Dance generation using Machine Learning

Abstract

Innovations in the field of generative machine learning have opened up many doors to build and refine many complex models that help us input a large corpus of data and generate similar data based on the given corpora. These generative models have allowed us to input various types of data such as music, text, dance, speech among other things and output data similar to the input. In this paper we speak specifically about dance moves corresponding to an Indian style of dancing *Bharatanatyam*. We first extract the human skeleton from dance videos to encode the data for machine learning and then feed it to different generation models to obtain similar *Bharatanatyam steps.* The various approaches we take are Deep artificial neural networks, autoencoders and Recurrent neural networks. Our final model is based on Andrej Karpathy's char-rnn which when given large amounts of textual data, generates sentences similar to the corpus of data using recurrent neural networks. We also take a small glimpse of audio encoding and building a neural network to correlation audio vector and dance position, but most of this is still work in progress. This model can provide inspiration to a choreographer and become the foundation to various entertainment systems. The three generative models and their performances are compared here to settle finally on the RNN model which gives satisfied results. This model can be altered to provide different types of data such as a fighter's moves and strategies to generate new ones.

Introduction

Bharatanatyam is an Indian Classical Dance form that originated in Tamil Nadu. The basic theme of Bharatanatyam revolves around religious themes, spiritual ideas and folklore. Bharatanatyam is noted for its fixed upper torso, bent legs or knee flexed out combined with intricate footwork and a sophisticated vocabulary of sign language based on gesture of hand, eye and face.

Basic Terms:

1. Thala - The underlying beat example - Dhruva Thala, Matya Thala, Rupaka Thala, Jhapme Thala, Triputa Thala, Atta Thala and Eka Thala
2. Laya - Speed or tempo of the music - Vilambit, Maddhya, Drut
3. Adavu - Steps or sequence of steps that are done in synchronization with sollu kattu (bol). There are sets of similar steps that are learnt as the basic building blocks of every dance performance. Example - Tattu, Mettu, Teermana etc.

Bharatanatyam consists of three dance types:

1. Nritya: Rhythmic with high dependency on Thala and Laya. The Nritta performance is an abstract, fast and rhythmic aspect of the dance. The viewer is presented with pure movement in Bharatanatyam, wherein the emphasis is the beauty in motion, form, speed, range and pattern. This part of the repertoire has no interpretative aspect, no telling of story. It is a technical performance, and aims to engage the senses (prakriti) of the audience. These are the dance types that we focus on and use in this project.
2. Nritya: Combination of rhythm and expression and conveys poetic meaning with the help of expressions, rhythmic gaits and postures. The Nritya is a slower and expressive aspect of the dance that attempts to communicate feelings, storyline particularly with spiritual themes in Hindu dance traditions. In a nritya, the dance-acting expands to include silent expression of words through gestures and body motion set to musical notes. The actor articulates a legend or a spiritual message. This part of a Bharatanatyam repertoire is more than sensory enjoyment, it aims to engage the emotions and mind of the viewer.
3. Natya: Has more dramatic elements and expressions and rasa. The Natyam is a play, typically a team performance, but can be acted out by a solo performer where the dancer uses certain standardized body movements to indicate a new character in the underlying story. A Natya incorporates the elements of a Nritya.

The basic and most common signature posture of Bharatanatyam is the Ardhamandala/ Aramandala which is a half sitting posture. The arms are also positioned in a particular manner such that the entire body can be divided into triangles. Bharatanatyam also uses a very concise set of hand gestures to go with every movement. A very basic way to describe movements in Bharatanatyam would be a

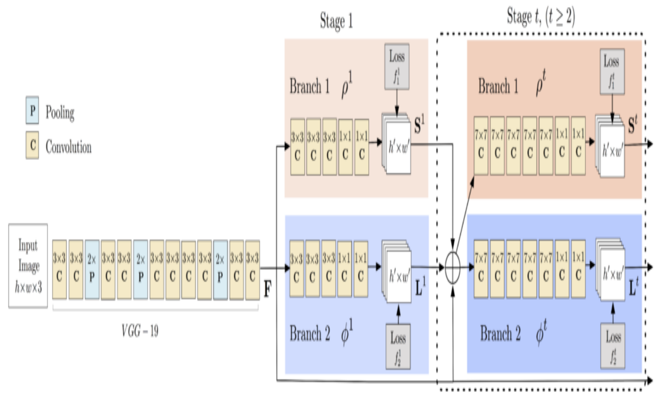


Fig 1.1 Above is the architecture for human pose detection. Here C corresponds to convolution later and P represents the pooling layer. The image is annotated with dimensions of each layer for corresponding stages and time steps

move for every thaala. The entire dance form is based on the sollu-kottu behind the adavus that make up the dance.

