Generation of genre-based dance sequences using Deep Neural Networks

Anjoe Anand Jacob, B.Tech

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Augmented and Virtual Reality)

Supervisor: Rachel McDonnell

August 2019

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Anjoe Anand Jacob

August 15, 2019

Permission to Lend and/or Copy

Acknowledgments

I would like to thank my supervisor, Rachel McDonnell for providing valuable suggestions and feedback during this research. I would also like to extend my gratitude to my friends and family for their consistent support throughout this term.

Anjoe Anand Jacob

University of Dublin, Trinity College
August 2019

Generation of genre-based dance sequences using Deep Neural Networks

Anjoe Anand Jacob, Master of Science in Computer Science

University of Dublin, Trinity College, 2019

Supervisor: Rachel McDonnell

This research project designs and implements a deep neural network that can generate dance movements based on the genre of input music. The model is given as input, STFT (Short-time Fourier Transform) of the audio signal and generates dance poses corresponding to it. The network uses a series of fully connected layers to process the audio and a series of LSTM (Long-short Term Memory) layers to predict dance poses. We also introduce a custom loss function based on Laban Movement Analysis, a theory to describe human movements like dance that penalizes the network when the energy of the generated movement does not match the ground truth. The network is able to generate long sequences of dance motions for pieces of music it was exposed to during the training. When using the custom Laban loss function, the network shows a possibility for improvement in the quality of result.

Contents

List of Figures

Chapter 1

Introduction

Bringing art and technology together garnered much interest in the last century across the world. It takes the form of art installations, interactive experiences, light-and-music shows, or creative software. Ever since technology is accessible at our fingertips through computers and mobile phones, research and innovation in the field has grown exponentially. With the advent of powerful and cost-effective processors and Graphics Processing Units (GPU) and availability of enormous amounts of data, Neural Networks and Deep Learning have gained significant popularity in terms of finding solutions to seemingly straightforward but complicated problems like image classification and recognition, text analytics, speech analysis. Although Artificial Intelligence has not yet reached the sensory capabilities of a human, it certainly surpasses human abilities in specific skills at a far higher speed and accuracy.

The human brain synthesizes information from past experiences. These experiences can come from various sensory inputs - vision, sound, touch, hearing, and taste. The mind combines its various simple historical inputs and comes up with novel conclusions [7]. A dancer's mind works no differently when coming up with new dance movements. A dancer listens to music, processes it in multi-dimensional mind space, and translates it into movements using his/her entire body. This exercise manifesting as dance, expresses emotions, and communicates to an audience.

Several researchers have worked on synthesizing dance motions from music in the past. However, there is comparatively very less work done using Deep Learning based methods. The traditional methods focus on generating plausible combinations of sequences by mixing and matching from motion databases, while deep learning would let a neural network learn by itself from the data to generate dance. From the general notion that technology doesn't create arts, humans do [8], this thesis explores the possibility of how technology can create arts by itself.

1.1 Overview

This research proposes a deep learning architecture to generate dance motion from an external signal, music. Chapter 2 provides a background of the work done on synthesizing animations from external signals using traditional and deep learning-based methods. Chapter 3 discusses the data used by the network, its formats, and the sources. We explain the design of the proposed architecture in detail with the reasons for specific design choices in chapter 4. Chapter 5 discusses implementing the discussed design. The results are visualized, discussed, and analyzed in chapter 6. Finally, chapter 7 concludes the thesis summarizing the findings before discussing future work.

1.2 Motivation

Traditional animations based on keyframes or motion capture would be static and mostly for specific use cases. It limits the creative potential for non-specific needs. It is a time-consuming process to define animation patterns manually. Contemporary architectures and technical advances in the field of Deep Learning and Animation could help us find a way out of this farrago. Neural Networks have outperformed a lot of traditional best-methods in the recent past. This fact stands as the fundamental motivation behind selecting the Neural Network route to generate dance movements. This research can throw some light into the possibilities of creating consumer applications that can generate novel dance animations for various purposes like motion graphics, TV ads, mobile games, or animated music videos.

1.3 Objective

In this research study, we aim to design and implement a deep neural network that can generate dance movements based on music input, and answer the following research questions:

- Can a deep neural network generate dance sequences according to the genre of the music given as input?
- Can the network synthesize novel meaningful dance motion?

Chapter 2

Background and Related Work

This chapter begins with a review of the literature in synthesizing animations from external signals, mainly music. It is followed by exploring work done on genre-based music classification and extraction of features from music. The chapter continues with techniques for animation synthesis in general and moves on to Neural Network based motion synthesis. Subsequently, Deep Learning is introduced briefly before moving on to the emotional, physical, and technical side of dance.

2.1 Animation synthesis from external signals

Synthesizing novel human motions based on external input signals like character trajectory [9], obstacles [10], speech [11], motion of other characters [12] is a well-studied area. Lucas Kover et al. [9] propose Motion Graphs that generate transitions between motion capture data and synthesize motion based on custom constraints. One of the main applications of Motion Graphs is interactive control of character locomotion along a defined path. It enables the animation of characters to be more realistic in digital environments, especially video games. In [13], a system is introduced to make robots dance to music. Using FFT (Fast Fourier Transform) and peak to peak time interval of low-frequency sounds that stands for beats or rhythm in an audio signal is used to select appropriate motion sequences.

Minhi Lee et al. [14] proposed an approach to generate dance sequence based on music similarity. The method involves creating a key-value pair dictionary of segmented music and motion. The music is stored as handcrafted acoustic features. On receiving a new musical input, it segments the audio signal and match the segments with the appropriate rows in the database. It works on the idea that similar musical pieces can have similar dance moves. Extraction of acoustic features and extraction of motion features like velocity, acceleration and directional change of motion help the system to create clusters of similar motions.

In [15], Shiratori T. et al. propose a system to generate dance synchronized to music. The system extracts motion and music features based on motion keyframes, motion intensity, music beats, chord progressions, and music intensity. It is followed by the construction of a motion graph to find out analogous poses from dance sequences to create transitions between matching motion. The system synthesizes dance steps by traversing the motion graph based on the correlation between motion and music features.

In an interesting work done by F Bevilacqua et al. [16], motion capture data is used to generate music. The proposed work creates a virtual 3D environment where the recorded motion is played back along with generated music. The authors broadly classify the parameters from the movement as Position/Angle/Distance, and Velocity/Acceleration. While the former can be used to deconstruct footsteps and arm movements, the latter can be used to find out movement qualities like "sustained" or "percussive". The motion information is transformed into MIDI parameters or signal processing control parameters. These parameters correspond to musical notes.

C Panagiotakis st al. [17] propose an approach to generate dance sequences synchronized to music in a three-step process. Firstly, the dance motion data is segmented temporally into spans of dance. Next, the audio signals are segmented based on the tempo. Lastly, it generates beat-synchronous motion by realigning motion spans from step one with the tempo of the music.

In general, the approaches mentioned above extract when the idea is to match specific musical features to motion features. We discuss more about this in section 4.8.

2.2 Music feature extraction

Typically, music analysis is done by extracting feature vectors from audio signals. It takes its inspiration majorly from speech recognition. [18] uses a Hidden Markov Model and cepstral coefficients to classify audio signals as "music", "laughter" and "speech". Wold et al. [19] use statistical features like mean, variance, and autocorrelation from the short time features of an audio signal. [20] introduces beat-IDs comprising of two components: average beat period length and the signal's energy distribution during the beat period. [21] analyzes tempo and extracts beats using bandpass filters and parallel-comb filters. This analysis can also predict the occurrence of beats in the future. [22] proposes a real-time beat tracking system that analyzes music with or without drums and extracts the beat structure using onset, chord progressions, and drum loops.

Dance driven by music needs several units to work together. Many researchers have worked on the various sub-fields related to this. Antonio Jose HomsiGoulart et al. [23] propose various approaches to classify music genre consisting of a pipeline that has a feature extraction stage connected to a classifier architecture. G. Tzanetakis et al. [24] propose statistical pattern recognition classifiers that use three feature sets for timbral texture, rhythmic content, and pitch content. Concerning structural music analysis, Yongwei Zhu et al. [25] use Short-time Fourier Transform (STFT) to extract the beats and other features from the audio which is followed by a conversion into 9-octave frequency bands. They calculate the musical energy of each band by summing up STFT's spectral coefficients. These extracted features are then used to identify motion sequences and synchronize motion with music. Gao and Lee [26] extract musical information like tempo using MAP estimation (Maximum a posteriori).

Since music belonging to the same genre possess similarities in rhythm, time signatures, structure and pitch distribution, audio signals belonging to the same genre share similar characteristics [27]. Tao Li et al. [28] conducted a study on music genre classification and proposed a system for automatic classification - Daubechies Wavelet Coefficient Histograms (DWCH). While traditional content-based feature extraction focused on Timbral Textural Features, Rhythmic Content Features, and Pitch Content Features, DWCHs compute histograms on the music signal's Daubechies wavelet coefficients.

In the comparison study [29], STFT spectrograms performed the best in extracting information to classify environmental sounds when used in the context of deep neural networks. Omid Alemi et al. [30] extract handpicked audio signal information manually and use them as inputs to their networks. While this looks like a precise process to extract information from a sound signal, this gives the network less information for audio analysis when compared to a larger feature set like STFT. This factor contributes to designing the audio feature set in the implementation of this thesis (section4.8).

2.3 Motion Capture

Motion Capture is the technique of recording movements with a real actor and mapping them to a 3-dimensional space. The recorded joint angles and position can then be mapped to any arbitrary 3D character and can be digitally animated. It is used heavily in the Film and Video Game industries. Typically, multiple cameras facing the center acting area are fit all around the room in a motion capture room. Human actors wear a black jumpsuit with special reflective markers attached to every bone joint. The information from the cameras placed around the actor is then combined to triangulate the position or angles of each joint in 3-dimensional space.

Nearly 100 years ago, in 1915, cartoonist Max Fleischer invented the rotoscope device [31] to aid the production of cartoon films. The device projects live-action films onto a light table one frame at a time. The cartoonist then traces the projected image onto paper. It is then stitched together to make animated cartoon films. Several decades later, in the early 1960s, Lee Harrison III created the first motion capture bodysuit [32] fitted with potentiometers and animated 3D characters on a CRT monitor in real-time. Several short films were made during the period using this technology. Eventually, Harrison won an Emmy award for his technical contributions.

In the 1980s, Ginsberg and Maxwell created an animation capture system [32] that uses flashing LEDs attached to an actor surrounded by a set of cameras. The cameras triangulate the 3-D position of the LEDs in real-time. Through the 90s and early 2000s, the technology matured into a full-fledged production-ready system for use in the entertainment industry.

2.4 Neural Network Motion Synthesis

2.4.1 A brief history of Deep Neural Networks

With the advent of advanced and powerful CPUs and GPUs, Neural Networks based learning has come a long way from the time McCulloch and Pitts proposed creating a Turing machine by combining neurons [33]. In 1958, Rosenblatt F. introduced the perceptron [34], a hypothetical nervous system that created a probabilistic model of information management in the brain. Nearly 30 years later in 1985, the backpropagation algorithm was proposed by Geoffrey Hinton et al. [35]. Neocognitron for visual pattern recognition [36] and Convolutional Neural Networks with backpropagation were introduced in the next 13 years by Geoffrey Hinton and Yann LeCun respectively. When the number of layers increases or goes deep in a Neural Network, we call it a Deep Neural Network. In 2006, GE Hinton et al. [37] introduced a fast algorithm using complementary priors that can teach Deep Belief Network layers, and also reduced the data dimensionality in Neural Networks [38] thereby solving the Deep Neural Networks learning problem.

For the generation of dance motion, we use a type of Neural Network called Recurrent Neural Network, which is explained in detail in section 4.1.

2.4.2 The state of Deep Learning Motion Synthesis

The current wave of Deep Learning started when Alex Krizhevesky et al. proposed AlexNet [39] that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It boosted the interest of researchers around the world to solve problems using Deep Learning based methods.

Daniel Holden et al. [40] proposed a Deep Learning framework for synthesizing character motion from high-level parameters. The model trains on a large consolidated dataset from CMU, HDM05 (Muller et al. 2007 [41]), and Berkeley MHAD (Ofli et al. 2013 [42]) using an autoencoder. A feedforward neural network is stacked on top of the trained autoencoder to learn the mapping between user input and motion data. The network applies constraints to the generated motion in the form of custom loss functions. The network is penalized using, position, bone length, and trajectory constraints. The total cost is calculated by summing the individual costs over the temporal domain. It is followed by gradient descent to minimize the cost function. Zimo Li et al. in [4], generate long sequences of motion data using a newly introduced training methodology, auto-conditioned LSTM (ac-LSTM). Vanilla RNN based motion generation suffer from noise accumulation over a period of time, and the generated motion comes to a standstill after a few seconds. ac-LSTM uses the network's output at periodic intervals and feeds it back to the network during training. The model can generate 18000 frames without getting frozen. Some of the problems are occasional jerky motion, self-collision of the skeletal bones and foot sliding. We discuss this in detail in the design section 4.2.

Omid Alemi et al. [30] captured synchronized music and motion sequences for training Factored Conditional Restricted Boltzmann Machines and LSTM-based Artificial Neural Networks. The audio features are extracted into an 84-dimensional vector, whereas the raw motion capture's Euler angle representation is converted to exponential maps. Subsequently, they replace the root joint's global position with its velocity along the axis perpendicular to the floor. The resulting vector is 52-dimensional. The system learned to perform well with training data with infrequent jerkiness. However, when untrained music is used to generate dance, the model doesn't generalize well, it creates extremely unstable sequences and unrealistic poses.

In [43], Tang T et al. use an LSTM-autoencoder model to learn the mapping between music and motion. A custom dataset comprising of 4 genres of dance along with the matching music is used to train the model. Acoustic features like MFCC, Constant-Q chromagram, etc. are extracted by hand and converted to a 16-dimensional vector. Tempo or beat information is extracted separately and given directly as input the network. The motion features are down-sampled from 41 joints to 21 joints. The loss function used to backpropagate is Euclidean distance loss, which is deemed insufficient to calculate the correctness of motion.

