Short-term Inbetweening of 3D Human Motions

Fabio Neves Rocha[1], **Valdinei Freire**[1], **Karina Valdivia Delgado**[1]

[1]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (USP)
CEP: 03828-000 – São Paulo – SP – Brazil

fnrocha@usp.br, valdinei.freire@usp.br, kvd@usp.br

Abstract. *Creating computer generated human animations without the use of motion capture technology is a tedious and time consuming activity. Although there are several publications regarding animation synthesis using data driven methods, not many are dedicated towards the task of inbetweening, which consists of generating transition movements between frames. A modified version of LSTM, called Recurrent Transition Network (RTN), solves the inbetweening task for walking motion based on ten initial frames and two final frames. In this work, we are interested on the short-term inbetweening task, where we need to use the least amount of frames to generate the missing frames for short-term transitions. We are also interested on different kinds of movements, such as martial arts and Indian dance. Thus, we adapt the Recurrent Transition Network (RTN) to require only the two firts frames and the last one, called ARTN, and propose a simple post processing method combining ARTN with linear interpolation, called ARTN+. The results show that the average error of ARTN+ is less than the average error of each method (RTN and interpolation) separately in the martial arts and Indian dance dataset.*

1. Introduction

Due to human movement has many degrees of freedom and stochastic nature, usually it represents a complex problem to model. For this reason, in order to produce humanoid 3D animations, the use of motion sensors to record the movements of an actor is more common [Meredith et al. 2001]. This technology denominated Motion Capture (MOCAP) is expensive and usually inaccessible [Harvey and Pal 2018]. An alternative solution in the animation, movie and video game industries is to manually produce the animation poses in a video or animation. Every picture in the image sequence is called a frame.

Recent advances in artificial neural networks have shown promising results in animation synthesis specially for the tasks of motion prediction, motion automatic control [Harvey and Pal 2018]. Motion prediction can be defined as a task where the model receives a number of past frames and attempts to predict the following movements [Fragkiadaki et al. 2015, Martinez et al. 2017, Li et al. 2017, Chiu et al. 2019]. Whereas motion automatic control is related to physics-based animations [Peng et al. 2018], an approach more related to dynamics simulation. Another task in animation synthesis is the process of creating intermediate frames between initial and final frames, called *inbetweening* [Betrancourt 2005] or transition generation. The software tools available to automatically generate transitions are usually rudimentary and typically require manual editing which consists of a labor intensive and time consuming work [Ciccone et al. 2019]. This task has only recently been receiving more attention

[Li et al. 2019, Harvey et al. 2020, Harvey and Pal 2018]. A Recurrent Transition Network (RTN), a modified version of LSTM to synthesize transitions in human locomotion, was proposed in [Harvey and Pal 2018]. This network was trained to generate a sequence of intermediate frames of walking motion based on ten initial frames and two last ones, they also take advantage of the cyclical and repetitive pattern inherent to the walking motion.

In this work, we are interested on the short-term inbetweening task, where we need to use the least amount of frames to generate the missing frames for short-term transitions. We are also interested on different kinds of movements, such as martial arts and Indian dance. Thus, we adapt the Recurrent Transition Network (RTN) to require only the two firts frames and the last one, called ARTN. The contributions of this work are: (i) the adaptation of the RTN to produce short-term animations with minimal past context frames; (ii) the proposal of combining RTN with linear interpolation, a common method for generating intermediate transitions in practical applications; and (iii) an experimental analysis in a dataset of martial arts and Indian dance movements that has a small quantity of animations.

Section 2 introduces the main concepts of animation, describes the main animation synthesis tasks and defines the short-term inbetweening task. Section 3 describes the proposed adaptation of the RTN. Section 4 describes our proposal that combines the adapted RTN with linear interpolation. Section 5 describes the dataset used. Section 6 shows the experimental results and Section 7 presents the final conclusions.

