Deep learning for music genre classification

Tao Feng
University of Illinois
taofeng1@illinois.edu

Abstract

In this paper we will present how to use Restricted Boltzmann machine algorithm to build deep belief neural networks.The goal is to use it to perform a multi-class classification task of labelling music genres and compare it to that of the vanilla neural networks. We expected that deep learning would out-perform however the results we obtained for 2 and 3 class classification turn out to be on par for deep neural networks and the vanilla version with small data set. By generating more dataset from the original limited music tracks, we see a great classification accuracy improvement in the deep belief neural network and its outperformance than neural networks.

1 Introduction

Artificial neural networks have great potential in learning complex high level knowledge from raw inputs thanks to its non-linear representation of the hypothesis function. Training and generalizing a neural network with many hidden layers using standard techniques are challenging. According to some related research literatures(2), training deep neural network with the traditional back-propagation algorithm tend to get the network stuck in a local minima. However with the development in pre-training algorithms such as auto-encoder and restricted Boltzmann machine one can achieve better results. Furthermore recently there is a surge in the interests of the deep learning algorithms in both academic and industries. This is partly due to the recent advancements in computational techniques and hardware such as GPU, a lot of deep learning algorithms can now be effectively trained. Many recent findings show that deep neural networks in some tasks outperform most of the traditional classification algorithms such as support vector machine and random forest. What we will be focused on in this project is a branch of automatic speech recognition. Specifically we would like to apply various deep learning algorithms to classify music genres and study their performances. In the first section of the paper, we will set up the problems of interest for the paper and describes the algorithms that we will compare. In section.3 data and data preprocessing are discussed. We then describe the implementation of the algorithms in section.4 and finally in section.5 and 6 I will report the results and discuss what could be done in the future to improve the classifier.

2 Setup and Theory

Here we will describe the general set up of the experiment. Early neural network arises from an attempt to simulate how the brain learns concepts. The way the brains learn is based on network of neurons. Each neuron receives signals from nearby neurons as its input which it then combines and transforms into its output, the activation signal. The activation signal is then fed to other neuron as the input. An individual neuron is represented as a perceptron unit. The only slight difference from the common perceptron is that the activation is computed using a logistic function:

$$h(\vec{x}) = \text{logistic}(\vec{w} \cdot \vec{x} + \theta) \tag{1}$$

This logistic function can be chosen according to a specific problem. In the case we will use a sigmoid function, $f(z) = \frac{1}{1+e^{-z}}$. A digram showing one of a neural network representations, feed-forward network is shown in figure.1. The diagram shows the different layers of a neural network: input layer, hidden layers, and an output layer. Each node of the input layer is an element of the feature vector of a data point. Specific to the problem, the feature vector is a vector of transformed amplitudes that we will discuss in more detailed in section. 3. The output layer is a logistic function that takes in activation values of the previous hidden layer and use a softmax function perform multi-class classification task. We expect it to be able to predict the label of the data(the music genre).

$$\begin{aligned} Pr(\text{label} = i|x, w, \theta) &= \text{softmax}_i(w \cdot x + \theta) \\ &= \frac{e^{w \cdot x_i + \theta_i}}{\sum_j e^{w \cdot x_j + \theta_j}} \\ y_{pred} &= \text{argmax}_i Pr(\text{label} = i|x, w, \theta) \end{aligned} \quad (2)$$

The network is considered deep if there is more than one hidden layer. By stacking more hidden layers, neural network is more capable to represent complex concept with high degree of correlation(such as the audio data). The goal here is by training this neural network, we would like to predict the genre accurately in an unseen test data.

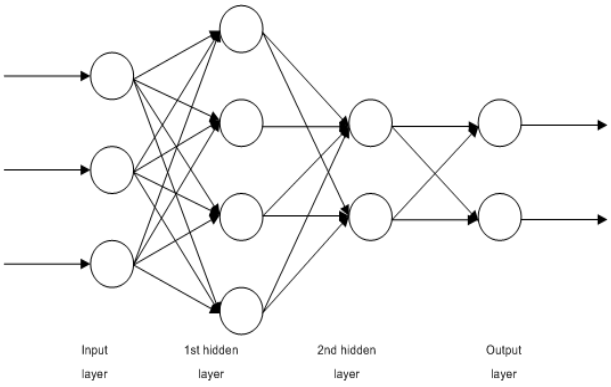


Figure 1: An illustration of the neural network hypothesis representation

2.1 Algorithms

As mentioned in the previous section traditional training method(backward-propagation with randomized initial weights) generally yields bad result when number of hidden layers is greater than 2. To train the neural network efficiently we explored two algorithms for "pre-training"