Motivation

As Bharatanatyam has a comparatively fixed posture as compared to other dance forms, it makes learning the human form easier. Furthermore, as in every dance posture in this form, the body is divided into a set of triangles and diamonds, the geometric cohesiveness allows the system to generate effectively. The angular posture also makes sure that all the skeletal points of the body are capable of being represented along a simple plane.
Bharatanatyam is performed to Carnatic music and the dance is often based on stories of people from Hindu Religious Texts. This assures fluidity and continuity in the movements of the dancer i.e. the dancer does not make very abrupt moves. The recording of transition from one posture to another can be done well. Therefore, a low frame rate can also capture the essence of the dance very well.
Lastly, Bharatanatyam is nothing if not a beat-based dance. There is a strong correlation between the beats and the steps. Almost every bol in the sollu-kottu is assured a step. Hence, there is constant change with good relation to music. Because all the steps are built on the basic blocks established by the adavus, the generative model has a good lay of the land when it comes to the data, meaning, the dataset is a very good reflection of the real-world conditions when it comes to Bharatanatyam.

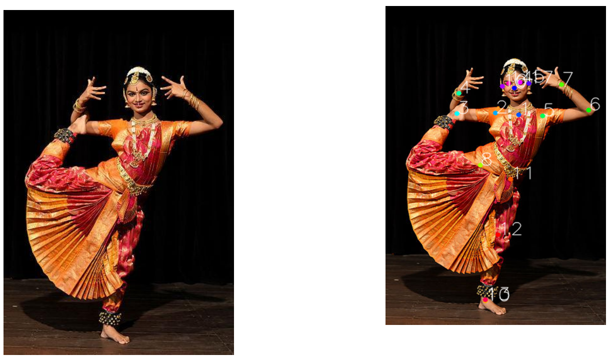
Literature survey

The literature survey consisted of looking through various papers mainly Andrej Karpathy's char-rnn which input a large corpus of data and outputs data similar to the input. Various papers were read to build a parallel model that inputs dance position in some format. Papers for human pose detection were looked through to build an encoding of dance position from a frame from a video to an encoding suitable for neural networks. In specific 'OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields by Zhe Cao, Student Member, IEEE, Gines Hidalgo, Student Member, IEEE, Tomas Simon, Shih-En Wei, and Yasser Sheikh' [2]. Several other models such as autoencoders and Deep convolutional networks were looked through and implemented to get best results. The best results were obtained from the Open pose implementation of human pose estimation.
The paper Art to Smart: An Automated Bharatanatyam Dance Choreography was surveyed to understand the basics of Bharatanatyam dance steps and its vectorization. The basics of various models such as Andrej Karpathy's char-rnn implementation was surveyed to study the working of plain rnn and its use in language modeling. Similar concepts were applied in human skeleton movement modeling.

Dance position encoding

Among various ways the dance position encoded. The most accurate way of encoding the dance position was to extract the solo human pose estimation of the image. Videos of Bharatanatyam solo dancers edited to crop out unnecessary frames not containing the dancer were taken. The videos were merged to form one long dance video. The whole video was processed frame by frame to extract a large set of images defined by the frame rate. The image was individually assessed and fed to the human pose estimation model. The architecture of this model and its parameters are provided in Fig 1.1.

The architecture can be segmented into three stages. The first stage consists of 10 layers VGGNet[4], the state-of-the-art model for image captioning to extract features from the input image. The second stage splits the model into two branches where the first branch will predict 2D positions of the human body parts and forms the confidence maps for them and the second branch builds an affinity map that tells the affinity of various body parts detected. The third stage the confidence and affinity maps are parsed using greedy inferencing to form the skeletal joint structure of the human in the input image. The image points are plotted on the image and the joining can be done according to the indices provided by the affinity maps of corresponding body parts to get the human pose skeleton. The output of this model from input image 1.2 (a) is shown in 1.2 (b). Transfer learning is incorporated to use pretrained weights for image captioning training of the dataset MSCOCO.



