Music-Driven Choreography using Deep Learning

by

Xueyao Jia
B.Sc., Northeastern University, 2016

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Music-Driven Choreography using Deep Learning

by

Xueyao Jia
B.Sc., Northeastern University, 2016

Supervisory Committee

Dr. George Tzanetakis, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Committee Member
(Department of Computer Science)

ABSTRACT

Deep learning is increasingly being used in a wide variety of tasks and application domains. In this project, the use of deep learning in automatic choreography is explored. Automatic music choreography has potential applications in a variety of areas such as robotics, computer graphics, and video games. Our goal is to generate dance movements automatically from analyzing music pieces. Towards this goal, we propose a model consisting of a 3-layer Long-Short Term Memory (LTSM) network to learn the relationship between the dance and the music. The trained model can then be used to generate new dance movements. We use STFT values as musical features and quaternion values as motion features. The model is then trained for 10 hours. The resulting generated motions can be viewed as animations using Blender, a well known free 3D creation software. The results show that our model is able to generate dance motions successfully but exhibits overfitting due to the small size of the data set considered.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My supervisor Dr. George Tzanetakis for supporting me in the low moments and providing constant encouragement. I am deeply grateful for his mentoring and patient guidance.

My parents Xinhui Li and Shaojie Jia for everything.

My friends Jiexing Hu and Feier Feng for believing in me.

DEDICATION

This work is dedicated to my grandmother, whom I deeply miss. To my parents and my grandfather. For always believing in me.

Chapter 1

Introduction

Dancing is an essential part of the lives of individuals and communities throughout the world [3]. Today, people usually dance for rituals, celebrations, fitness, brain development, mood enhancement, socializing, and ceremony.

Typically, dancing and music are in a close relationship. Throughout the human cultural development, dance and music always appear simultaneously [20], and the two activities are so closely integrated that many languages use just one word to describe both [20], therefore dance is closely bound to music in its structure, artistic expression, and interpretation. In order to closely combine dance with music, a choreographer needs to create dance movements according to the music features. Choreographers listen to the music and analyze music genres, features, inner emotions or messages, and then corresponding dance movements are designed according to the music information. This whole process is called choreography, which is the art of gathering and organizing movement sequences based on the music to embody or express ideas and emotions [12]. Researchers have used algorithms to simulate this process.

In recent years, many musical dance-related applications have been introduced for educational and entertained purposes, such as Second Life and MikuMikuDance. People can watch animated dance performances or even create their own. Therefore, a music-driven choreography framework would be popular and beneficial in various fields, including dancing teaching and performances, animated video games, robotics, etc. Specifically, it can help dancers with choreography, attract young generations interests in entertainment fields, and even improve the study of robotics.

There are many algorithms and methods that have been proposed in automatic choreography for decades and the main idea is almost the same: extract music features

and dance features, then find the mapping between the features. Interestingly, this main idea is similar to what choreographers do in real life.

In some early researches, Motion Graph [16][19] was one of the most popular methods to generate human-like motion sequences. Later machine-learning algorithms are widely used to generate movements, because it can dig the relationships between the music and the dance motions better, such as Hidden Markov Model (HMM) [21][18] and neural network. Recently, the use of deep learning is come up within choreography area, including Recurrent Neural Network (RNN) [10], Long Short-Term Memory (LSTM) [11][23][22], Convolutional Neural Network (CNN) [14], and Sequence to Sequence (Seq2Seq) Model (also known as Encoder Decoder Model) [15].

In this paper, we propose a music-driven choreography framework using deep learning. The structure is as follows.

Chapter 1 introduces the basic background, the relationship between music and dance, and the importance of, and recent developments in choreography applications.

Chapter 2 provides the literature review for the dance movements simulation. It provides a brief description of the major contributions. It also discusses the advantages or limitations of each approach.

Chapter 3 formulates the problem we are trying to solve. Then it introduces RNN and LSTM algorithms briefly and analyzes the reason we choose the LSTM algorithm. Finally, it describes the detailed structure of our network and the flow of each process.

Chapter 4 describes data acquisition, data pre-processing, feature extraction and training the network, as well as visualizing motion in Blender. It describes in detail the way we process the raw data and the problems we met during the experiments.

In chapter 5, we test the network using two kinds of data and analyze the results and limitations.

Chapter 6 concludes the achievements and limitations of the current project.

Chapter 7 gives detailed ideas about future work based on limitations in the current project and other related researches.

Chapter 2

Related Work

Simulations of human-like motions are widely used in various domains, including robotics, computer animation, virtual environment (prototyping) and video games. Generally, the main idea behind automatically generating dance steps is to find the mapping and relationships between the music and motions. This is a particularly challenging problem as the movements are continuous, high dimensional, and fundamentally expressive [10].

In early papers, researchers proposed various methods for generating realistic motion sequences. A popular approach was the Motion Graph concept based on motion capture data. To generate dance movements and consider human emotions when creating movements, based on the motion capture system, Takaaki Shiratori [19] described a new motion synthesis method synchronized to input music early in 2006. This system consists of three parts: a motion analysis, a music analysis, and a motion synthesis based on the results of the analyses. In these two analysis steps, motion intensity and features are extracted from the motion data, and music intensity, beats, and chord changes are extracted from the audio. Then a Motion Graph is constructed to search similar poses from given dance sequences and to connect them as possible transitions. Finally, when input music comes, the model calculates similarities between motion and music features, and traces the Motion Graph based on this correlation to generate new movements.

Similarly, in 2013, Minho Lee et al [16] used music similarity to find appropriate motions. The main idea is to construct music-motion database, which contains a number of segment-wise music-motion pairs. When a musical piece is given as an input, it will be divided into several small segments and each segment can find the most similar one in the previous music-motion database. Then the system can pro-

vide corresponding dance motion candidates and combine them together to form a new dance. The performance of this system is evaluated by a user study and got significantly higher ratings. However, the limitation of it is obvious: it cannot create novel dance movements that are not included in the database.

With advances in machine learning techniques, a variety of machine-learning algorithms have been proposed in order to generate dance movements in the form of motion capture data.

Hidden Markov Models (HMM) have also been used to generate dance movements. Early in 2005, Yi Wang et al [21] proposed a new model named NPHHMM (short for non-parametric Hierarchical Hidden Markov Model), which is a Hierarchical Hidden Markov Model with non-parametric output densities. They applied it to create a motion engine for learning and automatic recreation of human motion. In their research, they trained the NPHHMM on motion capture data containing ballet walk, ballet roll, disco, and complex disco movements. In 2012, inspired by early papers, Ofli et al [18] proposed a novel framework for learning many-to-many statistical mappings from musical features to dance figures. Based on these statistical mappings, they defined a discrete Hidden Markov Model and synthesize alternative dance figure sequences by employing a modified Viterbi algorithm [18]. This model is able to categorize the genre of music based on the Mel-Frequency Cepstrum Coefficients (MFCC) feature and then generate corresponding dance movements based on the music genres.

Later, with the development and popularity of artificial neural networks and deep learning, artificial neural networks have been successfully applied to the generation of dance movements. The significant advantage of using deep learning and artificial neural networks is that they enable the extraction of high-level features from raw sensor data (audio, motion capture). Also, the deep neural network is able to create new movements; whereas previous studies have a common limitation that they cannot create novel dance movements because the model reuses the choreographic samples in a predefined database when it generates new motions.

