nätverket bestod av 7000 noder för att undersöka om det gav bättre resultat.

3.2.3 Nätverk 3

Då Nätverk 2 inte heller lyckas testas ett nätverk som endast försökte skilja på A4 och C4.

3.2.4 Nätverk 4

Efter det att även Nätverk 3 misslyckats testades ett 4-lagers nätverk som försökte skilja på A4 och C4. Nätverket bestod alltså av 1375 indatanoder, 732 dolda noder i det första dolda lagret, 732 dolda noder i andra dolda lagret och 2 noder i utdatalagret. Nätverket tränades med den genererade listan *BList.txt* som bestod 10^5 testfall av de 30 sekvenser (av 31)genererade toner för A4 och C4 med en inlärningsfaktor $\alpha = 0.2$. Nätverket testades sedan med en lista *2outTL.txt* som gick igenom alla de 32 fallen av de inspelade tonerna för A4 och C4 respektive.

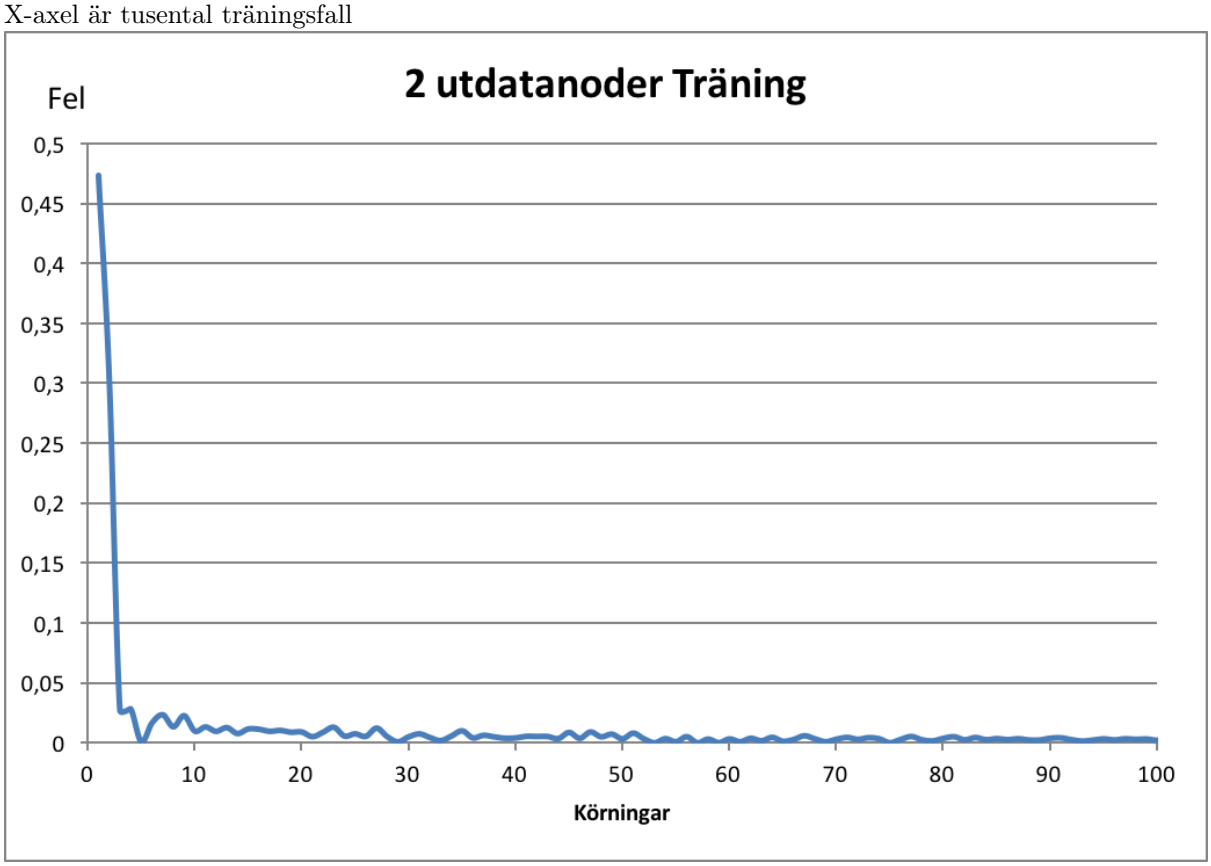
3.2.5 Nätverk 5

När Nätverk 3 lyckades skilja på A4 och C4 testades det att öka antalet utdatanoder till 12, alltså försökte det skapas ett nätverk som kunde skilja på tonerna i spannet C4-B4. Antalet dolda noder ökades också i varje lager eftersom att det är mer komplext att försöka känna igen 12 toner. Nätverket tränades med en slumpad lista *12List.txt* med 10^5 fall av de genererade tonernas 30 sekvenser (av 31). I första varvet av träning användes en inlärningsfaktor $\alpha = 0.2$. Nätverket testades sedan med en lista *TESTLIST.txt* som gick igenom alla de 32 fallen av de inspelade tonerna för C4 till B4 (resultat redovisas i bokstavsordning) respektive.
I nästa varv användes *12List.txt* igen men med en inlärningsfaktor $\alpha = 0.1$. Sedan testades *TESTLIST.txt*

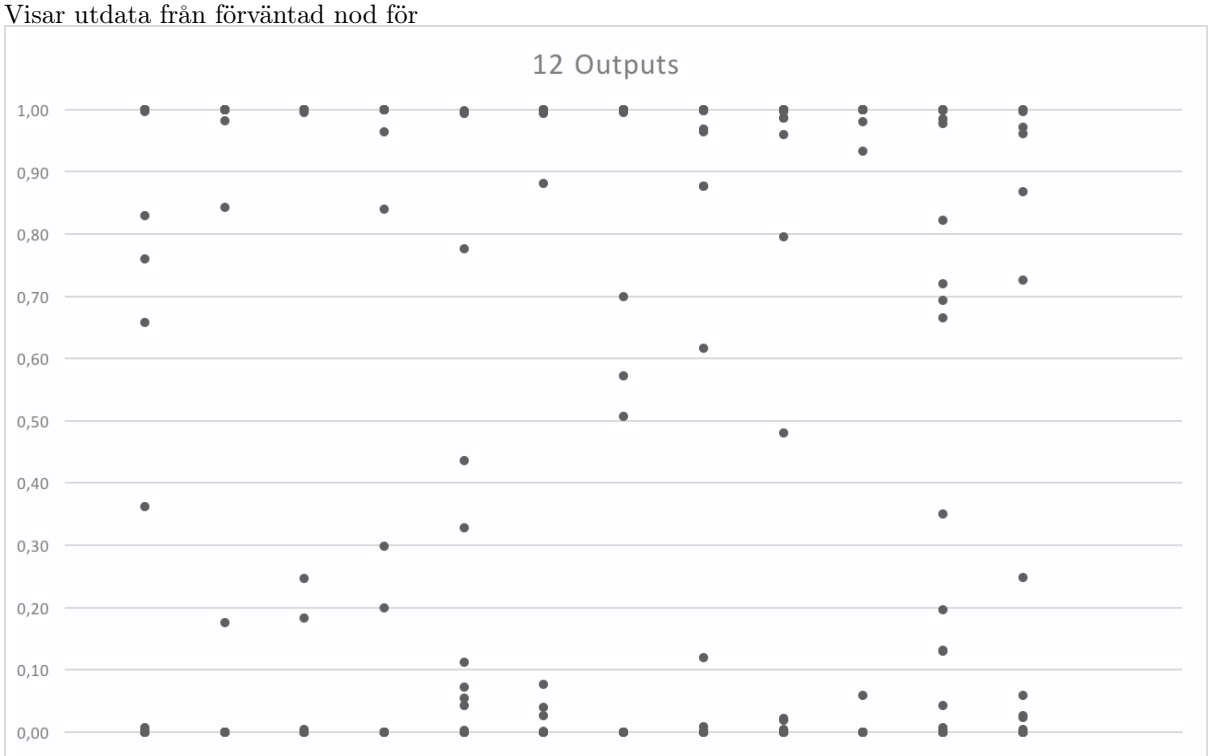
[19] Nielsen, *Neural Networks and Deep Learning.*

