Human Motion Reconstruction from Action Video Data Using a 3-Layer-LSTM

Jihee Hwang
Stanford Computer Science
jiheeh@stanford.edu

Danish Shabbir
Stanford Electrical Engineering
danishs@stanford.edu

Abstract

Data driven motion generation is a challenging problem. Typically the model is trained using motion capture (MOCAP) data, which is limited in scope and diversity. Solving this problem using motion data extracted from Youtube videos would enable motion generation pipelines to feed from a wider range of action data, allowing for greater variability in the actions generated. Human action is expressed through a sequence of bodily movements. Hence, in this work we explore three stages for the motion synthesis pipeline: first by extracting human pose from video data with Part Affinity Fields method by [1], and then by training a 3-Layer Recurrent Neural Network (RNN) that learns to generate human-like motion for a given class from annotated video data, and finally we explore a simple synthesis of motion from multiple classes. Ultimately, this approach could enable robust generation of a wide range of motions, capturing the the subtle and fine variations of human movement expressed in videos that populate the Internet to artificially produce life like motion.

1. Introduction

Understanding human motion and utilizing such knowledge to animate natural 3D models is a very crucial part of any animation or game development pipeline. However, automatic motion generation from data is still an incredibly challenging problem, as it is difficult to effectively parametrize motion information and learn the underlying structure of human actions due to lack of large readily available dataset and

Typically motion capture data is used for motion recognition and generation, but motion capture data is expensive to gather. On the other hand, the internet is saturated with video data of human activity. In present work, we investigate three stages of the motion synthesis pipeline. The first stage is to extract human pose from video data. The second stage is to learn a representation for a given action class from video data with joint positions from step 1. And our final step synthesizes generated motion from different action classes, for example, a motion sequence that displays walking and waving.

In this work, we seek to learn a representation for a given action class from labeled video data. Our method, used in conjunction with state of the art labeling techniques, can allow motion generation pipelines to feed in from a broad class of activity data available on the internet, opening up possibilities for robust automatic motion generation.

1.1. Approach

Motion is characterized by a sequence of movements. To capture the temporal information in our data, we will base our model on a Recurrent Neural Network (RNN). RNN cells are capable of preserving states from previous snapshot in time, hence they are suitable for sequential prediction tasks. As such, RNN models are popular for text generation, where text is seen as a sequence of individual characters.

2. Related Work

2.1. Background

Early approaches to motion generation involved manual methods such as setting the Degrees of Freedom of human joints in all key frames and generating continuous motion by interpolation[8]. While relatively simple, such a method requires hand crafting an algorithm for each type of motion and the results were rarely life like. Similarly, Physics-based approaches have also been introduced to generate motion by solving ODEs on the torque and trajectory of each joint position [2]. These models lacked detail and captured no individuality, thus unsuitable for subtle character creation.

2.2. Extracting Human Pose from Video Frames

Articulated human pose estimation from video frames is a long-standing problem in computer vision. In classical approaches, the pose is estimated by spatially correlating parts of the body, expressed as a tree-structured graphical model — what we call a 'skeleton' of joint positions in this work — with kinematic priors that couple connected limbs.

However, such methods are susceptible to double counting errors.

Recently convolutional architectures have been a popular approach towards this problem. We ran Convolution Pose Machines as described in [9] on our dataset and achieved the following articulated pose for a still frame from our dataset:

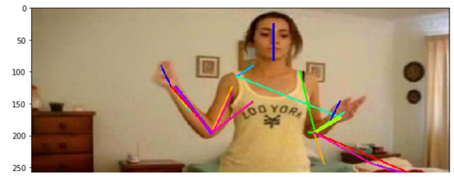
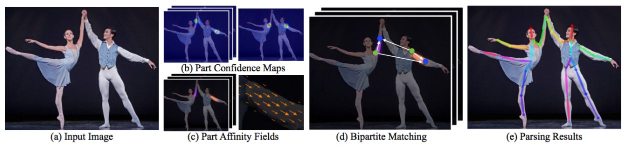


Figure 1. Result from CPM pose detection, as described in [9] on a still frame from our JHMDB dataset

This leaves more to be desired. 2-D Pose Estimation using Part Affinity Fields, as described in Cao et. al [1], was the winning method in 2016 MSCOCO Keypoints Challenge. The overall pipeline from is illustrated in the figure below.



