Hybrid Neural Network for Music Decade Recognition

ECE 9063 – Data Analytics Foundations

Team Member:

Yifang Tian 251000189
Yiqing Zhang 250985276
Mengjie Li 250984682
Junfei Wang 250943952

Abstract:

This paper aims to solve the problem of song classification by decade, so that online music apps who use this method can try different categories to catch users' interests then gain advantages in this competitive field. The data is from Million Song Dataset which involves time-series features and time independent features. To achieve the best result, an ensemble learning architecture with LSTM Neural Network and Feed-forward Neural Network is implemented to make predictions. This model works for this problem and outputs high accuracy. The main finding from the project is: motivated by how different parts of human brain co-operates a new ensemble Deep Learning architecture is adopted.

1. Introduction

Online music is a highly competitive field now. Given it is expensive to obtain new users online, companies turn to focus on making apps better, and one of crucial ways to do that is showing contents in various ways so that they can keep catching users' attractions.

Many Apps lay out categories of music by genres and artists, but a listener may not love just one genre, or one single artist. Instead, lots of people have their own favorite musical era which has songs in different categories.

Therefore, classifying music by period of time properly helps music lovers to find favorite music faster. It means the Apps may make users feel convenient, so keep using.

This project is a ten classification problem, and aims to classify music by decade. The Data is from Million Song Dataset [1], and one challenge is both time-series and non-time features are involved. For each song, we need to consider the time-series pattern which can be 2-dimensional (musical features and corresponding time steps), as well as non-time descriptions which are time-irrelevant. To make the best use of the dataset, a Hybrid Deep Neural Network is implemented – RNN for time-series features and NN for time-irrelevant features.

In this paper, the background about algorithms we used will be introduced in the chapter 2. Then in the Chapter 3, shows how we analyze the data, and use these algorithms to solve the problem. Finally, the evaluation and result will be given in Chapter 4.

2. Background

LSTM Recurrent Neural Network

Recurrent Neural Network (RNN) adds recurrent connections on feed-forward neural networks, giving the model the concept of time. RNN is known to have issues with the 'vanishing gradient problem', and the most effective way to get around this issue is to use the Long Short-Term Memory (LSTM) variant [4].

The critical component in LSTM is the memory cell, which is modulated by three gates structure: The input gate protects the unit from irrelevant input events, and the forget gate helps the unit forget previous memory contents. The output gate exposes the contents of the memory cell (or not) at the output of the LSTM unit.[5] The gate structure allows information to be retained across many time-steps, and consequently also allows gradients to flow across many time-steps.

Feed-forward Neural Network

Feed-forward Neural Network (FFNN) consists of an input and an output layer, as well as hidden layer(s). Except for the input nodes, each node is a neuron that uses a nonlinear activation function [7]. FFNN utilizes a supervised learning technique called Back-propagation for training. Like is shown in Figure 1, the Neural Network contains 6 input nodes, 2 hidden layers with 4 and 3 neurons respectively, and only one output node.

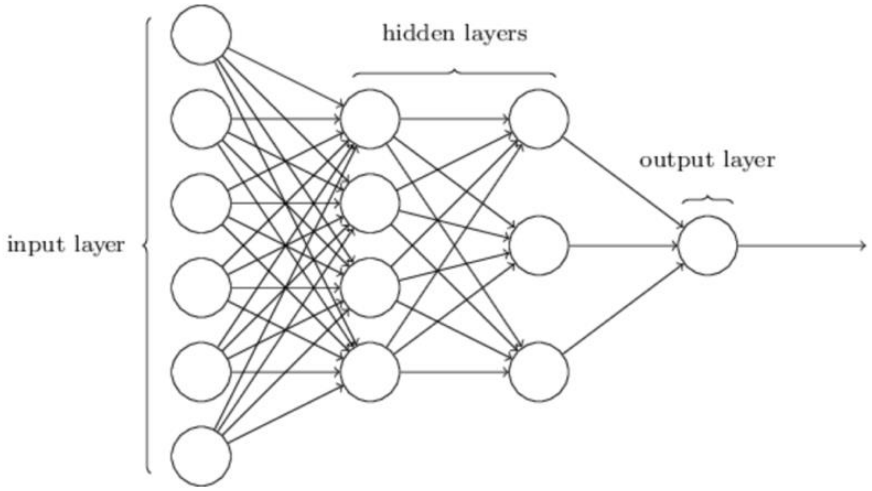

Figure 1. A typical 2 hidden-layer Feed-forward Neural Network

Ensemble Learning

Ensemble Learning is an algorithm that hybrids different models in some way, and there are several ways to ensemble different models, including bagging, boosting and stacking [2].