2. 3D human motion

An animation is classically defined as a sequence of pictures with progressive small changes that would generate the illusion of motion as they are scrolled over. Nowadays modern animation is produced using computational methods to generate movements in tridimensional space. Let a denote an animation representing a movement with frames $[x_1, x_2..., x_f]$, where each frame x_t is a vector of J joints $[j_{rt}, j_{1t}, j_{2t}, ..., j_{Jt}]$. Each joint j represents a tridimensional cartesian position with components x, y, z in $\mathbf{R}^3$, and the first one j_{rt} represents the hip joint which is considered the global position or root joint that acts as the center of gravity. The most well-known automated animation synthesis tasks are motion prediction, motion automatic control and generation of intermediate transitions.

2.1. Motion prediction

Motion prediction is defined as the inference of future frames based on a past context.

Definition 1 (Motion prediction) *Given a set of frames $[x_1, \ldots, x_f]$ as input, a solution should be able to generate k future frames $\hat{x}_{f+1}, \hat{x}_{f+2}, \cdots, \hat{x}_{f+k}$.*

Some works that focus on this task are [Fragkiadaki et al. 2015, Bengio et al. 2015, Martinez et al. 2017, Butepage et al. 2017, Jain et al. 2016, Li et al. 2017].

2.2. Motion automatic control

In some applications the objective is to generate controlled animations, providing control parameters to guide the algorithmic solution into reaching a certain desired result. These

approaches can be split mainly into three general tasks: physics-based animations, motion manifold solutions and parameter based approaches. Recent physics-based animation approaches such as [Liu and Hodgins 2018] use deep reinforcement learning techniques, creating a sort of physics simulation using mass, center of gravity and other data related to mechanical dynamics. Motion manifold approaches such as [Holden et al. 2015, Zhou et al. 2019, Xu et al. 2019] use techniques to prevent the production of inconsistent outputs. Some works use autoencoders to constrain the algorithmic solution into the subspace of valid motions. Parameter based approaches use customized constraints or external variables, for example, specific joint velocity, center of gravity trajectory or positions. These parameters are applied as a way to guide the model generated output. There are also solutions [Safonova and Hodgins 2007, Heck and Gleicher 2007] that use search algorithms in movement graphs. These techniques involve reproducing and mixing previously generated animations.

2.3. Generation of intermediate transitions (Inbetweening)

Learning transitions is defined as the process of creating intermediate movements between frames. This task is often called *Inbetweening* or *Tweening*.

Definition 2 (Inbetweening) *Given a set of initial frames* $[x_1, x_2, \cdots, x_t]$*, and a set of final frames* $[x_{t+k}, x_{t+k+1}, \cdots, x_f]$ *as input, a solution should be able to generate the intermediate frames* $[\hat{x}_{t+1}, ..., \hat{x}_{t+k-1}]$ *connecting the initial and final frames.*

Studies specifically focused on this task are scarce. Some works use Markov models [Chai and Hodgins 2007], Gaussian processes [Wang et al. 2007] and posterior probability optimization [Lehrmann et al. 2014] and show satisfactory results, but are limited to predetermined actions, such as jumps and specific kicks. Other works use neural networks [Zhou et al. 2020, Harvey and Pal 2018, Harvey et al. 2020]. Harvey and Pal [Harvey and Pal 2018] proposed the Recurrent Transition Network (RTN) which consists of combining LSTM neural networks and autoencoder network. The RTN is trained to generate transitions of walking motion. Based on a fixed number of initial frames called past context ($t = 10$) and two final frames, RTN generates a sequence of intermediate frames of a fixed length. An application of an adversarial recurrent neural network is proposed in [Harvey et al. 2020] also to synthesize transitions of locomotion movements (i.e walking and running) which tend to have a cyclical and repetitive nature. The architecture proposed in [Harvey et al. 2020] is actually more robust in terms of performance than [Harvey and Pal 2018]. However this model requires a significantly larger amount of data due to being a much more complex architecture.

In this work, we are interested on different kinds of movement (e.g. martial arts and Indian dance). Differently than [Harvey et al. 2020, Harvey and Pal 2018], we only consider the two initial frames and the final frame, this is supposed to emulate a scenario of manual animation authoring.

Definition 3 (Short-term Inbetweening) *Given the first frame* x_1*, the last frame* x_f *and the initial global velocity* $g_2 = x_2 - x_1$ *as input, a solution should be able to generate the intermediate frames* $[\hat{x}_2, ..., \hat{x}_{f-1}]$ *connecting the frames* x_1 *and* x_f*.*

Since different kinds of movements (e.g. martial arts and Indian dance) follows different paths from x_1 to x_f, we create a separated model, one for each movement style.