This method first computes confidence maps on the input image to detect body parts– each confidence map is a 2D representation of the belief that a particular body part occurs at a pixel location. Simultaneously, Part Affinity Fields (PaF) are also computed on the input image. PAF is a novel feature representation introduced in this work– each part affinity is a 2D vector field for each limb, this feature representation allows us to encode location as well as orientation information across different body parts. Then We ran the open source code on test images from our dataset and verified the results promised in this work.

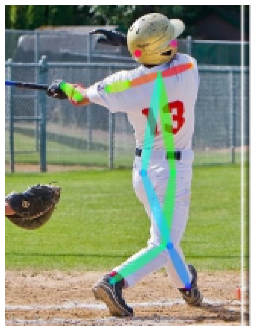


Figure 2. Results from [1]

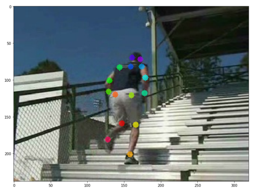


Figure 3. Evaluated on test frames from JHMDB dataset

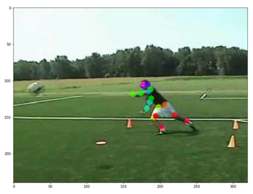


Figure 4. Evaluated on test frames from JHMDB dataset

This is a strong result on 2-D pose extraction and solves the first half of our problem, so instead we focused on generating human motion from these 2D poses using a recurrent network model.

2.3. Recurrent Network Models for Human Dynamics

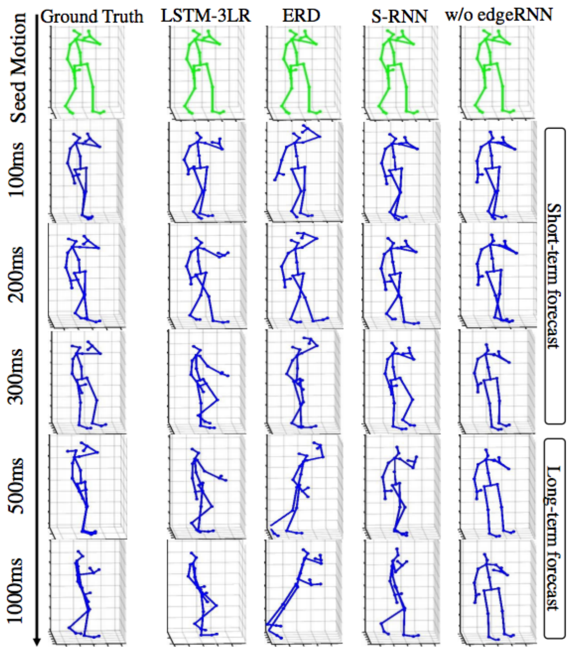
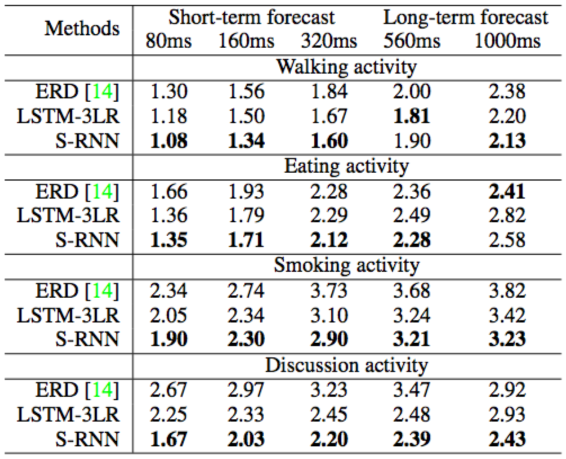