2.1.1 Auto Encoder

As we mention before that randomized configuration of the network is usually bad for training. Let us pose a problem of learning the good initial representation of the hidden layers. Vincent and collarborators suggested (2) a simple solution. One can consider an unsupervised learning algorithm that learns a sparse hidden representation of the input data itself. This can be done by devising a neural network, which its hidden layer is optimized to learn the input. Following from the algorithm of neural network one can concisely write down that

$$\begin{aligned} y &= s(\vec{w} \cdot \vec{x} + \theta) \\ \text{and } z &= s(\vec{w}' \cdot \vec{x} + \theta) \end{aligned} \quad (3)$$

The goal here is to learn weights and bias that minimize the reconstruction error loss, $\sum_i (z_i - x_i)^2$. This learned hidden representation is meaningful if there is a high degree of correlation in the data which we expect to be the case for audio data and that there is a constraint imposed that it has to be sparse. It is important to note that without the sparse constraint the auto encoder will just learn an identity mapping. To make a deep network we can stack these auto encoder on top of each others. This means that the code(learned hidden representation) of the $k^{th} - 1$ layer is fed as an input to the k^{th} layer which will also encodes it further. At the top most layer the logistic unit performs the classification. Once we pre-trained the hidden layers layer-wise we then fine tune the model by back-propagation. We will discuss how we can get around the constraint problem and how we could implement this efficiently in a later section. This can be described by the following diagram(figure.2). Since the implementation of auto-encoder has not been tuned to be successful yet, we restrict not to display the results from auto-encoder we have so far, but provide more results from Restricted Boltzmann Machine discussed below.

2.1.2 Restricted Boltzmann Machine

Here is another way to pre-train the parameters of the deep neural networks. Hinton and Salakhutdi-

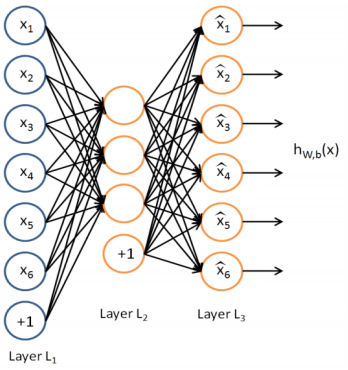


Figure 2: A neural network that maps an input feature vector onto itself.

nov(2) came up in 2006 with an energy based model. The probability distribution can be written as a function of an energy of a system as follow:

$$p(x) = \frac{e^{-E(x)}}{Z} \tag{4}$$

where Z is a normalization constant (5)

$$Z = \sum_{\text{all config}} e^{-E(x)} \tag{6}$$

from equations above we see that learning in this case corresponds to finding parameters that minimizes the the energy of the configuration of parameters. This can be done by minimizing the negative log-likelihood of the model.

$$\frac{1}{N} \sum_{\text{all config}} \log p(x)$$

This concept is a well known theory in physics called canonical ensemble approach in statistical mechanics. Minimizing the negative log-likelihood is the same as minimizing the free energy and hence finding the equilibrium state of the system. In restricted Boltzmann machine(RBM) we don't observed the model fully hence we have hidden variables(hidden layers). The specific energy function(Hamiltonian) we used is

$$E(v, h) = -b'v - c'h - h'Wv \tag{7}$$

where W is the weights in the neural networks. This Hamiltonian corresponds to the free energy of the form

$$F(v) = -b'v - \sum_i \log \sum_{h_i} e^{h_i(c_i + W_i v)} \tag{8}$$

Because of the fact that the visible unit and the hidden unit are conditionally independent given one-another we can write down

$$\begin{aligned} Pr(h|v) &= \prod_i Pr(h_i|v) \\ Pr(v|h) &= \prod_j Pr(v_j|h) \end{aligned} \tag{9}$$

If we can write down what the free energy is then we can minimize it using something like stochastic gradient descent. In general computing the free energy, eq.2.1.2 cannot be done analytically so we employ a Monte Carlo algorithm to find an expectation value of the stochastic gradient(2). The training algorithm of RBMs using Monte Carlo is described in more detailed in section.4.1

