Project 2: Neural network and music composition

Rafaël Mindreau - 3ICT1

Academiejaar 2016 - 2017

Inhoudsopgave

1 Inleiding

Dit rapport beschrijft de bevindingen, werkwijze en uitwerking van: Rafaël Mindreau. Voor het project: Neurale netwerken en compositie van muziek.

Binnen dit rapport beschrijf ik de verschillende technieken en technologiën waarmee ik in contact ben gekomen tijdens dit project. De uiteindelijke demo kan gevonden worden op de Portfolio op GIT, onder het opvolgingsdocument. De demo is een filmpje op youtube die de demonstrator uitlegt.

2 Introductie tot het project

Het doel van dit project was om er voor te zorgen dat een Artificiële Intelligentie - onder de vorm van een neuraal netwerk - de capaciteit bezit verder te improviseren over een bepaald akkoord of noot die gespeeld werd door de gebruiker. Deze opdracht was op zich al een serieuze uitdaging, waar vele nieuwe technologiën bij aan bod kwamen.

De bedoeling was om dit uit te voeren in de vorm van een VST/Audio applicatie in het JUCE-framework in C++. Echter bleek later dat dit focus meer legde op het configuren van projecten dan op neurale netwerken. Tijdens de loop van dit project werd besloten verder te gaan met een JavaScript implementatie van het probleem.

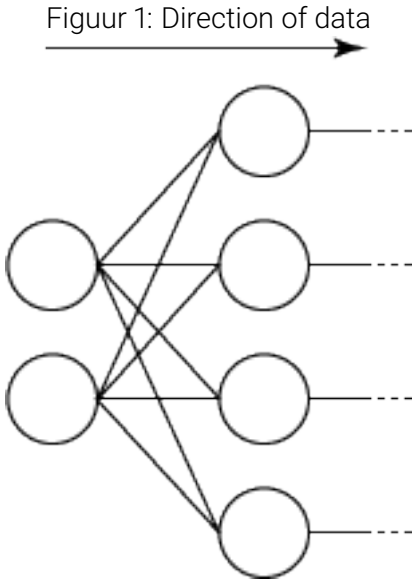
3 Neurale netwerken

Een neuraal netwerk is systeem van eenvoudige componenten die héél erg complex intern verbonden zijn, en dewelke zijn toestand afhangt van de dynamische toestanden die zich onderling tussen die componenten bevinden[1]. Het gelijkaardig aan de werking van echte neuronen, maar is in praktijk niet vergelijkbaar. Onder de motorkap blijft dit een verwerking van gegevens op binair niveau. De uitvoer van een neuron is gebasseerd op de verwerking van de invoer met een activatiefunctie. Dit maakt het toch nog heel verschillend met biologische netwerken.

3.1 Structuur van een neural netwerk

3.2 Multi-layer Perceptron MLP

De multi-layer Perceptron is het meest eenvoudige en fundamentele model van een neural netwerk. Het is een feed-forward netwerk wat betekent dat de informatie van links naar rechts gebeurt. Een neuraal netwerk beschikt over hiërarchische structuur. Verschillende lagen neuronen volgen elkaar telkens op. Dit houdt het overzichtelijk. De eerste laag is de input-laag



3.2.1 Input layer

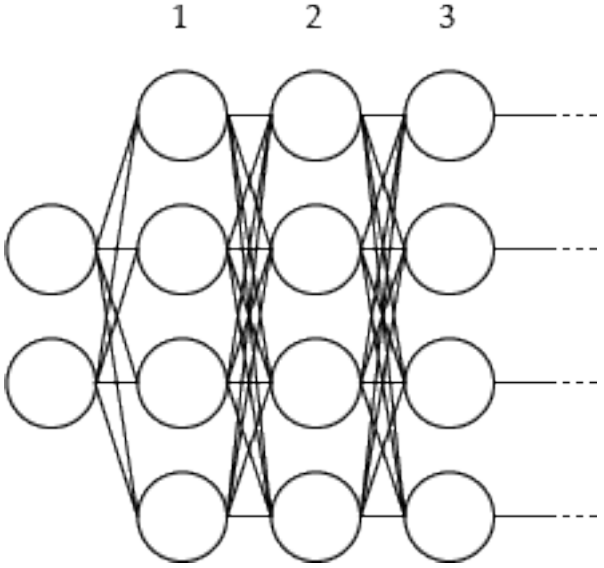
De input layer is een speciaal geval. Het gedraagt zich niet zoals de andere lagen omdat het enkel data aanlegt, en er niets mee doet behalve ze door te geven aan de volgende laag. De input layer is zoals de output layer, en hoort meestal ook het formaat te dragen waarin we data van de wereld kunnen omzetten naar data voor het netwerk.