(a) (b)

Fig 1.2 (a) Input image (b) Output joint tagged image

The input is provided to a CNN to jointly predict confidence maps for body part detection parts association. The parsing step performs a set of bipartite matchings to associate body parts predicted. The components are then finally combined to form the human skeleton for the given image. The output of each frame input consists of a vector containing the positions of the joints of the human skeleton. Each joint consists of a list of x-coordinate, y-coordinate and confidence of that joint. An example output for input frame Fig 1.2 is shown in Fig 1.3. Each list in the input corresponds to a joint in the skeleton. The indices of the lists are mapped to human body joints which are stored in a dictionary to create edges as and when necessary.

The mappings are as follows: Nose – 0, Neck – 1, Right Shoulder – 2, Right Elbow – 3,RightWrist–4,Left Shoulder – 5, Left Elbow – 6, Left Wrist–7, Right Hip–8, Right Knee – 9, Right Ankle –10, Left Hip–11, Left Knee–12, L Ankle – 13, Right Eye –14, Left Eye–15, Right Ear–16, Left Ear–17, Background – 18.

The feature vector per frame is now ready to be input to various machine learning models to generate new dance position vectors similar to these. The affinity maps are then used to reconstruct the skeletal structure given the output generated vector. Some joints with low prediction probability may be left and directly joined to predecessors to avoid abnormal non-human representing skeletons.

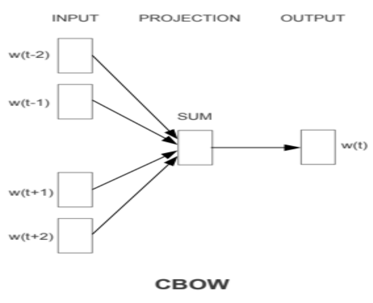
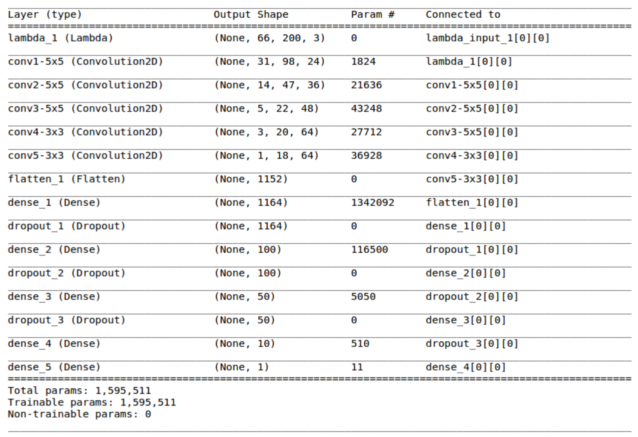


Fig 2.1 The CBOW architecture for language modeling.



```
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 66, 200, 3)    0           lambda_input_1[0][0]
____________________________________________________________________________________________________
conv1-5x5 (Convolution2D)        (None, 31, 98, 24)    1824        lambda_1[0][0]
____________________________________________________________________________________________________
conv2-5x5 (Convolution2D)        (None, 14, 47, 36)    21636       conv1-5x5[0][0]
____________________________________________________________________________________________________
conv3-5x5 (Convolution2D)        (None, 5, 22, 48)     43248       conv2-5x5[0][0]
____________________________________________________________________________________________________
conv4-3x3 (Convolution2D)        (None, 3, 20, 64)     27712       conv3-5x5[0][0]
____________________________________________________________________________________________________
conv5-3x3 (Convolution2D)        (None, 1, 18, 64)     36928       conv4-3x3[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 1152)          0           conv5-3x3[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 1164)          1342092     flatten_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 1164)          0           dense_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 100)           116500      dropout_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 100)           0           dense_2[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 50)            5050        dropout_2[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 50)            0           dense_3[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 10)            510         dropout_3[0][0]
____________________________________________________________________________________________________
dense_5 (Dense)                  (None, 1)             11          dense_4[0][0]
====================================================================================================
Total params: 1,595,511
Trainable params: 1,595,511
Non-trainable params: 0
____________________________________________________________________________________________________
```