3 Dataset

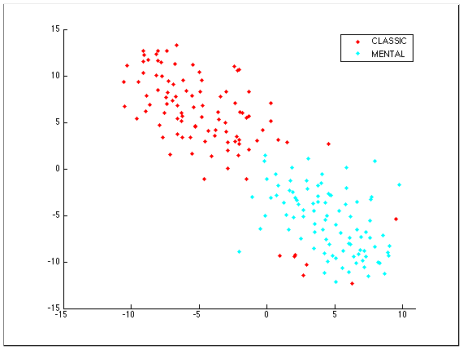
3.1 Data collection

We have compared several open sthece music dataset with associated metadata and select GTZAN Genre Collection, of which contains 1000 audio tracks each 30 seconds long. There are 10 genres represented, each containing 100 tracks. All the tracks are 22050 Hz Mono 16bit audio files in .au format. The 10 music genre includes: classical, jazz, metal, pop, country, blues, disco, metal, rock, reggae and hiphop.

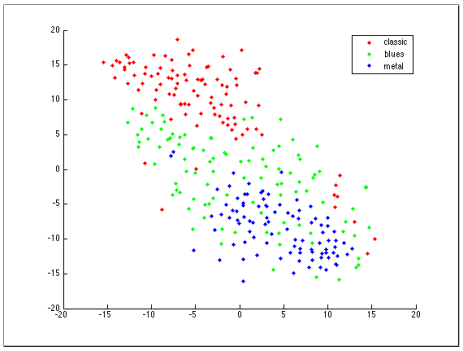
3.2 Feature selection: Mel frequency Cepstral Coefficient (MFCC)

As discussed in last section, each audio snippet could be represented as 30 seconds $\times$ 22050 sample/second $=$ 661500 length of vector, which would be heavy load for a convention machine learning method. From the acoustic literature we researched, the MFCC features is the most popular way to represent the long time domain waveform, reduce the dimension dramatically while still captures most information. As a pipeline, we first use a hamming window of 25 ms with 10 ms of overlap to generate consecutive smoothed frames. We than apply the Ftheier Transform over the frames to get frequency component and further map the frequency to mel scale, which models human perception of changes in pitch that is approximately linear

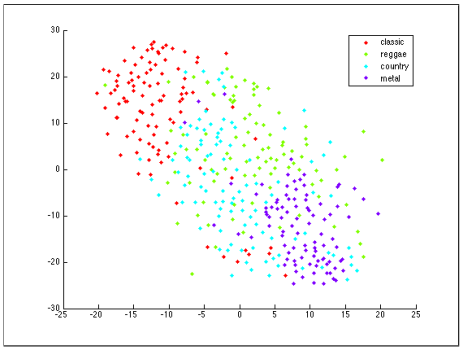
below 1kHz and logarithm above 1kHz. This mapping groups the frequencies into 20 bins by calculating triangle window coefficients based on the mel scale, multiplying that by the frequencies, and taking the log. We then take the Discrete Cosine Transform to de-correlate the frequency components. Finally, we keep the first 13 of these 20 frequencies since higher frequencies are the details that make less of a difference to human perception and contain less information about the song. This would result in 2600×13 features for each sample. In the experiment set up, we further divided the MFCC features into 4 roughly equal sized section and extracted first 40 of each section. In general, we generated a $13 \times 160 = 2080$ length of MFCC features to represent a 30 second audio file for the later experiment. We use the package t-SNE (Lvan der Maaten & Hinton, 2008) to scatter plot the data distributions. As shown in figure 4, the data set can be clearly differentiate in the 2-class cases, but then the data points gradually mix together as the number of classes increases. In the 10-class case, the dataset looks like squally spread out across different classes, which suggests it would be challenging to classify multi-class music genre.



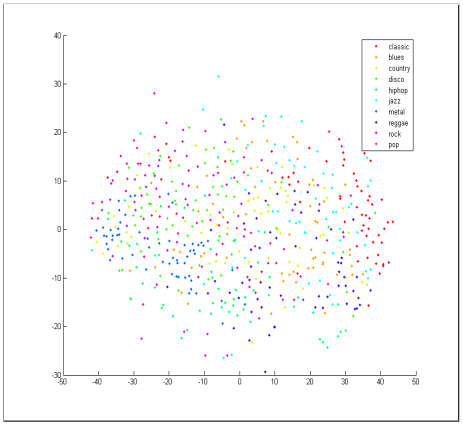
(a) 2 class scatter plot



(b) 3-class scatter plot



(c) 4-class scatter plot



(d) 10-class scatter plot

Figure 3: t-SNE (Lvan der Maaten & Hinton)

4 Implementation detail

4.1 Connect RBM with multilayer network

We build a 5-layer deep neural network (3 hidden layers) as the fundamental structure. We than train RBMs for each network layer except the output layer as the weight initialization. The key idea is to train RBMs iteratively between layers and stack them together on the multilayer architecture in the end. The steps of training is described below:

1. Pre-training
   (a) Train a RBM for the first layer with raw input data
   (b) Iteratively train another RBM for next layer with the hidden layer values from previous step
2. network-training
   (a) **Stack** the RBMs to corresponding layers as the initial weight of the network.
   (b) Use forward, backward propagation (or conjugate gradient method) to train the multilayer network.