Figure 5. A comparison of RNN Models for Human Dynamics trained on MOCAP data, as presented by Jain, et. al [5]

There has been considerable interest in RNNs for human motion forecasting, but most of this work has been done on MOCAP data. This figure illustrates the performance of various RNN architectures trained on 3-dimensional joint positions from the H3.6m motion data set. Existing implementations compared in the diagram include S-RNN, ERD, and a 3-Layer LSTM network [5, 3].



Methods	Short-term forecast			Long-term forecast	
	80ms	160ms	320ms	560ms	1000ms
	Walking activity				
ERD [14]	1.30	1.56	1.84	2.00	2.38
LSTM-3LR	1.18	1.50	1.67	**1.81**	2.20
S-RNN	**1.08**	**1.34**	**1.60**	1.90	**2.13**
	Eating activity				
ERD [14]	1.66	1.93	2.28	2.36	**2.41**
LSTM-3LR	1.36	1.79	2.29	2.49	2.82
S-RNN	**1.35**	**1.71**	**2.12**	**2.28**	2.58
	Smoking activity				
ERD [14]	2.34	2.74	3.73	3.68	3.82
LSTM-3LR	2.05	2.34	3.10	3.24	3.42
S-RNN	**1.90**	**2.30**	**2.90**	**3.21**	**3.23**
	Discussion activity				
ERD [14]	2.67	2.97	3.23	3.47	2.92
LSTM-3LR	2.25	2.33	2.45	2.48	2.93
S-RNN	**1.67**	**2.03**	**2.20**	**2.39**	**2.43**

Figure 6. Error comparison between motion prediction results of RNN models, as presented by Jain, et. al [5]

In present work, we take inspiration from this approach and extend the RNN work to generate motion data by training only on videos data labeled with joint positions.

3. Problem Statement

Our main challenge is to design a network that can robustly generate action of a certain class using only motion data extracted from videos. Using video data would allow us access to a wider range of semantic contextual information, which would help us to specify the type of motion that can be generated.

3.1. Dataset

For our investigation, we will work with the JHMDB dataset of videos labeled with joint positions. Joint-annotated Human Motion Database (JHMDB) contains 928 clips comprised of 21 action categories from the HMDB51 dataset. Each frame is annotated with 15 joints, providing scale, pose, coarse viewpoint, and dense optical flow for the humans in action. [6].

The dataset provides comprehensive coverage of the human silhouette with joints. The data are split into the following action categories: brush hair, catch, clap, climb stairs, golf, jump, kick ball, pick, pour, pull up, push, run, shoot ball, shoot bow, shoot gun, sit, stand, swing baseball, throw, walk, wave.

3.2. Evaluation Metric

We split the input action video data into a training set and a test set before training. Then we generated motion for a certain action class, such as 'clap', for 40 frames by feeding in the first 10 seed frames of an action video chosen from the test set, which means the seed frames are previously unseen by the network. We then calculate the mean-squared loss between the true joint positions and the generated joint positions at each frame, which is identical to how loss during the training stage is calculated.
It is important to note, however, that the goal of this project is not to *predict* the next frames of a certain video, but to robustly generate a generalized motion for a given action class. The nature of our problem makes it difficult to quantitatively evaluate generated data. This evaluation metric is simply a way to assess the general performance of our method.
Additionally, we will include generated samples for qualitative evaluation with our final submission.

4. Technical Approach

4.1. RNN

Human motion is best modeled as sequential data, as joint positions in a video frame are dependent on joint positions in earlier frames. Recurrent Neural Networks or RNNs are a family of neural networks for processing sequential data. A sequence of vectors is processed by applying a recurrence relation at each time step. In this way, an RNN can have a memory of the data ingested earlier in the sequence.