Bagging: It stands for Bootstrap Aggregating. It partitions the original data into subsets, and builds several models independently. By combining the result of each model, Bagging is a way to decrease the variance of predictions.

Boosting: It is a two-step approach, where one first uses subsets of the original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function. Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subsets contains the elements that were (likely to be) misclassified by previous models.

Stacking: It also applies several models to the original data. However, it uses another model/approach to estimate the input together with outputs of every model to estimate the weights or, in other words, to determine what models perform well and what badly given these input data.

Accuracy Measurement

In this classification problem, we calculate that how many the set of labels predicted for a sample strictly match with the corresponding set of labels in the true value, and calculate this number as a percentage of the total. This can be shown as the form in Equation 1 below:

$$\text{Accuracy} = \frac{\text{Num. of correctly classified songs}}{\text{Num. of all songs}}$$

Equation 1. Accuracy measurement

3. Methodology

Dataset Analysis

The dataset we used is in HDF5 (Hierarchical Data Format version 5). This set has 10,000 western songs from 1920s to 2010s. Features of a song are divided into two groups: time-series features and time independent features.

- **Time-series features**:

There are two time-series features: pitch and timbre, and each one has 100 time-steps.

Pitch: In a musical octave, there are twelve pitches from C to B. Like is shown below in the Figure 2, there are seven white buttons and five black buttons (totally 12) within an octave.

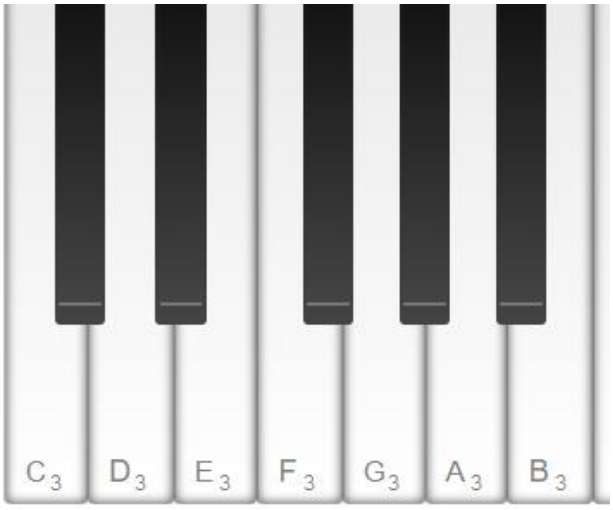

Figure 2. The 12 pitches within a musical octave

At one sample point (time step), pitch is a 12-dimension feature to represent whether every pitch from the octave appears.

Timbre: Timbre is the quality of a sound that distinguishes different types of instruments or voices. It is a sophisticated notion also referred to as sound color, texture, or tone quality. For example, almost everyone can recognize family members' voices because our brain is capable to distinguish these timbres. In this project, timbre has 12 sub-features, so it is also a 12-dimension feature in a single time-step[6]. In the Figure 3 below, shows these sub features, such as

loudness (the first one), brightness (the second one) and flatness (the third one).

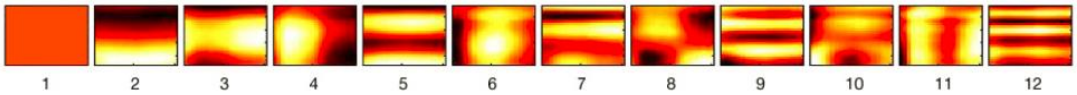

Figure 3. The graph of 12 timbre features

- **Non-time series features**:

There are more than 20 features in the raw dataset, we get rid of some which have large proportion sparse values, and remain ten of them whose correlations with decade relatively big. They can be divided into three groups below:
Length features (5): title length, term length, release length, name length and tags length
Music features (3): tempo, loudness, duration
Other (2): hotness (popularity), mode (major or minor)

Data Preprocessing:

- **Normalization**

For a better training process, the training data is normalized as in the equation 2:

$$\text{train\_nor} = \frac{train - Mean(train)}{Max(train) - Min(train)}$$

Equation 2. Normalization of training set

Similarly, the normalization process on validation set and test data, but use the statistic feature of training set, this process can be shown in the Equation 3:

$$\text{test\_nor} = \frac{test - Mean(train)}{Max(train) - Min(train)}$$

Equation 3. Normalization of test set

- **One-hot encoding**

Integer features need to use an one-hot encoding. Like is shown in the Figure 4, the input to this transformer is a matrix of integers, denoting the values taken on by categorical features. The output is a sparse matrix where each column corresponds to one possible value of one feature [10].