4 RESULTAT

X-axel är tusental träningsfall



Visar utdata från förväntad nod för



4.0.1 Nätverk 4

Följande tabell redovisar datan från det otränade nätverket och efter att nätverket har tränats en gång med $\alpha = 0.2$. Nätverken tar emot en av de 32 inspelade sekvenserna i tur och ordning. "Rätt" beskriver vad noden för den förväntade tonen gav på testfallet och "Fel" beskriver vad den andra gav på det testfallet. T.ex. Rätt 1 A4 är vad noden för A4 gav resultat på första fallet av den inspelade A4 tonen. Det önskade resultatet för Rätt 1 A4 är därför 1.
Fel 1 A4 är vad noden för C4 gav för resultat för första fallet av den inspelade A4 tonen. Det önskade resultatet för Fel 1 A4 är alltså 0.

	Rätt nod		Fel nod		Rätt nod, $\alpha = 0.2$		Fel nod, $\alpha = 0.2$	
Fall	*A4*	*C4*	*A4*	*C4*	*A4*	*C4*	*A4*	*C4*
1	0.530197	0.441537	0,447821	0,529853	0.924069	0.992079	0,076348	0,007774
2	0.528355	0.44502	0,442688	0,525828	0.064525	0.994155	0,936125	0,005715
3	0.528868	0.44783	0,452777	0,538675	0.592835	0.997007	0,416897	0,003021
4	0.528851	0.445894	0,443227	0,522045	0.996699	0.994699	0,003153	0,005171
5	0.524157	0.443848	0,449508	0,530202	0.957486	0.992996	0,042617	0,006833
6	0.533701	0.443743	0,44305	0,528417	0.002150	0.994996	0,997864	0,004937
7	0.522207	0.44727	0,444517	0,533847	0.158563	0.997109	0,836826	0,002880
8	0.530934	0.450109	0,448503	0,522099	0.955309	0.991146	0,045434	0,008695
9	0.530914	0.442246	0,446924	0,526682	6.41E-05	0.995466	0,999939	0,004415
10	0.525792	0.449017	0,445618	0,538008	0.999603	0.996711	0,000373	0,003310
11	0.52956	0.447838	0,445414	0,516878	0.9928	0.990002	0,007243	0,009715
12	0.528579	0.443131	0,447913	0,533056	0.989372	0.994965	0,010743	0,004983
13	0.52819	0.446004	0,44788	0,533912	6.12E-05	0.997142	0,999942	0,002873
14	0.528459	0.445664	0,447484	0,522907	0.999558	0.993909	0,000418	0,005956
15	0.526118	0.449245	0,444007	0,524843	0.953059	0.989591	0,046725	0,010184
16	0.529968	0.448834	0,448414	0,532862	0.96366	0.99631	0,036800	0,003698
17	0.527979	0.445778	0,4445	0,532014	0.000111	0.998189	0,999892	0,001796
18	0.528232	0.445666	0,450206	0,526373	0.99906	0.984567	0,000917	0,015137
19	0.528529	0.444762	0,442747	0,523086	0.817736	0.992016	0,179277	0,007792
20	0.527944	0.448757	0,449423	0,538247	0.989201	0.997474	0,010877	0,002572
21	0.530264	0.444677	0,446674	0,522857	0.000644	0.994722	0,999383	0,005201
22	0.526533	0.446885	0,445994	0,526291	0.997747	0.989485	0,002223	0,010286
23	0.529135	0.451268	0,449005	0,525382	0.999029	0.99505	0,000933	0,004897
24	0.527022	0.445682	0,443396	0,533444	0.972834	0.9979	0,027020	0,002087
25	0.529065	0.446183	0,446783	0,530342	0.969881	0.993137	0,030446	0,006786
26	0.526292	0.437518	0,448209	0,525356	0.000300	0.990682	0,999712	0,009031
27	0.532195	0.443127	0,443767	0,530157	0.996966	0.99479	0,003010	0,005185
28	0.523379	0.444124	0,45074	0,526417	0.960684	0.995993	0,038309	0,003936
29	0.532881	0.449827	0,441478	0,53203	0.973331	0.996948	0,026727	0,003055
30	0.523641	0.448685	0,450937	0,51989	0.041084	0.993572	0,959556	0,006283
31	0.532865	0.451292	0,443493	0,529569	0.892258	0.991648	0,109335	0,008270
32	0.523558	0.445709	0,44967	0,527829	0.972593	0.995818	0,026621	0,004130
Medelv.	0.528261	0.446161	0,446648	0,528418	0.691664	0.994071	0,308490	0,005831
Rätt	32	0	-	-	23	32	-	-
%	100	0	-	-	71,875	100	-	-

4.0.2 Nätverk 5

Följande nätverk har tränats 1 gång med $\alpha = 0.2$ beskriver utdatan som noden för den förväntade tonen gav på det specifika fallet av den inspelade tonen.