4.2. Long Short-Term Memory

Long Short-Term Memory (or LSTM for short), is a type of RNN originally conceived in the 1997 by Hochrieter and Schmidhuber, to improve the gradient flow of existing RNN networks. Unlike regular RNN cells, LSTM cells maintain 4 different 'gates' that control the flow of the gradients. This allows the network to maintain its stability even with multiple layers. It has become an industry standard, and thus we will be primarily using LSTM cells as opposed to vanilla RNN structures.

Even though RNNs perform well on sequential data, they have trouble capturing dependencies over longer sequences. The four gates of an LSTM allow us to control how much new input to take and how much of the previous hidden state to forget.

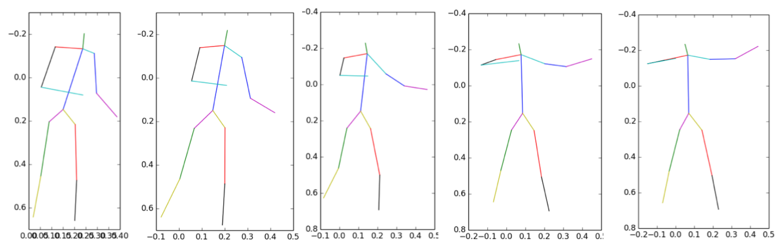


Figure 7. Every 5th frame of 'shooting bow' action animated with our module

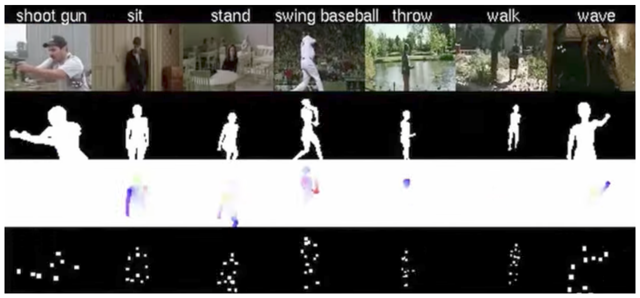


Figure 8. Example of different action classes and their motion notations given by the JHMDB dataset

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

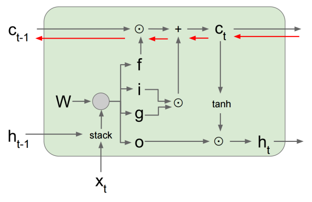


Figure 9. Four Gated LSTM cell and the associated data flow, from CS231N

5. Experiment Setup

5.1. Data Pruning

We were able to extract joint position information from the JHMDB motion dataset. For better results, we decided to preprocess our dataset and use videos that meet the following criteria: i) contains only one person, ii) facing one direction per action category (i.e. we pruned videos by orientation), iii) and is longer than 40 frames for a given action instance. However, we later found that the direction of the action does not affect the result of the generation motion in a significant manner, as will be discussed later.

5.2. Animation Module

We developed a simple animation module for testing purposes (Figure 4). The module is able to take a raw $2 * 15 * frames$ matrix into a visual representation of joint movements. This was done using pre-specified data from JHMDB on joint connectivity.

5.3. Baseline

As a baseline, we implemented a one-dimensional sequential RNN network with 3 layers, which effectively amounts to 120 layers as most motion data instance is comprised of 40 frames. The structure was benchmarked from an existing char-rnn implementation, modified accordingly to incorporate motion data with a much larger domain for the resulting probability density function. Because char-rnn takes one-dimensional time series inputs, each dimension was trained independently of each other.
Our final loss for each dimension overfit very quickly, as it reached 0.001 within only 50 epochs for all joint dimensions. We believe the biggest reason is that each joint is being trained independently of one another, which oversimplifies the problem. To address this issue, in our final network we used multidimensional LSTM cells where the weights can represent relationship between each joints.