Decade
1930s
1950s
1960s

→

1930s	1940s	1950s	1960s
1	0	0	0
0	0	1	0
0	0	0	1

Figure 4. The transformation of One-hot encoding

- **Data splitting**

Before building the model, the data is shuffled and split into three groups with the percentage of 60% training, 20% validation and 20% test. The validation set will be used to select models, and final result will be based on test set. It can be illustrated in the Figure 5 below:

Figure 5. The data splitting percentage

- **Mini-batch**

To reduce the complexity, RNN and NN use mini-batch to train: in each epoch, the training set is shuffled, and get a small portion from the set. The size of the batch is tuned in the experiment.

LSTM Recurrent Neural Network

- **Time-series input**

The input data of the LSTM RNN is in 3-dimension. As demonstrated before, there are two time-series features (pitch and timbre) with 24 features in total within a single time step. Given 100 time steps, the data for every single music is a 2-D surface, and another dimension is the mini batch. The input data is like the 3-D space in the Figure 6:

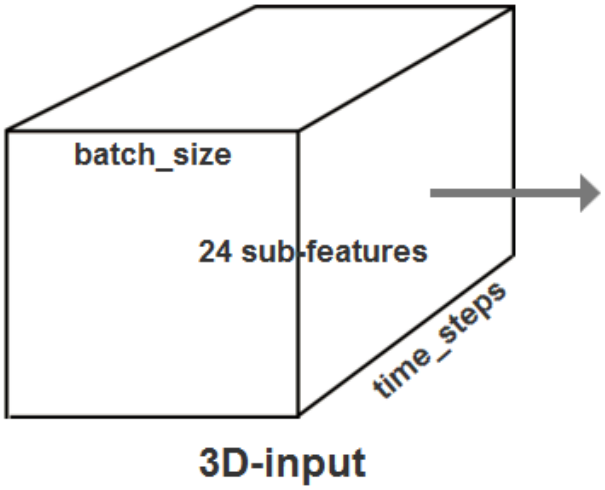

Figure 6. The 3-dimension input of LSTM RNN

- **Hidden Layer(s)**

Then LSTM memory cells are added into the hidden layer(s) to give the neural network capabilities to learn time relevant patterns. Forget bias (default: 1) is added to the biases of the forget gate in order to reduce the scale of forgetting in

the beginning of the training. It does not use peep-hole connections. The training begins at one hidden layer and 64 cells.

- **Softmax output**

For classification problem, the output of the Neural Network is a vector of size, instead of passing through activation function [9], it passes through the Softmax function, which is written in the form of Equation 4.

$$\operatorname{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

Equation 4. Softmax output

where z_i represents the i th element of the input to softmax, which corresponds to class i, and K is the number of classes. The result is a vector containing the probabilities that sample x belong to each class. The output is the class with the highest probability.

- **Cross-entropy loss function**

The Cross-entropy loss function for classification problem can be demonstrated as follow in the Equation 5.

$$\sum_{i}^{n} \sum_{k}^{K} -y_{true}^{(k)} \log\left(y_{predict}^{(k)}\right)$$

Equation 5. Cross-entropy loss function

Where k denotes the k-th classification, and i is the i-th training example.

Ensemble Model

There are two main reasons for ensemble RNN and Feed-forward Neural Network: Given the two types of features, using one type, and getting rid of another will lead accuracy lose. Secondly, ensemble model helps to avoid over-fitting [3].

After the analysis of data, the three mainstream ensemble methods are not suitable for the problem we are facing, so we come up with a new model, which is shown in the Figure 7 below: Firstly, all the time-series features are fed into RNN and fully trained. Its output is saved locally and can be considered as a less accurate prediction. We will use it as additional features and feed it into the FFNN model with non-time features to improve the accuracy.

When we get the output of RNN, which is in Softmax and can be seen as probabilities of decades, actually there are two options to use them: transferring them directly into the next stage, or encoding them into one-hot before use. After

testing, the second method gives us better efficiency, so we choose encoding them before ensemble. The testing result of Softmax input is shown in the next chapter.

Then we use these encoding outputs of RNN as additional non-time features to feed into Feed-forward Neural Network. Therefore, there are 20 features in the input layer of the Feed-forward Neural Network. In the hidden layer, we use fully-connected structure.

