Neural Network Genre Classification Engine

Connor Henderson

January 3, 2015

Abstract

The purpose of this experiment was to create a neural network that would be able to do basic genre classification of music files in the WAV format. The network receives 600 pieces of input data generated by a discrete Fourier transform in order to attempt this classification. The network is trained using backpropagation

Contents

1 Introduction

A neural network is a way to process information inspired by biological nervous systems like the brain. It is composed of a number of highly connected nodes (neurons) working to solve a given problem. In this case, it is being used to classify the genre of an audio clip. To do this, a discrete Fourier transform is performed to process the audio file, and resulting values are fed into the network.

A discrete Fourier transform is an algorithm that converts sound samples to sinusoidals. From this, the cosine, sine and harmonic amplitudes as well as the phase shift of the harmonic are used as input values for the neural network.

2 Background

2.1 The Network Structure

A basic feed-forward neural network consists of a set of input nodes and a set of output nodes potentially with layers in between known as hidden layers. The input nodes take the input values, and a weight is applied as the value is passed to each node in the next layer. The next layer will take the sum of these weighted values and send a value, based on the activation function of the node, to the next layer. This value will also have a weighted multiplier. This process repeats until the output layer is reached. In this experiment, the activation function used is the sigmoid function $1/(1 - e^{-x})$.

2.2 How the Network Learns

The neural network in this experiment uses backpropagation as its method of learning. The network is given a set of training problems with known answers and over a number of iterations, changes each of the weights on the connections between nodes to produce a more accurate result. Backpropagation involved running a set of data through the network, calculating the difference between the result and the correct answer, and assigning parts of that error to each node that contributed to the answer. This filters back through the network and the training process repeats.

2.3 Discrete Fourier Transforms

The discrete Fourier transform is a fundamental part of digital signal processing. It turns a waveform into its sinusoidal components. These components can be used to reconstruct the original waveform. In this experiment, the discrete Fourier transform breaks the waveform into 100 sinusoidal components, from which data is taken to serve as the input to the neural network.

3 Program Details

3.1 How to use it

The program opens with a simple user interface. There are a number of text boxes and buttons. The text boxes are preloaded with some default values.

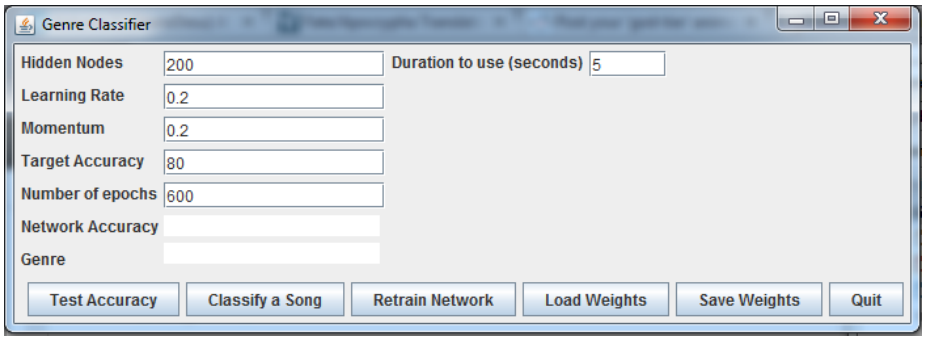

Figure 1: The user interface

These values produces some comparatively good results during experimentation and were used as a baseline. These values are also examples of valid input. The text boxes do not explicitly restrict input to valid data, so inputting things like characters or symbols will break the program.

The "hidden nodes" field represents the number of hidden nodes that will be used in the network. More hidden nodes will increase the runtime of the program. The "duration to use" field represents the amount of the audio clip to use for the discrete Fourier transform in seconds. The program will crash if this value exceeds the length of the audio file. The "learning rate" and "momentum" values are used to train the neural network and are intended to be values between 0 and 1. The "target accuracy" and "number of epochs" fields are the end conditions for training. If either of these values are reached, training will halt. The "network accuracy" and "genre" fields are non-editable fields that provide some information from the program. "Network accuracy" is the percentage accuracy achieved by the network at the end of training. "Genre" is the result of running a WAV file through the network. This is how the network decided to classify the input file.

The "Test Accuracy" button will run a single epoch on the network in order to gauge the accuracy of the set of weights. This is intended to be used after loading a set of weights from a text file, but can be used at any time. The "Classify a Song" button will open up a prompt to open a file. Anything other than a WAV file will not work. The "Retrain Network" button will run a full training cycle using the given parameters in the text fields. It begins with a random set of weights and trains using backpropagation from there. The "Load Weights" button will bring up a prompt to select a text file to load weights from. This files is expected to be one generated by the "Save Weights" button and thus will crash the program on an improperly formatted file. The "Save weights" button will save the current network weights to a selected text file through a file select window. This allows the user to save a set of weights deemed to be useful for later. An example weight file is included with the program. The "Quit" button will terminate the program. In the current state, if a new training set is desired, the program must be modified directly. This is a function that could be added to the interface in the future.