Fall	*A#4*	*A4*	*B4*	*C#4*	*C4*	*D#4*	*D4*	*E4*	*F#4*	*F4*	*G#4*	*G4*
1	1.10E-07	6.49E-05	0.999998	1.98E-15	0.076692	0.037060	0.52378	1.73E-07	0.024735	1.54E-13	9.67E-13	0.83583
2	0.999871	1.96E-12	0.993727	0.999936	1.73E-05	0.001426	2.86E-08	0.962536	4.63E-07	1.06E-15	2.96E-10	0.99843
3	6.07E-12	0.178397	4.73E-12	9.79E-09	1.04E-06	6.05E-14	3.10E-12	9.79E-05	0.98278	6.01E-15	1.81E-06	3.77E-08
4	0.999976	0.999999	1	6.68E-06	3.72E-05	0.068889	6.78E-14	4.93E-08	1.42E-10	4.55E-05	0.121349	8.07E-14
5	6.70E-10	0.000113	0.233541	0.999881	0.000131	0.99947	1.38E-13	0.009097	0.999959	1	0.977761	1.88E-06
6	0.342304	1.02E-08	7.67E-13	4.63E-14	0.000120	6.34E-17	8.71E-10	7.94E-05	3.39E-11	1	0.997871	0.996374
7	0.000968	0.999924	1	0.999986	0.000100	0.000347	0.672319	7.67E-10	0.999993	0.999999	0.999214	0.676149
8	0.712442	1.00E-12	8.68E-09	6.52E-16	0.003652	0.997878	0.99987	0.001664	4.83E-12	8.39E-11	0.999846	2.69E-09
9	3.86E-09	1.05E-09	2.37E-07	0.99997	0.000311	7.36E-15	0.999983	0.99934	0.999994	7.46E-16	0.999695	5.73E-10
10	0.997138	1	1	0.30181	0.046631	7.97E-07	0.999962	1.23E-09	1.03E-12	3.41E-15	0.966581	5.48E-07
11	4.55E-07	0.999977	1.04E-10	5.52E-14	0.748572	0.997839	5.49E-06	1.77E-12	0.999997	0.0622781	0.119313	0.004657
12	3.45E-05	1.95E-14	1.38E-07	0.999991	0.000534	1.57E-14	4.42E-15	0.856928	2.28E-12	1	0.007418	0.223541
13	0.994893	9.91E-10	1	3.26E-16	8.68E-06	2.03E-06	8.09E-16	0.999677	0.999994	1	0.001277	0.056083
14	8.36E-11	1	2.16E-11	0.999975	1.29E-06	0.999704	7.55E-14	1.85E-10	6.48E-11	0.999999	0.001192	6.08E-08
15	0.999464	0.999964	0.17106	0.213922	9.80E-08	2.26E-16	0.537378	3.03E-13	0.999793	8.81E-10	0.000393	5.02E-09
16	8.20E-06	1.89E-12	1	5.11E-14	1.20E-06	0.026365	0.999988	0.85713	1.55E-09	2.25E-15	0.175147	1.97E-06
17	6.89E-08	5.28E-10	1.97E-15	0.999991	0.118283	0.994722	0.999965	0.997755	0.982205	6.69E-16	0.632919	0.023741
18	0.621699	1	1	5.05E-16	0.993417	4.12E-15	0.995724	1.70E-09	1.11E-06	3.22E-13	0.60844	0.026152
19	0.007744	0.999997	9.47E-12	0.999929	0.061301	0.849981	5.21E-08	1.22E-11	0.006042	0.915432	0.774453	5.01E-05
20	1.71E-09	4.86E-11	0.000445	0.998672	3.68E-05	1.02E-05	1.95E-11	0.585034	0.763962	1	0.663805	1.27E-06
21	0.999973	1.18E-13	0.99971	3.43E-15	5.33E-06	5.31E-14	7.67E-08	0.997528	1.63E-06	1	0.311715	5.23E-08
22	8.79E-12	0.999978	0.004542	0.999991	9.84E-08	0.990295	0.992746	5.54E-09	0.996798	1	0.042266	8.70E-09
23	0.999493	1	3.52E-09	1.05E-12	1.01E-06	6.95E-10	0.995755	3.16E-12	4.74E-08	0.970736	0.002689	2.16E-05
24	3.72E-05	0.977592	0.999998	3.36E-07	0.002743	4.58E-13	8.93E-07	0.115062	0.995581	4.81E-13	7.69E-05	0.962292
25	1.56E-08	9.70E-13	0.001435	0.999892	0.998109	0.996815	6.13E-13	0.999339	2.82E-08	1.93E-15	7.60E-07	0.948256
26	0.999918	7.46E-12	1.25E-11	2.60E-14	0.99204	2.09E-10	6.56E-14	3.25E-06	0.949045	4.71E-15	4.14E-09	1.62E-09
27	1.73E-11	0.999946	1	0.936388	0.000330	8.84E-14	1.60E-09	5.94E-13	5.55E-07	1.65E-13	3.50E-11	1.89E-12
28	0.999937	1	4.46E-14	0.999971	8.64E-09	0.99933	0.998715	5.80E-08	0.453778	2.83E-12	8.04E-14	7.56E-07
29	4.53E-09	4.39E-05	1	1.75E-15	9.25E-08	2.12E-10	0.999987	0.999055	0.000152	7.84E-12	8.36E-14	0.999899
30	0.001101	2.18E-12	3.98E-09	0.999928	0.000641	4.58E-15	0.999986	0.958093	0.002977	3.31E-07	3.63E-13	0.999439
31	0.786122	0.8206	5.00E-05	0.796952	0.418373	0.999689	0.999174	7.43E-11	0.020662	0.999825	1.30E-12	9.52E-11
32	1.20E-07	0.999999	1	7.94E-15	0.331436	7.59E-09	3.25E-09	1.43E-11	9.00E-06	1	1.79E-11	1.75E-13
Medelv.	0.358222	0.468018	0.418890	0.476474	0.149797	0.311244	0.4286043	0.323075	0.380576	0.404634	0.293857	0.242216
Rätt	12	15	13	15	4	10	15	11	12	13	10	8
%	37.5	46.875	40.625	46.875	12.5	31.25	46.875	34.375	37.5	40.625	31.25	25