Een simpel voorbeeld is geschriftsherkenning[2], waarin het netwerk letter per letter een handgeschreven letter omzet naar een digitale voorstelling van de letter. Om dit te bekomen worden alle pixels van een afbeelding genomen als invoer van een neural netwerk, een input layer van 4096 neuronen dus.

3.2.2 Hidden layers

Tussen input en output layers zitten de hidden layers. Deze ”verborgen“ lagen zijn eigenlijk niet echt verborgen, ondanks hun naam. Ze bieden voor de gegevens een manier om van de ene kant naar de andere kant te gaan. De nodes zijn onderling verbonden allemaal verbonden. Elke node omvat dezelfde **activatiefunctie**. Het aantal neuronen in de hidden layers, alsook het aantal hidden layers, bepalen de **architectuur** van het netwerk.

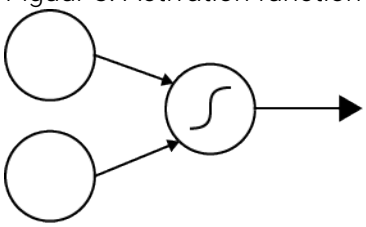
Figuur 2: Architectuur



3.2.3 Activatie-functies

Elke neuron in de hidden layer krijgt telkens de input van de vorige layer mee. Dit wilt dus zeggen elke neuron van de vorige layer. In het geval van handwriting recognition is dit dus 4096 verbindingen voor één neuron in de daarop volgende laag. Elk van deze verbindingen heeft een gewicht. Sommigen kunnen meer doorwegen dan anderen. De verhouding tussen de gewichten, en de waarde die ze mee geven aan de volgende laag, bepalen hoe het netwerk verder gaat reageren. De functie om deze verhoudingen om te zetten tot één getal dat dan doorgegeven wordt naar de volgende laag, is de **activatiefunctie**

Figuur 3: Activation function



De activatiefunctie die gebruikt wordt, is een sigmoïde functie:

$$\frac{1}{1+e^{-z}}$$

In deze functie is z de enumeratie/som van de scalaire product tussen de gewichten w_j en de invoer i_j.

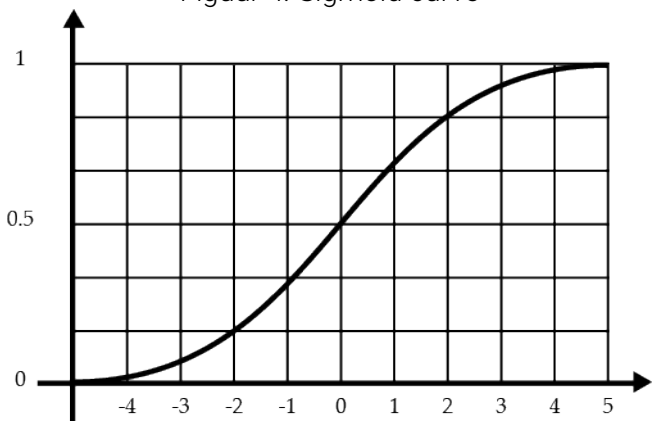
$$\sum w_j \cdot x_j - b$$

Waar b de bias voorstelt. Een grenswaarde dewelke overschreden moet worden om de neuron te activeren. Is de bias hoog, dan is de kans dat de neuron activeert hoger dan wanneer deze zéér negatief zou zijn. Gieten we deze in een functie dan de volledige versie:

$$\frac{1}{1+e^{\sum w_j \cdot x_j - b}}$$

Dit zorgt er voor dat kleine veranderingen in de invoer ook slechts kleine veranderingen worden in uitvoer tijdens de activatie van een neuron. De grafiek die deze functie voorstelt is voor die reden een afgeronde versie van een trigger-functie:

Figuur 4: Sigmoid curve



3.2.4 Training

Om het netwerk te laten doen wat we willen kunnen we moeilijk zelf al de waarden, verhoudingen en gewichten tussen de neuronen aanpassen. Hiervoor worden trainingsmechanismen gehanteerd. Een training maakt gebruik van een **Cost Function**. Deze kostfunctie gaat eenvoudig-weg na hoe goed de uitkomst bij activatie van het netwerk is. Dit door te vergelijken met een uitkomst uit de set training-data. Is de kost hoog? Dan is het netwerk slecht bezig, is deze laag? Dan scoort het netwerk goed en doet het wat het moet.

De volgende functie is een voorbeeld van een kostfunctie genaamd "Cross Entropy". Deze werd gebruikt voor de eerste trainingen binnen het netwerk.

$$C = -\frac{1}{n}\sum[y \ln a + (1-y)\ln(1-a)]$$

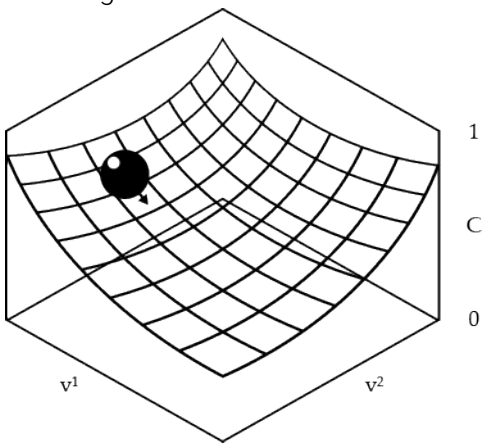
Na wat opzoekwerk blijkt Cross-Entropy niet geschikt voor regressie, maar eerder voor classificatie. MSE (mean-squared-error) is de geschiktere keuze bij problemen met regressie. Deze functie zit er als volgt uit:

$$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

Waar w en b respectievelijk de gewichten en de biases zijn van het netwerk. n het totale aantal aan training inputs. a is de vector van alle outputs in het netwerk, en x zijn dan alle inputs. Wat het belangrijkste in deze functie is het verschil tussen de output a en de verwachte ouput $y(x)$.

Nu we een functie hebben om de kost te bekomen, moeten we nog vinden welke richting het netwerk uit moet om de kost te minimaliseren. Welke gewichten moeten daarvoor omhoog of omlaag? Welke biases moeten worden aangepast en met welke waarde? Met de duizenden parameters waar men bij ons neuraal netwerk rekening mee moeten houden, is dit een moeilijke taak. Om het simpel te kunnen voorstellen gebruiken we de analogie met de bal in een vallei:

Figuur 5: Gradient descent



Hierop zijn de wetten van fysica niet belangrijk. Eerder willen we de bal naar beneden sturen tot dat het daar is, en dit zo rap mogelijk. De techniek die hiervoor gebruikt wordt, heet ”gradient descent“.

3.2.5 Gradient Descent

Jammer genoeg zijn er meer dan twee parameters in een neural netwerk waar men moet naar kijken. Er zijn vectoren van vele duizenden parameters waar rekening mee moet gehouden. Hier stopt dan ook de mooie analogie naar de bovenstaande figuur. Het principe blijft immers gelijk. We willen de kost minimaliseren.

We willen dat de verandering in kost kleiner of gelijk is aan 0 op zijn minst. $\Delta C \leq 0$. Zo gaat de kost geregeld omlaag.

$$grad(C) = \left(\frac{\Delta C}{\Delta v_1}, ..., \frac{\Delta C}{\Delta v_m} \right)$$

De bovenstaande formule verkrijgt voor elke kleine verandering in verhouding tot de verandering in kost, de gradient. Het verschil per parameter Δv is dan:

$$\Delta v = -\eta \cdot grad(C)$$

Waar η de **training rate** is. Deze mag niet te hoog zijn, want dan kan het zijn dat de ΔC positief gemaakt wordt. Dit fenomeen is waarneembaar in het demonstratie-filmpje. Deze mag ook niet te laag zijn, omdat de training dan té langzaam zal gaan.