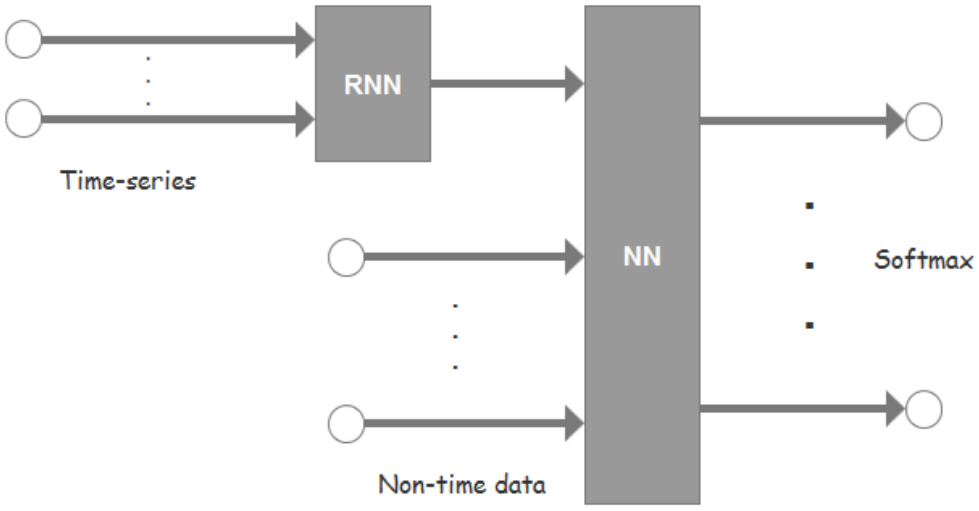

Figure 7. The architecture of the ensemble model

Feed-forward Neural Network also uses One-hot encoding, Cross-entropy loss function and Softmax in the output layer. Finally, back-propagation helps the model to update parameters.

4. Evaluation and Result

LSTM RNN

- **Number of LSTM memory cells**

We try only one hidden-layer in the project, and start the training process at 30 memory cells. Then continuously increases the neuron counts to 500, and 300 neurons achieve the best accuracy on both training set and validation set.

- **Optimizer of RNN**

Two optimizers (Gradient Descent and Adam) are tuned with variable-control and dynamically with other parameters. The Adam is a lot better in most of the situations as it won't stuck in a local minima.

- **batch size and iterations of RNN**

Batch size and iterations are tuned in the project. The batch size decides how many songs are trained together. If it is too small, the disadvantage is similar with Stochastic Training: the decrease of loss function becomes unstable. If it's too large, the training is time-inefficient. We try [2048, 1024, 512, 256, 128, 64, 32, 16, 8] as the batch size, and finally choose 64. As well, iterations is tuned during training, too large iterations wastes time, while too small value leads no convergence, we try [100, 500, 1000, 3000, 5000, 10000], then 5000 is the best.

- **Learning rate of RNN**

As for the learning rate, initial value is set from 0.1 to 0.0001 with 5 times smaller each time. 0.1 is too large so that the global minima might be skipped while 0.001 is too slow. It turns out that 0.0003 works the best especially with automatic update.

- **Result of RNN**

As in the Figure 8 and Figure 9, the loss of training process decreases rapidly in the first 1000 epochs, while the accuracy climbs. After that the algorithm converges slowly, and finally reaches the perfect convergence at around 5000 epochs. Similarly, the final accuracy of training set is about 98%.

