Spatio-temporal Manifold Learning for Human Motions via Long-horizon Modeling

He Wang [1] Edmond S. L. Ho [2] Hubert P. H. Shum [2] and Zhanxing Zhu [3]

Abstract—Data-driven modeling of human motions is ubiquitous in computer graphics and computer vision applications, such as synthesizing realistic motions or recognizing actions. Recent research has shown that such problems can be approached by learning a natural motion manifold using deep learning on a large amount data, to address the shortcomings of traditional data-driven approaches. However, previous deep learning methods can be sub-optimal for two reasons. First, the skeletal information has not been fully utilized for feature extraction. Unlike images, it is difficult to define spatial proximity in skeletal motions in the way that deep networks can be applied for feature extraction. Second, motion is time-series data with strong multi-modal temporal correlations between frames. On the one hand, a frame could be followed by several candidate frames leading to different motions; on the other hand, long-range dependencies exist where a number of frames in the beginning correlate to a number of frames later. Ineffective temporal modeling would either under-estimate the multi-modality and variance, resulting in featureless mean motion or over-estimate them resulting in jittery motions, which is a major source of visual artifacts. In this paper, we propose a new deep network to tackle these challenges by creating a natural motion manifold that is versatile for many applications. The network has a new spatial component for feature extraction. It is also equipped with a new batch prediction model that predicts a large number of frames at once, such that long-term temporally-based objective functions can be employed to correctly learn the motion multi-modality and variances. With our system, long-duration motions can be predicted/synthesized using an open-loop setup where the motion retains the dynamics accurately. It can also be used for denoising corrupted motions and synthesizing new motions with given control signals. We demonstrate that our system can create superior results comparing to existing work in multiple applications.

Index Terms—Computer Graphics, Computer Animation, Character Animation, Deep Learning

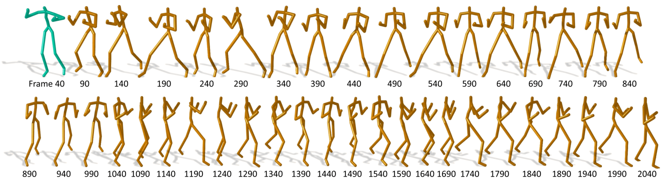


Fig. 1. Long-horizion motion generation: given the first **20** frames, STRNN generates the next **20000** frames (yellow) in an open-loop setting. Only the first 2000 frames are shown here.

1 Introduction

Modeling natural human motions is a central topic for data-driven character animation. It has been argued that natural human motions constitute a motion manifold [1]. Although there is a large body of research attempting to find good representations for this manifold such as Finite State Machines [2], subspace modeling [3] and statistical modeling [4], it is still challenging to find a representative subspace that leads to high-quality modelling.

Recently, some successes have been shown in obtaining a motion manifold using deep learning [1], [5], [6], but designing a good network is difficult. The difficulties lie in the spatial and temporal variances of the motion data. Spatially, unlike images, human motions are parameterized on a graph structure (i.e. the skeleton). The local similarity assumption, upon which powerful networks are built for feature extraction, is very different on skeletons comparing to images. The lack of spatial feature extraction leads to the system converging to a "mean posture" in which the spatial variation of the body movement disappears, as suggested by [6]. To mitigate such problems, Holden et al. [1] employed different *disambiguation networks* to recover different types of motions. Temporally, motions show two layers of variances, the first being multi-modality (i.e. the same motion prefix could lead to multiple motions); the second being dynamics variations (e.g. the same set of postures in a motion but with different timing profiles). When generating motions, under-estimating the variances results in a system that generates motions with "mean posture" [6], while over-estimating them leads to jittery motions. Previous work partially tackles this problem by switching the state of the network [5] based on the foot contact information during locomotion, but it is difficult to generalize to different types of motions.

We propose a new network called Spatio-temporal Recurrent Neural Network (STRNN) to model the spatial and temporal variances. In particular, we use networks to model segmented parts of the human skeleton to model the spatial variations within a frame effectively. This avoids generating a mean posture and thus eliminates the needs of a separate disambiguous network as in [1]. Our temporal network performs batch encoding, decoding and prediction, allowing us to use a new loss function to consider long-horizon prediction error and motion naturalness. As a result, our network can learn a high-quality motion manifold that can

- *1. H. Wang is with the University of Leeds, Leeds, UK. E-mail: h.e.wang@leeds.ac.uk. ORCID:orcid.org/0000-0002-2281-5679*
- *2. Edmond S. L. Ho and Hubert P. H. Shum are with Northumbria University, Newcastle Upon Tyne, UK.*
- *3. Zhanxing Zhu is with Peking University and Beijing Institute of Big Data Research, Beijing, China*

synthesize long sequences of motions without explicitly introducing extra temporally-related variables as in [5].

