# Machine Learning Engineer Nanodegree
# Capstone Proposal

Bas Donker
Date: 2018/01/10

## Domain Background:

Music is more than a sequence of random notes. There are many guidelines to follow when composing music, some of which can be taken loosely and can be bent or even broken, while others must be adhered to at all times in order to make something that sounds coherent.

Even before a composer writes the first note on paper, there are decissions about the musical piece that need to be made. For instance, they must consider what key the piece will be in, whether it will be major or minor and what time signature to use.

After the musical features described above have been decided on, there are other things to keep in mind when writing the music. For instance, the relationships between the tonic, predominant and dominant notes, chord progressions, and cadences.

It's also worth noting that repetition is something that occurs frequently in music. Basic musical ideas are generally repeated throughout a song in a number of different variations.

Personally, I'm sometimes involved in projects that require a piece of music to be written. I have enough of an understanding of Music Theory to be able to tell what makes a musical piece good and what makes it bad, but I lack the creativity to compose something myself. It would be useful if the creative process can be handed over to an algorhithm in a way that allows me to tweak its output in order to finalize the end product.

## Problem Statement:

Building an Artificial Intelligence that can write music following all of these different rules will be a challenge. If our model just plays random notes at random times, it will be very obvious to the listener that it's not music at all. Our model also might get stuck and start repeating the same note over and over again.

We need a machine learning model that can understand the underlying structures of a musical piece and is able to reproduce them. We need a model that can learn sequential data. It needs to be able to look at a whole sequence of notes and infer the next note that needs to be played.

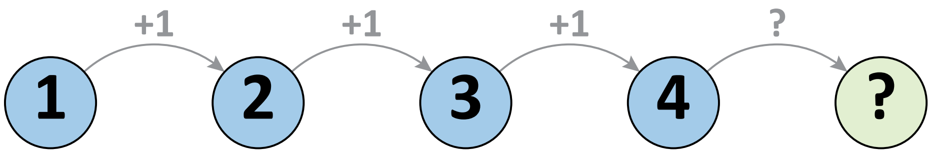
Traditional feed-forward neural networks are not able to do this. A traditional neural network could be given a sequence of notes and make a prediction on the next note, but it won't understand the underlying structure of the sequence, which is needed to compose something enjoyable to humans.

## Solution Statement:

Neural Networks work very well in many situations. For instance, they are able to tell apart handwritten digits with high accuracy and are able to tell images of cats from images of dogs.

However, Neural Networks are not good at is sequential data. We can't input [1, 2, 3, 4, ?] in a Neural Network and expect it to answer "5", even though 5 is the obvious next number in the sequence. Even if we train the network in a way that it does predict 5, it will be stumped when we input [5, 4, 3, 2, ?] or [1, 2, 4, 8, ?]. We as humans can tell that the outputs should be "1" and "16" respectively. This is because we understand sequences in a way a neural network can't.

When we look at a sequence of [1, 2, 3, 4, ?] and we're asked what the next number needs to be, we look at the rate of change of each number to determine the answer.
To illustrate:

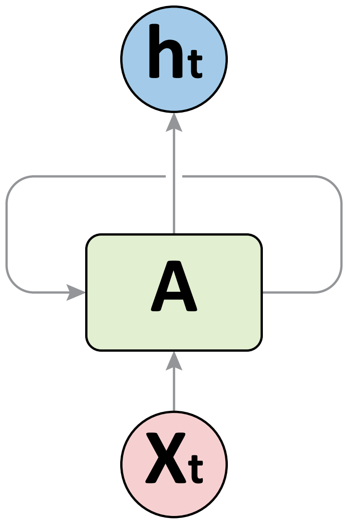


We see that 1 is added to each number, which leads us to predict one needs to be added to 4 to make the output 5.

It's easy to think of the grey numbers in between the other numbers as hidden states in a neural network. In order for a Neural Network to understand sequential data, it must use these hidden states in its calculations and destinguish patterns between them.

This is where LSTM comes in. LSTM stands for Long Short Term Memory and has been proven extremely effective in generating sequences such as in text.

This is what a simplified version of an LSTM cell looks like:



An LSTM cell takes some input x and produces some hidden state h. This hidden state then gets fed back into the network, along with the next input x.

This way, the network is able to remember how sequences change and develop over time. For example, it might learn that in the given sequence, there are a lot of notes played in quick succession and they range quickly between octaves. The LSTM will try to simulate that speed and varience in its output.

Inside cell A, there are more complex calculations. There are different variaties on LSTM cells, but in general, there are different gates that determine how important new information is and at what rate it should 'forget' old information.