4 Analysis

Since genre classification is a very complex problem, some liberties were taken with handling the analysis of this experiment. First, only six genres were used as categories. These genres are rap, alternative rock, classic rock, comedy, video game, and metal. These were chosen to provide varying degrees of similarities and differences, as well as due to availability. The comedy genre contains more spoken word than the other five. It contains work by artists like Tim Minchin. the video game music is all orchestral which is what differentiates it from the others. Alternative rock, classic rock and metal are used to attempt to determine whether the results can detect smaller differences. During the trials, the network was very frequently producing no conclusive result. To counteract this, instead of using a direct thresholding approach to determining whether a particular input corresponded to a specific genre, the highest valued output node was taken as the chosen genre. If the node did not explicitly activate, the result was labelled as uncertain, but the answer was still considered. During training, this improved accuracy from 15% to as much as 40%. This basically means the network would activate correctly at the rate of random chance, but, using the maximum value output, whether activated or not, did a much better job. The training methods were not altered for this analysis method.

5 Experimental Results

5.1 Input Manipulation

This experiment uses two types of input. For training, the program reads in a premade text file of input values and answer keys. This is to substantially improve efficiency as well as for space considerations. A training set was created using 5.33GB of WAV files and took a one minute sample of each to run the discrete Fourier transform on. This process took roughly 30 minutes and would otherwise have to be repeated for every epoch. The text file is roughly 500 KB and it takes less than one second to process an epoch using it. The duration used for the discrete Fourier transform for test files can be modified in the user interface.

5.2 Data

The variables used to generate results are as follows: number of hidden nodes, learning rate, momentum, number of epochs, and audio clip duration for the discrete Fourier transform. 135 sets of input data were used and 6 WAV files, one for each genre, were used for testing.

5.3 Methodology

To compare neural network training parameters, weights were initialized randomly and the network was trained based on the values given in the user interface. A set of six test songs, one of each genre was used to test generalization. For the duration of the audio file trials, the first set of weights acquired was saved using the save button and reused for all tests. Since the algorithm takes a random sample of the selected duration, this provides sufficient variance to

warrant multiple trials at each duration. Three trials were run at each set of parameters.

5.4 Discussion

All trials me single parameter modifications on a set of baseline parameters. This baseline was 200 hidden nodes, 0.2 learning rate, 0.2 momentum, 80% target accuracy, 600 epochs and 5 a second sample duration.

The baseline trials ran with internal accuracy from 25% to just over 50% in several trials. The testing set ran no better than random chance. This suggests overfitting was a problem.

5.4.1 Hidden Nodes

Trials were conducted at 150 and 250 hidden nodes. On the 150 node trials, the internal accuracy decreased by roughly 7% on average. Test accuracy was still equal with random chance.

The 250 node trials had roughly equivalent internal accuracy as the baseline set. The test set was, again, no better than random chance.

Changing the number of hidden nodes did seem to have a bit of an effect on internal accuracy, but on the trial set, it seemed to have little to no effect.

5.4.2 Learning

Learning rate trials were conducted at 0.1 and 0.4. The 0.1 trial had an internal accuracy roughly equivalent to baseline. However, the test set performed slightly better. One trial got 3 out of 6 genres correct. This may still have been luck, but at least it's an improvement.

The 0.4 learning rate trial had a slight improvement in internal accuracy as well as a slight improvement in testing accuracy, but it wasn't as much as the 0.1 trial. The test results are still in the realm of statistical noise, so it seems like a lower learning rate may be better.

5.4.3 Momentum

Momentum trials were conducted at 0.1 and 0.4 as well. The 0.1 momentum trial showed a similar improvement to the 0.1 learning rate trial. Again, one trial got 3 out of 6 genres correct.

The 0.4 momentum trial performed terribly. The internal accuracy showed substantial improvement, but the test result was well below statistical average.

The momentum trials suggest that lower momentum leads to better results, and higher momentum creates overfitting problems. Higher momentum increased internal accuracy and decreased test accuracy and low momentum kept internal accuracy equal, and increased test accuracy.

5.4.4 Number of Epochs

Trials were conducted at 400 and 800 epochs. The trials at 400 epochs showed a decrease in both internal accuracy and test results. The trials at 800 epochs showed similar decreases on average. However, one of the trials at 800 epochs gave the highest internal accuracy of any trial at 54.07%. It still didn't perform

any better on the test cases though. The 800 epoch trials also produced the set with the lowest internal accuracy at 22.22% This suggests that it was simply luck that produced the better result.

5.4.5 Sample Duration

This trial was conducted a little differently than the previous trials. Since this trial had entirely to do with the discrete Fourier transform and nothing to do with network training, the network weights were held static for these trials. The variance between sets comes with the random selection of the sample of the audio clip.

Trials conducted using 10 second samples performed much better on the test set than all previous trials, and seemingly more consistently so. This may still be due to luck, but it is the best lead of any trial in this experiment.

The 2 second sample trial was also more accurate overall. It was equal to the 10 second trial. Logically, luck would be an explanation for this since the results were less consistent, but perhaps 5 seconds is just exactly the wrong duration to get results.