D Pavllo et al. [44] propose QuaterNet that uses quaternions instead of Euler angles to train the network. Using joint angles or joint positions can result in the accumulation of error in the generated motion and can also result in bone stretching. This approach overcomes both limitations. The loss function also calculates the absolute position error after performing forward kinematics on the skeleton with predicted and ground truth data.

Nelson Yalta et al. [45] propose a weakly supervised RNN to generate dance steps. The authors pre-process the audio signals to extract the power spectrum (STFT). They pass it through an Encoder-Decoder configuration, in which the encoder contains convolutional layers that process the STFT of the audio signal. We discuss more about this approach later in section 4.8, The dimensionality-reduced output from the encoder is concatenated with motion frame quaternions and fed into the decoder network.

Finally, a contrastive loss function [46] is used to align motion with music beat.

2.5 The art and science of dance

Dance is a performing art form where performers express themselves using body movements. It's a form of non-verbal communication [47][48]. Typically, the dancer synchronizes their body movements with music during the performance. In some cases, music acts as a cue to perform the movement with a particular level strength, speed or flow whereas in other cases, it acts as a window of time wherein a sequence of similar movements takes place. Dance forms around the world are different and characterized by specific styles or genres. The treatment of space, time, and kinetic elements is an integral part of a dance performance [48][49].

While dance is a multi-dimensional thought process manifesting as body movements, science often deals with exploring reasons for invisible or unknown phenomena. Technical analysis of dance is best possible when these two dimensions of human activity overlap [50]. It helps to formalize movements and define parameters for studying dance better.

2.5.1 Laban Movement Analysis

Rudolph Laban, in the 1920s, came up with a dance notation system for analyzing movements. It is a method for describing, visualizing, interpreting, and documenting human movement [51]. Laban's belief that the body is an instrument for expression lead to its development. LMA draws its inspiration mainly from anatomy, kinesiology, and psychology. This was further expanded by his student Irmgard Bartneiff [52] by broadly classifying it into four categories [53] [54].

- Body: This describes the structural characteristics of the body in motion like simultaneous, successive, sequential movement, etc. of the body parts.
- Effort: This describes the overall quality of the movement and the energy used by the dancer. It further categorizes the energy of the dancer in four subsections, namely Flow, Time, Weight, and Space.

- Shape: This looks at how the human body changes its shape and the elements that drive the change.
- Space: This describes the interaction of the performer with the surroundings. It throws light into the personal performance sphere within which most of the movements take place.

This is discussed in detail in section 4.6.

Chapter 3

Data Collection

This chapter starts with a detailed explanation of the motion file format used for the implementation. It is followed by details about the sources of mocap and audio data. The chapter also discusses the absence of rhythmic synchronicity between motion and music due to the non-complimentary nature of the data sources. Lastly, we discuss techniques for extracting motion and music data from video files and its usability.

3.1 BioVision Hierarchy (BVH)

This thesis uses BVH (Bio Vision Hierarchy) file format to represent motion.

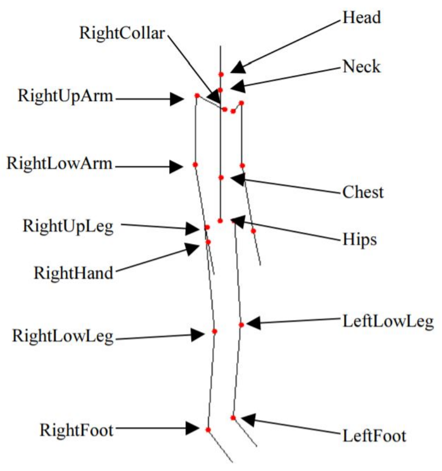
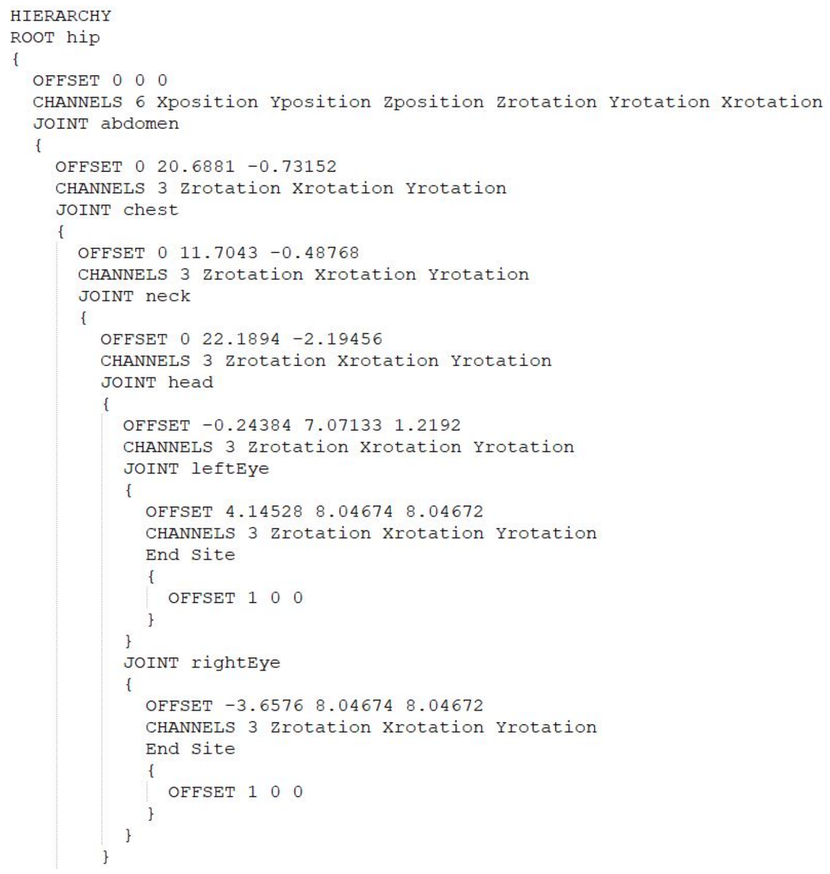


Figure 3.1: Skeleton structure of a sample BVH file. Image from [1]

The motion capture company called Bio Vision originally developed the BVH file format and its specifications. We chose this format because of its simplicity and straightforwardness to read the skeleton and joint angle information. A BVH file consists of two sections [1] [55]. The header section contains the skeleton joint hierarchy and the initial pose. The keyword HIERARCHY marks the beginning of the section. It is followed by the root joint keyword ROOT, which is generally the hip joint in case of a human skeleton. The hip/root is then cascaded with hierarchical joints with keyword

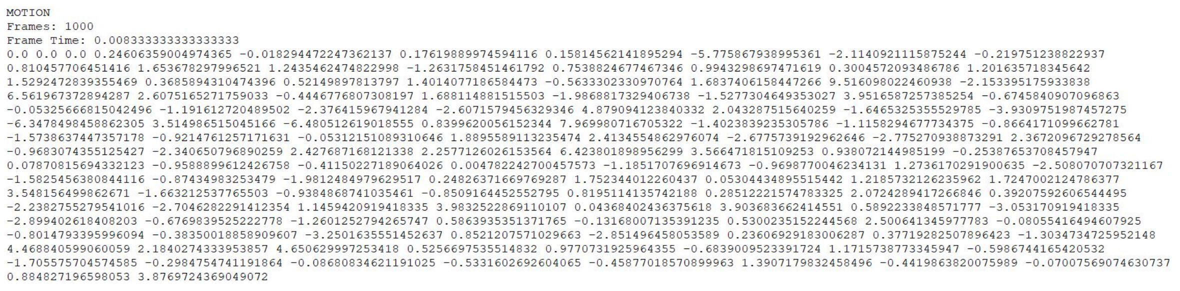


```
HIERARCHY
ROOT hip
{
  OFFSET 0 0 0
  CHANNELS 6 Xposition Yposition Zposition Zrotation Yrotation Xrotation
  JOINT abdomen
  {
    OFFSET 0 20.6881 -0.73152
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT chest
    {
      OFFSET 0 11.7043 -0.48768
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT neck
      {
        OFFSET 0 22.1894 -2.19456
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT head
        {
          OFFSET -0.24384 7.07133 1.2192
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT leftEye
          {
            OFFSET 4.14528 8.04674 8.04672
            CHANNELS 3 Zrotation Xrotation Yrotation
            End Site
            {
              OFFSET 1 0 0
            }
          }
          JOINT rightEye
          {
            OFFSET -3.6576 8.04674 8.04672
            CHANNELS 3 Zrotation Xrotation Yrotation
            End Site
            {
              OFFSET 1 0 0
            }
          }
        }
```

Figure 3.2: Header section of a BVH file

JOINT. Each joint, including the root joint, has an offset which is the translation of the bone's origin corresponding to its parent's origin. The keyword OFFSET represents this. The next line contains CHANNELS that represents the Degree of Freedom. The order of the axes is important because the motion data for a particular joint follows this structure. Different joints may have different orders. It should be accounted for while parsing the Euler angle data from the motion section. The keyword "End Site" marks the end-effectors in a joint chain.

The second section of the BVH file defines the actual motion data for each of the joint described above in the header section. The keyword MOTION marks the beginning of this section, which is followed by "Frames", defining the total number of frames in the BVH. The next line specifies the duration of each frame using the keyword "Frame Time", followed by a floating-point value.

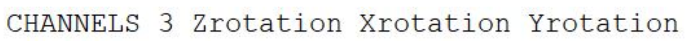


```
MOTION
Frames: 1000
Frame Time: 0.008333333333333333
0.0 0.0 0.0 0.24606359004974365 -0.018294472247362137 0.17619889974594116 0.15814562141895294 -5.775867938995361 -2.1140921115875244 -0.219751238822937
0.810457706451416 1.653678297996521 1.2435462474822998 -1.2631758451461792 0.7538824677467346 0.9943298697471619 0.3004572093486786 1.201635718345642
1.5292472839355469 0.3685894310474396 0.52149897813797 1.4014077186584473 -0.5633302330970764 1.6837406158447266 9.516098022460938 -2.153395175933838
6.561967372894287 2.6075165271759033 -0.4446776807308197 1.688114881515503 -1.9868817329406738 -1.5277304649353027 3.9516587257385254 -0.6745840907096863
-0.05325666815042496 -1.191612720489502 -2.376415967941284 -2.6071579456329346 4.879094123840332 2.043287515640259 -1.646532535529785 -3.9309751987457275
-6.3478498458862305 3.514986515045166 -6.480512619018555 0.8399620056152344 7.969980716705322 -1.4023839235305786 -1.1158294677734375 -0.8664171099662781
-1.5738637447357178 -0.9214761257171631 -0.05312151089310646 1.8895589113235474 2.4134554862976074 -2.6775739192962646 -2.775270938873291 2.3672096729278564
-0.9683074355125427 -2.340650796890259 2.427687168121338 2.2577126026153564 6.423801898956299 3.566471815109253 0.938072144985199 -0.25387653708457947
0.07870815694332123 -0.9588899612426758 -0.41150227189064026 0.004782242700457573 -1.1851707696914673 -0.9698770046234131 1.2736170291900635 -2.508070707321167
-1.5825456380844116 -0.87434983253479 -1.9812484979629517 0.24826371669769287 1.752344012260437 0.05304434895515442 1.2185732126235962 1.7247002124786377
3.548156499862671 -1.663212537765503 -0.9384868741035461 -0.8509164452552795 0.8195114135742188 0.28512221574783325 2.0724289417266846 0.39207592606544495
-2.2382755279541016 -2.7046282291412354 1.1459420919418335 3.9832522869110107 0.04368402436375618 3.903683662414551 0.5892233848571777 -3.053170919418335
-2.899402618408203 -0.6769839525222778 -1.2601252794265747 0.5863935351371765 -0.13168007135391235 0.5300235152244568 2.500641345977783 -0.08055416494607925
-0.8014793395996094 -0.38350018858909607 -3.2501635551452637 0.8521207571029663 -2.851496458053589 0.23606929183006287 0.37719282507896423 -1.3034734725952148
4.468840599060059 2.1840274333953857 4.650629997253418 0.5256697535514832 0.9770731925964355 -0.6839009523391724 1.1715738773345947 -0.5986744165420532
-1.705575704574585 -0.2984754741191864 -0.08680834621191025 -0.5331602692604065 -0.45877018570899963 1.3907179832458496 -0.4419863820075989 -0.07007569074630737
0.884827196598053 3.8769724369049072
```

Figure 3.3: Motion section of a BVH file

The rest of the file defines the root position and rotation information for each bone concatenated in the order defined in the header. Each line corresponds to a single frame in the chronological order. The first three space-separated values correspond to the root joint's position in the order mentioned in the header section. The rest of the line contains space-separated Euler angles for each joint as described in the header.

Using the data available, we can construct the rotation matrix, R by multiplying the rotation matrices for each axis. For a channel definition like below,



```
CHANNELS 3 Zrotation Xrotation Yrotation
```

The rotation matrix, R is calculated as

$$R = R_z R_x R_y \tag{3.1}$$

R_z, R_x, R_y are the rotation matrices of Z, X, and Y axes respectively.

Once the rotation matrix is calculated, using a homogeneous coordinate system, the transformation matrix can be calculated by aligning the translation values along the 4th column as below,

$$
M = \begin{bmatrix} R & R & R & T_x \\ R & R & R & T_y \\ R & R & R & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}
$$

T_x, T_y, T_z are the bone's base position added to the frame translation data. All BVH files used in this implementation contain 43 skeletal joints.

3.2 CMU Mocap database

The Carnegie Mellon University Graphics Lab (CMU Graphics Lab) Motion Capture database is an exhaustive collection of human motion data [56]. The database contains five types of motion - Human Interaction, Interaction with Environment, Locomotion, Physical Activities and Sports, Situations, and Scenarios. The subcategory called Dance under Physical Activities and Sports contains the dance BVH files. Two genres of dances used for the thesis are Indian classical dance and Latin dance.

CMU Mocap lab uses 12 Vicon MX-40 cameras with a maximum recording capability of 120 Hz at 4 Megapixels. The database contains 16 motion capture clips of Indian classical dance and 30 clips of Salsa dance. All the BVH files used are recorded at 120 fps. The total number of frames in the dataset is 52654.