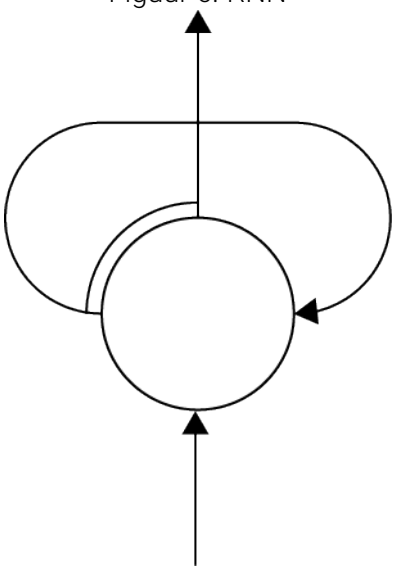
3.3 Andere architecturen

Naast de klassieke MLP die we zonet hebben besproken bestaan ook andere architecturen. De andere structuren berusten echter op het zelfde principe. De neuronen hanteren bij voorkeur de zelfde activatiefuncties, kostfuncties, en leermechanismen. Enkel is de flow vaak wel eens anders.

3.3.1 RNN

RNN staat voor Recurrent Neural Network[3] en is populair in het domein waar het het meest bruikbaar voor is: Tijd. Dit type neuraal netwerk heeft de structuur, en mogelijkheid om te werken als geheugenschakeling.

Figuur 6: RNN

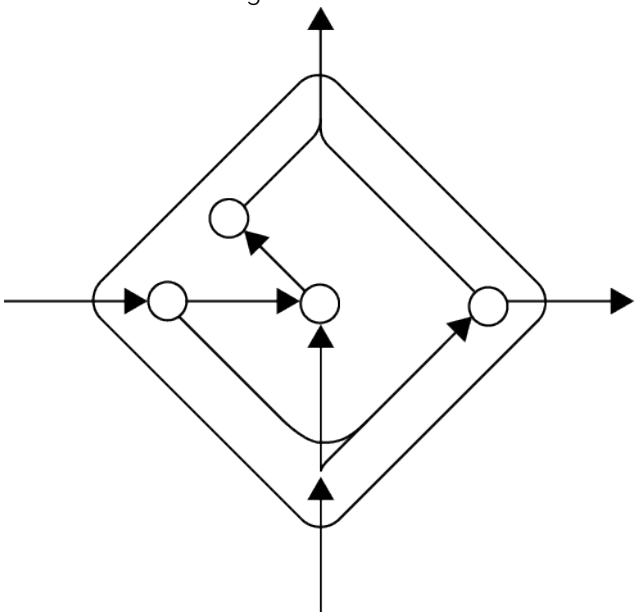


In bovenstaande figuur is zichtbaar dat de neuron zowel een input als output heeft; de pijl van beneden die naar boven loopt. De output van de neuron wordt ook gebruikt als input bij de volgende ronde. Dit impliceert dat er door de tijd heen een continuiteit zit in het netwerk. Met andere woorden kan het netwerk nu oordelen binnen de context op basis van wat vooraf ging. Dit maakt het interessant voor problemen waar het tijdsdomein van belang is.

3.3.2 LSTM

Long Short Term Memory, of kortweg LSTM[4] is een Recurrent Neural Network met langere geheugencapaciteit, waar klassieke RNN's de context over tijd verliezen. LSTM's komen in verschillende vormen en architecturen, maar ze zullen bijna altijd de mogelijkheid hebben om twee parallel lopende output-vectoren door te sluizen naar de volgende stap, waar deze op basis van de nieuwe input-vector andere beslissingen kan nemen. Één van deze twee output-vectoren is als een lopende band die blijft lopen, de andere bepaalt welke stukken van deze lopende band er bij de volgende stap wel of niet door mogen. Zo kunnen toestanden die dateren van enkele stappen terug nog invloed hebben op de komende stappen.

Figuur 7: LSTM



Zo kunnen deze netwerken bij voorbeeld het volgende woord in een zin beter

voorspellen als ze de context reeds weten. Hoewel LSTM's uitermate geschikt zijn in het improviseren op muziek binnen het tijdsdomein, zijn ze té complex geweest om volledig te implementeren binnen de limieten van de tijd voorzien voor de uitvoering van het project.