In 2016, Luka Crnkovic-Friis and Louise Crnkovic-Friis [11] presented a system called Chor-RNN for generating novel choreographic material in the nuanced choreographic language and a certain style. It uses Mixture Density Long Short-Term Memory (LSTM) type of RNN and is trained with motion-captured contemporary dance. After training for 48 hours, the model is capable of understanding and generating choreography syntax and style well and to some extent semantics. This paper is the first to try to combine deep recurrent neural network into the automatic choreography

area and it can produce novel choreographic sequences successfully. However, this paper only considers generating dance movements. It does not provide any methods to accompany any music. Inspired by Crnkovic-Friis, a variety of methods have been proposed. In 2017, Omid Alemi et al [10] presented an application named GrooveNet, a public interactive platform, in which an avatar can dance according to the music provided by audiences. This application uses Factored Conditional Restricted Boltzmann Machines (FCRBM) and Recurrent Neural Networks (RNN) to generate dance movements. The results show that this application is able to learn and generate the dance patterns from a very small training set. However, the performance is not satisfactory when using music that is not included in the training set. In 2018, Juheon Lee et al [15] proposed a neural network-based model that can generate novel and natural choreography. The researchers obtained audio-video data pairs from YouTube and used them as the training set. From the videos, the model extracts x, y coordinates as motion features, and then extracts Mel-spectrogram as audio features. Then it uses an autoregressive encoder-decoder network to generate skeleton sequences and compares the results with the ground truth data by using the L1 loss as a cost function. From the user study and online videos, this network shows good performance. However, it uses 2D rather than 3D skeleton positions as motion features. Also, the system is not stable if the training dataset is not big enough when using positions as features. Last year, Nelson Yalta et al [23] proposed a well-designed deep recurrent neural network based on their previous paper [22]. This model has one convolutional layer and multilayered Long Short-Term Memory (LSTM) layers as encoder part to process audio power spectrum, then it uses LSTM layers as decoder part to generate dance movements. In the decoder part, it uses auto-conditioned decode configuration to reduce the accumulation of feedback error. This model demonstrates reliable performance for motion generation and it can be used in real-time. Taoran Tang et al [20] also presented a well-designed LSTM-autoencoder model to extract a mapping between acoustic and motion features. Moreover, they use temporal indexes and a masking method to achieve better performance.

From the previous papers, we can find that in the recent year, the deep neural network is the main method in the computer choreography area. Inspired by Crnkovic-Friis [11] and Yalta [23], our project uses a multi-layer LSTM network to generate dance motions based on audio features extracted by using the Short-Time Fourier Transform (STFT).

Chapter 3

Problem Definition and Proposed Approach

In this chapter, we formulate the problem and introduce Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) algorithms briefly. Then, we describe the structure of our method at an abstract level.

3.1 Problem Definition

The input datasets are two sets of sequential features, consisting of motion features and music features. The music features are extracted from the input raw audio files and the motion features are derived from the input dance capture data. In our project, we use Biovision Hierarchy (BVH) files as input motion capture data. Our goal is to build a model, which can map from one music feature to one motion feature, to find the relationships between the music pieces and the motions. First, our model is trained by given collected data. Then, for any input music, our model can generate a new dance sequence based on the relationship we trained.

3.2 RNN and LSTM Algorithm

RNNs (short for Recurrent Neural Networks) are neural networks used to process sequential data, such as text, image, DNA sequence, music sequence, handwriting, speech, and stocks. Compared with traditional neural networks, they take time and sequence into account and have a temporal dimension [7]. For example, when peo-

ple need to analyze a word based on previous words and context, traditional neural networks would have bad performance because it has difficulty in remembering previous data, whereas RNN can address this problem well. It uses a loop to remember information, as shown in Fig 3.1, x_t is an input at time t, s_t is the hidden state at time t, which will be passed to the next time step $t+1$ as part of the input. It is used to remember the previous states. s_t is computed based on previous state s_{t-1} and current input x_t. o_t is the output at time step t.

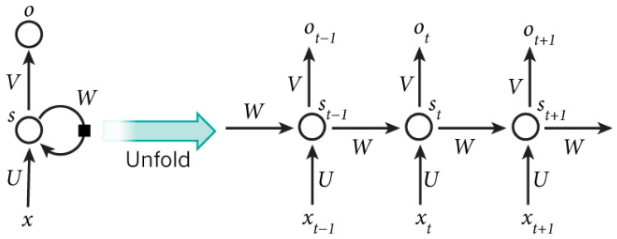


Figure 3.1: Structure of RNN cell [7]

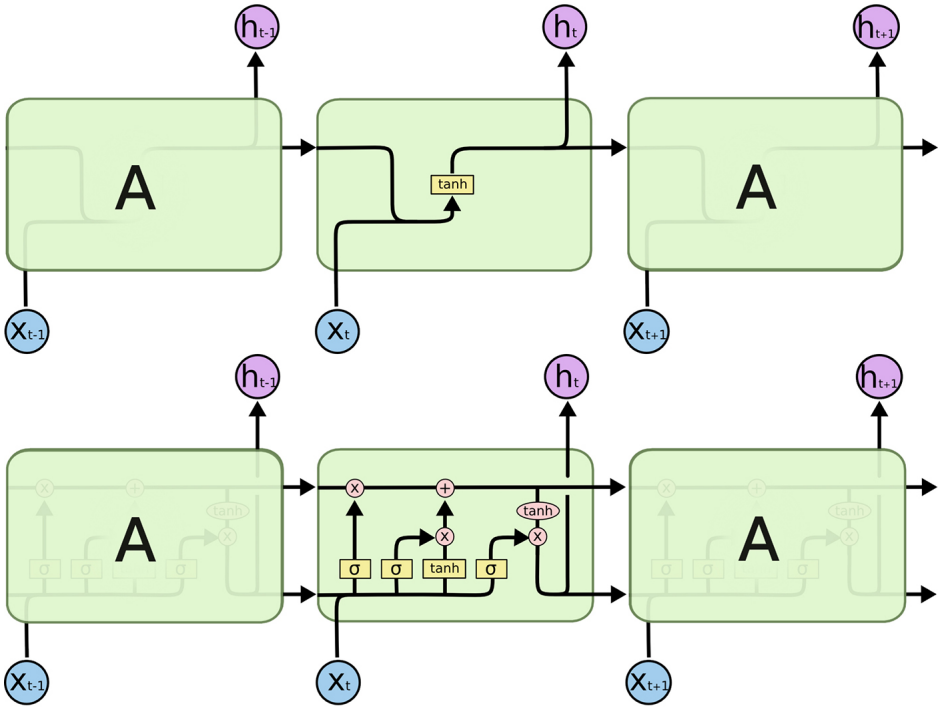


Figure 3.2: RNN cell structure (top) vs. LSTM cell structure (bottom) [9]

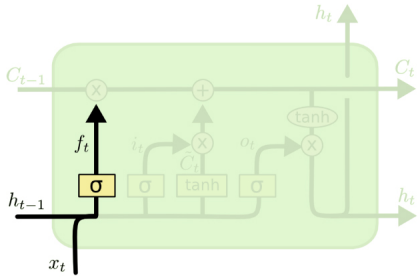
However, sometimes we only need to look at recent information to perform the present task and do not want to remember old data. Also, remember too much past

information may give rise to vanishing or exploding gradient problem because the model multiplies the same matrix too many times.

In this context, in 1997 Hochreiter and Schmidhuber introduced Long Short Term Memory networks (LSTM) based on Recurrent Neural Networks (RNNs).

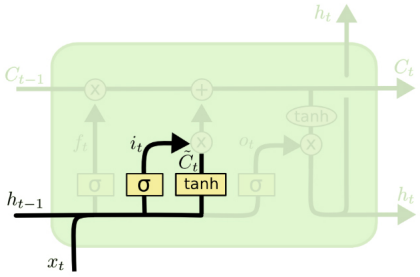
As shown in Fig 3.2, compared with the standard RNN that only has a tanh function in its cell, a normal LSTM has three gates (I.e. forget gate, input gate and output gate) to control the cell state.