In order to solve this short-term Inbetweening problem, we adapted the RTN architecture [Harvey and Pal 2018] and combined it with an interpolation method. In section 3, the adapted RTN architecture is described and in section 4, the proposed combination is introduced.

3. Adapted RTN Architecture to Short-term Inbetweening

In a simplified way, the RTN [Harvey and Pal 2018] joins a recurrent neural network trained to generate frames with an autoencoder to define valid human movements. In this section this network is adapted to solve the Short-term Inbetweening problem. This adapted architecture is called ARTN (adapted RTN). Figure 1 shows the architecture of the ARTN network at a given time step *t*.

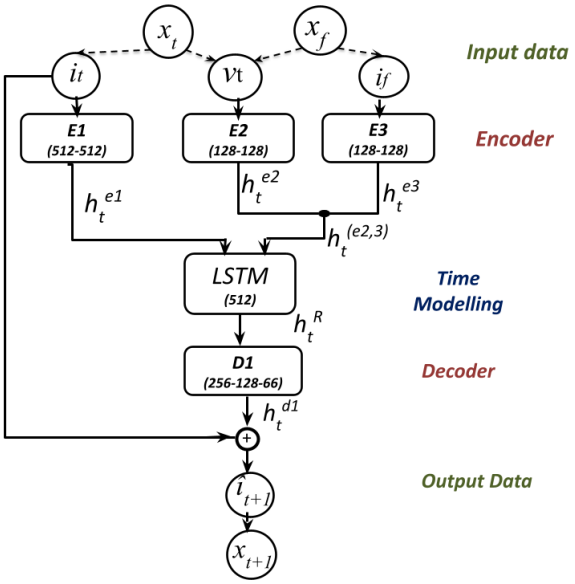

Figure 1. Adapted RTN Architecture (adapted from [Harvey and Pal 2018]).

3.1. Input data pre-processing

The input data pre-processing is carried out in 3 steps. In the first step, the global velocity $g_t = j_{rt+1} - j_{rt}$ is computed in which only the hip joints of the *frame* x_t and x_{t+1} are considered. In the second step, the hip position j_{rt} is subtracted from each joint of the *frame* x_t, which results in $[0, j_{1t} - j_{rt}, j_{2t} - j_{rt}, ..., j_{Jt} - j_{rt}]$. Thus now each joint is a representation of the relative position. The first coordinate is then replaced by the value of the global velocity g_t, obtaining $\tilde{x}_t = [g_t, j_{1t} - j_{rt}, j_{2t} - j_{rt}, ..., j_{Jt} - j_{rt}]$. In the third and last step, the usual normalization procedure is carried out to make the distribution of variables a zero-centered Gaussian, obtaining the vector $i_t = (\tilde{x}_t - \mu_t)/\sigma_t$. The input vector v_t is computed every time step, as the difference between the frame at time t and time f. To perform the inversion of this pre-processing process, we need to save the value of j_{r1}, in order to revert the velocity value g_t to the positional value x_t.

3.2. Components

E1, E2 and E3 represent MLP neural networks with the function of encoders, these encoders act by transforming the input data into another representation of latent space. Temporal dynamics is modeled by the LSTM element, which represents a recurrent neural

network. Finally, D1 represents an MLP network with decoder function, responsible for transforming the data back to the original format. The number of neurons in each layer is in parentheses and the vector $\hat{i}_{t+1}$ is the inferred output frame. Each component of this architecture is described next.

State encoder (E1). The frame corresponding to each instant of time t is transformed into a new representation h_t^{e1} by the E1 state *encoder*, which consists of an MLP network with an input layer of J size, one hidden layer and an output layer with 512 neurons each.

Future context encoders (E2 and E3). Future context encoders work analogous to the state encoder previously explained. The final frame i_f only needs to be processed once, remaining constant throughout the animation. The vector v_t on the other hand must be processed at each iteration. The outputs of E2 and E3 are h_t^{e2} and h_t^{e3}.