4 Uitvoering van het project

4.1 Probleemstelling

Het project beschrijft een artificiële intelligentie in staat om verder te improviseren op een menselijke ingave (of om het even welke vorige ingave). De imrpovisatie hoort conform te zijn met de regels van muziek, zonder deze te hoeven beschrijven in een regel-gebasseerd systeem. Hiervoor wordt bij voorkeur gebruik gemaakt van Neurale Netwerken.

4.2 Technieken

Er zijn verschillende technieken bedacht om binnen dit project aan de slag te gaan met neurale netwerken. Vooral in de voorschoteling van data aan het netwerk. Er waren drie technieken bedacht om het probleem uit te werken.

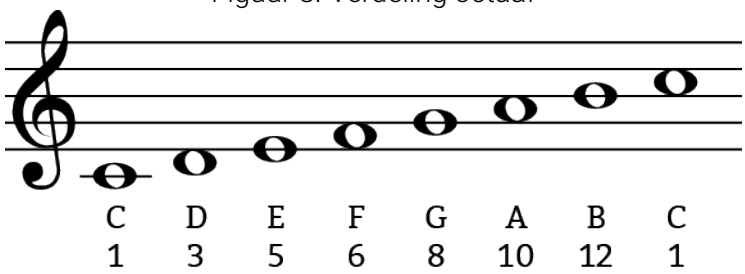
4.2.1 Primitive Octave

Een primitieve manier bedoeld als test. Primitive Octave beperkt de noten tot één octaaf. Een octaaf is per definitie een interval. Primitive Octave gebruikt enkel de noten die binnen **dat ene interval vallen**. De rest valt buiten kwestie. Deze techniek vereenvoudigt sterk de scope van muziek in het algemeen. Aangezien elk octaaf toch maar de dubbele frequentie is, en gelijkaardig klinkt, kunnen we eenvoudig het principe van toonladders bewijzen a.d.h.v. slechts één octaaf. We houden hiervoor harmonische frequenties, sympathetic resonance, en overtones buiten beschouwing. Uiteindelijk horen deze integraal bij wat muziek interessant maakt, de trillingen die horen bij verschillende instrumentatie.

De volgende figuur geeft één octaaf na de Do (of C) weer. We tellen precies 12 noten tussen zo'n interval. Omdat in westerse muziek de halve-intervallen

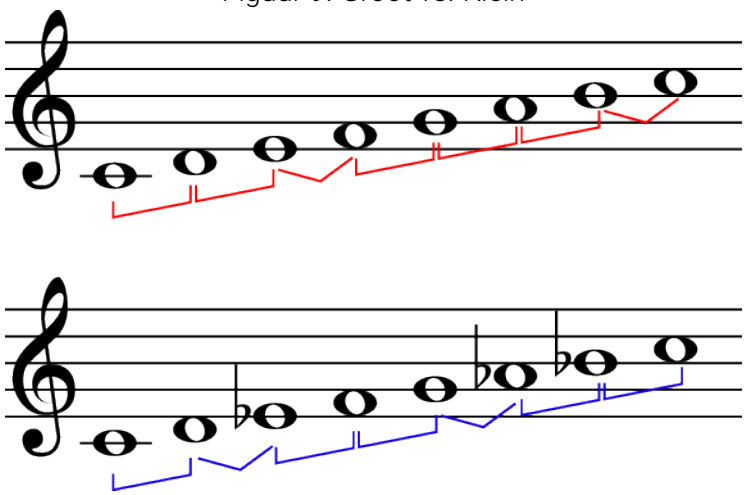
tussen de Mi en Fa, en de Si en de Do, **impliciet zijn.** Indien er geen mollen of kruisen aangegeven zijn, is dat per definitie altijd de regel.

Figuur 8: Verdeling octaaf



Echter, om zaken interessant te maken, kunnen we in plaats van Do-groot - want zo heet deze standaard toonladder in dit geval - Do klein maken. Het verschil tussen groot en klein, is de plaatsing van de intervals.