To demonstrate the quality and versatility of the learned motion manifold, we first synthesize long, natural and highly dynamic motions using an open-loop setup, in which we do not moderate the error using any run-time systems or disambiguous networks. Then, we show how the manifold can be used to denoise corrupted motion data and synthesize new motions with control signals. We also compare our methods qualitatively and quantitatively with existing methods.

The proposed network creates high-quality motion manifolds for a variety of applications in both computer graphics, including motion synthesis, motion denoising and motion prediction/extrapolation.

The major contributions of the work can be summarized as follows:

- We propose a new deep learning framework for generating a high-quality manifold of 3D skeletal human motion.
- We propose a spatial model for 3D motion by dividing the skeleton structure into parts with spatial proximity and semantics.
- We propose a new optimization strategy considering long-horizon prediction in the temporal domain to preserve long-term motion naturalness and dynamics.

The paper is arranged as follow. We review existing research in Section 2, particularly focusing on deep learning approaches. Then, we explain how we prepare training data in Section 3. We detail the design of our deep learning model and explain our optimization strategy in Section 4. We present experimental results of different applications and system evaluations in Section 5. Detailed comparisons are shown in Section 6, followed by the performance analysis in Section 7. Finally, we discuss our system in Section 8 and conclude this research in Section 9.

2 Related Work

2.1 Data-Driven Animation

Traditional human motion synthesis and classification are done in a feature space which is typically represented as a temporal sequence of 3D skeletal postures [7]. These systems allow some control over the motion, but both the data representation and control schemes are usually manually designed and application specific [8].

Later, researchers started to look for compact motion representations by statistical modeling [4], dimensionality reduction such as Gaussian Process Latent Variable Model [9] or representative posture landmarks [10]. High-level features such as styles are also modeled by, for instance, spectral-domain representation by Fast Fourier Transform [11] or local autoregression models [12]. A similar idea is proposed to represent the formation of a group of characters with a spectral-based manifold for formation interpolation [13]. To construct a posture space with a uniform density, selective resampling on a collection of human motion data is proposed [14].

However, due to the non-linearity of human motion, it is difficult to create a global manifold. As a result, local models assuming local linearity are proposed by considering only the data samples that are relevant to the control signals [15]. PCA has shown to be effective in reducing a motion into a low-dimensional space for better control and visualization [16], [17]. It is also applied to a dynamic set of samples selected based on the control signals to construct a low-dimensional space for motion synthesis [18]. Weighted PCA with a hierarchical k-d tree structure enables the local models to represent a large database [19]. Considering the reliability of the control signal, a more accurate set of data samples can be found in [20]. A mixture of Gaussian process is used to reduce the database size required for motion synthesis [21]. These methods ([20], [21]) focus on removing the noise from captured motions, which is a problem known as motion denosing [22]. We also demonstrate how our generated manifold can be used for motion denosing with superior results.

Generating local models require high run-time overhead and the accuracy depends heavily on how the local data is selected. In this paper, we are interested in deep learning based approaches, which has the advantage of low run-time overhead as a global model but also model the motion manifold as good as a local models.

2.2 Deep Learning

3D skeletal human motion is an effective representation for motion synthesis and classification, in which the movement is presented as temporal sequences of joint positions or joint angles. While deep learning methods using convolutional neural network (CNN) are effective for images [23], it is unclear how CNN can be performed for skeletal motion as the feature space does not explicitly encode proximity. Existing work shows CNN applied in the temporal domain through an autoencoder that learns the temporal features of a human motion [1]. The motion manifold learned by the new framework can be used for synthesizing new motions base on high-level parameters. Such an autoencoder, however, does not encode the spatial information of the joints within a frame, leading to the generation of featureless mean postures. A second disambiguation network is required to restore the motion. Another solution is to explicitly encode extracted control parameters such as stepping patterns and stepping phases [5], such that the network can output both motion and control parameters for better motion generation. However, such a method is application dependent and cannot be generalized to different types of motions. It is also possible to apply deep learning to learn the control mechanism of a dynamic controller to create dynamic locomotion on different terrains [24].