Recurrent neural network. The LSTM component is a recurrent neural network with one layer with 512 neurons and has the function of generating the sequence of frames considering the current state and the state to be reached. The output of LSTM is h_t^R.

Decoder (D1). The output h_t^R of the recurrent neural network is the input of the last MLP decoding network D1, which has two hidden layers of 256 and 128 neurons respectively, and an output layer of 66 neurons. The function of D1 is to transform the data to the original representation. The output vector of D1 is h_t^{d1}.

3.3. Output data

The output of the decoder D1, h_t^{d1}, can be considered as the estimate displacement between the instants t and $t+1$. The transformation of $\hat{i}_{t+1}$ into $\hat{x}_{t+1}$ is performed by inverting the pre-processing, i.e., inverting the normalization, adding the velocity and adding the global position.

4. Combining the ARTN Architecture with Interpolation

In the conducted experiments the ARTN architecture did not generate animations that would reach the target frame perfectly. In [Harvey and Pal 2018] a post processing method called target blend was used as a way to correct the gap between the last frame and the target position. However this method of target blend in the ARTN generated unnatural movements as the generated animations would slide abruptly during animations. On the other hand a simple linear interpolation works well in the last frames. So in this article, the target blend method was discarded and we propose the combination of the adapted network with the interpolation. We call this algorithm ARTN+ (ARTN plus interpolation).

ARTN+ first trains the ARTN architecture with the trained set of animations creating a model called NNmodel. Then ARTN+ calls Algorithm Frame Selection (Algorithm 1). This algorithm identifies the frame index, $f/2 \leq index \leq f$, that yields the minimum value of the average error of combining the ARTN with interpolation. Frame Selection receives a training set with n animations, the number of frames f that the animations have and the NNmodel. For each trans index $f/2 \leq trans \leq f$, the algorithm computes the error of using the combination for each animation a_k (line 5) by:

$$error_{a_k} = \frac{\sum_{j=1}^{f} ||\hat{i_j} - i_j||_2}{f} \tag{1}$$

And then the algorithm computes the average error over all the animations in the training dataset (line 8). Each trans index and the average are saved in a map. Then, in line 12, the algorithm get the frame index that has the minimum value between the averages that are in the map.

Algorithm 1 Frame Selection $(trainingDataSet = \{a_1, a_2, \ldots, a_n\}, f, NNmodel)$

1: **for** $trans = f/2, \ldots, f$ **do**
2: **for** $k = 1$ to n **do**
3: Generate $\hat{i}_1$ to $\hat{i}_{trans}$ using NNmodel with i_1 and i_f of a_k
4: Generate $\hat{i}_{trans+1}$ to $\hat{i}_f$ using interpolation with i_{trans} and i_f of a_k
5: $error_{a_k} = \frac{\sum_{j=1}^{f} ||\hat{i_j} - i_j||_2}{f}$
6: $errListAnimations$.add $(error_{a_k})$
7: **end for**
8: avg=average($errListAnimations$)
9: $errMap.add(trans, avg)$
10: **end for**
11: //frame index that yields minimum value of $errMap$
12: $index$=getTransMinAvg($errMap$)
13: return $index$

With the $index$ computed by Frame Selection and the NNmodel, algorithm Transition Generation (Algorihtm 2) is called to generate the intermediate transitions for a test instance a_k.

Algorithm 2 Transition Generation $(a_k, NNmodel, index)$

1: Generate $\hat{i}_1$ to $\hat{i}_{index}$ using NNmodel with i_1 and i_f of a_k.
2: Generate $\hat{i}_{index+1}$ to $\hat{i}_f$ using interpolation with i_1 and i_f of a_k.
3: return $\hat{i}_1$ to $\hat{i}_f$

5. Dataset and Training

The experiments were performed in the Carnegie Mellon University Mocap (CMU) dataset. Creating animations using only the first two frames and the last frame in the original format would not be possible since this dataset consists of long and diverse animations. As a solution, every animation file was subdivided in multiple smaller animations of length f, which were treated as independent observations. Aiming to extract the maximum amount of information from the data provided by the CMU dataset, a sliding window approach was used with padding of $\Delta = 15$ frames. Each frame has $J = 22$ joints, therefore there are $3 \times 22 = 66$ coordinates to represent each frame x_t.