Figuur 9: Groot vs. Klein



Uiteindelijk hangt het enkel af van de plaatsing van deze intervals. Zo is Mi-groot een getransponeerde versie van Do-klein. Deze informatie is belangrijk. Het vertelt ons dat er maar één eigenschap belangrijk is, namelijk of de toonaard groot of klein is. Waar de halve en de volle stappen zich bevinden ten opzichte van de **root note**. In de bovenstaande voorbeelden is de root note altijd Do geweest. Maar je kan dit eenvoudig transponeren, het effect blijft gelijk. De frequentie waarop het start zal anders zijn.

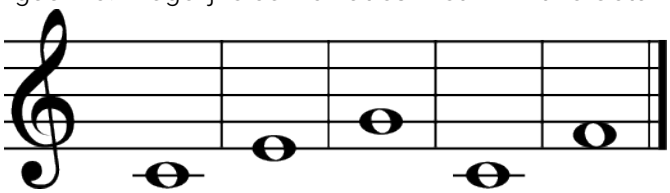
Het belangrijkste bij deze techniek is dus het kiezen van een base-note.

De basisnoot bepaalt de toonladder, maar dit is enkel voor ons hoorbaar. Het neuraal netwerk beschouwt het gewoon als de noot met waarde 0. Er volgen dan 12 noten die in beschouwing kunnen worden genomen, en dus ook als potentiële uitvoer kunnen optreden.

Door het netwerk te trainen met enkel noten binnen de groot-variant van de toonladder, dus met de rode figuur van hierboven, zal het netwerk leren om enkel met deze noten te reageren op een dergelijke input. Het zal ook terug vallen op deze variant indien men het een noot geeft buiten de toonladder die men heeft aangeleerd.

Deze techniek vult een input-vector van base-note tot laatste note. Waar gespeeld '1' is en niet gespeeld '0'. De output-vector bevat nummers waar het echter altijd tussen 0-1 zal liggen. Het hoogste getal wordt gekozen, en de positie ervan wordt dan ook gespeeld. Er kan zo telkens maar 1 antwoord zijn.

Figuur 10: Mogelijke combinaties met Primitive Octave



In dit voorbeeld hierboven, ziet men de gang van noten indien we het netwerk trainen met terts-intervallen zoals gedemonstreerd werd in de project-video. Het valt op dat soms andere intervallen plaats kunnen vinden. Het belangrijkste is dat er geen noten werden verkozen die vallen **buiten deze toonladder**. Met andere woorden is deze techniek geslaagd voor zowel groot als klein.

4.2.2 88-key Polyphony

Vertrekkende van het feit dat een **piano** kan beschouwd worden als **standaard instrument** die zowel hoge als lage klanken kan spelen, verkiezen we deze 88 toetsen als input voor ons netwerk. Deze techniek werd niet uitgevoerd. Het werd afgeraden om met grote input-vectoren te werken. Het zou inderdaad de zaken trager maken. Maar het had nog niet zo'n slecht idee geweest.

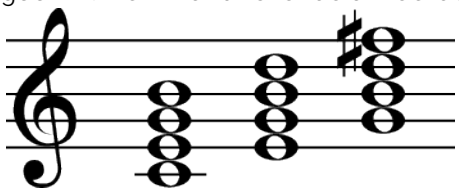
Het probleem had eerder gelegen in de output-vector. Waar, voor redenen van polyphony, in alle waarschijnlijkheid **héél veel toetsten ingedrukt zouden worden door het netwerk**. Er is geen probabiliteit vereist in deze kweste. Vanaf 0.5 of hoger, zou de noot spelen. Met 88-inputs zouden de mogelijkheden oplopen. Er zou héél veel training nodig geweest zijn.

4.2.3 Optimized 8-fingered

Deze techniek was de veel-belovende remedie op de eerder fout gemaakte redenering. Optimized 8-fingered vertrekt van het probleem dat zich voordeed bij 88-key polyphony: De output kan veel toetsen bevatten. Gezien dit menselijk niet mogelijk is beperken we de mogelijke outputs, en dus ook inputs, tot een realistische 8 toetsen. Er bestaan natuurlijk stukken van o.a. Debussy waar je voor één hand plots 7 vingers nodig hebt, maar dat houden we hierbij buiten beschouwing.