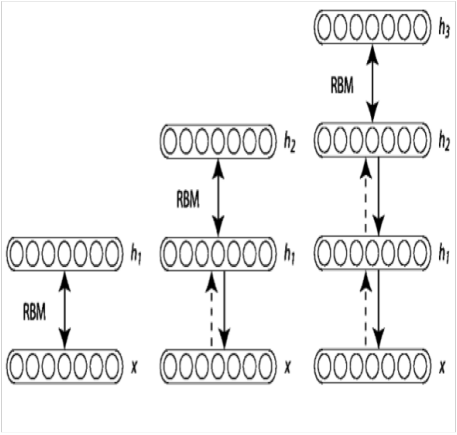


Figure 4: Iteratively train RBMs

4.2 Contrastive Divergence learning

To train an RBM, as discussed in previous section, we try to approximate $\hat{P}_{train}(v) \approx P(v)$ (the true, underlying distribution of the data) with the derived condition distribution

$$Pr(h_i|v) = g(b_i + \sum_j w_{i,j} v_j) \tag{10}$$

$$Pr(v_i|h) = g(a_i + \sum_j w_{i,j} h_j) \tag{11}$$

we apply the Contrastive Divergence learning, an variation of Gibbs sampling, to update the parameters:

Algorithm 1 Contrastive Divergence

1: For a training sample v1, compute the weight combination$W \times vi$ for hidden layer.
2: Generate hidden layer activation vector $h1$: $h1 = f(W \times v1)$
3: Map back from h1 to input layer to generate $v2 = g(W \times h1)$
4: Generate h2 again from v2: $h2 : h2 = f(W \times v2)$
   ** where we keep h1, v2 as binary value
5: let $m1 = h1 \times v1$, $m2 = h2 \times v2$
6: Update
   W with (m1 - m2)
   b with (v1 - v2)
   c with (h1 - h2)

As indicated by (2), 1 iteration of Contrastive Divergence would already generate good results. While in the experiment, we find more iterations would improve the performance, and we choose 5 iterations in the experiment.

4.3 Experiment setup

We use 60% of mfcc features as training set, and the rest 40% as testing set. 10 genres are equally weighted, each has 100 samples.
The number of iterations for RMB training fixed to be 5, the number of hidden nodes and the number of iteration of on the back propagation stage varies with different experiment and we report the parameters that achieves optimal result. We first test on 2 class classification, and then continue the experiment on 3 class classification and 10 class classification.

4.4 First experiment results

Classic VS. Metal	NN	DBN
	CG 3 line search	CG with 3 line search
Accuracy on train set	100% (120 out of 120)	100 % (120 out of 120)
Accuracy on test set	97.5% (78 out of 80)	98.75% (79 out of 80)
More stats: H hidden nodes T running time	H ~ [500 500 1000] T ~ 18 s	H ~ [1000 1500 2000] T ~ 7 min

1. DBN needs much larger network and more iterations (>50th) in the BP stage to get better performance. Much more time/computation consuming!
2. NN use smaller size network and converge very fast (after 6th).

(a) 2 class classification

Classic, Metal and Blues	NN	DBN
	CG 3 line search	CG with 3 line search
Accuracy on train set	90% (162 out of 180)	91% (164 out of 180)
Accuracy on test set	70.8% (85 out of 120)	69.16% (83 out of 120)
More stats: H hidden nodes T running time	H ~ [500 500 1000] T ~ 26 s	H ~ [1000 1000 2500] T ~ 20+ min

(b) 3-class classification

Classic, Metal, Blues and Disco	NN	DBN
	CG 3 line search	CG with 3 line search
Accuracy on train set	90.83% (218 out of 240)	69.17% (166 out of 240)
Accuracy on test set	63.75% (102 out of 160)	51.88% (83 out of 160)
More stats: H hidden nodes T running time	H ~ [500 500 1000] T ~ 26 s	H ~ [1000 1000 2500] T ~ 20+ min