The datasets for specific martial arts and indian dance are notably smaller than the ones used in other studies such as [Fragkiadaki et al. 2015, Martinez et al. 2017, Li et al. 2017, Chiu et al. 2019, Harvey and Pal 2018, Harvey et al. 2020]. Therefore, in order to expand the training and prevent overfitting, a data augmentation approach was used. This approach consists of a simple rotation of every original training animation in angles of 90^0. This implementation was adapted from the code provided in

[Li et al. 2017]. Note that this approach improves the inference on the global displacement (root joint movement), since the joint coordinates are already relative to the root joint and would not be affected by such rotations.

5.1. Weight initialization

A novel initialization method for weights in recurrent neural networks was proposed in [Harvey and Pal 2018]. An auxiliary single layer MLP is used to generate the initial weights of the recurrent neural network. This MLP is responsible for estimating the weight h_0 which corresponds to the time before the initial frame. The goal is to ensure that the initialization of the weights starts from a valid values at time $t = 0$. This network is trained alongside the main recurrent neural network, although in this case inference only happens in the first frame of the animation.

5.2. Scheduled sampling

While generating multiple elements of a sequence, a recurrent neural network will produce an inference $\hat{i}_{t+1}$ using a previous observation $\hat{i}_t$ as input. The model would try to produce the frame i_{t+1} from $\hat{i}_t$, thus generating an output $\hat{i}_{t+1}$ by: $RNN(i_t) = \hat{i}_{t+1} = i_{t+1} + \delta_t$, where RNN represents the recurrent neural network solution and δ_t the error between the inferred output and the ground truth at time $t + 1$.

While generating sequences, it is imperative to worry about the error δ_t accumulation between each time step t. In a test or validation scenario the past observations must be past inferences, however during training this is not always the case. A very useful technique frequently used in temporal supervised learning is to replace the model's past inference $\hat{i}_t$ by the actual ground truth for that given time step i_t [Jordan 1990]. This concept is often called *teacher forcing* and its goal is to ensure that the model inference does not produce large deviations from the original ground truth during training [Williams and Zipser 1989]. This technique should not be overused as it could prevent the model from learning from it's accumulated errors, making it useless in practice. With this consideration in mind, the schedule sampling methodology that uses two kinds of inputs during the training phase was proposed in [Bengio et al. 2015]. The methodology consists of sampling the input based on a probability p of selecting the previous ground truth frame i_t and $1 - p$ for selecting the past inference frame $\hat{i}_t$. At each time step a random choice is made based on p that would decrease over time [Bengio et al. 2015]. In our work, we used a fixed $p = 0.2$ throughout the whole sequence.

5.3. Loss function and hyperparameters

The loss function used is the Mean Square Error: $MSE = \frac{1}{f} \sum_{t=1}^{f} ||\hat{i}_t - i_t||_2$. Training was done using stochastic gradient descent with minibatches of 32. All the experiments were run for 400 epochs with 0.0005 as learning rate.

6. Experiments

The experiments were performed on two movement styles: Martial Arts and Indian Dance with one and two seconds of motion for each style. For one and two seconds of motion we have 30 frames ($f = 30$) and 60 frames ($f = 60$), respectively. The original animations in the data set was divided in 90% for training and 10% for test. Note that even using

90% of the data for training, we do not have the sufficient amount of data for learning. The training set input data has a large number of dimensions and a small amount of observations to train the network. Thus, the process of data augmentation was necessary due to the *curse of dimensionality* [Bellman 1966].

Table 1 shows the number of original and augmented animations in the training set (using the procedure described in section 5) and the number of animations in the test set. It is important to notice that a different model is created for each movement and seconds of motion.

Table 1. Number of animations used in the training set and test set

Movement Style	Training set	Augmented training set	Test set
Martial Arts (2sec)	1721	6884	191
Martial Arts (1sec)	1732	6928	192
Indian Dance (2sec)	2985	11940	331
Indian Dance (1sec)	3006	12024	334

6.1. Analyzing the error over all animations

Table 2 shows the average of $error_{a_k}$ (Equation 1) considering all the animations a_k in each test set using interpolation (INTER), ARTN and ARTN+. The last column shows the average of the $index$ frame found by the ARTN+ to combine ARTN with interpolation. The results show that ARTN+ and ARTN have better average than INTER. Additionally, ARTN seems to fail more in the final frames of Indian Dance than in Martial Arts movements. For the Martial Arts, the $index$ found is equal to f minus 2 or 3 frames in average. For the Indian Dance, the $index$ found is equal to f minus 6 or 7 frames in average. We observe that for Indian Dance (1 sec) 6 frames represents 20% of $f = 30$.