Fig 2.2 model summary of deep neural network

Deep Neural Networks for Generation

Deep neural networks are neural networks with a large number of hidden layers and can capture huge amounts of complexity. They can capture complex nonlinear relations among input. However, each input is considered as an individual input and there is no connectivity among the sequences of the inputs. Deep neural networks given the context of a word can predict the next word or word most related to the given input context. This is related to the cbow neural model [3] for language modeling's architecture of the COBOW model incorporated is shown in Fig 2.1. This model is enhanced to provide deeper layers to form a deep neural network so it can represent complex functions/ here we replace words with our encoded dance position vectors where the vectors of the previous dance positions are provided to predict the next dance step.

Here we transform our input vectors to another list of vectors appropriate to create context of dance vectors to generate the next dance vector. We take a lookback of 25 to create a vector of vectors consisting of 25 sequential dance vectors. This dance vector's output corresponds to the next dance vector is the 26th dance vector in the sequence. This is done sequentially to get a dance context vector for each dance vector in our corpus. Our corpus consists of 4500 dance vectors. After transformation and padding we get 4500 training dance context vectors and their corresponding outputs. These vectors are also normalized and shifted about the mean to center the vectors and get vectors apt for our deep learning model. The equations for normalization are as follows:

X -= np.amin(X, axis=(0, 1))

X /= np.amax(X, axis=(0, 1))

Here X is the training vector and the vectors are normalized between 0 and 1 centered about their

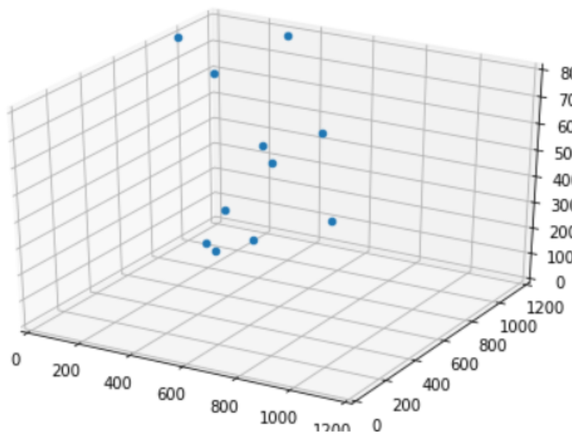


Fig 2.3 Sample output vector for a random input from training data

means to get a uniform distribution of dance vectors.

The model summary is given in Fig 2.2. The training vectors are provided to the deep neural networks. Various parameters are tuned to get optimum accuracy for the model. Various features of the model like loss functions, optimizers, decay rate, learning rate and dropout layers are tweaked to get optimal results. Table summarizes the results for some of the chosen parameters. While testing the context vectors are built by appending the newly generated output vector to the next training context dance vector.

The results of this model are quite poor and despite tweaking various parameters the model is only able to predict with an accuracy of 18.2%. The model is not able to detect or generate the human shape corresponding to the inputs provided. The model is also not able to generate appropriate dance steps with continuity in motion. Fig 2.3 shows a sample output for an input context vector. The data points are concentrated in a region but do not form a human form. The model is not able to capture the human form. The main reason for this can be the following.

The size of the dataset we have taken is extremely small and is not enough to train the deep neural network model we have built. Enormous amounts of data need to be fed for the model to even start recognizing a human form let alone a dance position. This model is not recommended for small corpus. The training times of this model are also very large. To train our dataset, the model took about 6.5 hours for 200 epochs. The loss function over epochs is shown in Fig 2.4, as we can see after the initial drop, the loss drops very slowly and requires a huge number of epochs to even reach an accuracy of 18%.