Overall, modifying duration had the most positive result on the trials.

6 Conclusions

Through all the trials involving the neural network training parameters, the overall result was roughly equal to random guessing. Some sets of parameters performed slightly better, but some performed worse.

The biggest impact came from the duration of the sample of the test clips. While the training parameter changes made minor improvements or decreases in accuracy, changing duration of the samples for testing improved test accuracy when both increasing and decreasing. This may be partly due to the random nature of sample selection, but could have something to do with the accuracy of the Fourier transform on the sample. A longer sample avoids anomalous sections of a song. For example, there may be a two second quiet spot in the middle of a metal song that would give a completely different result from the transform.

Overall, the trials did not provide very good results. There is a huge amount of variance between songs in the same genre and many different ways to classify each song. This was never meant to perform well, but merely to show that a neural network could be a viable way to perform this classification. This experiment did a better job of showing what doesn't work.

When the result was wrong, there was no good correlation between what was expected and what was guessed. It would be reasonable to see classic rock misclassified as alternative rock or metal, but this was frequently not the case. The network would classify a metal song as rap or comedy just as often as classic rock or alternative rock. It really did feel like it was just guessing blindly a lot of the time.

Overfitting was a huge problem in this experiment. The internal training accuracy consistently performed well above random chance. However, testing results did not. Most tests did equal to or worse than random chance. This is most likely due to the training set and its lack of diversity.

There are many things that could be improved in this experiment. Musical genres are an extremely large dataset with tons of variables and classification methods and this experiment only tested a very small subset of this field.

7 Future Improvements

Since this was only testing one way of classifying genre, and unsuccessfully at that, there are a large number of things that could be improved upon. Input data and more network variables are two examples.

7.1 The Raw Data Set

The training data was a relatively small set of songs from a relatively small set of genres. For each genre, there were three different artists. A proper dataset would consist of thousands of songs from hundreds of artists. This would provide much more variance and hopefully better results. It would also take far more resources. Ideally, there would be enough data to run a single epoch and get accurate results.

The dataset used seemed to run into huge problems with overfitting, and this is likely due to the training set. A more diverse and substantially larger training set should create a large improvement in results.

7.2 Input Data

The input data was created by running a discrete Fourier transform for 100 harmonics. There are many options for input data in this field. Many entirely different values could be used as well as the discrete Fourier transform data.

For the discrete Fourier transform, the number of harmonics makes a huge difference in accuracy to the original waveform. It may also make a difference for genre classification. Making the number of harmonics variable may make a difference in classification. This would require the ability to change the number of input nodes, which is not an option in this experiment. More harmonics creates more input variables and vice-versa. Some of the values obtained from the discrete Fourier transform could also be omitted. Perhaps the amplitude of the harmonic is the only important component retrieved from the transform. The other values may have no correlation at all, or even throw off the classification.

Some other possible parameters could be beats per minute, year, pitch range, key, beat strength, or many other things. Combining things like BPM and beat strength many also provide useful features to consider.[1] The problem is extracting some of these features. A number of different algorithms can be used for this in addition to or instead of the discrete Fourier transform used. One option is a short time Fourier transform. This could give information about signal changes over time. Spectral flux analysis is another option This would analyze the difference between normalized magnitudes of the signals over time.

A number of other filters could also be applied to the signal before running it though the neural network. Signals could be enveloped or run through a low pass filter. The signal could also be centred to zero.

Another common thing to use for genre classification is Mel Frequency Cepstral Coefficients.[2] This is another way to concisely represent time domain

waveforms. These values could be processed further and could provide a smaller number of inputs than the discrete Fourier transform used in this experiment. These different methods of processing inputs have shown to have accuracies as high as 97% [2], so this is clearly a big problem in this experiment. Since these other methods create fewer inputs for the network, it should also run substantially faster for this phase. These experiments used fewer genres that have more variance, but the results are still significant since accuracy was so much better.

7.3 Output Data

A much more complex model for outputs could also be used for this experiment. Having genres as the output nodes may not be a good way to model output. Gaussian mixture model states are commonly used for speech detection. Perhaps a similar method could be used for genre classification. This provides a probabilistic approach instead of a black and white yes or no classification. This would likes model the complexity of genres much more effectively.

7.4 Network type

A feed-forward neural network may not be the best way to organize a problem like this. Perhaps a self-organizing feature map would do a better job of sorting genres. This would give a more accurate display of the complex map that is musical genres. Since there is no clear line that separates genres and more specifically, the diverse sets of sub-genres within each genre, it may be much more efficient to, instead of hardline categories, use a method that shows gradual change. Genre classification is such a fuzzy problem that even humans only have accuracy around 70% at best.[1]

References

[1] George Tzanetakis, `http://dspace.library.uvic.ca:8080/bitstream/handle/1828/1344/tsap02gtzan.pdf?sequence=1`, 2002

[2] Michael Haggblade, Yang Hong, Kenny Kao, `http://cs229.stanford.edu/proj2011/HaggbladeHongKao-MusicGenreClassification.pdf`, 2011