Table 2. Average of $error_{a_k}$ considering all the animations a_k in each test set using INTER, ARTN and ARTN+

Movement style	INTER	ARTN	ARTN+	$index$
Martial Arts (2sec)	13.17	5.43	5.34	57
Martial Arts (1sec)	7.72	3.17	3.11	28
Indian Dance (2sec)	23.86	16.69	16.48	53
Indian Dance (1sec)	9.90	8.40	8.28	24

Figures 2 show the boxplot of the error $error_{a_k}$ obtained by INTER, ARTN and ARTN+ for Indian Dance considering 30 and 60 frames. We can observe that the minimum error excluding any outliers (0th percentile) is almost the same for INTER, ARTN and ARTN+, but the maximum (100th percentile) and third quartile is greatest for INTER. The median of INTER is greater than ARTN and ARTN+.

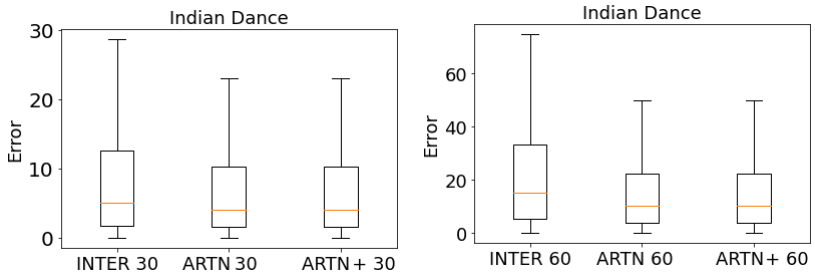

Figure 2. Error obtained by INTER, ARTN and ARTN+ for Indian Dance considering 30 (left side) and 60 (right side) frames.

Figures 3 show the boxplot of the error $error_{a_k}$ obtained by INTER, ARTN and ARTN+ for Martial Arts considering 30 and 60 frames. The behavior is similar with the one described for Indian Dance. However, the range of the error for Martial Arts is less than for Indian Dance. By comparing the error range it is clear that the Martial Arts dataset is an easier problem than the Indian Dance dataset. This can be explained by the fact that the Martial Arts movements into CMU dataset are much more similar among themselves than the Indian Dance movements.

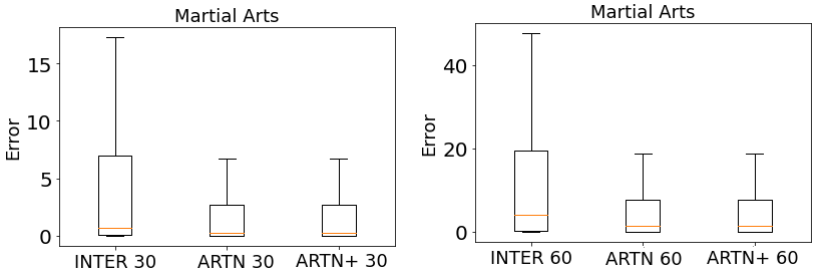

Figure 3. Error obtained by INTER, ARTN and ARTN+ for Martial Arts considering 30 (left side) and 60 (right side) frames.

Although the difference between ARTN and ARTN+ is imperceptible in boxplots, the difference exists in each individual animation, specially in the final frames as we will see in the next subsections.

6.2. Analyzing the final frames

Table 3 shows the average of $error_{a_k}$ (Equation 1) considering all the animations a_k in each test set using INTER, ARTN and ARTN+, but from $j = index$ to $j = f$. The last column shows the average of the $index$ frame found by the ARTN+ to combine ARTN with interpolation. The results show that ARTN+ has better average than ARTN.