The cell state C_t is the key in a LSTM cell. It is the horizontal line running through the top of the diagram with only some linear interactions. It can decide what information can be remained and which part should be dropped.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 3.3: Forget gate in LSTM cell [9]

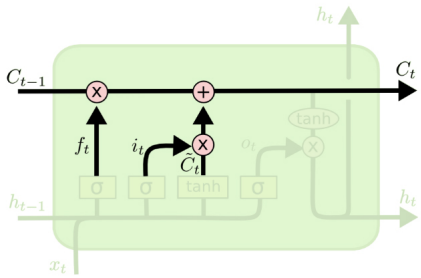


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 3.4: Input gate in LSTM cell [9]

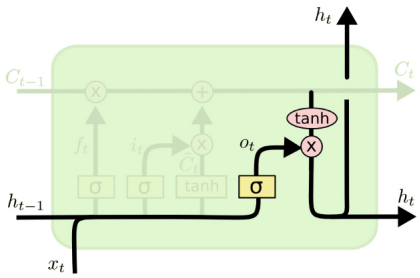
The first step in a LSTM is the forget gate f_t (Fig 3.3), which can decide what information to throw away from the cell state. It is a sigmoid layer using h_{t-1} and x_t as input information. It outputs a number between 0 and 1, 0 represents totally forget and 1 means completely remember.

The input gate (Fig 3.4) is to decide what new information to store in the cell state. It uses sigmoid and tanh functions to get i_t and $\widetilde{C}_t$ respectively. The value of $i_t \times \widetilde{C}_t$ is the information we will update, so called updated information.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.5: Update cell state in LSTM cell [9]



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3.6: Output gate in LSTM cell [9]

Next, we need to update the cell state. The new cell state C_t is the sum of forget information $f_t \times C_{t-1}$ and updated information $i_t \times \widetilde{C}_t$ (as shown in Fig 3.5), where C_{t-1} is the previous cell state. Then this new cell state will be passed to the next cell.

Finally, in the output gate (Fig 3.6), we can compute the hidden output h_t by using the updated cell state and o_t in sigmoid and tanh functions.

In this way, we can update the cell state C_t and get hidden state h_t in the LSTM cell.

3.3 Proposed Approach

In order to learn the relationship between the time-series data: music and the dance, we need a model that can perform sequence-to-sequence transformations. Therefore we create a three-layer LSTM network. The overview structure of our system is shown in Fig 3.7.

Generally, our model mainly consists of four parts: pre-processing, feature extraction, training and saving models, and visualizing motion in Blender.

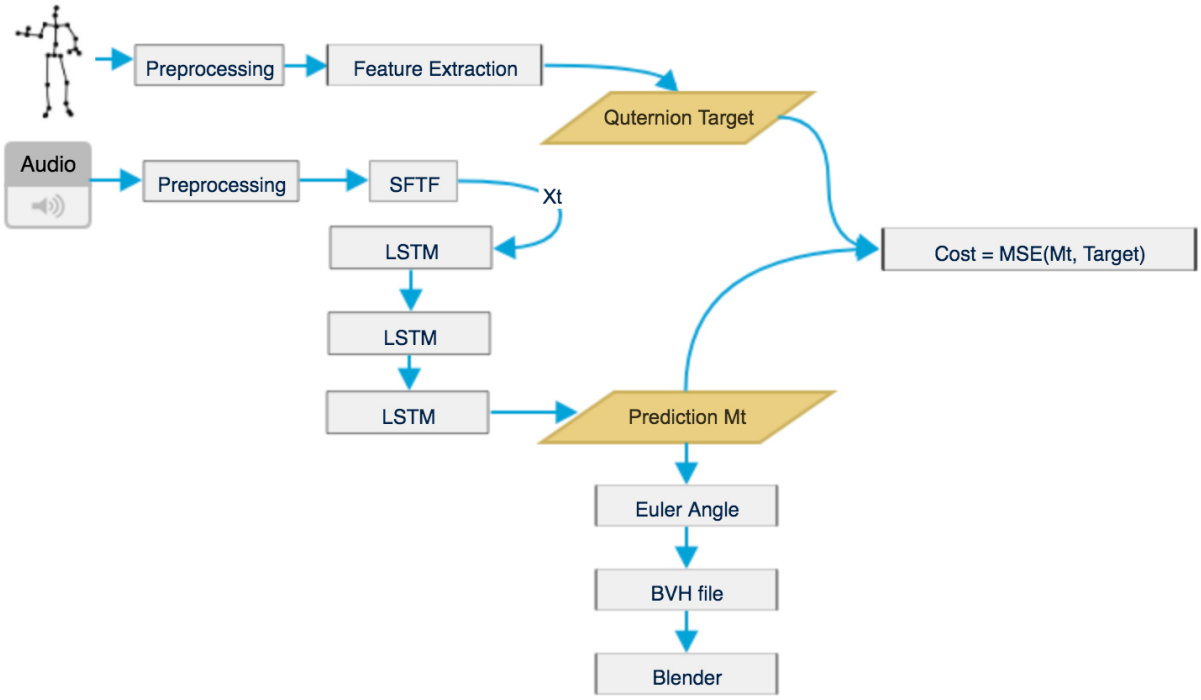


Figure 3.7: Model architecture

First, we need to do pre-processing on audio data and motion data. The pre-processing part for motion capture data contains clipping, unifying skeleton structure, setting frames number and frame time, centering the motions, setting rotation orders, converting to quaternions and padding. The pre-processing of audio data is to clip it into the corresponding length.

Then we extract features of audio and motion data respectively: get short-time Fourier transform (STFT) values X_t as audio features and get quaternion target values as motion features.

Next, to train the model, audio features X_t is passed to our three-layer LSTM network and we can get an output M_t. Then the mean-square error (MSE) can be computed by using M_t and target values. Our goal is to minimize the cost function, MSE.

In the last part, we need to change our result back to Euler angle to build a BVH file. Then we can implement visualizing motion by using MakeHuman and Blender.

Chapter 4

Experiments

In this chapter, we provide more details about the data acquisition, pre-processing, feature extraction, training and performance visualization in Blender. We list the problems when processing the data and the methods we choose to deal with them.

4.1 Data Acquisition

Motion capture data uses 2D data to represent motions in the 3D world. There are various motion capture data formats encoding gestures and motions, e.g. C3D file format, a public binary file format developed in the mid 1980s; FBX file format, represented as either binary or ASCII data and owned by Autodesk company; MVNX file format, a human-readable open XML based format; and VMD file format, used to store animations for models in the MikuMikuDance (Polygon Movie Maker) [4].

In our project, we use the Biovision Hierarchy (BVH for short) file format, which is currently one of the most popular motion data formats and has been widely used by the animation community due to its simple specifications [4]. Its name derives from its company and characteristic. It was originally developed by Biovision, a motion capture services company, to provide motion capture data with a hierarchical skeleton structure to their customers. Now it has become a standard representation of movements in the animation of human-like structures. From Fig 4.1, we can observe the clear hierarchy inside the BVH file.