Fall	*A#4*	*A4*	*B4*	*C#4*	*C4*	*D#4*	*D4*	*E4*	*F#4*	*F4*	*G#4*	*G4*
1	4.09E-08	3.62E-05	0.999999	2.12E-16	0.072891	0.0402907	0.507793	7.40E-08	0.022258	2.86E-14	1.86E-13	0.867475
2	0.999918	5.30E-13	0.996021	0.999984	8.44E-06	0.00126679	6.61E-09	0.968369	2.16E-07	1.76E-16	7.09E-11	0.998969
3	2.06E-12	0.175127	1.17E-12	1.81E-09	4.26E-07	1.28E-14	5.12E-13	6.21E-05	0.987159	9.85E-16	7.59E-07	1.51E-08
4	0.999984	1	1	2.60E-06	1.98E-05	0.0773748	1.01E-14	1.88E-08	4.31E-11	2.45E-05	0.130196	1.73E-14
5	2.48E-10	6.53E-05	0.246453	0.999966	7.49E-05	0.999662	2.06E-14	0.008100	0.999972	1	0.985411	9.81E-07
6	0.361802	3.50E-09	1.82E-13	5.56E-15	6.77E-05	1.35E-17	1.76E-10	4.99E-05	9.99E-12	1	0.998749	0.997461
7	0.000717	0.999949	1	0.999997	5.75E-05	0.000280676	0.699607	2.23E-10	0.999995	1	0.999555	0.725558
8	0.759771	2.57E-13	2.92E-09	6.98E-17	0.0027106	0.998624	0.999936	0.001370	1.39E-12	2.00E-11	0.999917	9.02E-10
9	1.46E-09	3.83E-10	9.76E-08	0.999993	0.0001884	1.56E-15	0.999992	0.999485	0.999996	1.27E-16	0.999834	1.69E-10
10	0.997999	1	1	0.29828	0.0422539	4.13E-07	0.999982	3.78E-10	2.93E-13	5.62E-16	0.978254	2.57E-07
11	2.26E-07	0.999985	2.86E-11	7.00E-15	0.776806	0.998602	1.56E-06	4.14E-13	0.999998	0.058516	0.132103	0.004111
12	1.83E-05	4.77E-15	5.48E-08	0.999998	0.0003382	3.35E-15	6.50E-16	0.87636	6.68E-13	1	0.006651	0.248402
13	0.996708	3.52E-10	1	3.53E-17	4.31E-06	1.13E-06	1.23E-16	0.99975	0.999996	1	0.000985	0.058633
14	2.87E-11	1	5.68E-12	0.999994	5.38E-07	0.99981	1.14E-14	5.24E-11	2.13E-11	0.999999	0.000915	2.44E-08
15	0.999642	0.999977	0.182771	0.199195	3.26E-08	4.79E-17	0.572193	7.01E-14	0.999858	2.31E-10	0.000274	1.70E-09
16	4.73E-06	4.70E-13	1	6.52E-15	4.80E-07	0.0270889	0.999994	0.876898	5.79E-10	3.80E-16	0.196923	1.01E-06
17	2.85E-08	1.78E-10	4.40E-16	0.999998	0.112804	0.996541	0.999983	0.998165	0.986522	1.10E-16	0.693672	0.023119
18	0.658196	1	1	5.47E-17	0.995244	8.67E-16	0.997352	5.34E-10	5.71E-07	5.79E-14	0.665578	0.026345
19	0.007559	0.999998	2.36E-12	0.999982	0.0543391	0.881046	1.07E-08	3.07E-12	0.004835	0.933282	0.821565	3.30E-05
20	5.89E-10	1.30E-11	0.000309	0.999506	2.01E-05	6.44E-06	3.35E-12	0.616402	0.79588	1	0.719606	6.38E-07
21	0.999983	3.11E-14	0.999838	4.11E-16	2.53E-06	1.12E-14	1.96E-08	0.998007	7.66E-07	1	0.350764	2.06E-08
22	3.07E-12	0.999987	0.003684	0.999998	3.26E-08	0.993215	0.995406	1.86E-09	0.997716	1	0.043027	3.05E-09
23	0.999654	1	1.12E-09	1.26E-13	3.94E-07	2.35E-10	0.997372	7.62E-13	1.92E-08	0.980596	0.002212	1.35E-05
24	2.60E-05	0.982619	0.999999	1.08E-07	0.0019326	1.01E-13	2.57E-07	0.12005	0.996831	9.07E-14	4.69E-05	0.972111
25	5.49E-09	2.43E-13	0.001052	0.999968	0.998704	0.997844	9.16E-14	0.999465	1.18E-08	3.19E-16	3.16E-07	0.961687
26	0.999947	2.12E-12	3.18E-12	3.03E-15	0.994199	6.55E-11	9.72E-15	1.64E-06	0.959851	7.68E-16	1.20E-09	5.37E-10
27	5.95E-12	0.999968	1	0.964372	0.0002036	1.81E-14	3.24E-10	1.41E-13	2.73E-07	2.88E-14	7.85E-12	4.36E-13
28	0.999958	1	1.01E-14	0.999993	2.60E-09	0.999559	0.999319	2.37E-08	0.480429	5.52E-13	1.48E-14	3.73E-07
29	1.81E-09	2.31E-05	1	2.02E-16	3.22E-08	6.67E-11	0.999994	0.999246	0.000105	1.64E-12	1.49E-14	0.999931
30	0.000761	5.77E-13	1.29E-09	0.999981	0.0004161	8.83E-16	0.999993	0.963543	0.002287	1.27E-07	6.59E-14	0.999639
31	0.829945	0.842738	2.91E-05	0.840568	0.436666	0.999795	0.99953	1.96E-11	0.019073	0.999892	2.45E-13	2.80E-11
32	5.10E-08	0.999999	1	9.49E-16	0.32775	2.91E-09	6.20E-10	3.66E-12	4.87E-06	1	3.70E-12	3.79E-14
Medelv.	0.362893	0.468764	0.419692	0.478180	0.1505532	0.3128439	0.430263996	0.325791395	0.382899065	0.405384677	0.303945055	0.246359146
Rätt	12	15	13	15	4	10	15	11	12	13	10	8
%	37.5	46.875	40.625	46.875	12.5	31.25	46.875	34.375	37.5	40.625	31.25	25

Fall	*G4*
1	0.887802
2	1.005143
3	0.034175
4	0.035148
5	0.139851
6	1.008482
7	0.732147
8	0.090149
9	0.301534
10	0.107102
11	0.224261
12	0.260468
13	0.070817
14	0.136160
15	0.097065
16	0.105636
17	0.303373
18	0.048460
19	0.040076
20	0.024852
21	0.218468
22	0.224696
23	0.043835
24	0.982971
25	0.971008
26	0.066624
27	0.103853
28	0.431247
29	1.005932
30	1.005942
31	0.025917
32	0.056114

5 DISKUSSION

5.1 Tolkning av resultat

5.1.1 Nätverk 1-3

Nätverk 1-3 gav mycket dåliga resultat. De var slumpartade och det var omöjligt att dra några andra slutsatser än att ingen av nätverken hade möjlighet att särskilja mellan de olika 87 och 2 tonerna resp. Eftersom att det inte fungerade oberoende av antalet utdatanoder verkar det sannolikt att man behöver ett mycket större antal dolda noder för att det ska vara möjligt att känna igen toner. Då det verkade mer gynnsamt att pröva flera lager gjorde vi det och anpassade koden eftersom att det var osäkert om det överhuvudtaget gick och hur många noder man skulle behöva om man endast hade ett dolt lager.

5.1.2 Nätverk 4

1. Sammanställning av mätdata för körning av nätverk 4.
2. Sammanställning av mätdata för körning av nätverk 4 tränat en gång med $\alpha = 0.2$.

	1		**2**	
Ton	*A4*	*C4*	*A4*	*C4*
Medelv.	0.528261	0.446161	0.691664	0.994071
Rätt	32	0	23	32
%	100	0	71,875	100

Inlärning
Den första grafen (2 utdatanoder) visar skillnaden mellan rätt svar och utdatan från noden (1-utdata) under träningen. Man kan tydligt se att felet minskar drastiskt i början och sedan planar ut och stabiliserar sig. Detta är det den bör göra då felet bör vara stort i början och sedan minska när den lär sig. Att den stabiliserat sig innebär med stor sannolikhet att den har hamnat i ett lokalt minimum som mycket väl kan vara det globala.

Testresultat
Om man jämför före och efter ser man att om vi räknar alla svar över 0.5 som rätt gav noden för A4 alltid rätt svar från början och C4 alltid fel. Trots det att A4 alltid gav rätt svar var medelvärdet nästan samma för dem båda (skiljer sig 0.0821 mellan dem). Om man sedan jämför med efter träning kan man se att antalet rätt för A4 har minskat men säkerheten hos svaren är mycket högre och att medelvärdet har ökat vilket tyder på att nätverket har lärt sig för den tonen. Om man sedan kollar på C4 ser man att den har lärt sig något otroligt med 100% korrekta svar med en mycket hög säkerhet. Dessutom ger C4 resultat som ligger nära 0 när A4 ger resultat som ligger nära 1 och tvärtom. Detta tyder på att nätverket är tränat. Frågan är bara vad det är tränat för det är väldigt intressant att summan av utdatan från de två noderna alltid blir ≈ 1. Det kan tydas som att nätverket har tränats att svara 1 om den andra noden svarar 0. Men eftersom att det gällde både när nätverket var tränat och otränat kan detta inte användas för att säga att det stämmer. Det är fullt möjligt att nätverket kan särskilja de två tonerna A4 och C4 men för att vara helt säker måste fler tester genomföras.