We investigated other Mocap databases like HDM05 and Simon Fraser University Mocap database. The former contains limited dance motions while the latter has various dance genres, but the quantity is insufficient to train neural networks. We chose CMU Mocap database because of the quantity and quality of the data available.

All BVH files downloaded from CMU Mocap DB use ZXY rotation order.

3.3 Audio acquisition

Since there is no direct music to motion mapping for the CMU dance motion data, we downloaded music corresponding to the genres of the motion from YouTube. The genres chosen for the experiments have a distinctly different type of music. Indian classical music for dance typically generally contains Hindustani percussion instruments like

Tabla, melody instruments like Sitar or Santoor, Carnatic instruments like Mrudangam, violin, and classical vocals [57] [58]. The rhythm can get extremely tricky and technical with non-typical time signatures. Some examples are farodast (14 beats), tala chautal (12 beats), sulfak (10 beats), etc. [59]. Latin music used for Salsa dance, on the other hand, is very different, uses another set of instruments with percussion playing on Congas and Bongo drums, melody on the piano, trumpet or trombone [60].

3.4 Pipeline to generate motion from video

Since online video sharing platforms are a good source of dance videos, we present a workflow to generate BVH files from videos here. Z Cao et al. proposed [2] techniques which culminated as the open-source library OpenPose. It can be used to extract human poses from images.

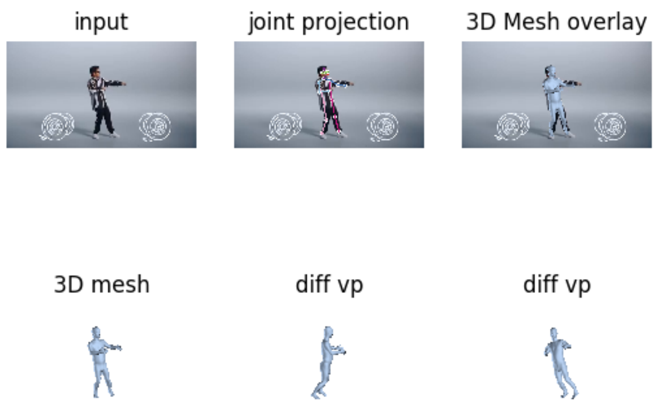


Figure 3.4: Estimated 3D pose using OpenPose [2] and HMR [3]. Video by Bruno Mars, downloaded from YouTube

A video downloaded from YouTube is processed with open source library FFmpeg to generate frame images. We feed these images into OpenPose, which generates CSV files with pose keypoints for each frame. It is then passed into a Human Mesh Recovery

(HMR) [3] pretrained model to generate another set of CSV files that contain the estimated 3D joint positions. Open source 3D creator software Blender [61] is used to read the joint information from the CSV file to export BVH files. A sample blender project is pre-created with a sample BVH file with ten frames loaded. We load the positions from the CSV files and create new frames in the scene, and insert keyframes. Finally, a BVH file is generated using the inbuilt BVH export functionality of Blender [62].

We run this entire pipeline on a Google Colaboratory environment. For a video of 35 seconds with frame size 426x240, it takes 7 hours to generate the BVH file on Google Colab. A square cropped, smaller image size around 150px works best and would reduce the processing time [3]. Although the generated motion is quite impressive, occasional jerkiness makes it necessary to do a manual cleanup of the data.

Since the whole workflow of generating BVH from video and its manual cleaning is a laborious and time-consuming process, we use the CMU dataset to train the Neural Network.

Chapter 4

Design

This chapter details the design choices and the intuition behind each component used. It starts with an overview of Recurrent Neural Networks (RNN) and discusses why it is the right choice for motion generation. Various techniques and configurations used in training to improve the stability, speed, and accuracy of predictions are discussed next. The chapter closes with an overview of the network architecture, making use of all the techniques discussed until then.

4.1 Recurrent Neural Network

Recurrent Neural Network (RNN) is a time-proven Neural Network architecture that works best with sequential data [5]. Typical feedforward neural networks do not have temporal context-specific information. RNNs can process a sequence of inputs on a temporal domain. At every time step in an RNN, the network takes in some information from the previous time step. This way, information from the previous layers gets passed on to future layers. It makes sure that the decision making process at every layer is dependent on its previous layers to some extent. RNNs thrive on the idea of parameter sharing where each input at every time step is a function of the output from the previous timestep.

The motivation behind choosing a Recurrent Neural Network architecture is that animation and music data are sequential and temporally related. When the sequence gets long, vanilla RNNs have the issue of forgetting what it learned prior in the sequence. It has the vanishing gradient problem where the gradient of the loss function decays gradually with time [63] and the earlier layers stop learning. It is not the ideal architecture to generate animation because the sequences can get very lengthy. We use LSTMs which implement a gating mechanism to avoid this problem.

4.1.1 Long Short Term Memory (LSTM) Network

An LSTM network is a type of Recurrent Neural Network that uses a set of gates to regulate the information coming in, going out, getting stored and deleted. These gates are logistic functions of weighted sums. Backpropagation updates the weights in the network during training. A comparison of a basic RNN cell and an LSTM cell is displayed below. The images below are from [64].

Consider a standard RNN that has a tanh function where X_t is the input at time step t and h_t is the hidden state at time t.

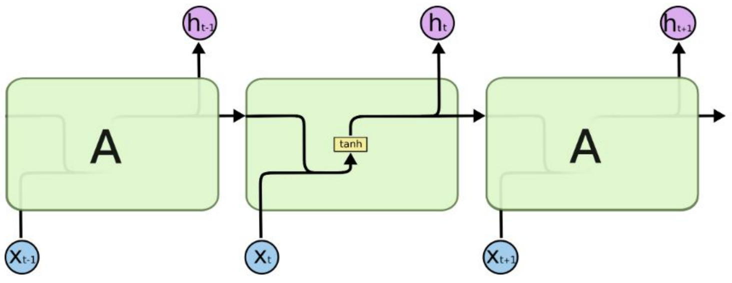


Figure 4.1: A standard RNN with a tanh layer

Instead of the tanh layer, LSTM has a different structure.

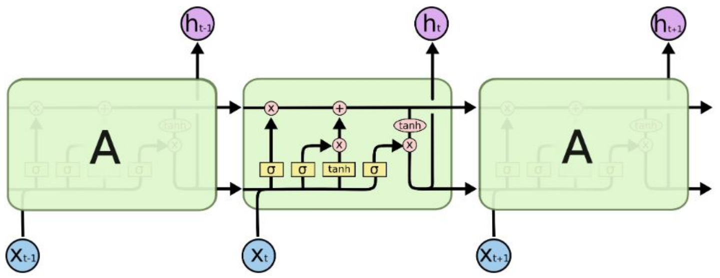


Figure 4.2: An LSTM with four interacting layers

The three gates involved in letting information in and out are forget gate, input gate, and output gate.

- Forget Gate : This gate outputs a number between 0 and 1 (sigmoid layer) which stands for "forget completely" and "keep completely" respectively.

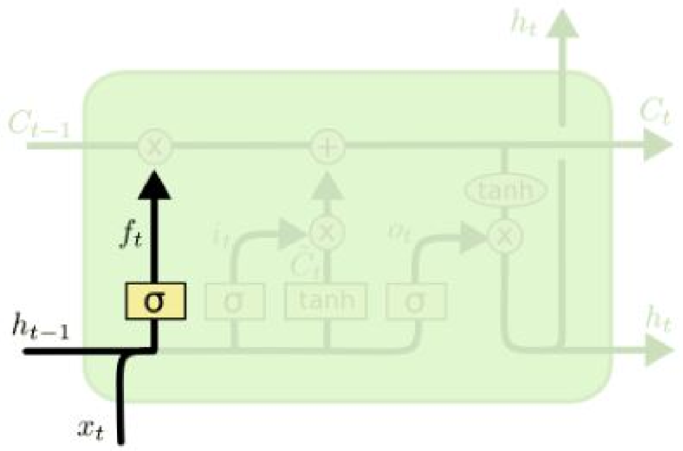


Figure 4.3: A forget gate in an LSTM

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{4.1}$$

- Input Gate : This gate decides what should be stored in the cell state. It consists of two parts which are combined to create the cell state. Firstly, a sigmoid layer to decide which values to update. Secondly, a tanh layer to create a vector, $\tilde{C}_t$ of values to be added to the state.

$$i_t = \sigma(W_f.[h_{t-1}, x_t] + b_i) \tag{4.2}$$

$$\tilde{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{4.3}$$

Now that we have both parts ready, the next step is to update the previous cell state, C_{t-1}. We multiply f_t with the previous cell state to forget, and multiply i_t with $\tilde{C}_t$ to add the new values.

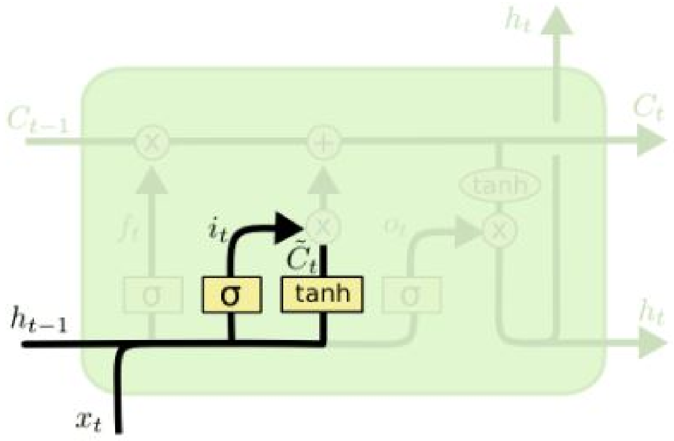


Figure 4.4: The input gate configuration in an LSTM

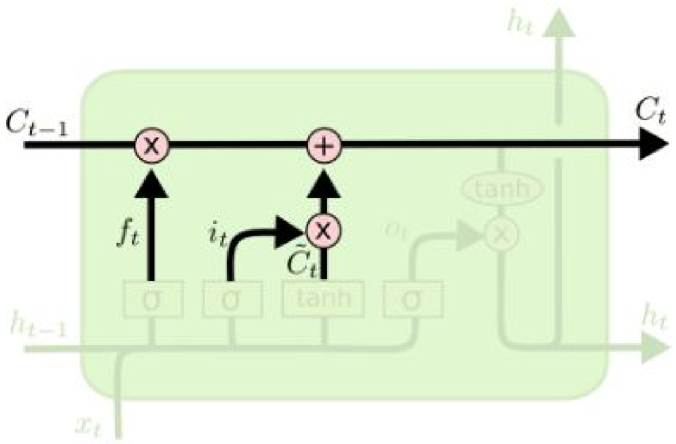


Figure 4.5: Forgetting and adding values in an LSTM

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4.4}$$

- Output Gate : This gate decides the output of the cell. Similar to the input gate, this is also a two-step process. First, we filter the values to be output. Then, we limit the cell state values to be in the range of -1 to 1 by passing it through a tanh gate.

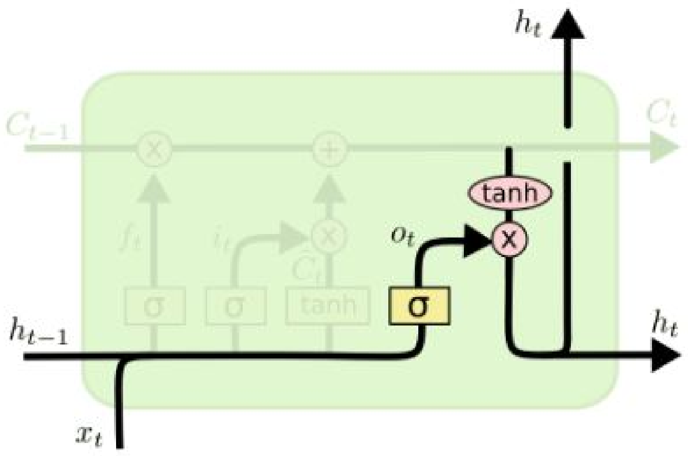


Figure 4.6: The output gate configuration in an LSTM

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{4.5}$$

$$\tilde{h_t} = o_t * tanh(C_t) \tag{4.6}$$

The gates described above make LSTMs remember relevant details from the training sequences and deliver better predictions. Hence, to train on frame sequences like dance motion data, LSTMs are better than Vanilla Recurrent Neural Networks.

4.2 Auto-conditioned configuration

Typical RNN based motion prediction has the issue of error accumulation when predicting long sequences. When a network is recursively trained with a sequence, it learns to predict sequences which are compared with the ground truth to calculate the loss. The backpropagation algorithm updates the model parameters with respect to the input sequences. The network parameters get familiarized with the ground truth, which it does not have access during the inference time.

When solving the problem of motion prediction, the skeleton can likely reach a mean pose and freeze if this is not taken into consideration as mentioned in [4].

An auto-conditioned configuration, as explained in section 2.4.2 makes use of the network's output from the previous timestep in addition to the cell state memory. Because the network is conditioned on its output, it learns to deal with the input better during inference when the ground truth data is not available.

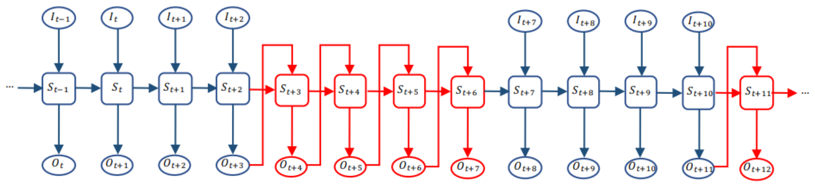


Figure 4.7: An auto-conditioned network. Image from [4]

In Fig 4.7, the network is conditioned with its own output for 4 timesteps (condition length = 4) after time t + 3. After the condition length, at 8th timestep, the LSTM inputs shift back to the original input.

4.3 Dropout

Nitish Srivastava et al. [65] proposed an inexpensive technique called Dropout to train neural networks for preventing the model from overfitting. The idea is to set the outputs of some neurons randomly in the network to zero during training. The thinned network learns to perform well even with missing information or in other words, the dropped neurons.