A BVH file mainly contains two parts: a header section (Fig 4.2), which describes the initial pose of a skeleton in a hierarchical structure; and a motion section (Fig 4.3), which contains a time-framed sequence of the pose data by using Euler rotations

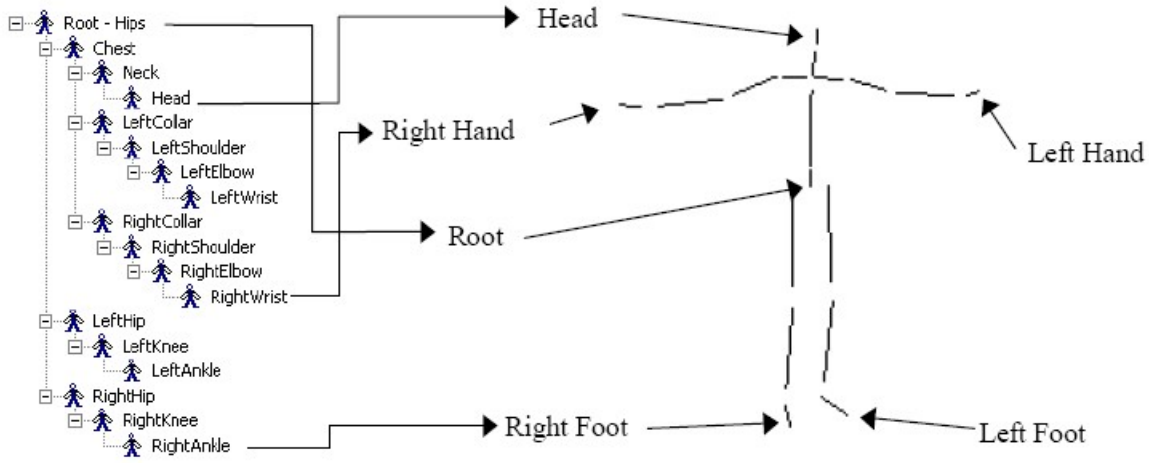


Figure 4.1: Hierarchy in BVH file [17]

in certain rotation order.

For example, as shown in Fig 4.2, after the keyword HIERARCHY we need to define the ROOT Hips as the basic position of the human [1], then all the other joints are defined in a children-parent structure by using relative position offsets,e.g. Right Hand, Left Hand, and Head.

There are three difficulties in acquiring a suitable dataset. First, although there are various motion capture datasets available online, it is hard to find BVH motion capture data for dance with associated music. For example, CMU capture datasets only provide BVH files without music; there are many dance-motion files on Miku-MikuDance application, but it uses its own motion file named VMD, which has a different format and can only be used on that application platform. Therefore, although the dance with corresponding music is necessary to train our model, a suitable motion capture dataset is difficult to obtain.

Second, various BVH files have different skeleton structures, because each BVH file can define its own hierarchical skeleton structure and joint numbers. For example, CMU datasets provide BVH files with 32 joints while BVH files in the SFU Motion Capture Database only use 25 joints. Therefore, unifying the structure of BVH files is a necessary step if we use the BVH files from different sources.

Third, rotation orders are different in BVH files from different sources. Rotation order is an important part of BVH files, which can influence the position of each joint. BVH standard format uses an unusual rotation order: it goes Z rotation, followed by the X rotation and finally the Y rotation, in this context, the rotation matrix of

```
HIERARCHY
ROOT Hips
        {
                OFFSET 0.00 0.00 0.00
                CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
                JOINT Chest
                {
                        OFFSET 0.000000 6.275751 0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT Neck
                        {
                                OFFSET 0.000000 14.296947 0.000000
                                CHANNELS 3 Zrotation Xrotation Yrotation
                                JOINT Head
                                {
                                        OFFSET 0.000000 2.637461 0.000000
                                        CHANNELS 3 Zrotation Xrotation Yrotation
                                        End Site
                                        {
                                                OFFSET 0.000000 4.499004 0.000000
                                        }
                                }
                        }
```

Figure 4.2: Header part in BVH file [17]

one join can be calculated as $R = YXZ$. Therefore, if the order changes then the way to calculate the rotation matrix and corresponding placements would be totally different. However, many BVH files online ignore the standard orders and set the order in a random way.

In this situation, we need to pre-process the skeleton data and music data to establish a small desirable dataset. In this project, we use the dataset from the SFU Motion Capture Database, because it has a uniform BVH format and also it provides several videos with dance motions and music. Following the provided demo on that website, we find the matching music by ourselves. Then we clip the BVH file and the music into the same length. In this way, we can get one-to-one music-motion pairs to build the dataset.

4.2 Data Pre-processing and Feature Extraction

4.2.1 Motion Capture Data

In this project, first, we clip the BVH file into a certain length to match the corresponding music. Then we need to unify skeleton structure, i.e. the number of joints in BVH files, because BVH files from different sources may have different numbers of

```
MOTION
Frames: 2
Frame Time: 0.04166667
-9.533684       4.447926        -0.566564       -7.757381       -1.735414       89.207932       9.763572
                6.289016        -1.825344       -6.106647       3.973667        -3.706973       -6.474916
                -14.391472      -3.461282       -16.504230      3.973544        -3.805107       22.204674
                2.533497        -28.283911      -6.862538       6.191492        4.448771        -16.292816
                2.951538        -3.418231       7.634442        11.325822       5.149696        -23.069189
                -18.352753      15.051558       -7.514462       8.397663        2.953842        -7.213992
                2.494318        -1.543435       2.970936        -25.086460      -4.195537       -1.752307
                7.093068        -1.507532       -2.633332       3.858087        0.256802        7.892136
                12.803010       -28.692566      2.151862        -9.164188       8.006427        -5.641034
                -12.596124      4.366460
-8.489557       4.285263        -0.621559       -8.244940       -1.784412       90.041962       8.849357
                5.557910        -1.926571       -5.487280       4.119726        -4.714622       -5.790586
                -15.218462      -3.167648       -15.823254      3.871795        -4.378940       22.399654
                2.244878        -29.421873      -6.918557       6.131992        4.521327        -18.013180
                3.059388        -3.768287       8.079588        10.124812       5.808083        -22.417845
                -15.736264      18.827469       -8.070700       9.689109        2.417364        -7.600582
                2.505005        -1.625679       2.430162        -27.579708      -3.852241       -1.830524
                12.520144       -1.653632       -2.688550       4.545600        0.296320        8.031574
                13.837914       -28.922058      2.077955        -9.176716       7.166249        -5.170825
```

Figure 4.3: Motion part in BVH file [17]

joints. For example, if we do not need toes in our files, then we need to delete corresponding data in the hierarchical part and rotation columns in motion part. This step can be done in an application called Bvhacker easily.

Next, we re-sample the data to set it as 26 fps, which means a certain setting in Frame and Frame Time in BVH files. Because we only focus on the pose of the body and do not consider the position shift, then we need to center the motion data by changing the offset of Xposition, YPosition and Zposition under the ROOT joint.

Then we need to unify the rotation order in ZXY order. In BVH files, the standard format of the rotation order is special: it goes Z-X-Y rotation. It means the rotation matrix of one joint is calculated as $R = Y \times X \times Z$ and the new position of one joint can be presented as $vR = vYXZ$ where v is an original position. Due to the non-commutativity of matrix multiplication, the same data with different rotation order can give rise to totally different positions. Therefore, we need to unify the rotation order in Blender. First, we import the BVH files into the Blender, and set the Z-axis points up and Y-axis as forward, because Blender uses a right-handed coordinate system, which is opposite to the coordinate system in BVH files. Then, we can export a Z-X-Y order BVH file.

Next, as we mentioned above, BVH files use Euler rotations to represent the position of each joint. Euler angles are three angles used to describe the rotation of a rigid body with respect to a fixed coordinate system [2]. However, it may give rise to Gimbal lock sometimes, especially when we generate new Euler rotations. To avoid Gimbal lock problem, our strategy is to change the rotation data into the quaternion version, and then use the quaternion data to train our model. After generating the

motions, we need to change the quaternion back to Euler rotations in order to match the format in BVH files.