(c) 4-class classification

Figure 5: Initial experiment result

From the 2 class classification result, we can see that both neural networks and deep belief neural networks almost perfectly classify the two genres, with both achieves 100% accuracy on training set and 97.5%, 98.75% for the testing set on NN and DBN respectively. This result is consistent with graph (a) in Figure 1, and get higher accuracy than the graph (a) indicates. However, the training of DBN is much more computational intensive that requires larger hidden layers and take more iterations than NN to achieve high performance.
For the 3 class classification, the NN and DBN is also competitive with each other, while DBN has the worse problem of over-fitting (higher accuracy on training set and less accuracy on testing set).
the experiment on 4 class classification shows that NN outperform DBN to a noticeable scale.

4.5 Second experiment results

Classic, Metal and Blues	NN	DBN
	CG 3 line search	CG with 3 line search
Accuracy on train set	92.89% (2508 out of 2700)	94.5% (2521 out of 2700)
Accuracy on test set	77.17% (1389 out of 1800)	77.94% (1403 out of 1800)
More stats: H hidden nodes T running time	H ~ [500 500 1000] T ~ 50+ min	H ~ [1000 1500 2500] T ~ 120+ min

(a) 3-genre classification with larger dataset

Classic, Metal, Disco and Blues	NN	DBN
	CG 3 line search	CG with 3 line search
Accuracy on train set	94.69% (3406 out of 3600)	75.3% (2712 out of 3600)
Accuracy on test set	60.46% (1451 out of 2400)	61.15% (1467 out of 2400)
More stats: H hidden nodes T running time	H ~ [500 500 1000] T ~ 60 min	H ~ [1000 1500 2500] T ~ 160+ min

(b) 4-genre classification with larger dataset

Figure 6: Second experiment result

The experiment result above indicate that DBN doesn't outperform NN as we expected before the exam. The first explanation we investigated was whether the it was because the relatively small dataset we have for the experiment. Use the same feature selection scheme disscussed in section **3.2**,

we chop the 2600×13 features into 15 160×13 subset of mfccs to simply generate more samples for the experiment, i.e. we now can generate 15 samples for each sound track with the same class so that we have 1500 samples for each genre instead of 100. We run the same experiment again and got some promising result:

- In both cases, with 15X more data set, the accuracy for both NN and DBN improves by an noticeable scale. And in both cases, the DBN outperforms the NN interms of train set accuracy and test set accuracy.
- For the DBN in 3-genre cases, the accuracy of training set improves by 3.5% and the accuracy of testing set improves by 8.78%, which significantly reduce the over-fitting problem with smaller dataset. Similar large improvement in 4-genre classification cases, with 6.13% improvement in train set and 9.27% improvement in testing set.

In general with larger data set, the DBN improves more than NN and gradually outperforms NN. And we are promising about the trend if we could create even larger dataset.

5 Future Works

We see the great improvement in experiment set 2 over experiment 1. But the original goal is to give correct classification for the 1000 (10 by 100) music track instead of the generated 10 by 1500 subsamples. The natural idea to solve this problem is we can think of the sub-examples created as in the context of ensemble method, in other words, we chop each music track to generate 15 samples with the same class labels, we then can use the majority vote of the 15 samples as the final prediction of the original music music track. This should be an effective method to combine ensemble method with deep neural networks architecture. Due the the time limitation, we haven't get time to implement this idea yet but will put it in schedule in near future.

Further we would like to improve the project on a more parallelepiped format to speed up the training process.

References

H.Lee, Y. Largman, P. Pham, A.Y. Ng *Unsupervised feature learning for audio classification using convolutional deep belief networks*. In NIPS 2009.

Yoshua Bengio *Learning Deep Architecures for AI*. Foundations and Trend in Machine Learning Vol.2, No. 1(2009)

Quac V. Le, et al. *Building High-level Features Using Large Scale Unsupervised Learning*

GE. Hinton and R. R. Salakhutdinov *Reducing the Dimensionality of Data with Neural Networks* 28 July 2006 VOL 313 Science

Vincent, H. Larochelle Y. Bengio and P.A. Manzagol, *Extracting and Composing Robust Features with Denoising Autoencoders*, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML?08), pages 1096 - 1103, ACM, 2008.

F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio. ?Theano: new features and speed improvements?. NIPS 2012 deep learning workshop

http://deeplearning.net/tutorial/rbm.html