Consider a standard Neural Network, as shown in fig 4.8.

After applying dropout to random neurons for all three layers, it would look like fig 4.9:

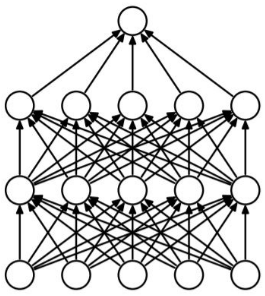


Figure 4.8: A standard Neural Net

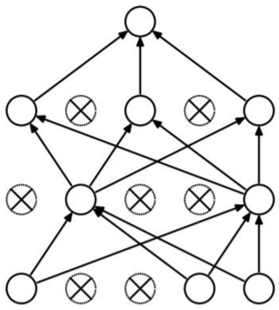


Figure 4.9: After applying dropout

Dropout helps neural networks to generalize better when operating on unseen data. Fig 4.8 and 4.9 are taken from the original paper [65] that introduced Dropout.

4.4 Teacher Forcing

Teacher forcing is a technique used to train RNNs that have connections from their outputs to their hidden states [5]. It injects the ground truth data into the layers at certain timesteps instead of the normal input. This approach reduces the amount of error fed back into the network [66] and speeds up the learning process.

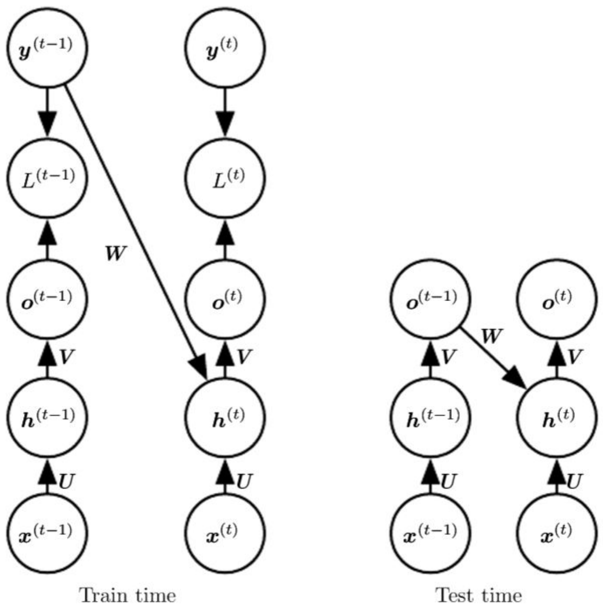


Figure 4.10: Teacher Forcing illustrated [5]

In Fig 4.10, where y stands for ground truth and o stands for network output, during training, the ground truth y^{t-1} from the training set is given as input to the hidden state of the next timestep. During the inference or test time, since there is no access to ground truth, the connection switches back to the output from the previous state, o^{t-1}.

If teacher forcing is used for all of the input sequences, the network is less likely to learn from information in the past because it relies on the ground-truth instead of its hidden-to-hidden state connections. Hence, the design in this thesis uses Teacher Forcing for 30 % of the training time.

4.5 Representing rotations as Quaternions

Irish Mathematician, William Rowan Hamilton first conceived the idea of quaternions, in 1843 [67] as a number system extending complex numbers. Quaternions are represented generally as :

$$a + b\,\mathbf{i} + c\,\mathbf{j} + d\,\mathbf{k} \tag{4.7}$$

a, b, c and d are real numbers and i, j, k are the fundamental quaternion units.

A quaternion can be used to represent any orientation in a 3-dimensional coordinate system [68].

In the context of generating motion using a neural network, using Euler angles can cause gradient explosion and can get tricky to train the model because of its discontinuities and singularities. Hence, taking inspiration from the work of D Pavllo et al. [44], as described in section 2.4.2, we preprocess all skeletal joints in the mocap dataset by converting the Euler angle representation to quaternions. Furthermore, Edward Pervin et al. [69] show that quaternions can streamline derivations in robotics related to rotations over Euler angle representations. A custom loss function as explained in 4.6 should be differentiable inorder to backpropagate. Interested readers are referred to the paper by Diebel J, [70] for information about conversion to and from Euler angles and quaternions respectively.

4.6 Custom loss function based on Laban Effort

A loss function in the context of a Neural Network is a function that evaluates the model's prediction with respect to the input. The general idea is to minimize the cost function and update the weights of the network. The network continues to get trained until the cost reaches a low value that is satisfactory enough for the type of problem it is trying to solve.

Typically, we do a mean squared error difference between the ground truth and the predicted joint angles to calculate the loss. It works well when the network is only expected to perform well within the training set. When a model handles complex multi-dimensional motion data like dance, which has more artistic meaning than mathematical meaning, a loss function that can represent the artistic value in a mathematical form is necessary. This section proposes the design of a custom loss function towards meeting that goal.

As discussed in section 2.5.1, Laban Movement Analysis (LMA) is a technique used to technically observe, describe, and evaluate a physical performance like dance. Out of the four categories that constitute LMA, Laban Effort describes the quality of the movement. Effort is further classified into four - Space, Weight, Time, Flow.

Nakata T et al. in their study conducted in 2001 [71], validated the importance of Laban Movement Analysis using a small robot. The study found out that the excitement of the motion is strongly related to Laban "weight effort". Laban, in his theory, described weight effort as the energy in the movement. Mathematically, it is the linear sum of angular velocities of each skeletal joint. Subsequently, Takaaki Shiratori et al [72] used this concept in the paper "Dancing-to-Music Character Animation" to select motion sequences based on motion intensity which is essentially matching Laban weight effort to the energy of the music.

For an object that has rotated θ degrees in a given time t, the Angular velocity is calculated as:

$$\omega = \theta/t \tag{4.8}$$

When representing rotation as Euler angles, this is straightforward to calculate because we have the angle values for x, y, and z-axis directly. Here, since we are using quaternions instead of Euler angles, it should be dealt slightly differently.

According to the differentiation theorem of quaternions [73] [74], for a unit quaternion function $q(t)$, and angular velocity $\omega(t)$, the derivative of $q(t)$ is determined by:

$$\dot{q} = \frac{1}{2}\omega q \tag{4.9}$$

This can be rewritten as

$$\omega = 2\dot{q}q^{-1} \tag{4.10}$$

For a unit quaternion, the inverse of the quaternion is its conjugate [75] [76]. Hence, the angular velocity can be written as

$$\omega = 2\dot{q}\bar{q} \tag{4.11}$$

The derivative of the quaternion here is the change in rotation over a small time period. In the case of a skeletal joint, it is the change in rotation from frame n to frame n + 1. Given 2 quaternions q_1 and q_2, the rotation that rotates q_1 to q_2 is given by

$$\Delta q = q_2 q_1^{-1} \tag{4.12}$$

From equations 4.10 and 4.11,

$$\Delta q = q_2 \bar{q_1} \tag{4.13}$$

Finally, angular velocity can be rewritten as

$$\omega = 2\Delta q \bar{q} \tag{4.14}$$

This is essentially the crux of Laban Effort parameter. A loss function that penalizes the network based on the energy of the dance would be a bridge between the artistic and mathematical side of dance.

4.7 Variable learning rate and storing model parameters

The parameters of a model can be saved to the file system for inference later. We make use of this feature to continue training from a past epoch by storing the model parameters every 10 epochs. It like having a time machine of training because we can stop and start the training again if the model is diverging after some point. The saved model from a time point before it started diverging can be loaded again and retrained with a different learning rate. This approach saves training time because if the model diverges after for eg. 10 hours, and there is no way to start training from that point, the entire model needs to be retrained from scratch with different hyperparameters.

4.8 Audio features

Inspired from the traditional work done on synthesizing animations from music, We considered two approaches to extract audio information. The first method is to extract a set of handcrafted acoustic features. The second method is to use the audio signal's STFT. In traditional methods, when the user can manually match music and motion features, acoustic features would be of great help. For eg, if the user has to match music intensity with motion intensity, it makes the implementation logical by comparing an extracted acoustic feature "music intensity" with another extracted feature "motion intensity". However, in a neural network, more input parameters mean more representations of the data. However, this is an exception in the case of huge networks like GoogleNet and NIN [77]. Extracting a representation like STFT gives the network more representations to learn.

While CNNs are good to extract the local properties of spectrograms, a series of fully connected layers work best in representing high-dimensional data [78]. It means CNNs are useful when extracting features like onset of beats or chord changes that are spatially not too far apart in a spectrogram. Since the idea is not to focus on local features like beats and more on the global features like genre, we go ahead with the traditional dimensionality-reduction approach using a series of fully connected layers. An additional reason lie in the fact that CNNs are computationally expensive [79].

Because of the distinct difference in the fundamental nature of the genres chosen, the neural network is expected to focus more on the styles of the motions produced and less on the timing of the steps. In the case where the generated dance is expected to match the beats, custom motion data should be recorded synchronized with music, and timing specific labels should be added to the data.

4.9 Training Infrastructure

PyTorch, the deep learning framework from Facebook is used to implement the network. In addition to the simplicity in design, PyTorch is selected over other frameworks like TensorFlow and Keras is that it uses dynamic computation graphs. The computation graph is created on-the-fly during an epoch (one forward pass and one backward pass) and freed up after finishing the epoch. It means that the overall memory usage would be less.

To train the network, one compute machine on Google Cloud with Intel(R) Xeon(R) CPU @ 2.30GHz (4 CPUs) with one Nvidia Tesla T4 GPU is used. Additionally, a Google Colaboratory environment is used to train the model. However, Google Cloud was used mostly because a Google Colaboratory session lasts only for 12 hours, and the environment gets reset.

4.10 Network Design

The design proposed here (Fig 4.11) is divided into 3 parts. Firstly, the audio and dance motion data are pre-processed before passing into the network. The audio files are loaded, resampled to 16 kHz, and the Short-Time Fourier Transform is calculated. Motion files are parsed and joint angles are converted to quaternions. The data is resampled to 30 fps to improve robustness of the model predictions [80]. This process is explained in detail in section 5.1

The proposed network design has fully connected layers before and after the LSTM layers. The fully connected layers before the LSTMs reduce the dimensionality of the audio data(section 4.10), and the one after LSTMs assembles the data corresponding to the 43 skeletal joints and it 4 quaternion values. As mentioned in section 4.3, a dropout is applied to the hidden layer of the first LSTM with a value of 0.3. This means 30 percent of the neurons will be dropped when training and the model learns to predict data with missing information. As discussed in 4.4, Teacher forcing is applied to the second LSTM layer by replacing the hidden state with ground truth motion data.

The predictions after each epoch are passed into the loss functions to calculate the loss values. Subsequently, this is used by the optimizer to backpropagate to update the weights so that the cost is minimized. Mean Squared Error (MSE) and a custom Laban Effort based loss function discussed in section 4.6 are used to optimize the network.

Once the model is trained, the output from the network is then converted from quaternions to Euler angles representation. This is then passed through a Butterworth filter followed by exporting a BVH file.

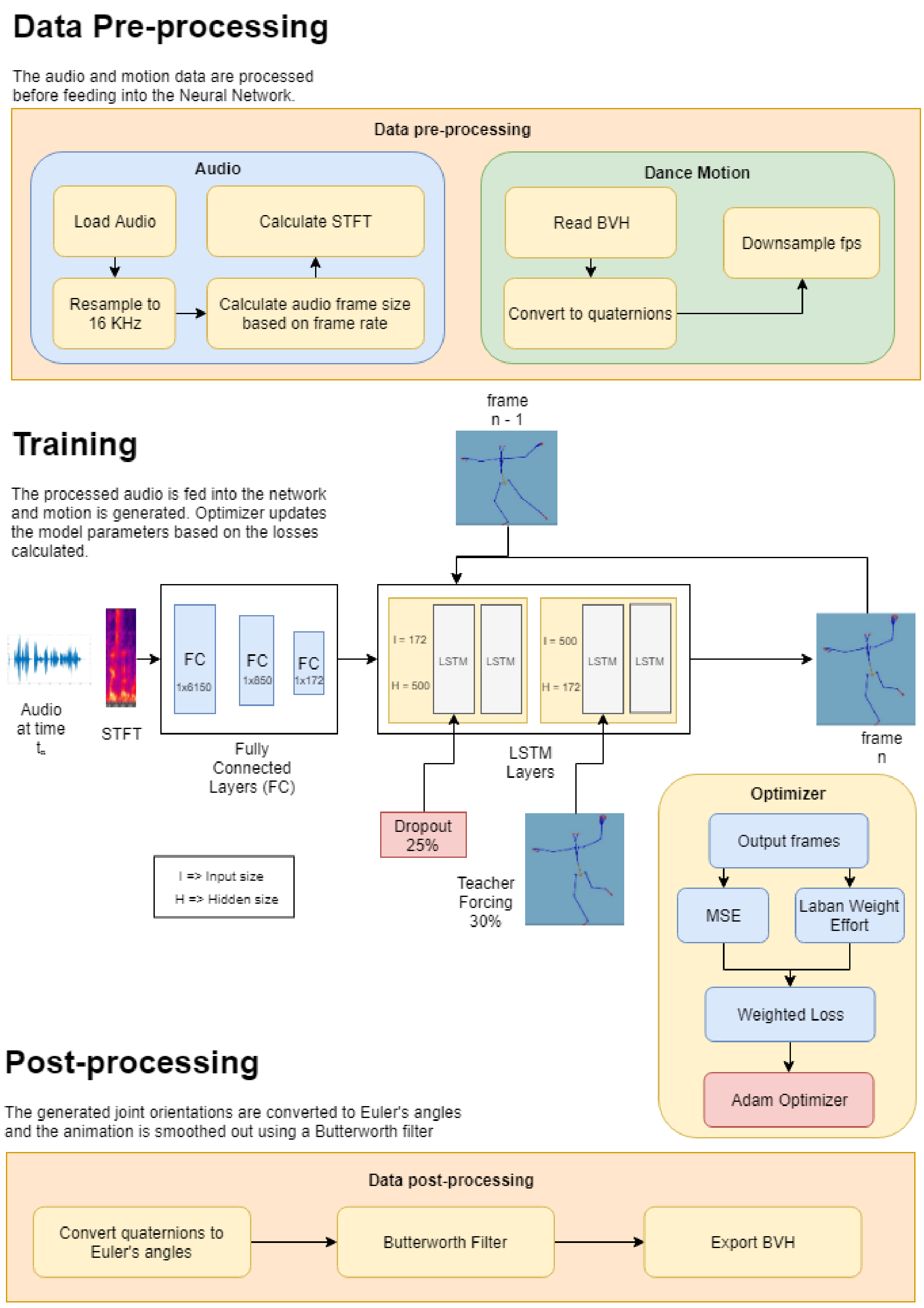


Figure 4.11: Network Design

Chapter 5

Implementation

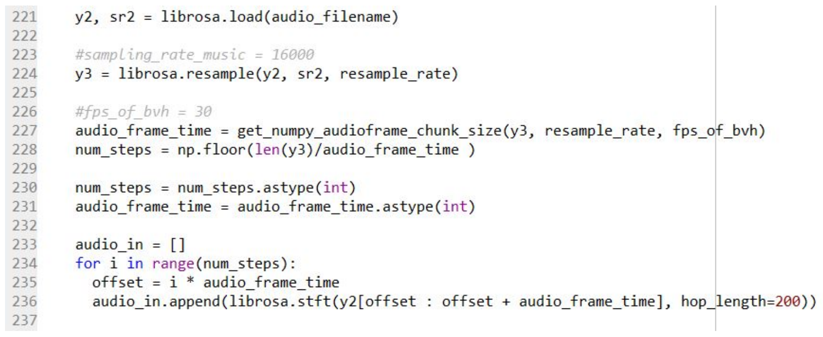
This chapter starts with the implementation details of the pre-processing applied to audio and dance motion data. It is followed by the implementation of the network and its training. Once the network is trained, the output is fed through a post-processing filter. Lastly, a method to monitor the performance of the model is detailed.