In mathematics, the quaternions are a number system that extends the complex numbers [6]. Generally, quaternions are represented in the form: $x\ \mathbf{i} + y\ \mathbf{j} + z\ \mathbf{k} + w$, where x,y,z, and w are real numbers, and i, j, and k are the fundamental quaternion units [6]. Unit quaternions (also known as rotation quaternions) is usually used to represent 3D rotation. In this project, we use the transformation formula given by NASA in 1977 (Fig 4.4) and the corresponding codes are as follows (Fig 4.5).

$$(9)\quad M = M(Z(\theta_1),\ X(\theta_2),\ Y(\theta_3)) = ZXY$$

Axis Rotation Sequence: 3, 1, 2

$$M = \begin{bmatrix} -\sin\theta_1\sin\theta_2\sin\theta_3 + \cos\theta_1\cos\theta_3 & -\sin\theta_1\cos\theta_2 & \sin\theta_1\sin\theta_2\cos\theta_3 + \cos\theta_1\sin\theta_3 \\ \cos\theta_1\sin\theta_2\sin\theta_3 + \sin\theta_1\cos\theta_3 & \cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2\cos\theta_3 + \sin\theta_1\sin\theta_3 \\ -\cos\theta_2\sin\theta_3 & \sin\theta_2 & \cos\theta_2\cos\theta_3 \end{bmatrix}$$

$$q_1 = -\sin\tfrac{1}{2}\theta_1\sin\tfrac{1}{2}\theta_2\sin\tfrac{1}{2}\theta_3 + \cos\tfrac{1}{2}\theta_1\cos\tfrac{1}{2}\theta_2\cos\tfrac{1}{2}\theta_3$$
$$q_2 = -\sin\tfrac{1}{2}\theta_1\sin\tfrac{1}{2}\theta_3\cos\tfrac{1}{2}\theta_2 + \sin\tfrac{1}{2}\theta_2\cos\tfrac{1}{2}\theta_1\cos\tfrac{1}{2}\theta_3$$
$$q_3 = +\sin\tfrac{1}{2}\theta_1\sin\tfrac{1}{2}\theta_2\cos\tfrac{1}{2}\theta_3 + \sin\tfrac{1}{2}\theta_3\cos\tfrac{1}{2}\theta_1\cos\tfrac{1}{2}\theta_2$$
$$q_4 = +\sin\tfrac{1}{2}\theta_1\cos\tfrac{1}{2}\theta_2\cos\tfrac{1}{2}\theta_3 + \sin\tfrac{1}{2}\theta_2\sin\tfrac{1}{2}\theta_3\cos\tfrac{1}{2}\theta_1$$

$$\theta_1 = \tan^{-1}\left(\frac{-m_{12}}{m_{22}}\right)$$

$$\theta_2 = \tan^{-1}\left(\frac{m_{32}}{\sqrt{1-m_{32}^2}}\right)$$

$$\theta_3 = \tan^{-1}\left(\frac{-m_{31}}{m_{33}}\right)$$

Figure 4.4: Transformation formulas between rotation matrices, Euler angles and quaternions[13]

```
def to_quat(self, alpha, beta, gamma, params):
    # alpha = x, beta = y, gamma = z
    if params['from_deg'] == True:
        alpha = deg2rad(alpha/2)
        beta = deg2rad(beta/2)
        gamma = deg2rad(gamma/2)

    ca = math.cos(alpha)
    cb = math.cos(beta)
    cg = math.cos(gamma)
    sa = math.sin(alpha)
    sb = math.sin(beta)
    sg = math.sin(gamma)

    q0 = -sg*sa*sb + cg*ca*cb
    q1 = -sg*sb*ca + sa*cg*cb
    q2 = sg*sa*cb + sb*cg*ca
    q3 = sg*ca*cb + sa*sb*cg

    self.rotmat = [q1,q2,q3,q0]
```

Figure 4.5: Codes to implement transformation formula

Finally, we need to do padding before we transfer the data into our neural network. Although LSTM in TensorFlow can handle variant sequences inside the network, it only accepts input data with a certain (maximal) length of data. Therefore padding is necessary to deal with this situation when data has different frame numbers in training and testing. The way of padding is to fulfill zero numbers in empty positions. In this way, we can get clean input data for our neural network.

In conclusion, the pre-processing part for the motion capture data consists of clipping it into a certain length, unifying skeleton structure, setting frames number and frame time, centering the motions, setting rotation orders, extracting quaternions and padding (as shown in Fig4.6). As a result, the motion data contained 25 joints and each has a quaternion rotation vector in a format as (x, y, z, w). The feature of dance motions is extracted from BVH files as quaternion rotations and the input format is (1,1299,100) where 1299 is frame number and 100 represents the number of all the quaternions.

4.2.2 Music Data

In our project, the music piece is clipped into a certain length to correspond to the dance motions. We use short-time Fourier transform (STFT) as the feature of the

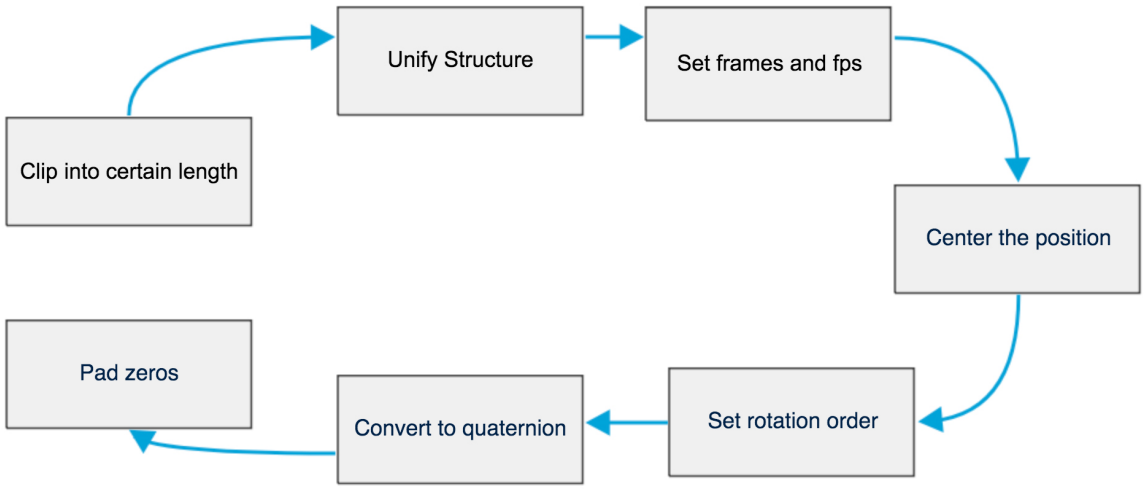


Figure 4.6: FlowChart for pre-processing

music because Mel-spectrogram is more focused on the tone and pitch and we do not want that.

Short-time Fourier transform (STFT) is a transformation of Fourier transform, also called windowed Fourier transform or time-dependent Fourier transform. Basically, the way to compute STFT is to divide a long signal into several short equal-length signals, and then compute Fourier transform separately on each short signals segment. Compared with Fourier transform, STFT can be used in a time-frequency joint analysis, because it not only gets frequency features but also provides time information. [8].

We re-sample each audio at 16000 Hz and then extract power features from these files. To synchronize the audio with the motion in 26 fps, we divide the audio into small slices with the same frame numbers and then use STFT to get features in each slice.