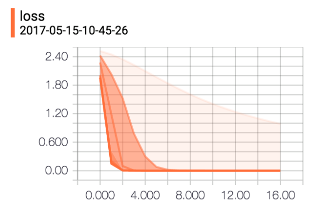


Figure 10. Loss curve for char-rnn run on the dataset

6. Training

6.1. Network Architecture

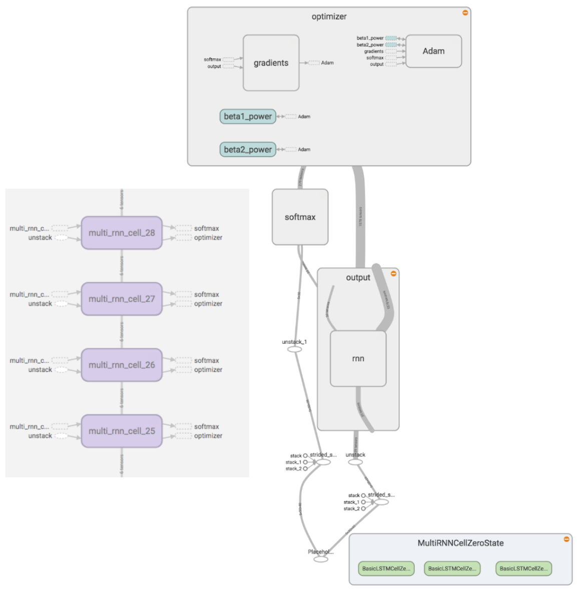


Figure 11. Our final network architecture

Our final network design involves a 3-Layer-LSTM network structure implemented in Tensorflow. The LSTM cell has a *tanh* activation function, with a total series of 39 cells in order to generate 40 timesteps of human motion. Each input is sized *batch_num* $*$ *30* $*$ *40*, and all x, y positions of each 15 dimensions (total 30) can be taken in at once, allowing the network to learn the relationship between each joints for better preservation of spatial information. Prediction at each timestep produces a mean-square loss with the ground truth joint positions, which are then fed to an Adam optimizer.

The nature of our dataset made it difficult to generate batches. Traditional RNNs are trained in a way so that each batch is taken from a random window along a single long sequence; however, our data is comprised of each action instance, 40 frames each. Thus, during each epoch, a batch_num random selection of action instances was chosen for training.

6.2. Hyperparameter Optimization

We tuned the hyperparameters across the following dimensions of our network: learning rate, hidden state size, and number of LSTM layers. The model performance improved as we increased state size and number of layers. By weighing tradeoffs between computational efficiency and highest achievable performance, we settled on a model with 3 layers. In conclusion, we found the most optimal hyperparameters to be state size of 200 and learning rate of 1e-3.

7. Results

7.1. Training Result

With Orientation Pruning

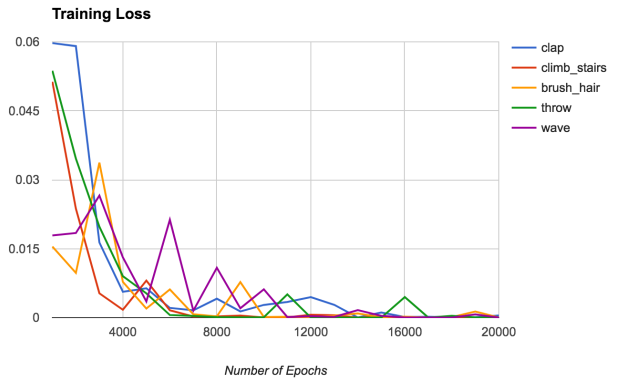


Figure 12. Training loss curve with orientation pruning

This is the loss curve corresponding to our initial training pipeline, which includes the orientation pruning step, detailed in the next section. Over a course of around 2 hours and 30000 epochs, the loss decreases significantly down to around 1.83e-04 in average. Notice some classes start off with higher losses than others, such as *throw* over *wave*; we believe this is due to higher intra-class variation within certain motions than others, as there are more different kinds of throwing motions than waving motions within the dataset. However, we observe that after enough training, all of the action classes converge to a fairly minimal loss.