5.1 Data preprocessing

5.1.1 Audio

Matching music for the two genres of motion is selected (refer section 3.3) according to the length of BVH motion files. Each audio file is split into lengths according to the selected frame rate. Since the frame rate is 30, each frame duration is 33.33 milliseconds. This would translate to splitting the audio every 33.33 milliseconds to setup the input dataset. As mentioned in section 3.3, since there is no direct mapping for the CMU dataset with audio, we resorted to selecting publicly available music from youtube. The files are first read as a time series of floating-point values. Since the sampling rate of the original audio is 44KHz, we resample it to 16KHz to make the data concise and efficient enough to be passed through multi-layer neural networks. This resampling process retains as much information as possible while greatly reducing the size of data. Python library librosa is used to process the audio files here.

After the audio is split based on the frame rate of the motion capture files, we calculate the STFT (Short Time Fourier Transform) of each of the audio chunk. For



```
y2, sr2 = librosa.load(audio_filename)

#sampling_rate_music = 16000
y3 = librosa.resample(y2, sr2, resample_rate)

#fps_of_bvh = 30
audio_frame_time = get_numpy_audioframe_chunk_size(y3, resample_rate, fps_of_bvh)
num_steps = np.floor(len(y3)/audio_frame_time )

num_steps = num_steps.astype(int)
audio_frame_time = audio_frame_time.astype(int)

audio_in = []
for i in range(num_steps):
  offset = i * audio_frame_time
  audio_in.append(librosa.stft(y2[offset : offset + audio_frame_time], hop_length=200))
```

Figure 5.1: Code to resample audio and conversion to STFT

an audio chunk of 33.33 ms, converted to STFT, the size of the array is 1025x6. This is then converted to a 1-D vector of size 6150 before feeding into the Neural Network.

5.1.2 Motion

The Indian classical and Salsa dance BVH files are read from their corresponding directories. The joint angles are extracted using BVH python libraries used by Holden et al. [81] [82]. As discussed in section 4.5, the next step is to convert Euler's angles to quaternions. The helper libraries from the same dataset are reused to do the conversion.

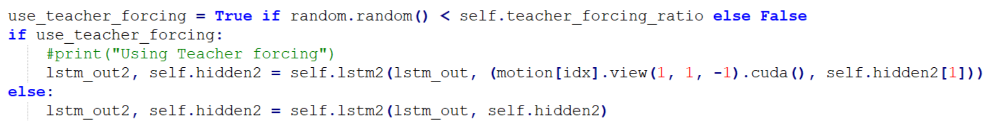
For a BVH file with 5000 frames, 43 joints and Euler's angles, the size of the array would be (5000, 43, 3). This gets changed to (5000, 43, 4) after converting to quaternions where the 3rd dimension represents the four quaternion values.

As discussed in section 4.10, the every 4th frame is extracted from the dataset to down-sample from 120 to 30 fps.

5.2 Training

The music STFT tensor is unrolled from 1025x6 to 1x6150 to pass through the first fully connected layer. The output which is a tensor of size 1x1200 is then passed through the second layer. The output of size 1x600 from the second layer is connected to the third layer. The third layer outputs a tensor of size 1x172. At this point, we have a low dimensional representation of the music segment for which we are trying to generate dance. It is then connected to a 2-layer LSTM with an input size of 172 and an output size of 500. We apply a dropout of 25% to the hidden state of this LSTM.

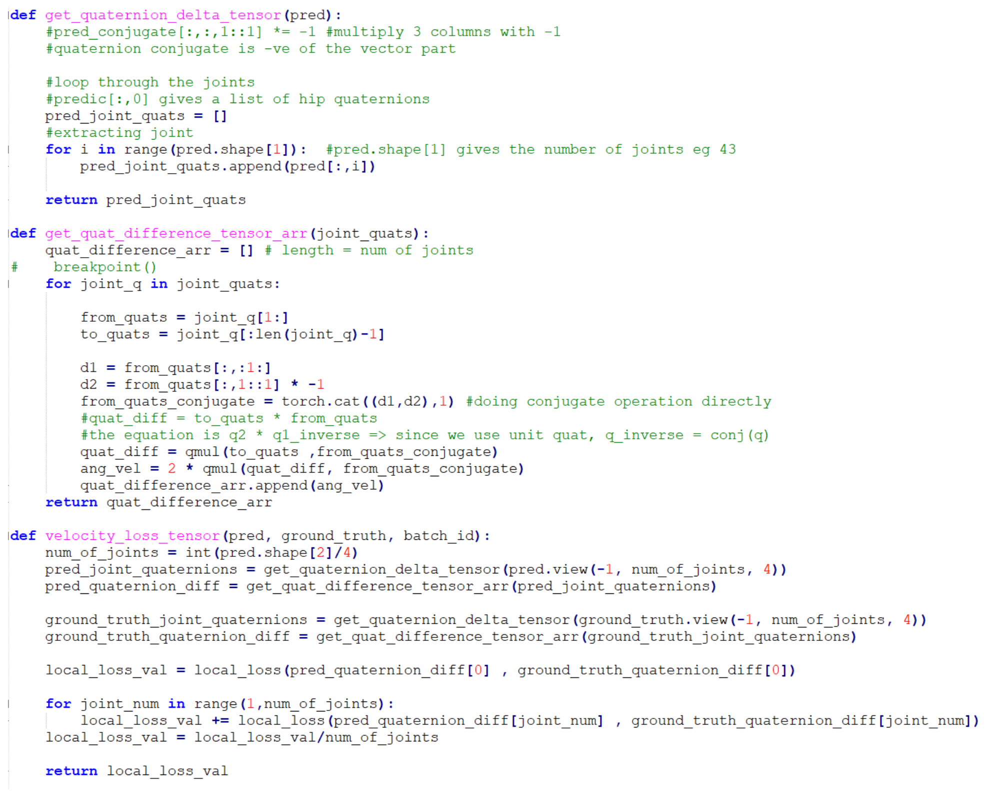
The output of size 1x500 is then passed into a 2-layer LSTM where we apply Teacher Forcing to the hidden layer 4.4 by taking the ground truth and replacing the hidden state based on a probability of 30%. Whenever we are not forcing ground truth data, we feed the outputs back into the hidden states in the next iteration as mentioned in section 4.2.



```
use_teacher_forcing = True if random.random() < self.teacher_forcing_ratio else False
if use_teacher_forcing:
    #print("Using Teacher forcing")
    lstm_out2, self.hidden2 = self.lstm2(lstm_out, (motion[idx].view(1, 1, -1).cuda(), self.hidden2[1]))
else:
    lstm_out2, self.hidden2 = self.lstm2(lstm_out, self.hidden2)
```

Figure 5.2: Teacher Forcing applied inside the forward pass

The differentiable custom loss function *velocity_loss_tensor* is implemented as in fig 5.3



```
def get_quaternion_delta_tensor(pred):
    #pred_conjugate[:,:,1::1] *= -1 #multiply 3 columns with -1
    #quaternion conjugate is -ve of the vector part

    #loop through the joints
    #predic[:,0] gives a list of hip quaternions
    pred_joint_quats = []
    #extracting joint
    for i in range(pred.shape[1]):  #pred.shape[1] gives the number of joints eg 43
        pred_joint_quats.append(pred[:,i])

    return pred_joint_quats

def get_quat_difference_tensor_arr(joint_quats):
    quat_difference_arr = [] # length = num of joints
#    breakpoint()
    for joint_q in joint_quats:

        from_quats = joint_q[1:]
        to_quats = joint_q[:len(joint_q)-1]

        d1 = from_quats[:,:1:]
        d2 = from_quats[:,1::1] * -1
        from_quats_conjugate = torch.cat((d1,d2),1) #doing conjugate operation directly
        #quat_diff = to_quats * from_quats
        #the equation is q2 * q1_inverse => since we use unit quat, q_inverse = conj(q)
        quat_diff = qmul(to_quats ,from_quats_conjugate)
        ang_vel = 2 * qmul(quat_diff, from_quats_conjugate)
        quat_difference_arr.append(ang_vel)
    return quat_difference_arr

def velocity_loss_tensor(pred, ground_truth, batch_id):
    num_of_joints = int(pred.shape[2]/4)
    pred_joint_quaternions = get_quaternion_delta_tensor(pred.view(-1, num_of_joints, 4))
    pred_quaternion_diff = get_quat_difference_tensor_arr(pred_joint_quaternions)

    ground_truth_joint_quaternions = get_quaternion_delta_tensor(ground_truth.view(-1, num_of_joints, 4))
    ground_truth_quaternion_diff = get_quat_difference_tensor_arr(ground_truth_joint_quaternions)

    local_loss_val = local_loss(pred_quaternion_diff[0] , ground_truth_quaternion_diff[0])

    for joint_num in range(1,num_of_joints):
        local_loss_val += local_loss(pred_quaternion_diff[joint_num] , ground_truth_quaternion_diff[joint_num])
    local_loss_val = local_loss_val/num_of_joints

    return local_loss_val
```

Figure 5.3: Custom Laban Effort loss function

5.3 Data postprocessing

The network output could be shaky at times. Hence, we need to post-process the generated dance motion to smooth out jerks and jitters. As commonly used by researchers, an n-order Butterworth Filter can be used to smooth the unwanted jerks from the motion sequence.

5.3.1 Butterworth Filter

Butterworth filters are a specific type of low pass filter. The magnitude of the frequency response is defined as:

$$|H(j\omega)| = \frac{1}{\sqrt{1 + (\frac{\omega}{\omega_c})^{2n}}} \tag{5.1}$$

ω_c is the cutoff frequency of the filter and n is the order of the filter.

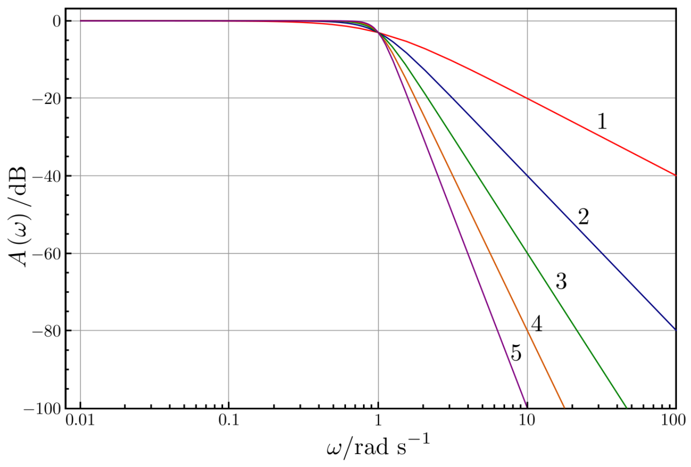


Figure 5.4: Butterworth filter - Gain plotted for order 1 to 5 [6]

At low frequencies, the gain is closer to 1 and at high frequencies, the gain of the filter goes down. In Fig 5.4, When the order is one or two, the gain rolls off smoothly whereas for higher order, the change is quicker and sharper.