For example, one BVH file contains 1299 frames and each frame is 38 ms (26 fps), then we correspondingly divide the audio into 38 ms segments. Each segment is computed by STFT with a window size 300 and hop length of 150, after getting the magnitude of it, this 2D result is flatted into 1D data. This 1D result represents the feature of one frame. After stacking all the frames, the input of our model is a (1, 1299, 755) dimensional file, where 1299 is the number of frames and 755 is STFT result in each frame (38ms).

Parameter Name	Parameter Value
Shape of input vector	(1,1299,755)
Shape of output vector	(1,1299,100)
Learning rate	0.006
Hidden units	300
Dropout	0.6
Numbers of LSTM layer	3
Numbers of hidden layer	2
Training Time (hours)	10

Table 4.1: Parameters and training statistics of LSTM

4.3 Training and Saving

In our project, we use TensorFlow to build a three-layer LSTM model. TensorFlow is an open-source machine learning library. The reason we choose TensorFlow rather than Keras is TensorFlow has higher accuracy and training speed. Keras runs on the top of the TensorFlow library, thus it still needs time to recall the functions in TensorFlow.

One thing to notice is that in TensorFlow, the input data and the ground truth data that we need to feed into LSTM should be three-dimension data. The format of them is (B, F, D), where B presents batch numbers, F is frame numbers (i.e. the length of our time-series data) and D is the dimension of the features in each frame.

The input can be batches of audio features, but in our project, we only create one batch of training data. As explained in the previous section, the dimension of audio features is (1,1299,755), where 1 is the batch number; 1299 is the maximal frame number (time steps); 755 is the dimension of SFTF results in each frame. The format in ground true motion data is (1,1299,100) because it has 25 joints and each has one quaternion rotation vector.

As shown in Fig 4.8, our model is built with 2 hidden layers with 3 LSTM layers, the number of hidden units is 300 and the learning rate is 0.006 (as shown in Table 4.1). Before passing the data to the LSTM cells, we add an input hidden layer for input to cells using $W \times X + b$. By adding a hidden layer, we are able to decrease the dimension of the audio features in a rough way. Then we can pass the input data to the MultiRNNCell containing three LSTM cells. Following the LSTM cells, there is an output hidden layer for output as the final results. In this way, we can get a motion prediction with a shape of (1,1299,100). Finally, we use Adam optimizer to

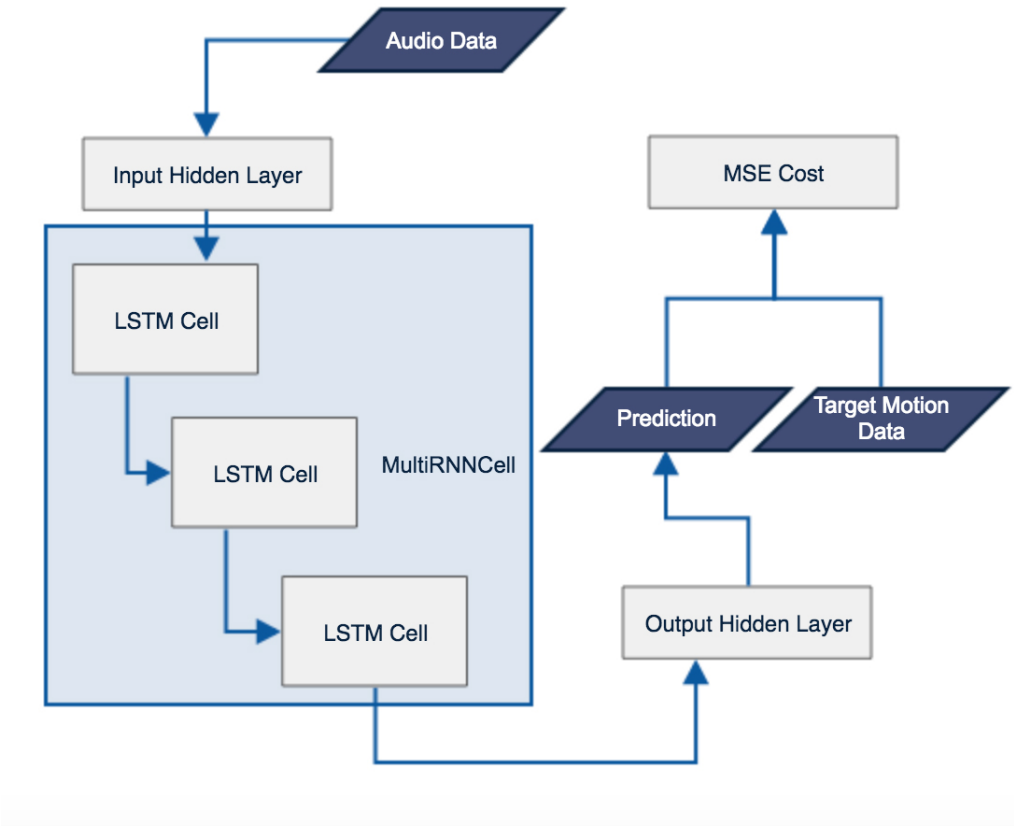


Figure 4.7: Training model flowchart

minimize the MSE cost between the prediction and original ground true motion data.

```
saver = tf.train.Saver()
save_path = saver.save(sess, model_path)
```

Figure 4.8: Save the model

After training, we save the model by using tf.train.Saver Class, which can save the graph and variables in the model. Also, we can use tf.add_to_collection to save the tensor. Then next time we can continue to train the model with other data or test the model directly.

4.4 Creating Avatar and Performing in Blender

In order to get a BVH file, we change our quaternion results to Euler angles and add a header for it. Now we can use a BVH player to show the animation of our

result. In our project, we use BVHplay, which is a free, lightweight, Python-based and open-source BVH player.