5.1.3 Nätverk 5

1. Sammanställning av mätdata för körning av nätverk 5 tränat med $\alpha = 0.2$.
2. Sammanställning av mätdata för körning av nätverk 5 tränat ytterligare med $\alpha = 0.1$.

	1											
Ton	*A♯4*	*A4*	*B4*	*C♯4*	*C4*	*D♯4*	*D4*	*E4*	*F♯4*	*F4*	*G♯4*	*G4*
Medelv.	0.358222	0.468018	0.418890	0.476474	0.149797	0.311244	0.428604	0.323075	0.380576	0.404634	0.293857	0.242216
Rätt.	12	15	13	15	4	10	15	11	12	13	10	8
%.	37.5	46.875	40.625	46.875	12.5	31.25	46.875	34.375	37.5	40.625	31.25	25

	2											
Ton	*A♯4*	*A4*	*B4*	*C♯4*	*C4*	*D♯4*	*D4*	*E4*	*F♯4*	*F4*	*G♯4*	*G4*
Medelv.	0.362893	0.468764	0.419692	0.478180	0.150553	0.312844	0.430264	0.325791	0.382899	0.405385	0.303945	0.246359
Rätt	12	15	13	15	4	10	15	11	12	13	10	8
%	37.5	46.875	40.625	46.875	12.5	31.25	46.875	34.375	37.5	40.625	31.25	25

Testresultat
Efter en träning kan man se att nätverket inte svarar över 50% rätt för någon av tonerna. Man kan också se att medelvärdet av resultaten för varje testfall av respektive ton ligger mycket lågt, i spannet 0.15-0.48. Dessa två faktorer tillsammans innebär att nätverket efter en träningssession inte är kapabelt att särskilja på 12 toner trots att när det svarar rätt är svaret mycket pålitligt, exempelvis ger B4 1 på ett antal toner. Om man jämför nätverket efter en träningssession och efter två träningsessioner med avseende på medelvärdet av alla testresultat för t.ex. A#4 kan man se att det har skett en ökning av medelvärdet även om den procentuellt sätt träffar rätt på exakt lika många toner. Som man kan se i tabellen som är summan av alla resultat när man testade G4 gick summan från Nätverk 5 bara mot 1 när G4 ger rätt svar till skillnad från Nätverk 4 där summan av alla resultat alltid går mot 1. Detta implicerar att Nätverk 4 kan se skillnad på A4 och C4 eftersom att båda dessa nätverk tränas efter samma princip, dock förklarar det inte varför C4 ger ≈ 1 när A4 ger ≈ 0 om man spelar en A4 ton. Mer studier och tester måste fortfarande göras.

5.2 Metodanalys

De tester som gjorts på de neurala nätverken verkar ha fungerat som de ska då det för säkerhets skulle skrivits ut vilken ton som testats och resultaten skiljer sig mycket vilket är starka bevis för att det gått igenom olika toner och olika sekvenser. Testlistorna består alla av ett antal namn på toner och därefter går nätverket igenom var och en av filerna i motsvarande mapp. Den reservation som finns är att det skulle finnas något fel i koden som missats men det verkar mycket otroligt då koden har bearbetats mycket.

Ett problem med de tester som gjordes är dock att det är en relativt liten mängd data som testas (32 sekvenser av varje ton). Om man skulle öka antalet inspelade sekvenser kanske man skulle få fram resultat som var mer säkra eftersom att det blir minskar risken av att resultaten påverkas av en ton som var onormal. T.ex. är det fullt tänkbart att orsaken till att A4 i Nätverk 4 gav färre träffar var att de inspelade tonernas kvalité var dålig på grund av något som inte märktes av oss när vi spelade in.
Ett problem som dock finns är att bland resulaten räknas det som att nätverket svarat rätt när resultatet för noden som var förväntad att ge det korrekta svaret kommer fram till ett resultat som ligger över 0.5. Detta innebär att det inte utesluter att det skulle kunna finnas ett resultat från en annan nod som var större än den noden som förväntades ha rätt.

När det kommer till inlärningen skrevs datan ut var 1000:e gång, oberoende av vilken fil som testades. Den gav också endast ut skillnaden mellan önskat svar (1) och resultatet för den noden som förväntades ha rätt så eventuellt kan grafen vara något missvisande. Grafen visar inte hur den totala inlärningen gick utan använde endast de olika nodernas inlärningsprocess och använde dessa för att försöka skapa en överblick av inlärningens effektivitet.

5.3 Bedömning av metodens giltighet

Det finns ett fåtal problem som kan uppkomma i projektet på grund av hur vi handskas med ljud. Men normaliseringen födde nya problem. Ett problem var exempelvis att vi inte visste hur vi skulle normalisera all data. Till en början ville vi normalisera över hela den inspelade datan. Att normalisera med denna metod visade sig vara helt onödigt då vi senare delar upp filen i en massa små filer som var för sig inte var normaliserade. Enbart den absolut högsta toppen i hela filen fick värdet ett. Alla andra toppar kommer ha betydligt lägre värden. Men om vi däremot bestämmer oss för att normalisera över de mindre intervallen vi skär ut så kommer vi få åtminstone få att alla segment vi jobbar med är normaliserade, alltså att alla värden ligger mellan noll och ett. Problemet vi fann med denna metod är att den högsta punkten i intervallet inte nödvändigtvis är en topp på en våg. Detta kan möjligtvis komma att försvåra saker för nätverket. Ett annat problem vi kom att tänka på är att var att de inspelade tonerna kommer innefatta svävning medan de genererade inte innefattar det. Vi är osäkra huruvida nätverket påverkades av det.

En annan felkälla som finns är att NN ibland ger ut olika resultat för samma testfall. Resultaten kan skilja sig och under (10^{-3}). Denna skillnad kan ha påverkat nätverkets inlärning om den ger skillnad för samma

testfall men eftersom att samma testfall dyker upp flera gånger och att inlärningsfaktorn inte alltid är särskilt stor innebär det att NN har potentialen att rätta de felen som skulle kunna tänkas uppstå. Orsaken bakom felet verkar kunna vara avrundningsfel vid inläsningen från filerna men det förklarar inte varför den avrundar olika från en gång till en annan. Orsaken till att det beter sig oberäkneligt skulle också kunna vara att ett antal av vikterna för nätverken ligger mellan -1 och 1 samtidigt som all data som vi tillhandhåller nätverket ligger inom samma område. Detta kan resultera i att deras produkter ligger väldigt nära noll vilket riskerar att det blir väldigt dålig precision.