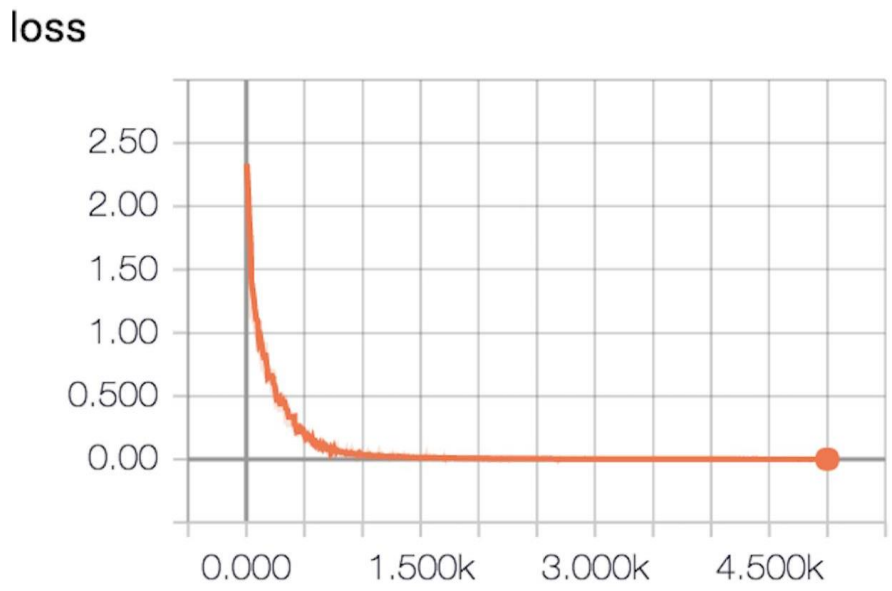

Figure 8. The training loss curve of LSTM RNN

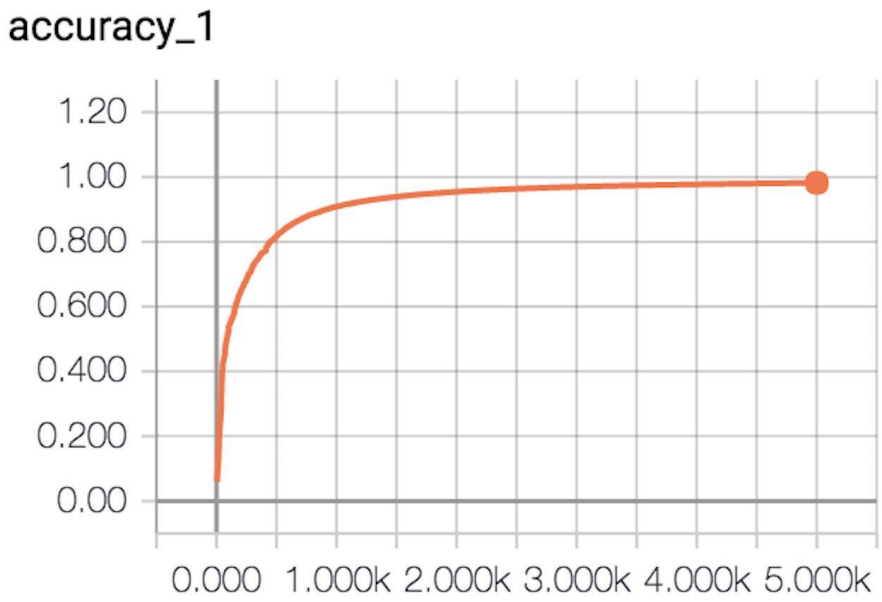

Figure 9. The training accuracy curve of LSTM RNN

However, the accuracy for Cross-validation set is not perfect: about 86%, which can be seen as over-fitting.

Ensemble Model

- **The result with only non-time features**

We've got the accuracy of RNN which is 98% on training set and 86% on validation set. This is the result comes from RNN structure using only time-series features.

The result of FFNN with only 10 time-irrelevant features is also evaluated by the validation set and recorded. In this step we didn't use the output of RNN as additional features. When we train it, firstly fix the number of neurons at [128, 256, 512, 1024], and then change the number of layers from 1 to 4. It turns out 4 hidden layers with total 1024 neurons gives us the best result on the training set. However, the problem is no matter how we tune the model, the accuracy on validation set is always under 60%. So the conclusion is Feed-forward Neural Network model with non-time features suffers from over-fitting, and reason can be no enough features and the relevancy of these features and music decade is not strong enough.

- **The result with decade probability inputs**

Like the demonstrations in the last chapter, when we design the connection part between RNN and NN, there are two options – Possibility or One-hot encoding. This section shows the result of using the Softmax output directly as the input of NN.

In Figure 10, it shows that the algorithm converges really slowly, and when

epochs = 10,000, the training accuracy is about 85%, while in Figure 11, the algorithm turns out to converge with an accuracy of 98% at about 90,000 iterations.

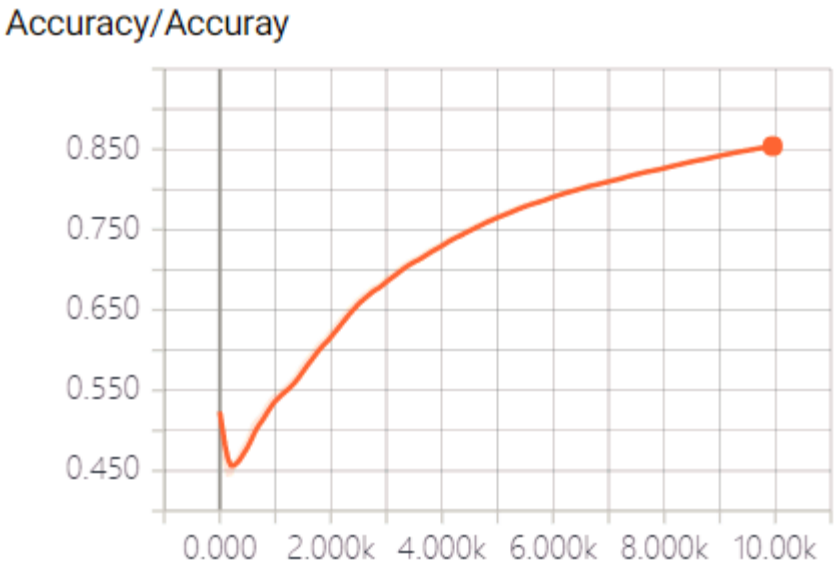

Figure 10. The training accuracy curve of ensemble model in 10k epochs without One-hot encoding