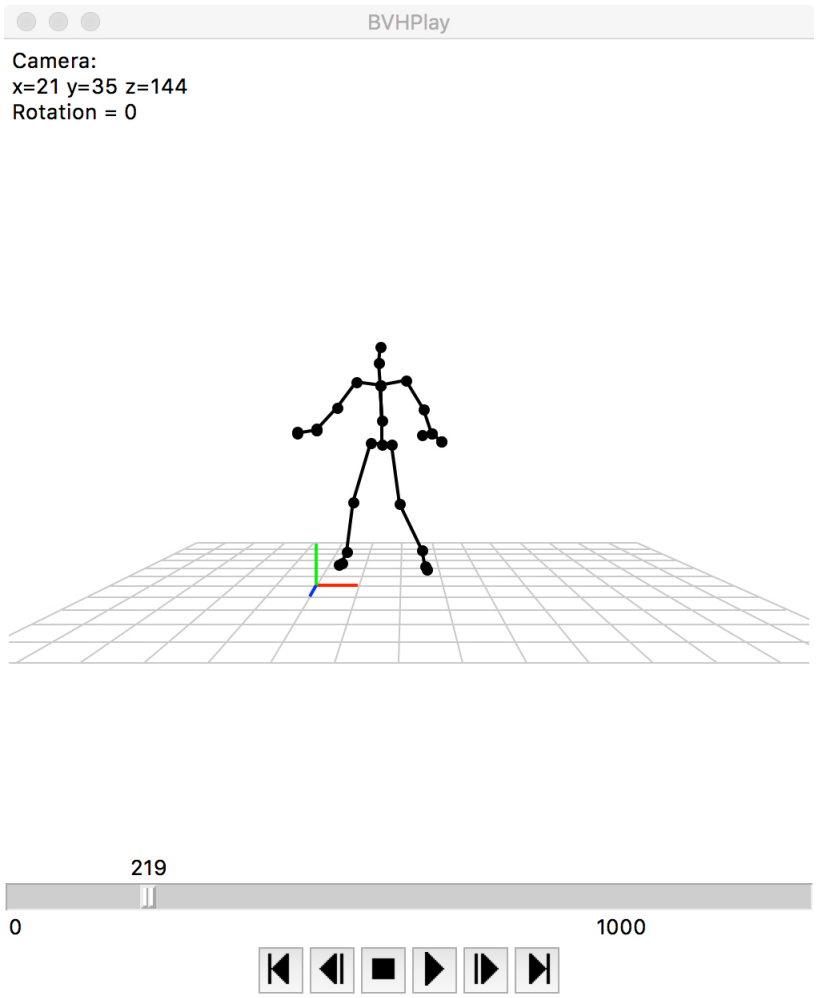


Figure 4.9: BVH file in BVHplay player

However, the BVHplay can only show the movements of a skeleton (as shown in Fig 4.9). To show a human-shaped dancer performing our results, we use MakeHuman to create avatar model and combine it with BVH file in Blender. MakeHuman is open-source 3D modeling software designed for the prototyping of realistic humanoids [5] and Blender is a free and open-source 3D computer graphics software toolset. The interaction between these two applications is well designed and MakeHuman models can be easily imported into Blender by using Blender side plugins.

In our project, first we create a human model with 31 bones in MakeHuman, and then import it into the Blender and map our BVH files into it. In Blender, avatars

joints are able to match our importing BVH file automatically, and move according to our BVH file. In order to combine the music and our dance motion, we also use Video Sequence Editor (VSE) to add the music into the motion in Blender. The screenshots of the video are shown in Fig 4.10.

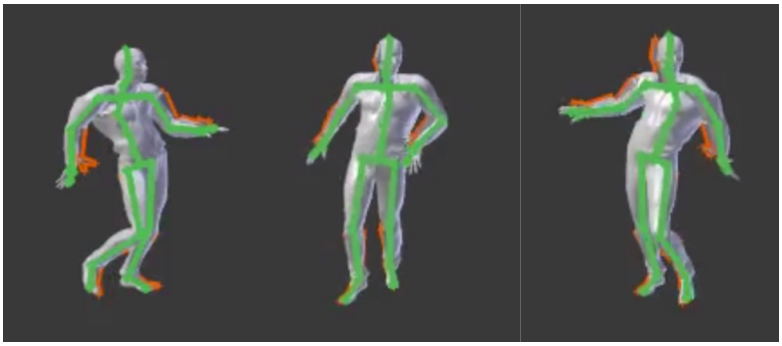


Figure 4.10: Original BVH file in Blender

Chapter 5

Results

5.1 Testing with Trained Music

After training 10 hours with our small dataset, the MSE cost reaches a very low value: 0.001. We choose the cost values in the last 5000 training rounds and build a line chart (as shown in Fig 5.1). From the chart we can figure that when training our model, the cost value decreases dramatically at first, then it drops slowly but still maintains a small value in the cost function.

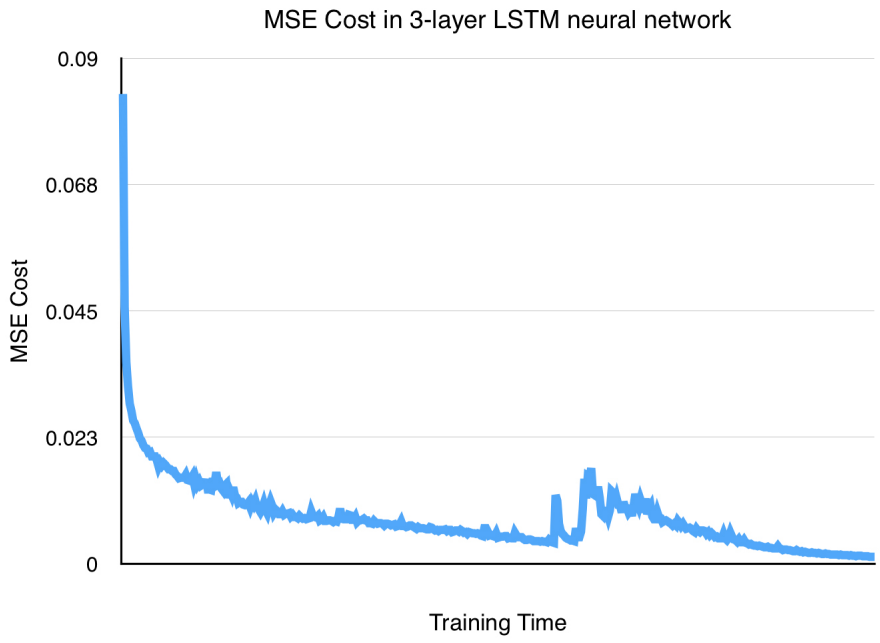


Figure 5.1: MSE cost in last 5000 rounds

```
if __name__ == '__main__':
    sess = tf.Session()
    new_saver = tf.train.import_meta_graph('model/test2.meta')
    new_saver.restore(sess, tf.train.latest_checkpoint('model/'))
    graph = tf.get_default_graph()

    predict = tf.get_collection('predict')[0]
    xs = graph.get_operation_by_name("xs").outputs[0]
    seqlen = graph.get_operation_by_name("seqlen").outputs[0]
    ys = graph.get_operation_by_name("ys").outputs[0]
    average_cost = graph.get_tensor_by_name("average_cost:0")
```

Figure 5.2: Relevant codes on importing the model

Before testing the model, we import the meta graph and the latest checkpoint to restore the whole model network. The relevant codes are shown as Fig 5.2. Then we use get_operation_by_name function to handle the feed values; use get_collection and get_tensor_by_name functions to get tensors. In this way, we can reuse our model directly without building the whole graph again.

Then we can get the result as follows (Fig 5.3). We do the testing ten times, the MSE is pretty low and all of them are around 0.0001.

```
==Test==
The shape of input audio is:  (1299, 755)
The shape of output motion is:  (1299, 100)
The SSE is:  17.91287
The MSE is:  0.00013789721

Process finished with exit code 0
```

```
==Test==
The shape of input audio is:  (1299, 755)
The shape of output motion is:  (1299, 100)
The SSE is:  17.904373
The MSE is:  0.00013783171

Process finished with exit code 0
```

Figure 5.3: Test outputs

As we explained in the previous section, in order to show the results in animation, we add a header to the result and get a BVH file. We can play the BVH file both in the Blender and BVHplay. Our output result performs similar to the original motion file, but the motion is not fluent enough, the avatar would shake sometimes. Several screenshots are shown as follows (Fig 5.4).

Figure 5.4: Animation in Blender

5.2 Testing with Slow Down and Speed Up Music

To generate the slow and fast version of original music, we use librosa.effects.time_stretch function to generate a new song and then extract its features.

If we set the fast music 1.2 times faster than the original one, then we can get the feature in format (1083, 755). Correspondingly, we can get an output motion with 1083 frames, which should have the same frames numbers as the audio. The result in

```
==Test FAST==
The shape of input audio is:  (1083, 755)
The shape of output motion is:  (1083, 100)
The MSE is:  0.0136408685

Process finished with exit code 0
```

Figure 5.5: Console output of speed up music

the console is shown in Fig 5.5. From the console, we can find that the MSE is pretty low, which means the motion in the fast version music is similar to the motion in the original one.

Also, motions are shown in the BVHplay application. At the same frame, the motions are almost the same but still have small differences (as shown in Fig 5.6).

The results are similar in the slow version of music. From the console output (Fig 5.7) we can see the MSE value is very small but still there are differences exits. The differences at the same frame time can be observed in BVHplay as shown in Fig 5.8.