Another reason for the poor performance of deep neural networks to form continuous motions may be because of loss of connectivity among training data. Neural networks consider each training example as independent training example and there may be a loss of connectivity when erroneous inputs are considered

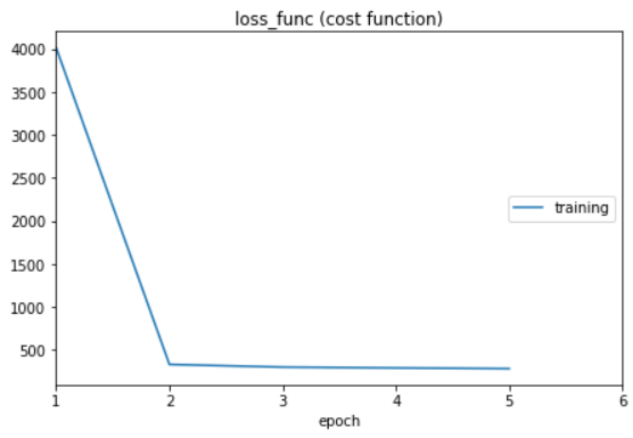


Fig 2.4 graph of loss function

in the context and the model may be severely misled as we are appending the output subsequently to predict the next step. This provides an ability to diverge from the learned target function to diverge step by step to produce erroneous outputs accumulated over time to completely form a wrong context vector for testing in the x^{th} step.

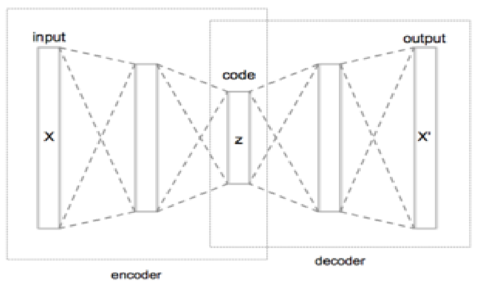


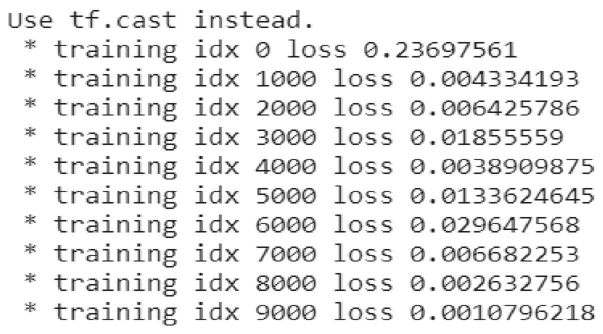
Fig 3.1 General autoencoder architecture.

Autoencoders

Autoencoders are neural networks that use two neural networks where one is the mirror of the other [5]. General architecture of an autoencoder is shown in Fig

3.1. The output of one network is the input to the second network which is concatenated to the first. The first neural network is called the encoder which takes the input and compresses it to a lower dimensional latent space. In our case we will take our original input dance vector and feed it to our encoder decoder network. The encoder takes this dance vector and encodes it to a hidden representation. The decoder takes this hidden representation as an input and tries to reconstruct output similar to the input. However latent variables z is parameterized by a generative network. This produces a variation which results in slightest different generative outputs which aren't exactly like the inputs.

The architecture of our model is summarized in Fig 3.2. Like our previous approach each input vector consists of the compressed version of context dance vectors. We keep a stack of predictions which grows over time as outputs are seen. The outputs are stacked on top for each step. The context vector then includes the top look back number of vectors to compress and form the next input. The dance vectors are encoded by the encoder network. The encoder network uses dense layers with LeakyReLU activations. The decoder then takes this embedding and tries to reconstruct input to get the dance vector with variation. The decoder network is a mirror image of the encoder with slight variations. It also uses a LeakyReLU activation function and dense layer for the decoding task.



```
Use tf.cast instead.
 * training idx 0 loss 0.23697561
 * training idx 1000 loss 0.004334193
 * training idx 2000 loss 0.006425786
 * training idx 3000 loss 0.01855559
 * training idx 4000 loss 0.0038909875
 * training idx 5000 loss 0.0133624645
 * training idx 6000 loss 0.029647568
 * training idx 7000 loss 0.006682253
 * training idx 8000 loss 0.002632756
 * training idx 9000 loss 0.0010796218
```

Fig 3.3 shows losses for different epoch checkpoints.