Recurrent Neural Networks (RNN) has been applied to modeling ([25], [26]) and recognizing 3D skeletal human motion [27]. In [27], the skeleton in each frame is divided into different body parts to construct a hierarchical structure of RNN. We follow this idea but designed a network to explore spatial features of the body parts. Modeling temporal information using deep neural network has also been applied in handling RGB videos. Recurrent Convolutional Neural Networks (RCNN) are proposed to handle video data, in which each frame is fed into a CNN followed by one or more layers long short-term memory (LSTM) that are recurrently connected [28]. A similar RCNN structure is applied for human identification using 4D depth video, in which a spatial distribution of 3D point cloud obtained by a single depth is used at the input of the CNN [29]. A recent supervised learning approach [30] is proposed for synthesizing character animation with interactive control. In particular, the motion synthesis framework models the spatio-temporal motion structure as well as constraints for interactive control by RNN-based networks. While we share similar interests in using RNN for synthesizing high-quality human motion, our research is orthogonal to theirs. In [30], they trained the model for the tasks using task specific objectives. In contrast, our method aims to learn a versatile and natural motion manifold without any task-specific supervisory information. We propose a RNN model equipped with a spatial network, allowing it to directly utilize skeletal information of human motions to obtain a high-quality manifold.

Spatio-temporal graphs that explicitly represent the relationships of human and environment objects have shown to be effective in activity recognition [31]. Structural RNN can be used to recognize activities based on graphs representing the trajectories of skeletal motions and object movements [32]. Another graph implementation is to consider the spatial and temporal relationship as intra-frame and inter-frame edges respectively [31]. However, these graphs focus more on the interaction between multiple instances, instead of focusing on the movement of a single human. Also, they are not suitable for motion synthesis due to the abstract representation.

3 Data Preparation

We follow the practice in [1] to build our motion dataset. We use several datasets including CMU [33], HDM05 [34], MHAD [35], Action3D [36] and Edinburgh [1]. Since the captured data comes from different actors with different skeletons, the data is first scaled then mapped onto one standard skeleton. Then inverse kinematics is used to bring the joints of the standard skeleton to the joint positions of the source skeleton. Frame rates from different datasets vary from 30 to 120Hz. We resampled them to 30Hz. To unify posture presentations, we use joint positions defined with respect to the body's local coordinate system where we project the root onto the ground as the coordinate system origin. Some other methods used joint angles instead joint positions. We also tried joint angles but found that it is easier for neutral nets to learn a stable manifold of joint positions. Also, it is also easier for motion control, explained in the next section. Our character contains a total of 73 Degrees of freedom (Dofs). The first 66 Dofs are the joints including 6 Dofs (the global positions and orientation) of the root joint and 3 Dofs for the 3D positions of the remaining 20 joints (left/right toes, foot, knees, hips, fingers, wrists, forearms, arms and spine, spine1, neck, head). Then we also have a global velocity which is a 2D vector in the x-z plane (assuming that the y axis points upwards) and an angular velocity which is a 1D vector around the y axis. Finally, we use 4 binary variables to record binary foot contact information for left heel, left toes, right heel and right toes respectively. After processing, we obtained approximately 60k motion clips that are further divided in ratio 80:10:10 for training, validation and testing.

4 Spatio-temporal Recurrent Neural Network (STRNN)

4.1 Motion Manifold Modeling

We start by parameterizing the motion manifold as a time series: $P(X_{t+n},...,X_{t+1}|X_t,...,X_{t-m})$ where X_t is the motion frame at time t and P is the conditional probabilistic distribution of n frames from $t+1$ given $m+1$ frames before $t+1$. What the model captures is the dependencies between the past $m+1$ frames and the future n frames. Many existing data-driven models fall under this umbrella. Most of them consider the situation when $n = 1$ and $m = 0$, such as the motion graphs [2] and autoregression [12]. Some consider $m > 0$, such as the dynamic Bayesian networks [37] and high-order Markovian models [38]. Our model generalizes them by setting both n and m much bigger than 1, resulting in a *batch prediction* framework that forces the model to look forward into a further future and capture the mid-term and long-term frame dependencies. This allows us to improve the model representation of the motion dynamics. Here, we propose the Spatio-temporal Recurrent Neural Network (STRNN) that learns these dependencies.

4.1.1 Spatio-temporal Recurrent Neural Network (STRNN)

STRNN consists of three sub networks: spatial, temporal and residual, shown in Figure 2. The temporal network aims at learning the temporal dependencies between long motion sequences. It is structured as a Recurrent Neural Network composed of long short-term memory (LSTM) cells [39].