5.4 Förslag på metodförbättringar

En sak som skulle kunna förbättra nätverkens inlärning skulle vara att istället för att slumpa en träningslista, slumpa 10^5 olika testfall och bara undersöka hur väl nätverket svarar när man ger den en lista med data som endast används för att testa nätverken med.

5.5 Reflektion över arbetet

Neurala nätverk är ett väldigt hett forskningsområde och alla sätt som de applicerats bidrar till att utveckla nya metoder och användningsområden. Att kombinera NN med ljud kan mycket väl vara ett ovanligt område men som definitivt bidrar med grundforskning. Genom att visa att identifieringen av ljudkurvor, som in princip är funktioner, är ett abstrakt problem som kräver ett komplext nätverk kan bidra till att man får en annorlunda syn på vad som krävs av ett nätverk som ska lösa en uppgift som ter sig vara mycket mer abstrakt. Men det kan också ge en ny inblick i vad som krävs av oss som konstruktör för att identifiera vad som är ett abstrakt problem eller inte.

5.6 Framtid

Något som vore väldigt intressant vore att undersöka hur nätverket reagerar om man ger den toner som den inte alls tränats för att känna igen. Det skulle vara ett intressant försök att se dels hur väl tränat nätverket är och om det går att urskilja likheter mellan tonerna om det ger utslag. Det vore också mycket intressant att försöka använda en rekursiv struktur för nätverket, så kallade rekursiva nätverk, för att se om det blir lättare eller svårare för nätverket att fullföra sina uppgifter. Arbetet öppnar också upp en ny vinkel att analysera nätverken, tränar vi dem att göra det som vi tror? Finns det flera nätverk som kan identifiera toner trots att den är tränad för något helt annat? Möjligheterna som står framför att utveckla och bidra till nya områden är lika många som antalet uppgifter som vi potentiellt kan applicera dessa nya neurala nätverk på.

Referenser

[1] Yoshua Bengio och Yann LeCun. "Scaling Learning Algorithms towards AI". I: *Large-Scale Kernel Machines* (aug. 2007), s. 3,25.

[2] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[3] Jonathan Rogness. *Trigonometry in Nature*. Email: rogness@math.umn.edu. URL: http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.

[4] R. Rojas. *Threshold Logic*. Okt. 1996. URL: http://cgm.cs.mcgill.ca/~godfried/teaching/dm-reading-assignments/Threshold-Logic.pdf.

[5] Daniel Shiffman. *The nature of code*. 2012. Kap. 10.

[6] Wikipedia. *Artificial neural network*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.

[7] Wikipedia. *Backpropagation*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Backpropagation.

[8] Wikipedia. *Deep Learning*. Jan. 2016. URL: https://en.wikipedia.org/wiki/Deep_learning.

[9] Wikipedia. *Gradient descent*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Gradient_descent.

[10] Wikipedia. *Perceptron*. Okt. 2015. URL: https://en.wikipedia.org/wiki/Perceptron.

[11] Wikipedia. *Svävning*. [Online; hämtad 29-januari-2016]. 2015. URL: //sv.wikipedia.org/w/index.php?title=Sv%C3%A4vning&oldid=32054228.

[12] Douglas Y'barbo. *Stack Exchange Forum, How to choose the number of hidden layers and nodes in a feedforward neural network*. Not: baserad på Bilaga C. Aug. 2010. URL: http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.

A Backpropagation härledning

Låt w_{ij} vara alla vikter mellan två noder i och j.
Låt y vara utdatan.
Låt t vara önskad utdata.
Låt o_i vara utdatan från noden i.
Låt σ vara en sigmoid funktion.
Låt errorfunktionen vara:

$$E(w_{ij}) = (\frac{1}{2})(y-t)^2$$

Enligt gradientsökning så kommer:

$$E(w_{ij}) \geq E(w_{ij} - \alpha \nabla E(w_{ij}))$$

Så för att minska felet så behöver vi få ut $\nabla E(w_{ij})$.
Vilket bara är de partiella derivatorna $\frac{\partial E}{\partial w_{ij}}$.
Vilket är, enligt kedjeregeln:

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

då $o_j = \sigma(net_j)$,
och $net_j = \sum_{k=1}^{n}(w_{kj}o_k)$ utom för indata där net_j är givet.
Vi beräknar de tre derivatorna var för sig.
Vi börjar med den längst bak:

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial \sum_{k=1}^{n} w_{kj}o_k}{\partial w_{ij}} = o_i$$

då allt utom $w_{ij}o_i$ (d.v.s. när k=i) är konstanter och försvinner.

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = \sigma'(net_j)$$

$\frac{\partial E}{\partial o_j}$ behöver delas upp i två fall, en då j är utdata och en då det är i vårat gömda lager.
Låt oss börja med utdatanoden.
Notera att utdatan för ett utdatalager är y.

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial (\frac{1}{2})(y-t)^2}{\partial y} = y - t$$

Nu, om den är i det gömda lagret istället.
Notera att E är en funktion av indatan till alla neuroner som får indata av j.
Låt L vara mängden av alla dessa tal

$$\frac{\partial E}{\partial o_j} = \frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(net)}{\partial o_j}$$

Notera att vi nu kan derivera totalt och med hjälp av kedjeregeln få:

$$\frac{\partial E}{\partial o_j} = \sum_{l \in L}(\frac{\partial E}{\partial net_l}\frac{\partial net_l}{\partial o_j}) = \sum_{l \in L}(\frac{\partial E}{\partial net_l}\frac{\partial \sum_{k=1}^{n} w_{kl}o_k}{\partial o_j}) = \sum_{l \in L}(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l}w_{jl})$$

notera att $\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l}$ redan är uträknad.
då har vi

$$\frac{\partial E}{\partial w_{ij}}$$

till

$$(o_j - t_j)\sigma'(net_j)o_i$$

om det är utdata och

$$\frac{\partial E}{\partial w_{ij}}$$

till

$$(\sum_{l \in L}\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l}w_{jl})\sigma'(net_j)o_i$$

om det är i det gömda lagret.

B Pseudokod för körning

w_{pq} är en vikt mellan två noder p och q
i är en godtycklig nod i indatalagret I
d är en godtycklig nod i dolda lagret D
u är en godtycklig nod i utdatalagret U
b_n är biasnoden för lager N o_n är utdata för nod n i lager N
σ är sigmoid funktionen $\frac{1}{1+e^{-z}}$
Alla indatanoder får var sitt decimaltal som den kommer ha som output

for $d \in D$ **do**
    $resultat \leftarrow 0$
    for $i \in I$ **do**
        $resultat \leftarrow resultat + o_i \cdot w_{id}$ ▷ Viktade summan av inputs
    end for
    $o_d \leftarrow \sigma(resultat + w_{b_i d})$ ▷ Lägg till tröskel eller bias, spara utdata
end for

for $u \in U$ **do**
    $resultat \leftarrow 0$
    for $d \in D$ **do**
        $resultat \leftarrow resultat + o_d \cdot w_{du}$ ▷ Viktade summan av inputs
    end for
    $o_u \leftarrow \sigma(resultat + w_{b_d u})$ ▷ Lägg till tröskel eller bias, spara utdata
end for

C Komplexitet för körning

Tillåt oss inspektera vad som händer under forward propagation:
Vi ignorerar alla konstanter