This module applies a filter that smooths out the frequencies. The below explanation assumes a use case of 2000 frames with 43 joints and 3 angle (x, y, z)

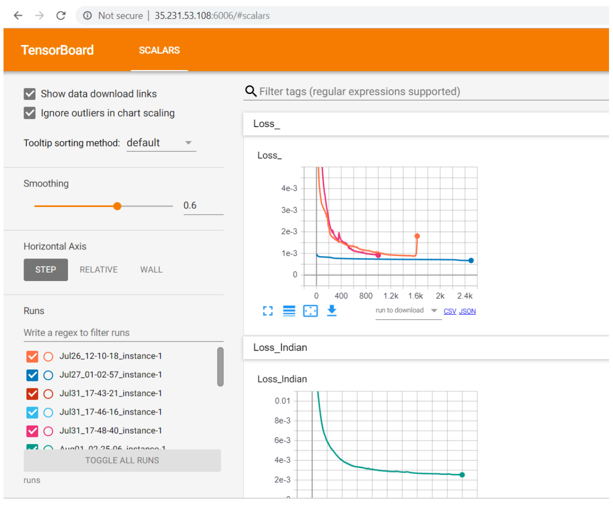
The generated quaternions are converted to Euler angles and are stored in an array of size (2000,43,3). Now, for every joint's Euler angle component, we extract the values in degrees and add to an array of size 2000. The joint-wise array now has a shape of (43,2000,3). The next step is to convert the values to fourier domain by calculating the FFT (Fast Fourier Transform). Butterworth filter is then applied to every joint as in fig 5.5.

```python
def butter_worth_filter(t, u0, n):
    H = np.zeros(t)
    for u in range(int(t/2)):
        H[u] = 1/math.sqrt(1 + (u/u0)**(2*n))
    for u in range(int(t/2),t):
        H[u] = 1/math.sqrt(1 + ((t-u)/u0)**(2*n))
    return H
```

Figure 5.5: Butterworth filter implementation

5.4 Monitoring training performance in real-time

We use Tensorboard from Tensorflow to monitor the training in real-time. The training loop logs the loss values into a file in the local directory which is then picked up by the locally running tensorboard server. The plots can be visualized on the browser as in fig!5.7


Not secure | 35.231.53.108:6006/#scalars

TensorBoard SCALARS

✓ Show data download links
✓ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

✓ Jul26_12-10-18_instance-1
✓ Jul27_01-02-57_instance-1
✓ Jul31_17-43-21_instance-1
✓ Jul31_17-46-16_instance-1
✓ Jul31_17-48-40_instance-1

TOGGLE ALL RUNS

runs

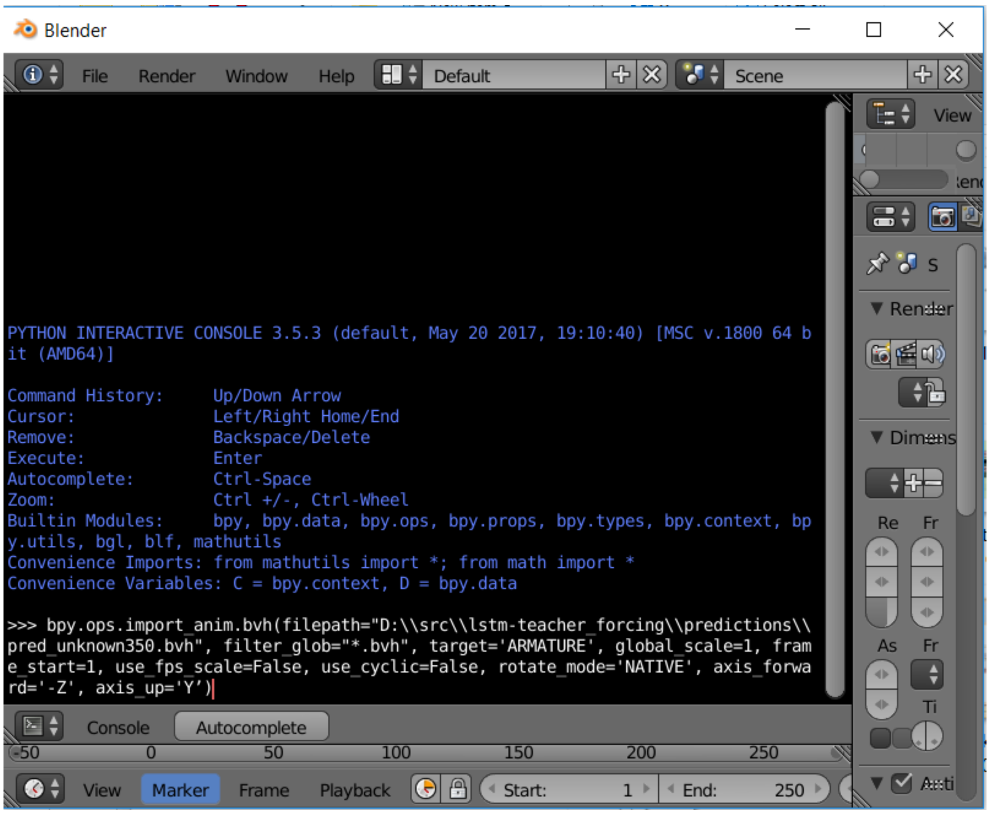
Filter tags (regular expressions supported)

Loss_

Loss_Indian

Figure 5.6: Tensorboard interface

We export BVHs from training and validation set during the training after every 10 epochs. Subsequently, this is read by Blender using the code in fig. A manual qualitative analysis of the training can be performed with Blender using this method. This can be run automatically every few minutes to import the BVH skeleton into Blender GUI.



```
PYTHON INTERACTIVE CONSOLE 3.5.3 (default, May 20 2017, 19:10:40) [MSC v.1800 64 b
it (AMD64)]

Command History:     Up/Down Arrow
Cursor:              Left/Right Home/End
Remove:              Backspace/Delete
Execute:             Enter
Autocomplete:        Ctrl-Space
Zoom:                Ctrl +/-, Ctrl-Wheel
Builtin Modules:     bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bp
y.utils, bgl, blf, mathutils
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> bpy.ops.import_anim.bvh(filepath="D:\\src\\lstm-teacher_forcing\\predictions\\
pred_unknown350.bvh", filter_glob="*.bvh", target='ARMATURE', global_scale=1, fram
e_start=1, use_fps_scale=False, use_cyclic=False, rotate_mode='NATIVE', axis_forwa
rd='-Z', axis_up='Y')
```

Figure 5.7: Code to import BVHs to Blender

Chapter 6

Results and Analysis

This chapter discusses the outcome of implementing the neural network discussed in chapters 4 and 5 to generate dance sequences and poses matching to the genre of the input music. The output generated by the model is compared with ground truth wherever relevant.

We assess the model's ability to generate dance steps by analyzing the rotation, angular velocity, acceleration, and jerks of selected skeletal joints. Firstly, we feed into the model song sequences that it already heard during the training as input. Next, the model is given unheard songs from the validation set. The general process of evaluation in this chapter follows the sequence mentioned above.

The average training time for the model is 11 hours with MSE loss and 15 hours with custom Laban Loss. The model generates frames at speeds well past the real-time requirement. The speed of frame generation averaged at 1175 frames per second. Hence, the design supports real-time applications.

6.1 Mean Squared Error (MSE)

In this configuration, we train the model with an MSE Loss function that calculates the mean squared error between the generated quaternions and the ground truth quaternions. For an Indian classical music piece from the training set, the network generated data with matching joint angles, velocity, and acceleration. In fig 6.2, the velocity, acceleration, and jerk of the left forearm joint are plotted. We chose only one skeletal joint for the ease of representation. We pass the output produced by the network through the post-processing filter (section 5.3.1) and export the BVH file.

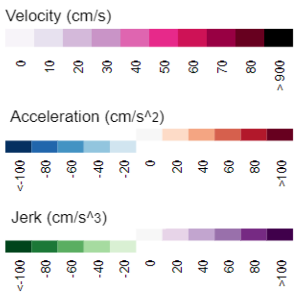


Figure 6.1: Legend for all motion comparisons in this chapter

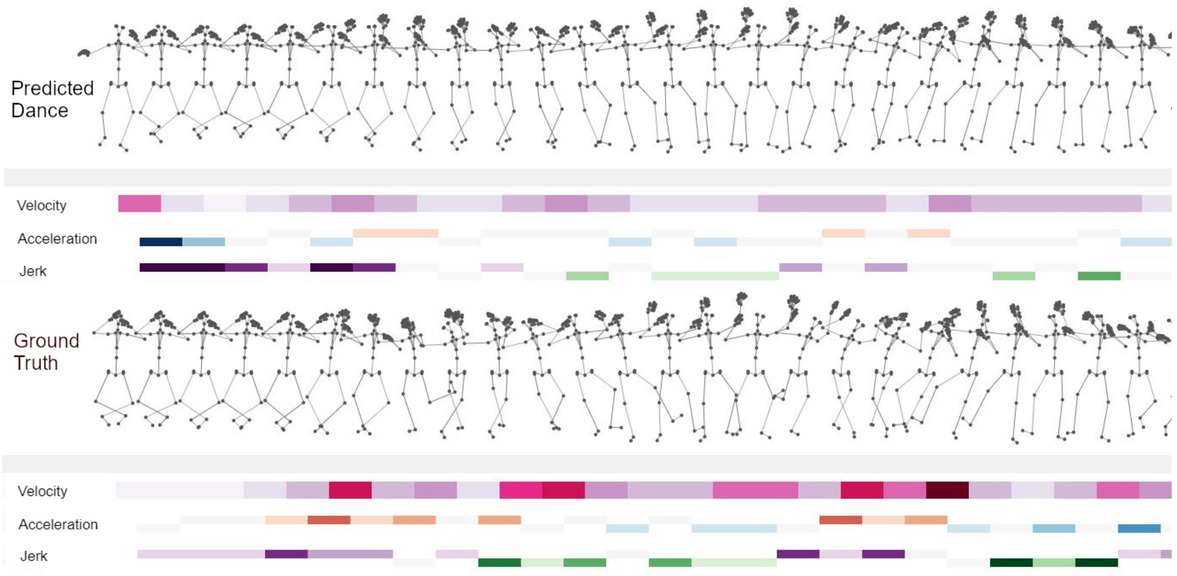


Figure 6.2: Angular velocity, acceleration and jerk of the predicted dance and ground truth (every 5th frame is plotted) of left forearm joint

Comparing the joint velocities, the corresponding intensities of the frames are higher in the ground truth than the prediction as visualized in fig 6.2. The acceleration and deceleration of the joint are comparable to each other. It follows a similar trend as that of velocity in terms of the intensity. The degree of sudden movements or jerks in the motion also similarly corresponds to each other. It means that the generated dance is a 'lazy' or a less intense version of the ground truth.

Interestingly, a slight noise in the ground truth (In fig 6.2, frame 8 from the left, and frame 5 to 7 from the right) is fixed in the predicted motion because the network learned to link and fill in between adjacent frames.

For an unheard piece of Salsa music, the network generated a sequence of very jittery poses. It is interesting to note that the jitters of the hip joint affect the overall aesthetics of the movement the most. Since the hip is the root joint of the skeleton, its instability affects the entire skeleton much more than any other joint.

6.2 Laban Effort loss function

After MSE loss, we train the network with the custom loss function as designed in section 4.6. The network converged faster in 500 epochs as opposed to 2500 epochs as compared to MSE loss. Nevertheless, the generated poses oscillate within a short range of movements with high-intensity jerks for both training set as well as validation set inputs.

For an unheard salsa music from the validation set visualized in fig 6.3, the predicted dance's hip joint is found to be highly unstable. Sometimes, it flips the skeleton's facing direction due to a sudden 180 degree rotation of the hip in adjacent frames with no continuity. It is because the network learned to match the overall energy of the movement and cares less about the ground truth's joint orientations.

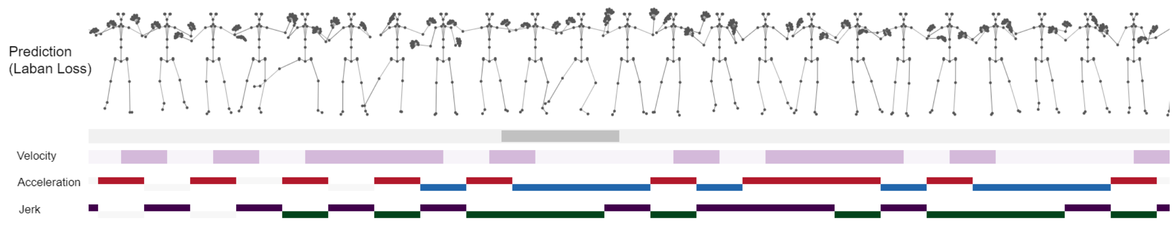


Figure 6.3: Angular velocity, acceleration and jerk of the predicted dance and ground truth (every 5th frame is plotted) of the abdomen joint

Fig 6.3 shows the front view of the sequence of frames predicted using Laban loss function. The acceleration and jerk frequently reach their extremes every few frames. It can be inferred that changes of this scale to any of the skeletal joints directly connected to the hip mean extreme instability. The stick figures in the visualization look unchanged because the flips happen exactly opposite to the viewing angle. This is visualized from a different angle in fig 6.4

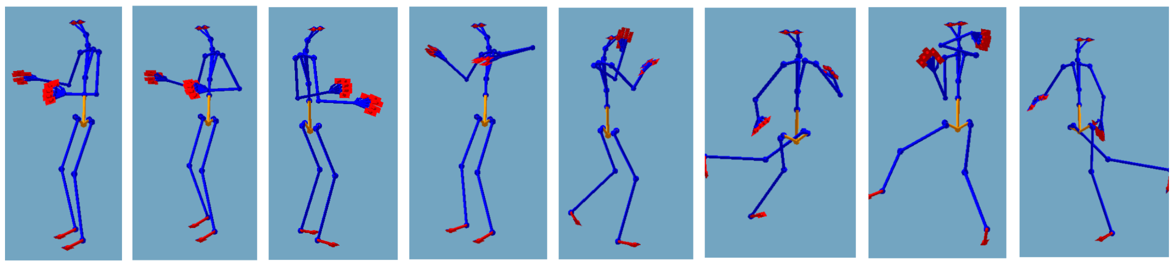


Figure 6.4: Consequent frames with 180 degree hip rotations and extreme angles

6.3 MSE + Laban Effort Loss

The following experiments use different combinations of the loss functions. From the previous section, we found out that, using only Laban loss function disregards the ground truth poses. Here, we consider the following two cases:

- **Sum of MSE and Laban Loss**: We add both losses together and backpropagate to minimize the new loss. In multiple trials, we found that the network stopped learning once it reached 50 epochs. It could not optimize for either MSE or Laban loss.

- **One Laban loss every 4 epochs**: In this configuration, we use MSE loss for three iterations while every fourth iteration uses a sum of MSE and Laban loss. For an unheard salsa music from the validation set, the model generated dance poses with considerably less flickers than section 6.2. Although the hip flickers at times, the preliminary analysis shows that the poses generated by the model belong to the distribution of poses in the salsa dataset. Furthermore, the generated data has comparatively more temporal coherence. Fig 6.7 shows the left forearm joint's energy visualizations.

In fig 6.5, the acceleration and jerk are more dynamic and less intense than fig 6.3. Although there are hip flips, the movement of hands became stabler. Furthermore, the adjacent frames show continuity. It is interesting to note that, after manually fixing hip, the movements of the hands and the continuity looks more plausible.