7.2. Sampling Result

Action Class	**Train Error***	**Avg Test Error**
shoot_bow	2.54e-04	4.62e-03
climb_stairs	6.00e-07	2.02e-04
walk	4.87e-04	2.95e-03
clap	4.65e-04	2.87e-03
brush_hair	3.14e-05	2.17e-02
throw	1.50e-05	4.19e-04

Figure 13. Train and test error comparison chart, on orientation pruned data

Our numeric sampling evaluation was carried out through the methodology explained above in the **3.2 Evaluation Metric** section. While the network does seem to be generally overfitting judging from the lower train errors compared to test errors, a certain degree of overfitting is unavoidable given the low number of training data and high intra-class variation within each action class. As mentioned before, the purpose of this project is to derive a generalized motion of a certain action, not to perfectly predict next frames of a given video. Thus, a better evaluation can be done in a subjective manner by actually observing the sampled motion sequence.

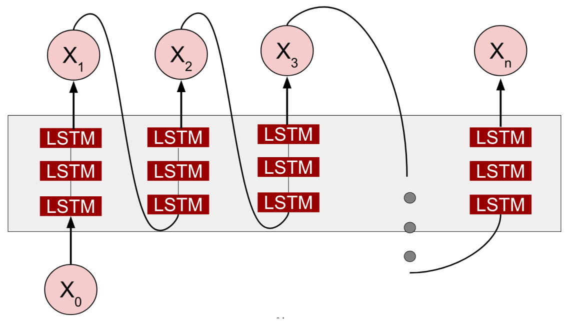


Figure 14. Our sampling pipeline from a trained LSTM network

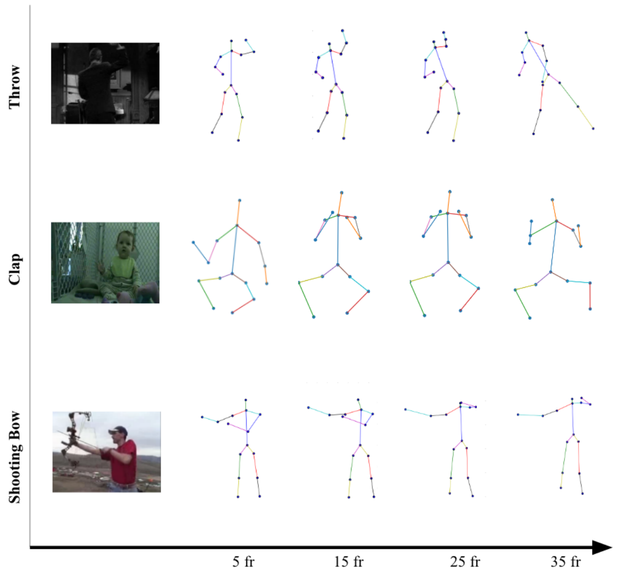


Figure 15. Motion Forecast Results from our method on three different classes, sampled every 10 frames

Using our animation module, for each class we can observe the motion that the network generates given an initial seed of 10 frames from an unseen action video. The result for the three classes *Throw*, *Clap* and *Shooting Bow*—snapshot every 10 frames—is presented above.

The network generates surprisingly accurate recreations of the action even when the initial seed frames remain fairly stationary. The impressive generalization capabilities of the network are even apparent from its ability to effortlessly synthesize motion from different action instances. For example, the test video for the clap motion involves a baby clapping multiple times; however, a large number of clapping motions in the training set involve a single clap. As a result, provided the initial frames of the baby's joint positions, the network develops a lifelike motion of the baby clapping a single time as opposed to multiple times, while maintaining the sitting posture of the baby.