1: **for** $d \in D$ **do**
2: $\quad resultat \leftarrow 0$
3: $\quad$ **for** $i \in I$ **do**
4: $\quad\quad resultat \leftarrow resultat + o_i \cdot w_{id}$ ▷ $D \cdot I$ beräkningar
5: $\quad$ **end for**
6: $\quad o_d \leftarrow \sigma(resultat + w_{b_i d})$ ▷ D beräkningar
7: **end for**
8:
9: **for** $u \in U$ **do**
10: $\quad resultat \leftarrow 0$
11: $\quad$ **for** $d \in D$ **do**
12: $\quad\quad resultat \leftarrow resultat + o_d \cdot w_{du}$ ▷ $U \cdot D$ beräkningar
13: $\quad$ **end for**
14: $\quad o_u \leftarrow \sigma(resultat + w_{b_d u})$ ▷ U beräkningar
15: **end for**

$D \cdot I + D + D \cdot U + U$ enl def för tidskomplexitet $\frac{I+D\cdot I+D+D\cdot U+U}{D\cdot U+D\cdot I} <$ någon konstant för $L, I, J > 1$
dvs programmet är i komplexitet $O(D(U + I))$

D Pseudokod BP

w_{pq} är en vikt mellan två noder p och q
i är en godtycklig nod i indatalagret I
d är en godtycklig nod i dolda lagret D
u är en godtycklig nod i utdatalagret U
o_n är utdata för nod n i lager N
t_u är det korrekta svaret för nod u
B_n är en lista för förändringen av en nod n i lager N
α är inlärningskoefficienten

1: $B \leftarrow empty$
2:
3: **for** $u \in U$ **do**
4: $\quad B_u \leftarrow (o_u - t_u)(1 - o_u)o_u$ ▷ Beräkna för utdatanoderna
5: **end for**
6:
7: **for** $d \in D$ **do**
8: $\quad sum \leftarrow 0$
9: $\quad$ **for** $u \in U$ **do**
10: $\quad\quad sum \leftarrow sum + w_{du}(o_u - t_u)o_u(1 - o_u)$ ▷ Med avseende på utdata
11: $\quad$ **end for**
12: $\quad B_d \leftarrow sum * (1 - o_d)o_d$ ▷ Beräkna för dolda noderna
13: **end for**
14:
15: **for** $d \in D$ **do**
16: $\quad$ **for** $i \in I$ **do**
17: $\quad\quad w_{id} \leftarrow w_{id} - \alpha o_i B_d$ ▷ Uppdatera vikterna mellan $i \in I$ och $d \in D$
18: $\quad$ **end for**
19: **end for**
20:
21: **for** $u \in U$ **do**
22: $\quad w_{du} \leftarrow w_{du} - \alpha o_d B_u$ ▷ Uppdatera vikterna mellan $d \in D$ och $u \in U$
23: **end for**

E Komplexitet för BP

w_{pq} är en vikt mellan två noder p och q
i är en godtycklig nod i indatalagret I
d är en godtycklig nod i dolda lagret D
u är en godtycklig nod i utdatalagret U
o_n är utdata för nod n i lager N
t_u är det korrekta svaret för nod u
B_n är en lista för förändringen av en nod n i lager N
α är inlärningskoefficienten

b_n är biasnoden för lager N
σ är sigmoid funktionen $\frac{1}{1+e^{-z}}$
Alla indatanoder får var sitt decimaltal som den kommer ha som output

Nätverket består av I indata noder, D hidden layer noder, U utdata noder.

Tillåt oss inspektera vad som händer under back propagation:
Vi ignorerar alla konstanter

1: $B \leftarrow empty$
2:
3: **for** $u \in U$ **do**
4: $\quad B_u \leftarrow (o_u - t_u)(1 - o_u)o_u$ ▷ U beräkningar
5: **end for**
6:
7: **for** $d \in D$ **do**
8: $\quad sum \leftarrow 0$
9: $\quad$ **for** $u \in U$ **do**
10: $\quad\quad sum \leftarrow sum + w_{du}(o_u - t_u)o_u(1 - o_u)$ ▷ $D \cdot U$ beräkningar
11: $\quad$ **end for**
12: $\quad B_d \leftarrow sum * (1 - o_d)o_d$ ▷ D beräkningar
13: **end for**
14:
15: **for** $d \in D$ **do**
16: $\quad$ **for** $i \in I$ **do**
17: $\quad\quad w_{id} \leftarrow w_{id} - \alpha o_i B_d$ ▷ $D \cdot I$ beräkningar
18: $\quad$ **end for**
19: **end for**
20:
21: **for** $u \in U$ **do**
22: $\quad w_{du} \leftarrow w_{du} - \alpha o_d B_u$ ▷ U beräkningar
23: **end for**

$U + D \cdot U + D + D \cdot I + U$ enl def för tidskomplexitet $\frac{U+D\cdot U+D+D\cdot I+U}{D\cdot U+D\cdot I} <$ någon konstant för $L, I, J > 1$
dvs programmet är i komplexitet $O(D(U + I))$

F Neural Nets FAQ part 3, utdrag

Subject: How many hidden units should I use?
==========

The best number of hidden units depends in a complex way on:
o the numbers of input and output units
o the number of training cases
o the amount of noise in the targets
o the complexity of the function or classification to be learned
o the architecture
o the type of hidden unit activation function
o the training algorithm
o regularization

In most situations, there is no way to determine the best number of hidden units without training several networks and estimating the generalization error of each. If you have too few hidden units, you will get high training error and high generalization error due to underfitting and high statistical bias. If you have too many hidden units, you may get low training error but still have high generalization error due to overfitting and high variance. Geman, Bienenstock, and Doursat (1992) discuss how the number of hidden units affects the bias/variance trade-off.
Some books and articles offer "rules of thumbfor choosing an architecture; for example:
o "A rule of thumb is for the size of this [hidden] layer to be somewhere between the input layer size ... and the output layer size ..."(Blum, 1992, p. 60).
o "To calculate the number of hidden nodes we use a general rule of: (Number of inputs + outputs) * (2/3)"(from the FAQ for a commercial neural network software company).
o "you will never require more than twice the number of hidden units as you have inputs"in an MLP with one hidden layer (Swingler, 1996, p. 53). See the section in Part 4 of the FAQ on The Worst books for the source of this myth.)
o "How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer."(Berry and Linoff, 1997, p. 323).
o "Typically, we specify as many hidden nodes as dimensions [principal components] needed to capture 70-90% of the variance of the input data set."(Boger and Guterman, 1997)

These rules of thumb are nonsense because they ignore the number of training cases, the amount of noise in the targets, and the complexity of the function. Even if you restrict consideration to minimizing training error on data with lots of training cases and no noise, it is easy to construct counterexamples that disprove these rules of thumb. For example:
o There are 100 Boolean inputs and 100 Boolean targets. Each target is a conjunction of some subset of inputs. No hidden units are needed.

o There are two continuous inputs X and Y which take values uniformly distributed on a square [0,8] by [0,8]. Think of the input space as a chessboard, and number the squares 1 to 64. The categorical target variable C is the square number, so there are 64 output units. For example, you could generate the data as follows (this is the SAS programming language, but it should be easy to translate into any other language): \- **EDITERAD, KOD BORTTAGEN**
No hidden units are needed.

o The classic two-spirals problem has two continuous inputs and a Boolean classification target. The data can be generated as follows: \- **EDITERAD, KOD BORTTAGEN**
With one hidden layer, about 50 tanh hidden units are needed. Many NN programs may actually need closer to 100 hidden units to get zero training error.

o There is one continuous input X that takes values on [0,100]. There is one continuous target Y = sin(X). Getting a good approximation to Y requires about 20 to 25 tanh hidden units. Of course, 1 sine hidden unit would do the job.