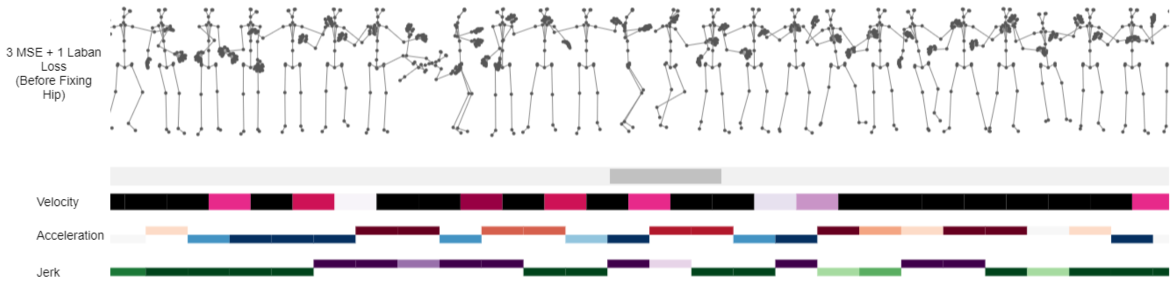


Figure 6.5: One Laban Loss every 4 epochs (Before Fixing Hip, Before Post processing)

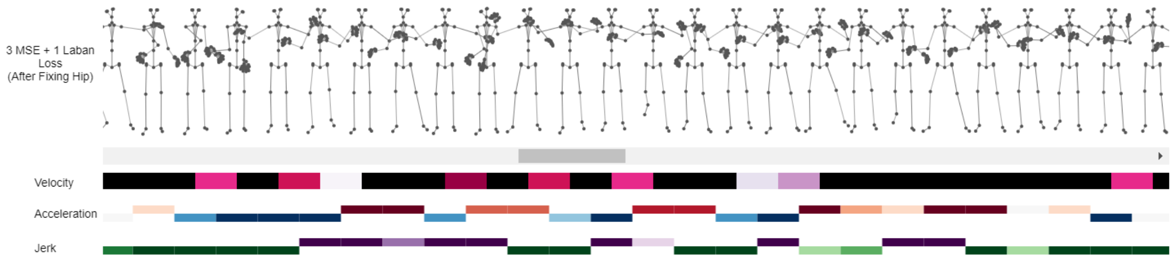


Figure 6.6: One Laban Loss every 4 epochs (After Fixing Hip, Before Post processing)

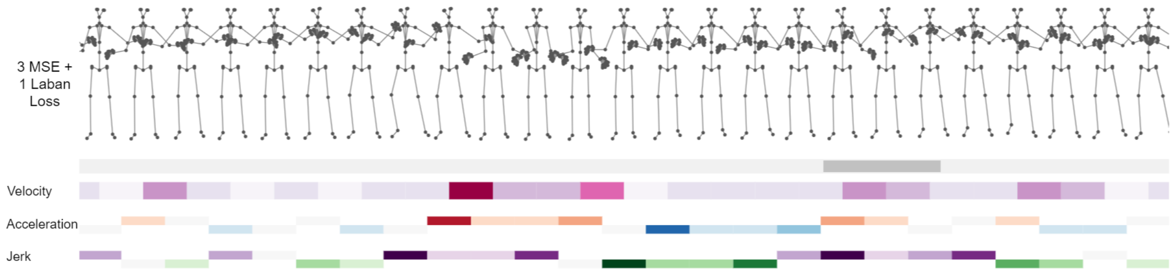


Figure 6.7: One Laban Loss every 4 epochs (After Fixing Hip, After Post processing)

6.4 The effect of datasets

To check the effect of the type of dance available to the model, we trained the model with three different configurations:

- Only Indian dataset
- Only Salsa dataset
- Combined dataset

The training set performed equally well in all the three configurations. However,the hip joint is found to be comparatively stabler when the combined dataset is used. This shows that more input data can improve the quality of the generated motion.

6.5 Generation of long sequences

We give the network a long audio sequence to verify the design choice - auto-conditioned configuration, as discussed in section 4.2. Fig 6.8 analyzes the network output of 13000 continuous frames. It is found that the network can generate long sequences of motion without getting frozen or attaining a mean pose. The velocity, acceleration, and jerk remain continuous and dynamic throughout the sequence.

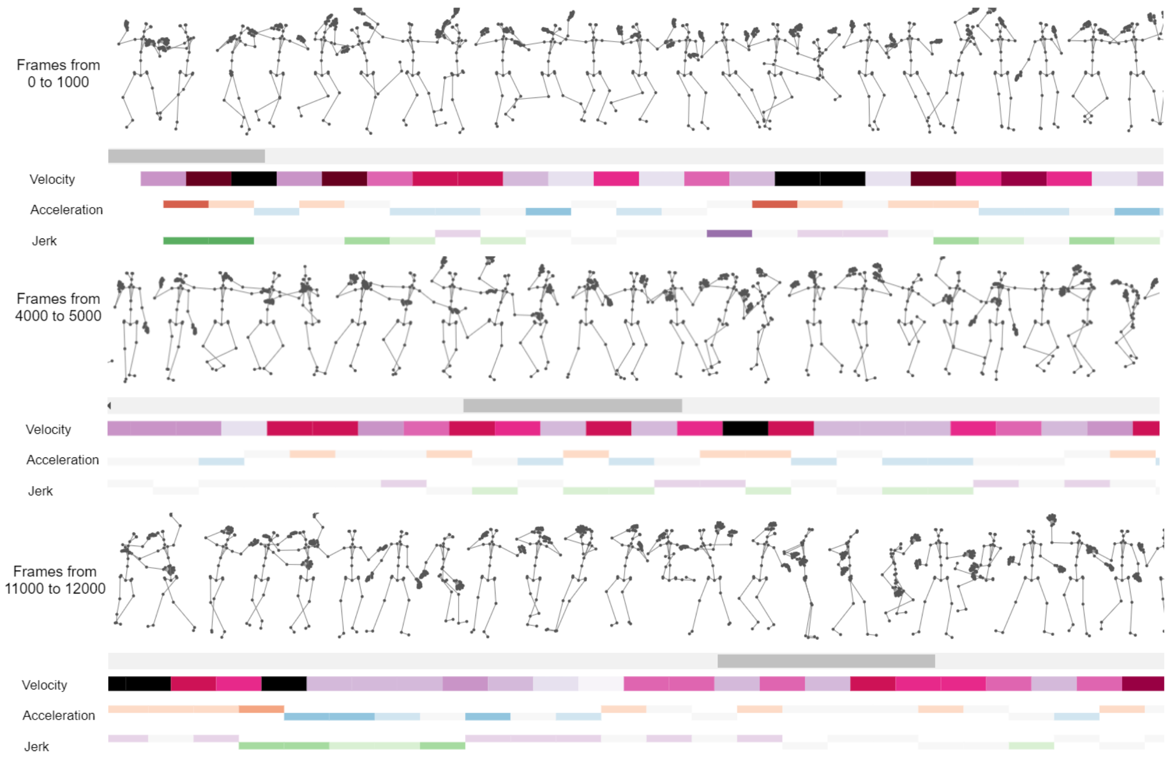


Figure 6.8: Angular velocity, acceleration and jerk of the predicted dance of left forearm joint of a 12000-frame long sequence

6.6 Teacher Forcing

To analyze the effect of Teacher forcing, we trained the network with 0% and 30% Teacher forcing ratio. We found that the network converged faster when using Teacher forcing. Nonetheless, the results generated are very similar to each other. This shows that Teacher forcing can be used as a technique to slightly accelerate the learning, but it has no visible changes in the performance of the model.

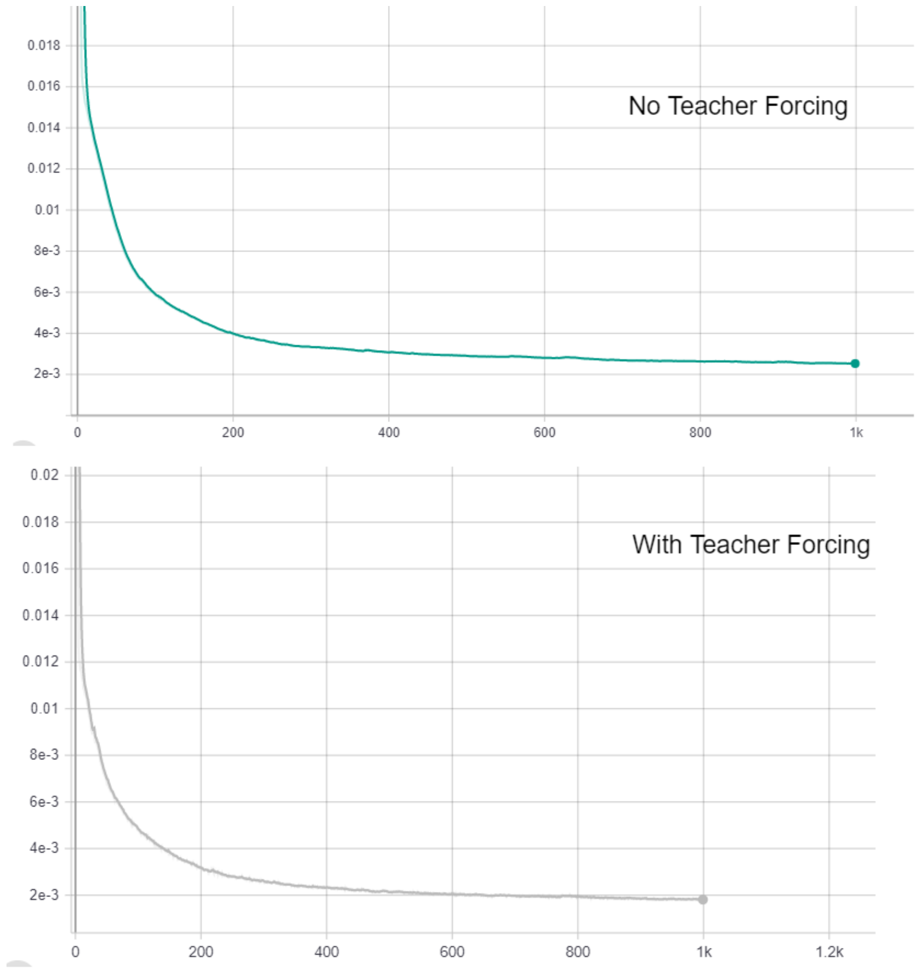


Figure 6.9: Loss plotted with and without Teacher Forcing

6.7 Network output vs post-processed

This section shows a comparison between the network's output and post-processed output. Butterworth filter with order 1 and 2 are applied and visualized in fig 6.10. From the plot, alternating dark green and purple bands show that the generated dance is jerky, and hence the corresponding velocities and accelerations are at the highest end of the scale. After 1-order Butterworth filter is applied, the jerk is smoothed out. The intensity of the jerk is reduced, and the corresponding colored bands have become longer. The longest black band in the network output corresponds to very high velocities whereas after applying 1-order Butterworth, the transition of velocities between frames is much smoother.

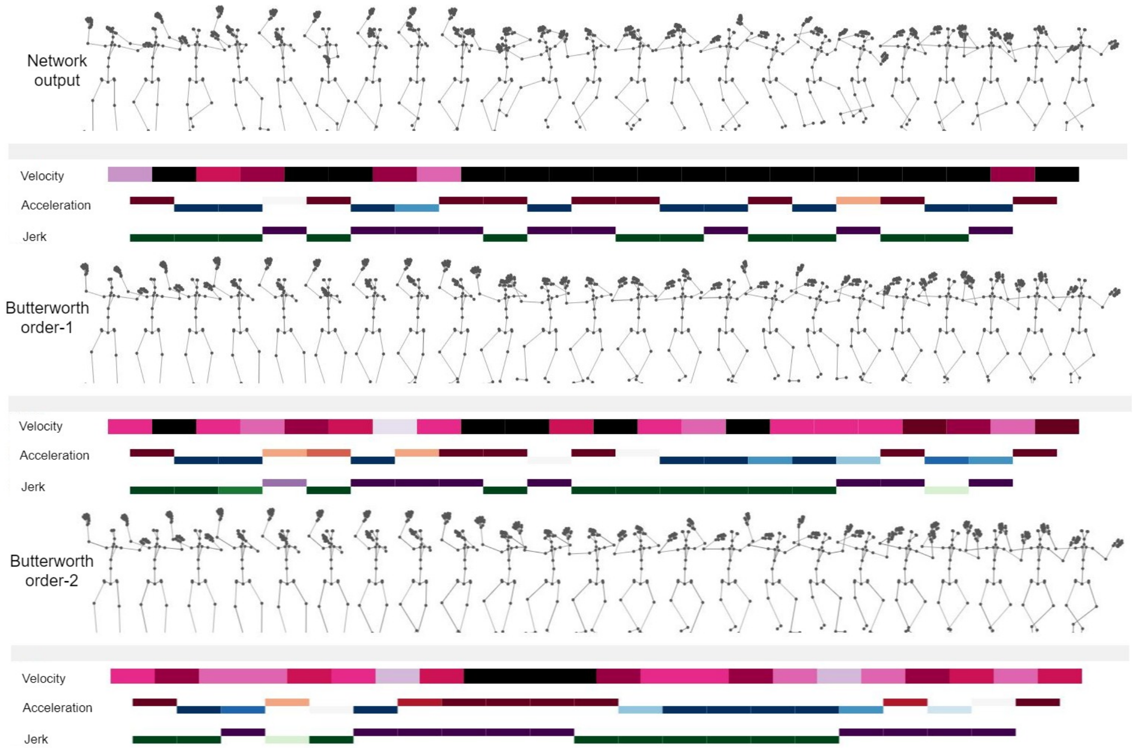


Figure 6.10: Angular velocity, acceleration and jerk of the predicted dance, post-processed dances with 1 and 2 order Butterworth filters (every 5th frame is plotted) of left forearm joint

6.8 Summary of results

From the results described in the previous section, using MSE loss, the model can generate dance sequences specific to the genre of the music that it already heard during the training. However, for a new piece of music from the validation set, the model generated results that are jittery and non-continuous. When using the custom Laban Loss function, the model generated highly unstable movements, but when used in conjunction with MSE, it behaved slightly better. Although not totally smooth, the hand movements seen are realistic and visually closer to salsa. We need more experiments to fully verify the custom Laban Loss function. The way we achieved clean result is by manual hip fixing.