However, the ideal result should be a slow version or fast version of the movement sequence. The possible reasons may as follows.

1. Overfitting. Because there is not enough training data and too much training

Figure 5.6: Motions of fast speed music (top) vs. normal speed music (bottom) in BVHplay at the same frame time

on one dataset leads to overfitting. The network can generate the original dance very well but cannot 'jump out of' it. Although we add a 'dropout' layer, it does not help a lot in this project.

2. LSTM network remembers the sequence of the motions. In this three-layers LSTM network, the previous states can influence the output. Also due to overfitting, the LSTM network remembers the sequence too well and the music features can only influence the result in a limit range.

```
==Test SLOW==
The shape of input audio is:  (2599, 755)
The shape of output motion is:  (1299, 100)
The SSE is:  1862.2244
The MSE is:  0.014335796

Process finished with exit code 0
```

Figure 5.7: Console output of slow down music

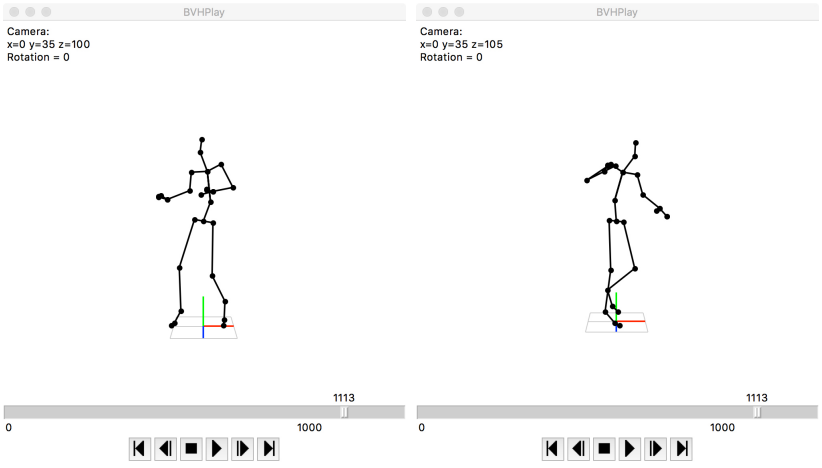


Figure 5.8: Motions of slow speed music (left) vs. normal speed music (right) in BVHplay at the same frame time

Chapter 6

Conclusions and Future Work

In this project, we propose a multi-layer LSTM network to find the relationship between the motion and the music. First, we acquire the motion-music pairs and clip them into the same length as a dataset. In the pre-processing part, we need to pre-process music and motion data respectively: resample the music piece in 16000Hz; in the motion data, we need to unify skeleton structures, re-sample the data to set its frame number and frame time, centering and unify the rotation order. In our project, we compute the abstract magnificent of STFT as a musical feature; and use quaternion sequences as motion feature. Then we pass these two features into our three-layer LSTM network and use Adam optimizer to minimize the MSE cost function, during which, we also use padding to process variable length sequence and dropout layer to decrease outfitting. Next, we can generate new dance motion data and make it into a BVH file. Finally, we use Bvhacker to play BVH file and also use MakeHuman and Blender to create avatar performing the new dance.

The results show that our model is able to generate dance motions based on music, and the cost can be very low when using trained music. However, outfitting is a big problem in our model due to the limit size of the dataset. Also, because the dimensions of the features are too big, the training speed is very slow.

In future work, first, we need to enlarge our dataset and set proper dropout parameters to avoid overfitting. Also, we can add velocity features to represent the movements on the ground, i.e. use 3D velocities as features to represent Xposition, Yposition, and Zposition of ROOT in the BVH file. From the performance aspect, the model with seq2seq and attention mechanism is suggested in some papers, which can extract and compact the features better in the encoder section and consider previous results in the decoder part. To speed up our model, we can use a convolutional

neural network (short for CNN) to decrease the dimension of the features better and use auto-encoder to compact data information. We would also like to explore training our model using Compute Canada to speed up training time and facilitate experimentation.

Bibliography

[1] Biovision bvh. https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html. Accessed April 6, 2019.

[2] Euler angles. https://en.wikipedia.org/wiki/Euler_angles. Accessed April 6, 2019.

[3] history of dance. https://en.wikipedia.org/wiki/History_of_dance.

[4] List of motion and gesture file formats. https://en.wikipedia.org/wiki/List_of_motion_and_gesture_file_formats_C3D_file_format. Accessed April 6, 2019.

[5] Makehuman. https://en.wikipedia.org/wiki/MakeHuman.

[6] Quaternion. https://en.wikipedia.org/wiki/Quaternion. Accessed April 6, 2019.

[7] Recurrent neural networks tutorial, part 1introduction to rnns. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. Accessed April 6, 2019.

[8] Short-time fourier transform. https://en.wikipedia.org/wiki/Short-time_Fourier_transform. Accessed April 6, 2019.

[9] Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed April 6, 2019.

[10] Omid Alemi, Jules Françoise, and Philippe Pasquier. Groovenet: Real-time music-driven dance movement generation using artificial neural networks. *networks*, 8(17):26, 2017.

[11] Luka Crnkovic-Friis and Louise Crnkovic-Friis. Generative choreography using deep learning. *arXiv preprint arXiv:1605.06921*, 2016.

[12] Anastasia H Dalziell, Richard A Peters, Andrew Cockburn, Alexandra D Dorland, Alex C Maisey, and Robert D Magrath. Dance choreography is coordinated with song repertoire in a complex avian display. *Current Biology*, 23(12):1132–1135, 2013.

[13] DM Henderson. Euler angles, quaternions, and transformation matrices for space shuttle analysis. 1977.

[14] PVV Kishore, KVV Kumar, E Kiran Kumar, ASCS Sastry, M Teja Kiran, D Anil Kumar, and MVD Prasad. Indian classical dance action identification and classification with convolutional neural networks. *Advances in Multimedia*, 2018, 2018.

[15] Juheon Lee, Seohyun Kim, and Kyogu Lee. Listen to dance: Music-driven choreography generation using autoregressive encoder-decoder network. *CoRR*, abs/1811.00818, 2018.

[16] Minho Lee, Kyogu Lee, and Jaeheung Park. Music similarity-based approach to generating dance motion sequence. *Multimedia tools and applications*, 62(3):895–912, 2013.

[17] Maddock Meredith, Steve Maddock, et al. Motion capture file formats explained. *Department of Computer Science, University of Sheffield*, 211:241–244, 2001.

[18] Ferda Ofli, Engin Erzin, Yücel Yemez, and A Murat Tekalp. Learn2dance: Learning statistical music-to-dance mappings for choreography synthesis. *IEEE Transactions on Multimedia*, 14(3):747–759, 2012.

[19] Takaaki Shiratori, Atsushi Nakazawa, and Katsushi Ikeuchi. Synthesizing dance performance using musical and motion features. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3654–3659. IEEE, 2006.

[20] Taoran Tang, Jia Jia, and Hanyang Mao. Dance with melody: An lstm-autoencoder approach to music-oriented dance synthesis. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 1598–1606. ACM, 2018.

[21] Yi Wang, Zhi-Qiang Liu, and Li-Zhu Zhou. Learning hierarchical non-parametric hidden markov model of human motion. In *2005 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3315–3320. IEEE, 2005.

[22] Nelson Yalta. Sequential deep learning for dancing motion generation. 11 2016.

[23] Nelson Yalta, Shinji Watanabe, Kazuhiro Nakadai, and Tetsuya Ogata. Weakly supervised deep recurrent neural networks for basic dance step generation. *arXiv preprint arXiv:1807.01126*, 2018.