Without Orientation Pruning

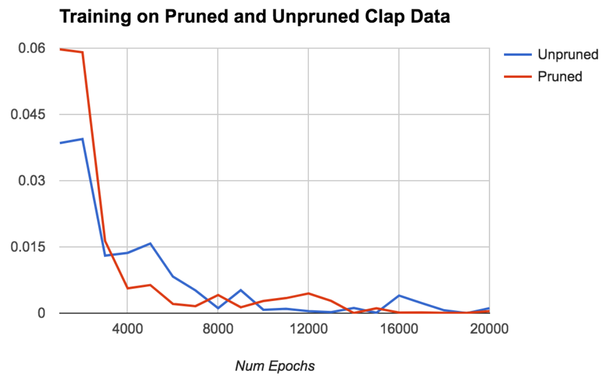


Figure 16. Training loss curve on unpruned motion data

Each action video in the JHMDB dataset comes with an action direction parameter. This is the direction the motion is being filmed in relation to the person in action. For example, a clapping action could be filmed from either the front or the side of the subject. This annotation is especially crucial as we are dealing with 2-dimensional joint position data, instead of 3-dimensional positions generated from motion capture, and our input can drastically fluctuate depending on the camera orientation. Our initial decision to prune motions by orientation, i.e. only consider motions with the same direction, was inspired to help the network more easily establish a single model of motion.

Contrary to our expectations, however, our tests later show that orientation pruning has relatively minimal effects on the training abilities of the LSTM network. In fact, pruned data sets initially have a higher loss — we postulate that this is due to a much smaller number of data in the pruned case. Not surprisingly, pruned data converge much quicker than unpruned data; however, in the long term, there are not many visible differences between the two.

Motion sampling tests indicate that even without pruning, the model is still able to generate life-like actions, even though the performing skeleton is a bit unstable. Amusingly, for the baby clap example that was discussed in the section above, the unpruned model generates a baby's motion that claps multiple times instead of just once. This is likely due to the fact that there were many multiple clap action videos that were filmed from the side, which were then pruned away in the above orientation-pruned model.

7.2.1 Motion Synthesis

While generating motion for each pre-classified action is a difficult problem in itself, a robust algorithm to effectively synthesize motion from different action classes would allow a far wider range of motion generation without having to extensively label each action in a detailed way. Such algorithm could also be potentially crucial in the problem of motion generation from description, made possible by the use of rich textual background information that can be derived from videos along with joint positions.

Baseline Synthesis Pipeline

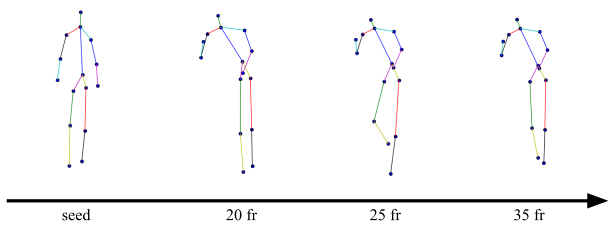


Figure 17. Motion synthesis result from **walk** and **wave**

To demonstrate that our 3-Layer-LSTM is robust to simple motion synthesis methods, we tested our model's capability to combine two existing action classes and generate novel motion trajectories. The synthesis was performed by training two models on two separate actions; then, relevant joints were selected for each action class. For example, in our combination pipeline of **walk** and **wave**, the upper body joints were selected for the wave motion and the lower joints were selected for walk.

We then fed each separate model the same initial frames taken from a test walk video. The final generation is produced by combining motions generated by each model, selecting the relevant joints from each.

The result from this process is as above. The motion is fairly realistic; one can observe that the character is walking and waving at the same time. However, there is a slight offset between the upper and lower body of the character. This is because the upper and the lower body were trained on completely separate models, and the relationship between the two (positional constraint) could not be preserved with the current synthesis algorithm.

To solve such issues, incorporating a more sophisticated synthesis pipeline is unavoidable, of which the potential approaches are discussed below.