Figure 4 shows the last frame found by INTER, ARTN and ARTN+ and the ground truth last frame for a 2 second Indian dance animation. Note that while INTER and ARTN+ reach the same final pose, ARTN does not.

6.3. Analyzing the error at each frame in some Indian Dance animations

Figure 5 shows $error_t = ||\hat{i}_t - i_t||_2$ at each frame t ($1 \leq t \leq 60$) of 4 indian dance animations (sampled from the test set) with two seconds of motion ($f = 60$) using INTER, ARTN and ARTN+, i.e., each curve represents the error along the frames of an animation

Table 3. Average of $error_{a_k}$ considering only the final frames from the index frame onward in each test set using INTER, ARTN and ARTN+

Movement style	INTER	ARTN	ARTN+	$index$
Martial Arts (2sec)	0.34	0.55	0.42	57
Martial Arts (1sec)	0.58	0.63	0.61	28
Indian Dance (2sec)	3.28	5.54	4.63	53
Indian Dance (1sec)	3.05	3.89	3.30	24

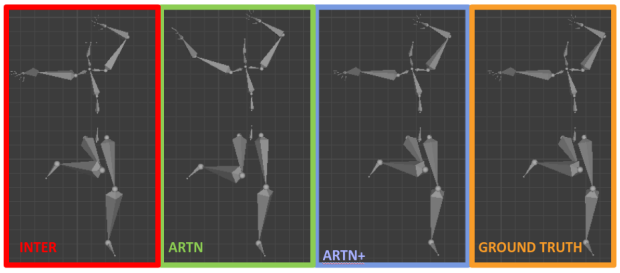

Figure 4. The last frame found by INTER, ARTN and ARTN+ and the ground truth last frame for a 2 second Indian dance animation. The animations were generated using the software Blender3D.

using INTER (red curve), ARTN (green curve) and ARTN+ (blue curve). Figure 6 shows $error_t$ for 4 animations with one second of motion ($f = 30$).

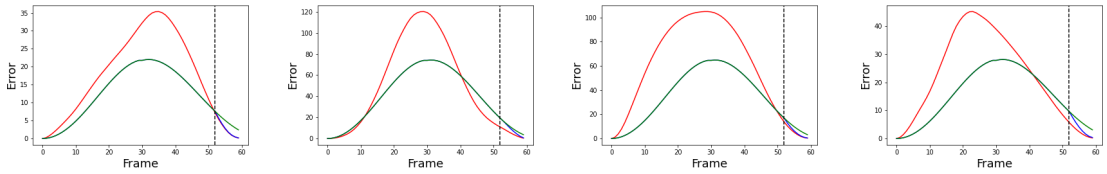

Figure 5. Error at each frame for 4 Indian Dance animations with 60 frames using INTER (red curve), ARTN (green curve) and ARTN+ (blue curve). The dashed black line represents the $index$ frame found by the ARTN+.

The results show that ARTN works better than linear interpolation except in the last frames. ARTN+ combines the best features of both, generating animations that would reach the target frame better than ARTN.

7. Conclusion

This study has shown the capability of the RTN architecture in producing short-term realistic animations using minimal information (the two initial frames and the final frame). The experimental results have also demonstrated that the performance of this neural network model in the final frames tends to be inferior than the naive linear interpolation. Therefore, a post processing method is crucial for the completion of the Inbetweening task. Thus, in this article we have proposed ARTN+ that combines the naive interpolation method with the ARTN architecture and produces better overall results. One fact that requires emphasizing is that the advantage of the proposed solution does not reside

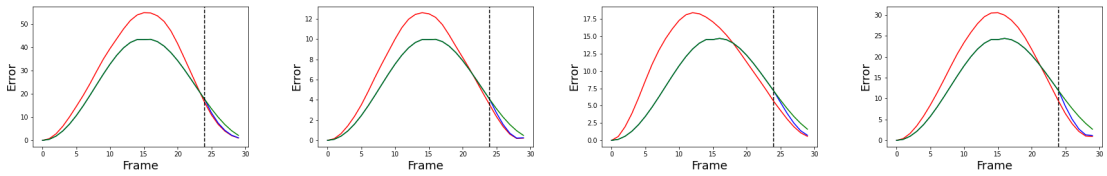

Figure 6. Error at each frame for 4 Indian Dance animations with 30 frames using INTER (red curve), ARTN (green curve) and ARTN+ (blue curve). The dashed black line represents the $index$ frame found by the ARTN+.

only in the performance gain, but rather in ensuring the completion of the transition task without losing general performance. The model comparison boxplots provide evidence that the ARTN and ARTN+ architectures are strongly influenced by the movement style and transition lengths.