Optimized 8-fingered deelt elke input-waarde door 88, omdat er 88 mogelijke toetsen zijn. Na training zal het netwerk leren werken met complexe akkoorden. Zelfs akkoorden in combinatie met bas-octaven. Hiermee kan direct al gevorderde en complexe muziek mee gecreëerd worden.

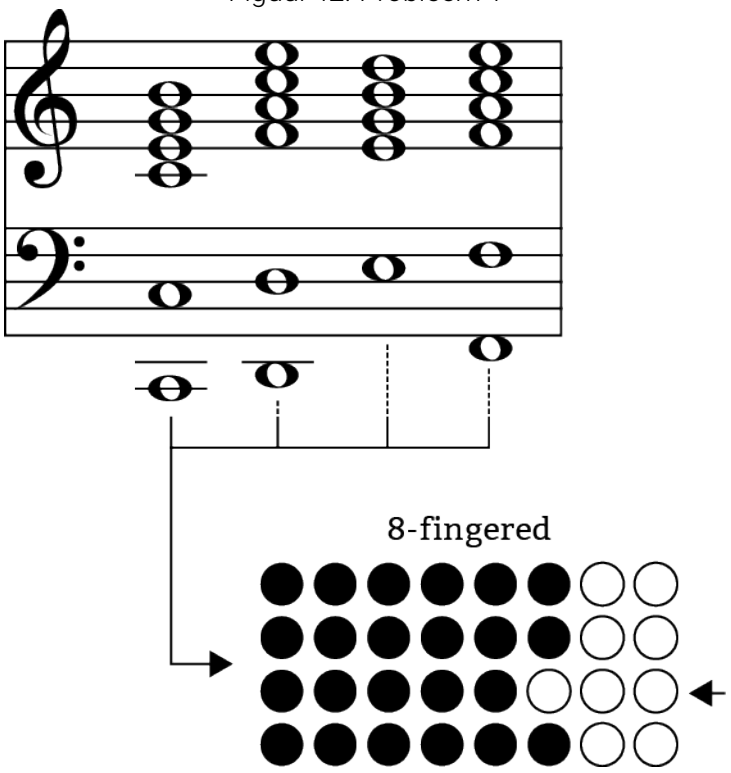
Figuur 11: Dominant-zevende akkoorden



Merk op dat hier de toonladder - groot of klein - niet meer uitmaakt. We leren het netwerk aan te denken binnen muziek door akkoord na akkoord complementair te spelen, zonder fouten. Het laatste akkoord in het bovenstaande voorbeeld is niet meer in Do-groot. Er is dus een verandering van toonladder gebeurd. Dit gebeurt impliciet. Zoals aan het begin van dit rapport aangegeven werd, maken we geen regel-gebasseerde engine. Het neurale netwerk kan echter wel enkel omgaan met wat het kreeg. Als de training-data niet netjes op elkaar past, zal het netwerk ook nooit leren deze combinaties te hanteren.

Dit alles klonk veelbelovend, maar er zijn een aantal problemen met zowel de implementatie ervan, als de structuur als neuraal netwerk. Het eerste probleem is de positie van de noten. Deze is altijd van links naar rechts uitgelijnd. Er kan geen rekening gehouden worden per vinger. We kunnen dus niet de input-vector beschouwen als vier vingers van links naar rechts. Het is enkel mogelijk om de toetsen in een vector te steken van zodra ze geregistreerd worden. Ee nmogelijkheid had geweest, om de noten te spreiden volgens hoe ver ze zijn van de basis-noot. Interessant zou zijn om daarvan de effecten te kunnen waarnemen.

Figuur 12: Probleem 1



In de bovenstaande figuur zien we dat het derde akkoord in de rij een bas-noot mankeert. Daaronder zien we hoe de input-vectoren opgevuld zijn volgens de methode geïmplementeerd binnen de applicatie. Merk op dat, hoewel deze combinatie bijna identiek had geklonken als wanneer deze bas-noot niet had ontbroken, de input-vector héél erg anders wordt. De inputs zijn geshift naar links. Dit brengt het netwerk ”in de war“.