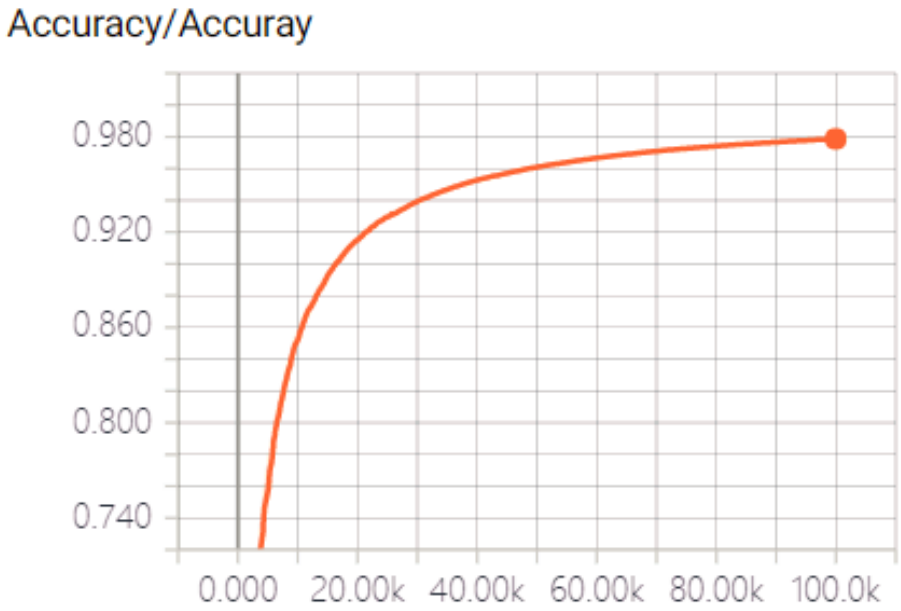

Figure 11. The training accuracy curve of ensemble model in 100k epochs without One-hot encoding

Considering the inefficiency of this ensemble model, we managed to do some pre-processing to the output of RNN before feeding into FFNN – the one-hot encoding.

- **Number of layers and neuron counts per-layer of ensemble model**

To build a Neural Network without underfitting and overfitting problems, the architecture starts at one hidden layer with two neurons, then increase both parameters to test (the upper limits for layers is 5, for the total number of neurons is 512). Finally, the best combination is two layers and neurons counts equals [64, 32].

- **Activation functions**

There are three activation functions commonly used at the beginning of deep learning. They are Sigmoid, Tanh and Relu whose equations and curves can be shown in the Figure 12:

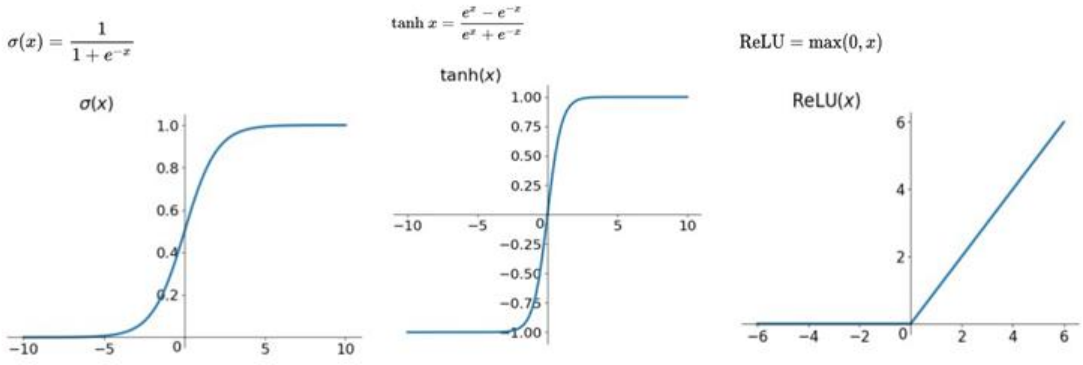

Figure 12. The 3 different activation functions

We observed the procoss and results of the 3000 iterations to compare the performances of these 3 activation functions. Sigmoid function performs best: both convergence speed and accuracy are OK, while Tanh converges slowly with worse accuracy. Additionally, Relu function has the same performance on time, but provides higher accuracy.