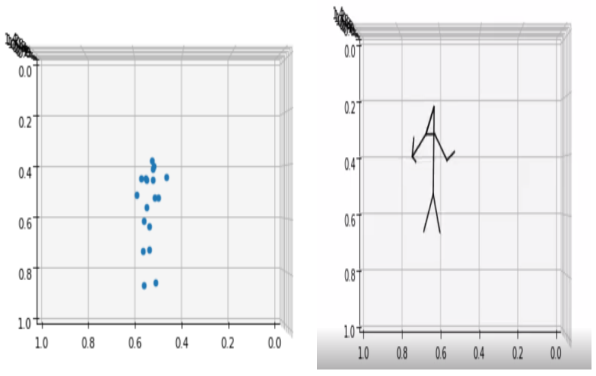
The model is then trained over the training examples for 10000 epochs and the loss function varies with epochs as shown in Fig 3.3. We can see that the loss values are very jumpy and stabilize as the number of epochs increase.

Results

The output of the model represents some form of human shape and dance position. The model learns the relationship between input and output human forms and learns to produce output vectors that resemble human form. Fig 3.4 shows an output of the input frame during testing.

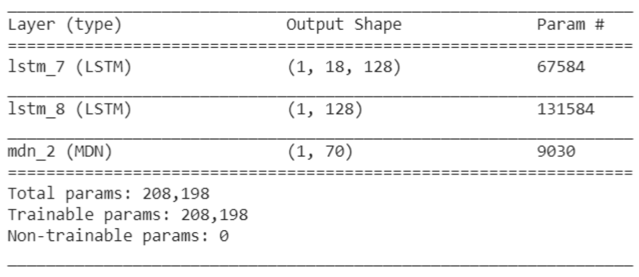
Even though the auto encoder learns to represent the human form output dance vector, the subsequent dance vectors have no relations among them. The auto encoder doesn't learn how to connect dance vectors to produce smooth continuous flow. The output video is dodgy and repetitive.

This may mainly be because the model doesn't consider connectivity among training examples and only learns how to reconstruct input vectors with variations. The connectivity among sequences is not established as the autoencoder considers the training examples as independent and doesn't remember the state of the context. As a new training example is seen, the current state is lost and hence nothing held in memory to connect to the next dance vector.



(a) (b)

Fig 3.4 (a) Output dance vector (b) Vector values connected by edges using affinity map



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (1, 18, 128)              67584
_________________________________________________________________
lstm_8 (LSTM)                (1, 128)                  131584
_________________________________________________________________
mdn_2 (MDN)                  (1, 70)                   9030
=================================================================
Total params: 208,198
Trainable params: 208,198
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.1 Architecture of RNN LSTM network

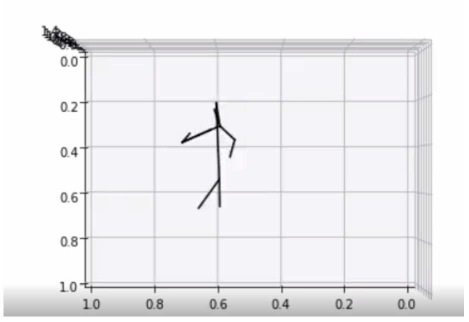


Fig 4.2 Output of sample frame

Recurrent neural networks

Recurrent neural networks are architectures used to get state of the art results for time series of various types. The various types include stock market predictions, speech recognition, language models, speech translations, music encoding etc. They are the only networks with internal memory and are able to capture dependencies in a sequence of related inputs. In