S-RNN

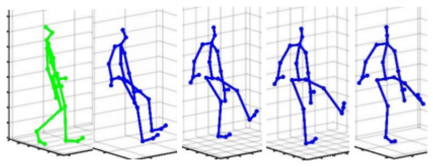


Figure 18. Hybrid motion of lifting a leg and jumping at the same time, a generation result from S-RNN as described in Jain, et. al [5]

A work by Jain et al.[5] introduces s-RNN, short for Structural-RNN. The idea is to represent the spatio-temporal graph describing the problem as a mixture of RNNs. Spatiotemporal graphs (st-graphs) are a general tool for representing such high level structure. The nodes of the graph typically represent the problem components - in our case joints on the body - while the edges capture their spatio-temporal interactions. This way, the relationship between each joint movements can be preserved, leading to a more realistic synthesis result.

Motion Parametrization

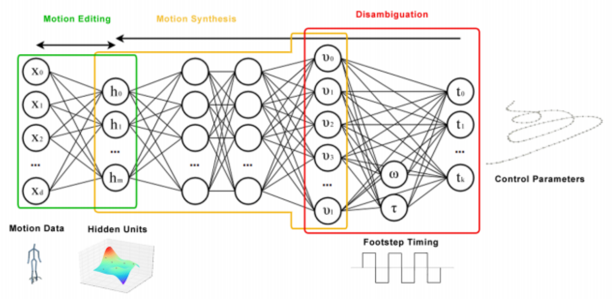


Figure 19. High-level motion parameterization pipeline as described in Holden, et. al [4]

The work by Holden, et. al [4] produces extremely realistic motion generation results from high-level motion parameters, such as generation walking motion given the trajectory of the character. By designing a separate network structure to learn motion parameters, a more robust motion synthesis method would be made possible.

8. Discussion

8.1. Conclusion

Our 3-Layer-LSTM network is an extremely robust implementation that can generate life-like recreations of a certain action class, trained only on 2-D information derived from videos. While motion data from videos can be extremely noisy due to low detail, high sensitivity to action orientation and inherent imperfection in joint detection algorithms, our network is able to construct a stable, smooth motion for any given reasonable initial seed data. We believe this project demonstrates the versatility and the capability of RNNs, performing fairly well even with noisy two-dimensional motion inputs. Given that videos of human motion are not only extremely available and accessible on the net today, but also carry a significant amount of contextual information that can be extracted with computer vision techniques, we hope this project opens the problem of motion generation to be integrated into the wider field of computer vision.

8.2. Steps Forward

One of our next goals is to design a better network to solve this problem. A series of one-dimensional LSTM cells is not suited to handle spatial-temporal information, so we hope to better capture the high level structure of our problem with a Grid-LSTM or s-RNN. An s-RNN might better capture spatial relations between the joints and temporal edges between frames of the video. We will evaluate the comparative performance of the models, if we find a high performing model we could train it further with adversarial loss to generate motion data. Finally, following the initial motivation for this project, we believe taking advantage of existing computer vision and graphics research would lead to a very exciting field of semantic motion generation. Using semantic captioning networks[7] along with motion synthesis implementations[4], one could possibly generate a motion sequence from a given semantic description of such action.

9. Appendix

Github Repo
https://github.com/alarmringing/text2motion

References

[1] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1611.08050*, 2016.

[2] P. Faloutsos, M. Van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. pages 251–260, 2001.

[3] K. Fragkiadaki, S. Levine, and J. Malik. Recurrent network models for kinematic tracking. *CoRR*, abs/1508.00271, 2015.

[4] S. Holden and Komura. A deep learning framework for character motion synthesis and editing. 2016.

[5] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[6] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. *Proceedings of the IEEE International Conference on Computer Vision*, pages 3192–3199, 2013.

[7] Y. Pan, T. Yao, H. Li, and T. Mei. Video captioning with transferred semantic attributes. *CoRR*, abs/1611.07675, 2016.

[8] K. Perlin and A. Goldberg. Improv: A system for scripting interactive actors in virtual worlds. pages 205–216, 1996.

[9] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. pages 4724–4732, 2016.