Het tweede probleem is gelijkaardig aan het eerste. In plaats van de positie in de vectoren, gaat het bij dit probleem eerder over de toonhoogte. Namelijk, een klein verschil in het netwerk, brengt een groot verschil teweeg voor ons gehoor. Stel dat we één van de noten zouden transponeren, een halve noot omhoog of omlaag, wegens een afronding of wegens approximatie, dan kan het resultaat verkeerd klinken. Merk op dat het niet radicaal anders is. Het gaat hem hier om het feit dat het **verkeerd is**. Het gaat dan om een akkoord dat niet mocht worden gespeeld.

Dit is zichtbaar tijdens het trainen van het netwerk. De training gaat niet op omwille van de dichtheid van de data. We kunnen de data niet specifieker maken in deze opstelling. Het netwerk probeert te convergeren maar de regels zijn niet duidelijk herkenbaar in de cijfers. Er moet dus worden nagedacht over een complexer systeem waar rekening gehouden wordt met harmonie.

4.3 Netwerkarchitectuur

De architectuur die schuilt achter elk van deze technieken is een MLP. Telkens met 2 hidden layers die het dubbele bevatten van de hoeveelheid neuronen in de input-vector[5]. Via synaptic.js is het trainen van een netwerk altijd het zelfde, ongeacht de architectuur.

5 Testen

5.1 Primitive Octave

Zoals eerder vermeld zijn er hier maar twee valide testen die kunnen worden uitgevoerd. Of het netwerk al dan niet groot-toonladders en klein-toonladders kan onderscheiden.

Test #1 Primitive Octave Minor	
Training Data	minorsFinal.dat
Final Error	2.063480
Begin of test data	
Input	**Output**
C	G
D	C
Eb	F
F	C
G	Eb
Ab	G
Bb	Ab

In elke test schuilt zich wel altijd een vorm van repetitie. Hier is de cyclus gesloten na 4 noten.

$$C \rightarrow G \rightarrow Eb \rightarrow F \rightarrow C$$

Andere noten keren terug naar noten die wel deel uitmaken van deze cyclus. Het resultaat is met andere woorden nogal eenzijdig.

Test #2 Primitive Octave Major	
Training Data	primitive.dat
Final Error	2.25081521
Begin of test data	
Input	**Output**
C	E
D	C
E	C
F	E
G	E
A	C
B	E

Bij deze worden de patronen minder interessant. Het hangt sterk af van de training die gebeurt in het netwerk.

$$C \rightarrow E \rightarrow C$$

5.2 8-Fingered Optimized

Bij deze techniek convergeert het netwerk niet omwille van redenen die eerder besproken waren. Dit maakt het onmogelijk om testen uit te voeren. De errors blijven hoger dan 3.5, en dit zorgt er voor dat de outputs neigen naar eenzelfde noot, zonder veel interessante variaties.

6 Future work

6.1 Architectuur

Hoewel het origineel de intentie was om te werken met RNN's is dit niet gebeurd. Er kon nog in het tijdsdomein gewerkt worden, maar daarvoor moet er eerst een deftige implementatie komen voor enkelvoudige antwoorden.

6.2 Voorstelling in data

Data voorstellen als toetsen die ingedrukt worden werkt makkelijk en direct van input naar output. Maar het heeft zoals eerder aangetoond problemen. Als men enkel met 1 & 0 werkt voor de toestand van de input, dan hebben we veel inputs nodig (88). Dit zorgt er voor dat er meer tijd nodig is om te trainen. Met 88 toetsen kunnen er véél combinaties voordoen, tenzij we deze limiteren in de output.

We willen dus zoeken naar een andere en betere techniek. Dit kan a.d.h.v. een totaal andere techniek door bij voorbeeld rekening te houden met harmoniën.

7 Referenties

[1] Wisc.edu, A Basic Introduction To Neural Networks. Online. 02/01/2017

http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

[2] Michael Nielsen, Using neural nets to recognize handwritten digits. Online. 01/2017

http://neuralnetworksanddeeplearning.com/chap1.html

[3] Wikipedia, RNN. Online. 01/2017

https://en.wikipedia.org/wiki/Recurrent_neural_network

[4] Colah's blog, LSTM. Online. 27/08/2015

http://colah.github.io/posts/2015-08-Understanding-LSTMs

[5] Jeff Heaton. Introduction To Neural Networks in Java. Heaton Research. 2008.