Some rules of thumb relate the total number of trainable weights in the network to the number of training cases. A typical recommendation is that the number of weights should be no more than 1/30 of the number

of training cases. Such rules are only concerned with overfitting and are at best crude approximations. Also, these rules do not apply when regularization is used. It is true that without regularization, if the number of training cases is much larger (but no one knows exactly how much larger) than the number of weights, you are unlikely to get overfitting, but you may suffer from underfitting. For a noise-free quantitative target variable, twice as many training cases as weights may be more than enough to avoid overfitting. For a very noisy categorical target variable, 30 times as many training cases as weights may not be enough to avoid overfitting.

An intelligent choice of the number of hidden units depends on whether you are using early stopping or some other form of regularization. If not, you must simply try many networks with different numbers of hidden units, estimate the generalization error for each one, and choose the network with the minimum estimated generalization error. For examples using statistical criteria to choose the number of hidden units, see ftp://ftp.sas.com/pub/neural/dojo/dojo.html.

Using conventional optimization algorithms (see "What are conjugate gradients, Levenberg-Marquardt, etc.?"), there is little point in trying a network with more weights than training cases, since such a large network is likely to overfit.

Using standard online backprop, however, Lawrence, Giles, and Tsoi (1996, 1997) have shown that it can be difficult to reduce training error to a level near the globally optimal value, even when using more weights than training cases. But increasing the number of weights makes it easier for standard backprop to find a good local optimum, so using "oversizenetworks can reduce both training error and generalization error.

If you are using early stopping, it is essential to use lots of hidden units to avoid bad local optima (Sarle 1995). There seems to be no upper limit on the number of hidden units, other than that imposed by computer time and memory requirements. Weigend (1994) makes this assertion, but provides only one example as evidence. Tetko, Livingstone, and Luik (1995) provide simulation studies that are more convincing. Similar results were obtained in conjunction with the simulations in Sarle (1995), but those results are not reported in the paper for lack of space. On the other hand, there seems to be no advantage to using more hidden units than you have training cases, since bad local minima do not occur with so many hidden units.

If you are using weight decay or Bayesian estimation, you can also use lots of hidden units (Neal 1996). However, it is not strictly necessary to do so, because other methods are available to avoid local minima, such as multiple random starts and simulated annealing (such methods are not safe to use with early stopping). You can use one network with lots of hidden units, or you can try different networks with different numbers of hidden units, and choose on the basis of estimated generalization error. With weight decay or MAP Bayesian estimation, it is prudent to keep the number of weights less than half the number of training cases.

Bear in mind that with two or more inputs, an MLP with one hidden layer containing only a few units can fit only a limited variety of target functions. Even simple, smooth surfaces such as a Gaussian bump in two dimensions may require 20 to 50 hidden units for a close approximation. Networks with a smaller number of hidden units often produce spurious ridges and valleys in the output surface (see Chester 1990 and "How do MLPs compare with RBFs?") Training a network with 20 hidden units will typically require anywhere from 150 to 2500 training cases if you do not use early stopping or regularization. Hence, if you have a smaller training set than that, it is usually advisable to use early stopping or regularization rather than to restrict the net to a small number of hidden units.

Ordinary RBF networks containing only a few hidden units also produce peculiar, bumpy output functions. Normalized RBF networks are better at approximating simple smooth surfaces with a small number of hidden units (see How do MLPs compare with RBFs?).

There are various theoretical results on how fast approximation error decreases as the number of hidden units increases, but the conclusions are quite sensitive to the assumptions regarding the function you are trying to approximate. See p. 178 in Ripley (1996) for a summary. According to a well-known result by Barron (1993), in a network with I inputs and H units in a single hidden layer, the root integrated squared error (RISE) will decrease at least as fast as $H^{-1/2}$ under some quite complicated smoothness assumptions. Ripley cites another paper by DeVore et al. (1989) that says if the function has only R bounded derivatives, RISE may decrease as slowly as $H^{-R/I}$. For some examples with from 1 to 4 hidden layers see How many hidden layers should I use?änd 'How do MLPs compare with RBFs?"

For learning a finite training set exactly, bounds for the number of hidden units required are provided by Elisseeff and Paugam-Moisy (1997).

References:

Barron, A.R. (1993), "Universal approximation bounds for superpositions of a sigmoid function,"IEEE Transactions on Information Theory, 39, 930-945.

Berry, M.J.A., and Linoff, G. (1997), Data Mining Techniques, NY: John Wiley and Sons.

Blum, A. (1992), Neural Networks in C++, NY: Wiley.

Boger, Z., and Guterman, H. (1997), "Knowledge extraction from artificial neural network models,"IEEE Systems, Man, and Cybernetics Conference, Orlando, FL.

Chester, D.L. (1990), "Why Two Hidden Layers are Better than One,"IJCNN-90-WASH-DC, Lawrence Erlbaum, 1990, volume 1, 265-268.

DeVore, R.A., Howard, R., and Micchelli, C.A. (1989), "Optimal nonlinear approximation,Manuscripta Mathematica, 63, 469-478.

Elisseeff, A., and Paugam-Moisy, H. (1997), "Size of multilayer networks for exact learning: analytic approach,"in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) Advances in Neural Information Processing Systems 9, Cambrideg, MA: The MIT Press, pp.162-168.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", Neural Computation, 4, 1-58.

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1996), "What size neural network gives optimal generalization? Convergence properties of backpropagation,Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, http://www.neci.nj.nec.com/homepages/lawrence/papers/minima-tr96/minima-tr96.html

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1997), "Lessons in Neural Network Training: Overfitting May be Harder than Expected,Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97, AAAI Press, Menlo Park, California, pp. 540-545, http://www.neci.nj.nec.com/homepages/lawrence/papers/ove aaai97/overfitting-aaai97-bib.html

Neal, R. M. (1996) Bayesian Learning for Neural Networks, New York: Springer-Verlag, ISBN 0-387-94724-8.

Ripley, B.D. (1996) Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press,

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting,Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics, 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a very large compressed postscript file, 747K, 10 pages)

Swingler, K. (1996), Applying Neural Networks: A Practical Guide, London: Academic Press.

Tetko, I.V., Livingstone, D.J., and Luik, A.I. (1995), "Neural Network Studies. 1. Comparison of Overfitting and Overtraining,"J. Chem. Info. Comp. Sci., 35, 826-833.

Weigend, A. (1994), "On overfitting and the effective number of hidden units,Proceedings of the 1993 Connectionist Models Summer School, 335-342.