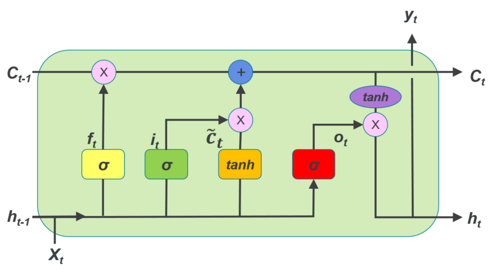


Fig 4.3 Structure of an LSTM memory cell

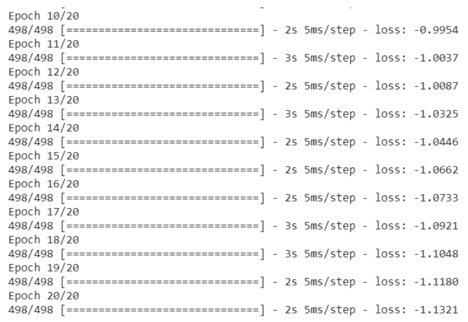
recurrent neural networks the information cycles in loops within the network to make future predictions based on previous data and current input. This is the most important property because the sequence of data contains crucial information about what is coming next, which is why an RNN can do things other algorithms can't. This architecture seems very apt for our task as we will require architectures that relate the previous dance vector to the current dance vector which is the input to provide the next dance vector which will continue the previous and current dance vector to perform smooth dance movements. Since we are dealing with complex data with multidimensional inputs as the dance vectors, we will require a deep RNN to try and predict a dance vector. Even though practically RNNs must hold all information related to all previous inputs, they only hold short term memory due to vanishing gradient problems. They are not able to capture long term dependencies. In dance videos especially of Bharatanatyam the dances usually tend to enclose a concept or story within them. These concepts may not be directly closely placed within the video; hence the network must learn the long-term dependencies and expression and concept of the overall dance video to produce continuing dance steps that follow the same.

To solve this problem, we use LSTM cells which stand for long short-term memory that allows the network to selectively store important information along the previous inputs. It solved the RNNs vanishing and exploding gradient problem. It enables the network to capture long term memories. LSTMs are stable over long training runs and can be stacked to form deeper networks without loss of stability . The LSTM controls the signal flow through the hidden memory state to ensure only important information is captured and the unnecessary information forgotten based on input vectors. An LSTM has three gates that enable it to read, write and forget information from its memory the assigning of the importance of the hidden information also happens through the algorithms based on the nature of previous inputs and outputs during training.

The three gates are input gate, output gate and the forget gate. Figure 4.3 showcases an LSTM cell. The LSTM cell will help our network to capture long term dependencies among dance position vectors enabling it to capture concepts of dance and expression.

Training

The input vectors of our initial corpus are modified in the same way we had modified it for autoencoders and deep neural networks. The lookback is specified to form the context dance vector and normalized to center the vectors about mean and form values between 1 and 0. To help build the required model the LSTM class is imported from https://github.com/cpmpercussion/keras-mdn-layer . The class is instantiated and initiated with



```
Epoch 10/20
498/498 [==============================] - 2s 5ms/step - loss: -0.9954
Epoch 11/20
498/498 [==============================] - 3s 5ms/step - loss: -1.0037
Epoch 12/20
498/498 [==============================] - 2s 5ms/step - loss: -1.0087
Epoch 13/20
498/498 [==============================] - 3s 5ms/step - loss: -1.0325
Epoch 14/20
498/498 [==============================] - 2s 5ms/step - loss: -1.0446
Epoch 15/20
498/498 [==============================] - 2s 5ms/step - loss: -1.0662
Epoch 16/20
498/498 [==============================] - 2s 5ms/step - loss: -1.0733
Epoch 17/20
498/498 [==============================] - 3s 5ms/step - loss: -1.0921
Epoch 18/20
498/498 [==============================] - 3s 5ms/step - loss: -1.1048
Epoch 19/20
498/498 [==============================] - 2s 5ms/step - loss: -1.1180
Epoch 20/20
498/498 [==============================] - 2s 5ms/step - loss: -1.1321
```

Fig 4.4 Loss over last 10 epochs

appropriate values to train our network. The training data is then fed to the network and trained over 20 epochs to reduce the loss. The performance over the last 10 epochs is shown in Fig 4.4.

RNNs even with LSTMs still sometimes give exploding gradient problems leading to the loss function turning into Nan. To solve this, we do gradient clipping and reduce the batch size to get 20 epochs of proper reduction of loss. Various parameters are tweaked to get optimum results such as changing the loss function and optimizers. Generally, it can be shown using a mean square error metric, the output will stagnate and converge to an average output in a significant number of epochs. The network is trained for 20 epochs which takes about one hour. Each batch size is kept minimally to 2 to avoid exploding gradients which goes through 498 training examples to update the weights according to stochastic gradient descent.

The outputs of the network are stacked and provided to the network to generate smooth short dance moves. The output dance vectors are connected using affinity maps built earlier to form the stick figure as given in Fig. 4.2. Additional libraries are used to demonstrate the dance moves. These libraries include matplotlib python libraries and their 3d axes modules and animation modules.