References

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

Betrancourt, M. (2005). *The Animation and Interactivity Principles in Multimedia Learning*, page 287–296. Cambridge Handbooks in Psychology. Cambridge University Press.

Butepage, J., Black, M. J., Kragic, D., and Kjellstrom, H. (2017). Deep representation learning for human motion prediction and classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chai, J. and Hodgins, J. K. (2007). Constraint-based motion optimization using a statistical dynamic model. In *ACM SIGGRAPH 2007 papers*, pages 8–es.

Chiu, H.-K., Adeli, E., Wang, B., Huang, D.-A., and Niebles, J. C. (2019). Action-agnostic human pose forecasting. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1423–1432.

Ciccone, L., Öztireli, C., and Sumner, R. W. (2019). Tangent-space optimization for interactive animation control. *ACM Trans. Graph.*, 38(4).

Fragkiadaki, K., Levine, S., Felsen, P., and Malik, J. (2015). Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4346–4354.

Harvey, F. G. and Pal, C. (2018). Recurrent transition networks for character locomotion. In *SIGGRAPH Asia 2018 Technical Briefs*, SA '18, pages 4:1–4:4, New York, NY, USA. ACM.

Harvey, F. G., Yurick, M., Nowrouzezahrai, D., and Pal, C. (2020). Robust motion inbetweening. *ACM Transactions on Graphics (TOG)*, 39(4):60–1.

Heck, R. and Gleicher, M. (2007). Parametric motion graphs. volume 2007, pages 129–136.

Holden, D., Saito, J., Komura, T., and Joyce, T. (2015). Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, page 18. ACM.

Jain, A., Zamir, A. R., Savarese, S., and Saxena, A. (2016). Structural-RNN: Deep learning on spatio-temporal graphs. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 5308–5317.

Jordan, M. I. (1990). *Attractor Dynamics and Parallelism in a Connectionist Sequential Machine*, page 112–127. IEEE Press.

Lehrmann, A. M., Gehler, P. V., and Nowozin, S. (2014). Efficient nonlinear Markov models for human motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1314–1321.

Li, Y., Roblek, D., and Tagliasacchi, M. (2019). From here to there: Video inbetweening using direct 3d convolutions. *ArXiv*, abs/1905.10240.

Li, Z., Zhou, Y., Xiao, S., He, C., Huang, Z., and Li, H. (2017). Auto-conditioned recurrent networks for extended complex human motion synthesis. *arXiv preprint arXiv:1707.05363*.

Liu, L. and Hodgins, J. (2018). Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4):1–14.

Martinez, J., Black, M. J., and Romero, J. (2017). On human motion prediction using recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2891–2900.

Meredith, M., Maddock, S., et al. (2001). Motion capture file formats explained. *Department of Computer Science, University of Sheffield*, 211:241–244.

Peng, X. B., Kanazawa, A., Malik, J., Abbeel, P., and Levine, S. (2018). Sfv: Reinforcement learning of physical skills from videos. *ACM Trans. Graph.*, 37(6).

Safonova, A. and Hodgins, J. K. (2007). Construction and optimal search of interpolated motion graphs. In *ACM SIGGRAPH 2007 papers*, pages 106–es.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2007). Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

Xu, Y. T., Li, Y., and Meger, D. (2019). Human motion prediction via pattern completion in latent representation space. *arXiv preprint arXiv:1904.09039*.

Zhou, D., Feng, X., Yang, X., Zhang, Q., Wei, X., Fang, X., and Yang, D. (2019). Human motion data editing based on a convolutional automatic encoder and manifold learning. *Entertainment Computing*, 30:100300.

Zhou, Y., Lu, J., Barnes, C., Yang, J., Xiang, S., et al. (2020). Generative tweening: Long-term inbetweening of 3d human motions. *arXiv preprint arXiv:2005.08891*.