- **Learning rate**

Different learning rates (0.01, 0.03, 0.1, 0.3, 0.5) are tested in the project. The Figure 13 shows the learning curve for different learning rates:

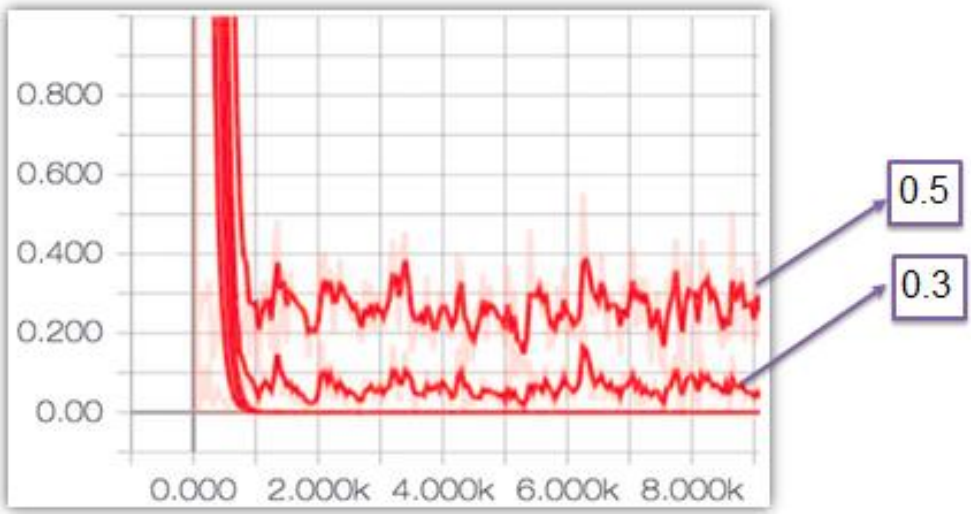

Figure 13. The training loss curve under different learning rates

When learning rate are 0.3 and 0.5, the loss curve doesn't converge. Instead, their fluctuations in the graph are obvious. The other three learning rates all give us good results. However, learning rate=0.1 converges faster, compared with 0.01 and 0.03. Meanwhile, they three get the same accuracy result. Therefore, we choose 0.1 as our learning rate.

- **The best combination and its result**

After tuning, the best model for the ensemble model is shown in the Table 1.

Parameters	Num. of layers	Num. of neurons	Iteration	Learning rate	Batch size	Activation function
Value	2	(64, 32)	10000	0.1	64	ReLU

Table 1. The best combination of the ensemble model

The learning curve of this model can be drawn as in the Figure 14.