## Datasets and inputs:

The LSTM will be trained on the following dataset:
http://abc.sourceforge.net/NMD/nmd/reelsh-l.txt

It's a collection of songs in ABC Notation. Since LSTM's have been proven successful in generating sequences of text, it should be able to produce music in ABC notation, as it's entirely text based.

A problem we come across when looking at individual notes in music is that different notes have different meanings if the song is in a different key. Note "C" in the key of C has a completely different meaning as oposed to the same note in the key of "A#".

One way we can overcome this is by transforming all the songs in the training data into the same key in the pre-processing stage.

We can also measure for each note how many steps are in between it and the key. This way it won't matter what key the song is in. For instance, in the key of "E", a note sequence of:
[E E F F | G D E D]
will turn into:
[0 0 2 2 | 4 -2 0 -2]

## Evaluation Metrics:

Traditional Neural Networks are not ideal for generating sequences due to the reasons described earlier, but they are good at classifying data into separate categories. We can create a Neural Network that is able to tell whether a piece of music was composed by a human or a machine and we'll use this as an evaluation metric for our model.

The more often this network classifies something that was composed by the LSTM model as having been composed by a human, the better the LSTM is at composing music.

## Benchmark model:

To test the effectiveness of an LSTM, we will also create a traditional Feed-Foward Neural Network that will try to attempt to compose music as well. I predict that the LSTM model will perorm much better than a traditional Neural Network.

# Project Design

## Pre-Processing:

We take all the songs in ABC notation from the following dataset:
http://abc.sourceforge.net/NMD/nmd/reelsh-l.txt

We will implement a few options of transforming the data and we can later check what combinations of options achieve the best results.

A few options we might have is transforming the notation from notes as A, B, and C to how many steps are in between those notes and the key of the song. We can also look at how much the pitch changed compared to the previous note. Or we might keep the letters, but shift them in pitch so that all songs will be in the same key.

Take the following sequence in the key of “E”:
E, E, F, B | C, D, E, G

If we transform this sequence to be in the key of “C”, it will look like this:
C, C, ^C, G | ^G,, ^A,, C, ^D

If we take for each note the distances from the key, it will look like:
0, 0, 1, 7 |-4, -2, 0, 3

If we take the distance from the previously played note, our new sequence looks like this:
0, 0, 1, 6 |-11, 2, 2, 3

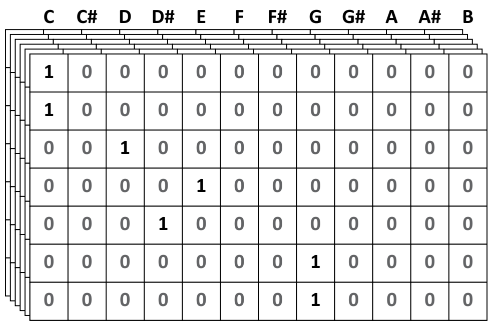
After transforming the sequences, we’ll store the transformed file somewhere so we can use it as training data.

Next, we need to format this data in such as way that the LSTM can take it as input. Our LSTM will take a 3 dimensional matrix as input.

We’ll take the long sequence of notes and split them up into smaller sequences of size maxlen. For each of these smaller sequences ‘x’, we record what the next note in the sequence is, so we can use that as our data lables ‘y’.

We will also one-hot encode all the possible notes and characters in the sequence.

Here is a visualisation of what the model’s input would look like:



| C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

## Building the models:

We will create three different neural networks. For simplicity, let's give name them A, B and Z.

Algorithm A is our LSTM Network that will attempt to compose sequences of music.
Algorithm B is a traditional Feed Forward Neural Network, attempting to do the same.

Z is also a traditional Neural Network, but it's trained on being able to tell whether a given sequence is composed by a human or by a machine.

To prevent A and B from overfitting, both will have a dropout layer.

## Training the models:

A and B will start learning from the training data. After each epoch, they will both compose a piece of music of their own.

After having composed some music, A and B will send their creation, along with some human-composed sequences to Z.

Z will then look at the sequences, and try to determine whether they were made by a human or a computer. Once Z has recorded its predictions, it will use the new data it got to train, so it will be better at being able to tell human composed music from AI composed music next time.

In the beginning, both algorithms will appear to do well, as Z hasn't learned much and is randomly guessing whether music is created by a human or a computer, getting it right about 50% of the time. After a few iterations, Z will be better at telling these sequences apart, but at the same time, A and B should now be better at creating human-sounding music.

In other words; A and B will try to convince Z that they are humans, not machines, while Chuck is getting better at telling humans and machines apart.

The model that is most consitently able to fool Z into thinking they are human is the better model.

Once Z has trained sufficiently, we can choose between A and B as the model to develop further on. We can take the best model and tweak the hyperparameters to see if it becomes even better at passing Z's 'humanity test'.