The Indian Classical Music data has most of its steps with hands stretched out, at or above shoulder level while most of the Salsa steps have the hands closer to the chest and below the shoulder level. Although not the best way, this is a visual metric that can be used to quickly identify the dance genre in this thesis' context. Furthermore, the model is able to generate long sequences without getting the mean-pose issue because of the auto-conditioned configuration. Surprisingly, it is also learnt that using Teacher Forcing has no effect on the network output, but it improves the network convergence time.

Chapter 7

Conclusion

We have designed, implemented and tested a novel neural network architecture to generate dance motion corresponding to the input music. To answer the original research questions, the network can produce dance sequences for the music it heard during the training, but for unheard music, the network produces results that are not so plausible. However with the use of custom loss functions based on theories of physical movement, there is an improvement in the generated dance, but still not realistic. Further experiments need to be conducted with different parameter tuning to evaluate this. However, this is a challenge with respect to compute resources because one experiment takes 11 to 15 hours. In this research experiment, user studies were not conducted due to the lack of time, but in the future experiments, it should be taken in consideration.

As compared to traditional methods where the user has much control for engineering the input feature and its handling, the potential for user input is limited with Deep Learning-based methods. Instead, they detect correlations in the data by itself and use that as a feature vector to generate dance poses. Generating dance is a much bigger and deeper problem than locomotion just because of the nature of the art. Dance is an expression of emotions, feelings, and sensations. It bears meanings at multiple levels in addition to synchronizing with music. To evaluate a movement and consider it as a dance movement, we need to look at the science of the art. Typical mathematical ways of error calculation using joint angle or positional differences are meaningful only in verifying movements against a ground truth. For instance, when a dancer performs a step where the left arm waves over the head, and then another step where the same hand waves across the hips. Let us assume a case where the former is predicted by the model, and the latter is the ground truth the model is expected to output. Although both are valid dance steps, when we calculate the mean square error between these two steps, the resulting value would suggest that one is different from the other by quite a considerable amount. Typically, a large error value suggested by the mean square function means there is something significantly wrong with the output. In the context of dance, due to this reason, mean square error does not provide a meaningful measure of validating a dance sequence. Hence, movement theories like Laban Movement Analysis should be incorporated into loss functions while training Deep Neural Networks.

7.1 Future Work

The amount of frames used as dataset is tiny compared to what a recurrent neural network can handle. A huge dataset of 2 hours of dance would be a good experiment to do in future. More experiments with different hyperparameters can be conducted in future to fully validate the Laban loss function. There are more parameters in Laban Movement Analysis to describe dance movements. It would be worthwhile to implement more Laban parameters as loss functions. The network architectures of Encoder-Decoder and sequence2sequence are also possible future extensions. Another interesting work to build upon this is to implement Adverserial learning. When using a GAN (Generative Adverserial Network), the generator can be the proposed network in this thesis while the discriminator can be trained on a larger dataset of motion and dance so that it can discriminate between valid motion and invalid motion.

Bibliography

[1] M. Meredith, S. Maddock, *et al.*, "Motion capture file formats explained," *Department of Computer Science, University of Sheffield*, vol. 211, pp. 241–244, 2001.

[2] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7291–7299, 2017.

[3] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik, "End-to-end recovery of human shape and pose," in *Computer Vision and Pattern Regognition (CVPR)*, 2018.

[4] Y. Zhou, Z. Li, S. Xiao, C. He, Z. Huang, and H. Li, "Auto-conditioned recurrent networks for extended complex human motion synthesis," in *International Conference on Learning Representations*, 2018.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[6] Wikipedia, "Butterworth filter - graph plotted."

[7] E. Masicampo and R. F. Baumeister, "Conscious thought does not guide moment-to-moment actions—it serves social and cultural functions," *Frontiers in psychology*, vol. 4, p. 478, 2013.

[8] M. Kitagawa and B. Windsor, *MoCap for artists: workflow and techniques for motion capture.* Routledge, 2012.

[9] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, (New York, NY, USA), pp. 51:1–51:10, ACM, 2008.

[10] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, (New York, NY, USA), pp. 491–500, ACM, 2002.

[11] M. Stone, D. DeCarlo, I. Oh, C. Rodriguez, A. Stere, A. Lees, and C. Bregler, "Speaking with hands: Creating animated conversational characters from recordings of human performance," in *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, (New York, NY, USA), pp. 506–513, ACM, 2004.

[12] E. Hsu, S. Gentry, and J. Popović, "Example-based control of human motion," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04, (Goslar Germany, Germany), pp. 69–77, Eurographics Association, 2004.

[13] J.-H. Seo, J.-Y. Yang, J. Kim, and D.-S. Kwon, "Autonomous humanoid robot dance generation system based on real-time music input," in *2013 IEEE RO-MAN*, pp. 204–209, IEEE, 2013.

[14] M. Lee, K. Lee, and J. Park, "Music similarity-based approach to generating dance motion sequence," *Multimedia tools and applications*, vol. 62, no. 3, pp. 895–912, 2013.

[15] T. Shiratori, A. Nakazawa, and K. Ikeuchi, "Synthesizing dance performance using musical and motion features," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 3654–3659, IEEE, 2006.

[16] F. Bevilacqua, L. Naugle, and I. Valverde, "Virtual dance and music environment using motion capture," in *Proc. of the IEEE-Multimedia Technology And Applications Conference, Irvine CA*, 2001.

[17] C. Panagiotakis, A. Holzapfel, D. Michel, and A. A. Argyros, "Beat synchronous dance animation based on visual analysis of human motion and audio analysis of music tempo," in *International symposium on visual computing*, pp. 118–127, Springer, 2013.

[18] D. Kimber, L. Wilcox, *et al.*, "Acoustic segmentation for audio browsers," *Computing Science and Statistics*, pp. 295–304, 1997.

[19] E. Wold, T. Blum, D. Keislar, and J. Wheaten, "Content-based classification, search, and retrieval of audio," *IEEE multimedia*, vol. 3, no. 3, pp. 27–36, 1996.

[20] D. Kirovski and H. Attias, "Beat-id: Identifying music via beat analysis," in *2002 IEEE Workshop on Multimedia Signal Processing.*, pp. 190–193, IEEE, 2002.

[21] E. D. Scheirer, "Tempo and beat analysis of acoustic musical signals," *The Journal of the Acoustical Society of America*, vol. 103, no. 1, pp. 588–601, 1998.

[22] M. Goto, "An audio-based real-time beat tracking system for music with or without drum-sounds," *Journal of New Music Research*, vol. 30, no. 2, pp. 159–171, 2001.

[23] A. Jose Homsi Goulart, C. Maciel, R. Guido, K. Cristina Silva Paulo, and I. Nunes da Silva, "Music genre classification based on entropy and fractal lacunarity," pp. 533–536, 12 2011.

[24] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.

[25] Y. Zhu, C. M. Manders, F. Farbiz, and S. Rahardja, "Music feature analysis for synchronization with dance animation," in *TENCON 2009-2009 IEEE Region 10 Conference*, pp. 1–5, IEEE, 2009.

[26] F. Ofli, E. Erzin, Y. Yemez, and A. M. Tekalp, "Multi-modal analysis of dance performances for music-driven choreography synthesis," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2466–2469, IEEE, 2010.

[27] W. J. Dowling and D. L. Harwood, *Music cognition.* Academic Press, 1986.

[28] T. Li, M. Ogihara, and Q. Li, "A comparative study on content-based music genre classification," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, (New York, NY, USA), pp. 282–289, ACM, 2003.

[29] M. Huzaifah, "Comparison of time-frequency representations for environmental sound classification using convolutional neural networks," *arXiv preprint arXiv:1706.07156*, 2017.

[30] O. Alemi, J. Françoise, and P. Pasquier, "Groovenet: Real-time music-driven dance movement generation using artificial neural networks," *networks*, vol. 8, no. 17, p. 26, 2017.

[31] A. Menache, *Understanding motion capture for computer animation and video games.* Morgan kaufmann, 2000.

[32] D. Sturman, "The state of computer animation," *COMPUTER GRAPHICS-NEW YORK-ASSOCIATION FOR COMPUTING MACHINERY-*, vol. 32, pp. 57–61, 1998.

[33] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec 1943.

[34] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[35] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[36] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.

[37] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[38] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[40] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 138, 2016.

[41] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber, "Documentation mocap database hdm05," 2007.

[42] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy, "Berkeley mhad: A comprehensive multimodal human action database," in *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, pp. 53–60, IEEE, 2013.

[43] T. Tang, J. Jia, and H. Mao, "Dance with melody: An lstm-autoencoder approach to music-oriented dance synthesis," in *2018 ACM Multimedia Conference on Multimedia Conference*, pp. 1598–1606, ACM, 2018.

[44] D. Pavllo, D. Grangier, and M. Auli, "Quaternet: A quaternion-based recurrent model for human motion," in *BMVC*, 2018.

[45] N. Yalta, S. Watanabe, K. Nakadai, and T. Ogata, "Weakly supervised deep recurrent neural networks for basic dance step generation," *arXiv preprint arXiv:1807.01126*, 2018.

[46] S. Chopra, R. Hadsell, Y. LeCun, *et al.*, "Learning a similarity metric discriminatively, with application to face verification," in *CVPR (1)*, pp. 539–546, 2005.

[47] H. Thomas, *The body, dance and cultural theory.* Macmillan International Higher Education, 2003.

[48] J. L. Hanna, *To dance is human: A theory of nonverbal communication.* University of Chicago Press, 1987.

[49] K. Vatsyayan, "Notes on the relationship of music and dance in india," *Ethnomusicology*, vol. 7, no. 1, pp. 33–38, 1963.

[50] K. Laws, *Physics and the art of dance: Understanding movement.* Oxford University Press, 2002.

[51] R. Von Laban and R. Lange, *Laban's principles of dance and movement notation.* Princeton Book Co Pub, 1975.

[52] I. Bartenieff, "Dance therapy: A new profession or a rediscovery of an ancient role of the dance," *Dance Scope*, vol. 7, no. 1, pp. 6–18, 1972.

[53] E. Davies, *Beyond dance: Laban's legacy of movement analysis.* Routledge, 2007.

[54] N. Y. University, "Nyu movement : Intro to lma."

[55] U. of Wisconsin-Madison, "Course - cs-838-1999."

[56] C. M. University, "Cmu graphics lab motion capture database."

[57] S. Krishnaswami, "Musical instruments of india," *Asian Music*, vol. 2, no. 2, pp. 31–42, 1971.

[58] J. Salamon, S. Gulati, and X. Serra, "A multipitch approach to tonic identification in indian classical music," in *Gouyon F, Herrera P, Martins LG, Müller M. ISMIR 2012: Proceedings of the 13th International Society for Music Information Retrieval Conference; 2012 Oct 8-12; Porto, Portugal. Porto: FEUP Ediçoes; 2012.*, International Society for Music Information Retrieval (ISMIR), 2012.

[59] P. Agarwal, H. Karnick, and B. Raj, "A comparative study of indian and western music forms.," in *ISMIR*, pp. 29–34, 2013.

[60] M. Berrios-Miranda, "Salsa music as expressive liberation," *Centro Journal*, vol. 16, no. 2, pp. 158–173, 2004.

[61] O. Source, "Blender - open source 3d creator.."

[62] Dene33, "Hmr - github."

[63] S. Hochreiter, "Recurrent neural net learning and vanishing gradient," *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[64] C. Olah, "Understanding lstm networks."

[65] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[66] N. Toomarian and J. Barhen, “Fast temporal neural learning using teacher forcing,” June 27 1995. US Patent 5,428,710.

[67] W. R. Hamilton, *Elements of quaternions.* Longmans, Green, & Company, 1866.

[68] J. B. Kuipers *et al.*, *Quaternions and rotation sequences*, vol. 66. Princeton university press Princeton, 1999.

[69] E. Pervin and J. A. Webb, “Quaternions in computer vision and robotics,” tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1982.

[70] J. Diebel, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[71] T. Nakata, T. Mori, and T. Sato, “Analysis of impression of robot bodily expression,” *Journal of Robotics and Mechatronics*, vol. 14, no. 1, pp. 27–36, 2002.

[72] T. Shiratori, A. Nakazawa, and K. Ikeuchi, “Dancing-to-music character animation,” in *Computer Graphics Forum*, vol. 25, pp. 449–458, Wiley Online Library, 2006.

[73] K. I. Kou and Y.-H. Xia, “Linear quaternion differential equations: Basic theory and fundamental results,” *Studies in Applied Mathematics*, vol. 141, no. 1, pp. 3–45, 2018.

[74] I. S. Unitversity, “Cs577 - quaternions.”

[75] J. Voight, “Quaternion algebras,” *Version v0*, vol. 9, 2018.

[76] university of illinois at Urbana-Champaign, “Introduction to quaternionial algebra.”

[77] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, “Firecaffe: near-linear acceleration of deep neural network training on compute clusters,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2592–2600, 2016.

[78] M. Espi, M. Fujimoto, K. Kinoshita, and T. Nakatani, “Exploiting spectro-temporal locality in deep learning based acoustic event detection,” *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2015, no. 1, p. 26, 2015.

[79] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[80] Z. Wang, J. Chai, and S. Xia, “Combining recurrent neural networks and adversarial training for human motion synthesis and control,” *arXiv preprint arXiv:1806.08666*, 2018.

[81] D. Holden, “Bvh dataset compiled.”

[82] D. Holden, J. Saito, T. Komura, and T. Joyce, “Learning motion manifolds with convolutional autoencoders,” in *SIGGRAPH Asia 2015 Technical Briefs*, 2015.

Appendix

Google Drive link for results - https://drive.google.com/open?id=1c20GpTtwxKIQw-xi-vmQ-EwhcHnuaOq2