Results

It is seen that when trained over one long dance video of 20 minutes containing 498 frames or input dance vectors the networks learn the human form as well as short sequences of dance steps of continuous movements. An example output for the input frame is provided in figure 4.2. Our small network with small training data is only able to produce small relevant Bharatanatyam steps ranging over 2 or 3 seconds due to limited resources. Most of the output dance steps are based around the most general dance step in Bharatanatyam shown in Fig 4.5. The dance moves are small dance moves that are frequently performed in the video with some variation.

Future work

Better training with more resources: Due to limited computational resources and complexity of extraction of human poses we were only able to train our model with limited resources. The model performs the required task for a small dataset and can perform much more complex dance steps, encoding long dependency concepts given enough training data and computational power.

Audio synchronization: We intend to encode audio of the training video to match the beats of the audio with the dance vectors of the video. We would need to match the frame rate of the video human posture encoding with the sample rate of the audio encoding to produce training vectors. The audio vectors and the dance posture vectors need to be encoded to form the final training vector to be fed to the recurrent neural network with LSTM cells. This will help produce dance vectors appropriate to the music given. More complex network architecture may be needed to accommodate this multi feature multidimensional input vector to predict the output dance vector for the given beat of the audio.

Multiperson dance choreography: The project can be extended to extract multiple people and their dance vectors. The synchronized dance steps of a group of dancers can be learned along with their interaction. This would require a very large corpus of data and computational resources to train but has a lot of scope as it can be used not only to choreograph dance moves but also fighting sequences and motions of humans for strategic wrestling etc.

Conclusions

The given task is a very complex task of generating a dance sequence given a plethora of training corpus. The system needs to learn to generate smooth dance moves in parallel to capturing the concept exhibited by the dance. It involved multiple dimensions and relations among input vectors. This task requires a very high amount of computational resources and training data to even exhibit decent dance steps. Three models were tested to generate such a dance sequence. The deep neural network given the context dance vector was neither able to represent the human form nor able to generate a dance sequence. Despite a long training time and tweaking of parameters the output dance vector was arbitrary and sparse motion. The second approach was to use an auto encoder which was only able to represent the human form but not generate smooth dance steps. This was mainly because it didn't have the power to connect the dance steps and only tried to regenerate the input dance vector. The third

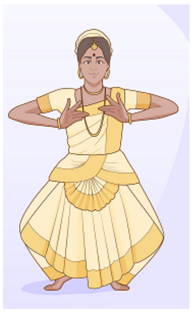


Fig 4.5 Basic posture of Bharatanatyam

and last approach used recurrent neural networks that connected dance vectors and had LSTM cells to capture long term dependencies. This model was able to capture the human form as well as produce small sequences of dance with smooth continuous motion relevant to the input dance vectors. We can conclude that given much more training data and higher complexity of the model the network can generate complex steps and capture the concept of the dance.

References

[1] OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields Zhe Cao, Student Member, IEEE, Gines Hidalgo, Student Member, IEEE, Tomas Simon, Shih-En Wei, and Yaser Sheikh

[2] Jadhav, Sangeeta & Joshi, Manish & pawar, jyoti. (2015). Art to SMart: An Automated BharataNatyam Dance Choreography. Applied Artificial Intelligence. 29. 10.1080/08839514.2015.993557

[3] Two Improved Continuous Bag-of-Word Models Qi Wang, Jungang Xu, Hong Chen, Ben He School of Computer and Control Engineering University of Chinese Academy

[4] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, Kuala Lumpur, 2015,pp730-734.10.1109/ACPR.2015.7486599
keywords: {image classification;neural nets;image classification;Krizhevsky;imagenet large scale visual recognition challenge;deep convolutional neural networks;D-CNNs;ImageNet datasets;CIFAR-10;VGG-16 network;batch normalization;Convolution;Training;Error analysis;Computational modeling;Neural networks;Acceleration;Data models},
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7486599&isnumber=7486438

[5] B. O. Ayinde and J. M. Zurada, "Deep Learning of Constrained Autoencoders for Enhanced Understanding of Data," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 9, pp. 3969-3979, Sept. 2018.
doi: 10.1109/TNNLS.2017.2747861
keywords: {Computer architecture;Feature extraction;Encoding;Training;Decoding;Data mining;Data models;Deep learning (DL);part-based representation;receptive field;sparse autoencoder (SAE);white-box model},