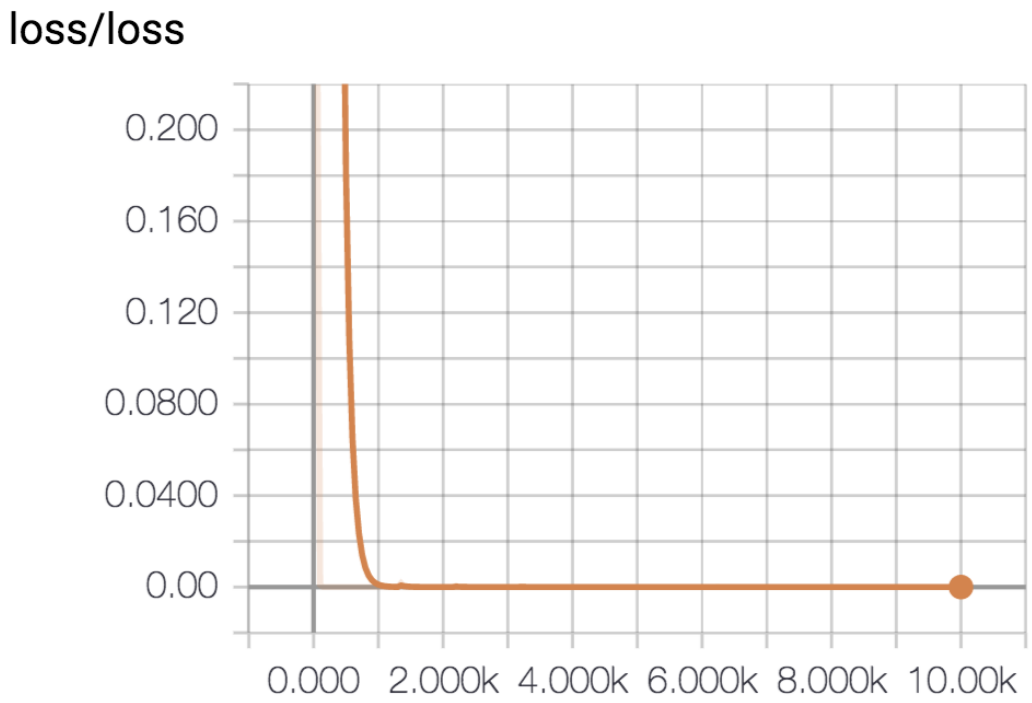

Figure 14. The training loss curve under different learning rates

The final result is shown on the Table 2 below, the network outputs perfectly performance on training set, whose accuracy achieves 100%. Validation accuracy and test accuracy are 96% and 94% respectively.

Accuracy	Training Set	Validation Set	Test Set
Percentage	100%	96%	94%

Table 2. The final result of the ensemble model

Libraries and Tools

The project is in Python language. We use Tensorflow lib to implement the LSTM RNN and Feed-forward Neural Networks. Also, Numpy lib helps up to process matrix calculations and transformations.

Additionally, when the coding work has been done, we try our work in a GPU on a laptop. It is a NVDIA GEFORCE 920 MX, and we use CUDA lib to deploy. When it runs, the training process is accelerated: actual time of each epoch at least halved.

5. Summary and future works

This project aims to solve a music decade classification problem. This is a valuable topic for most online music Apps, because they are in need of classifying music in different categories so that they can keep attracting their users.

The dataset we are using is a 1% subset from Million Song Dataset. Considering different two types of features inside it, we split the features of dataset into two groups: time-relevant features and time-independent features, then deal with the problem in a two-step ensemble model. After tuning and many trials, the accuracy for the test set is 94%.

LSTM RNN and Feed-forward NN are combined in this model. This ensemble model is actually an innovative approach from us; it is motivated by how different parts of human brain work together. We can also generalize this structure to any dataset with both time-relevant and time-independent features. This makes it much more valuable in reality. For example, the time-relevant traffic and time-independent features of website can be captured and analyzed for information security requirement.

The project also gives us an idea that there is no any fixed model for all the problems in real life. Based on the dataset and build up a tailored model can be the most effective way to achieve the best result.

For future works, we will try to solve this problem in a big data challenge. The whole set of Million Song Dataset is 300 GB, we may need to face the problem of volume and veracity in big data environment. During the process, some big data tools and techniques like Hadoop and MapReduce may be involved, which will be challenging experience for us.

6. References

[1] C. Doll, "Million Song Dataset. Columbia University. http://labrosa.ee.columbia.edu/millionsong," Journal of the Society for American Music, vol. 8, (1), pp. 121, 2014.

[2] D. West, S. Dellana and J. Qian, "Neural network ensemble strategies for financial decision applications," Computers and Operations Research, vol. 32, (10), pp. 2543-2559, 2005.

[3] J. Yang et al, "Effective Neural Network Ensemble Approach for Improving Generalization Performance," IEEE Transactions on Neural Networks and Learning Systems, vol. 24, (6), pp. 878-887, 2013.

[4] H. Park and C. D. Yoo, "Melody extraction and detection through LSTM-RNN with harmonic sum loss," in 2017, . DOI: 10.1109/ICASSP.2017.7952660.

[5] J. Dai et al, "Long short-term memory recurrent neural network based segment features for music genre classification," in 2016, . DOI: 10.1109/ISCSLP.2016.7918369.

[6] Y. Jia et al, "Long Short-Term Memory Projection Recurrent Neural Network Architectures for Piano's Continuous Note Recognition," Journal of Robotics, vol. 2017, 2017.

[7] S. Shanmuganathan, S. Samarasinghe and SpringerLink (Online service), Artificial Neural Network Modelling. (1st 2016. ed.) Cham: Springer International Publishing, 2016628.

[8] J. Lee et al, "SampleCNN: End-to-End Deep Convolutional Neural Networks Using Very Small Filters for Music Classification," Applied Sciences-Basel, vol. 8, (1), pp. 150, 2018.

[9] A. R. Naghsh-Nilchi and A. R. Kadkhodamohammadi, "Cardiac Arrhythmias Classification Method Based on MUSIC, Morphological Descriptors, and Neural Network," EURASIP Journal on Advances in Signal Processing, vol. 2008, (1), pp. 1-10, 2009;2008;

[10] Defence Research Establishment Ottawa and W.E. Thorp Associates Ltd, Evaluation of Neural Network Performance in Target Classification Applications. Ottawa: DREO, 198990-603.