LSTM networks have been used for modeling time series data in many fields [28]. Two common approaches are (1) encoding/decoding [40] where information is forced to be recovered by every time step, and (2) sequence-to-sequence [41] where the network takes all input first, then decodes them into a different time series. The former focuses on learning the inter-frame dependencies while the latter targets at the mappings between sequences. Our temporal network is a combination of both, so that the network is forced to reconstruct the motion manifold while taking the future motion into consideration, as shown in Figure 3. It also enables us to impose constraints in a longer time span to stabilize the network.

The temporal network is named *Two-way Bidirectional Temporal Network* (TBTN), consisting of three parts: the temporal encoder (TEncoder), the temporal decoder (TDecoder) and the temporal predictor (TPredictor) (Figure 3). The training is done by iterations of forward and backward passes. The forward pass goes through an encoding phase and then a decoding/predicting phase. It starts by taking $m + 1$ frames into TEncoder. Note that unlike some RNNs [6], the decoding and predicting only start *after* the TEncoder takes all the input, making it a sequence-to-sequence model. After the encoding phase, the internal state of TEncoder is copied to TDecoder and TPredictor as a *good/reasonable initialization*. Then, the forward pass continues on TDecoder

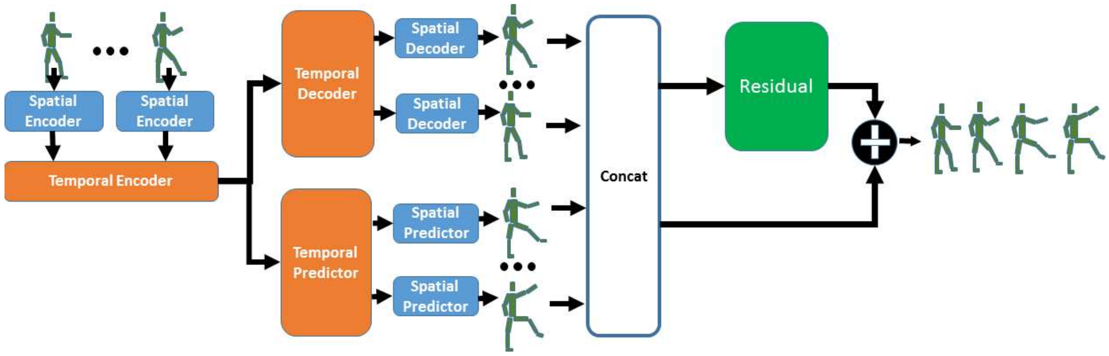


Fig. 2. The network architecture of the Spatio-temporal Recurrent Neural Networks. Detailed network structures of the Temporal Encoder/Decoder/Predictor and the Spatial Encoder/Decoder/Predictor can be found in Fig. 3 and Fig. 4 respectively.

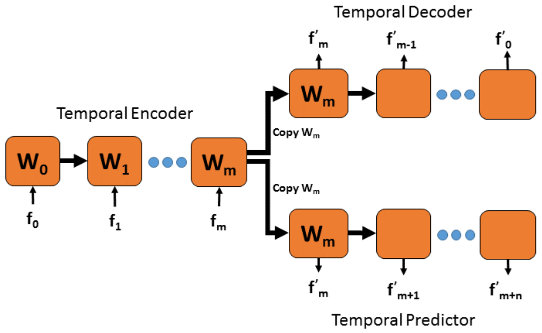


Fig. 3. Two-way Bidirectional Temporal Network (TBTN). Rectangles are LSTM cells. Ws are the corresponding weights.

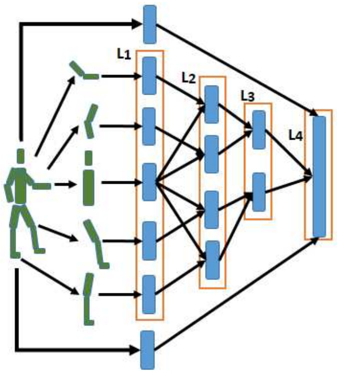


Fig. 4. Spatial encoder. A hierarchical neural network on graph structure.

and TPredictor simultaneously. The decoding in TBTN unrolls in both directions in time. The task of TDecoder is to decode the frames *backwards* in time and the task of the TPredictor is to predict the frames *forwards* into the future. The backward decoding improves the convergence speed as it first decodes the last few frames that the encoder just sees. Therefore, it forces the model to learn the short-term correlations between frames first, and then progresses onto their longer-term correlations. Similar ideas have been employed in video prediction problems [42]. During the decoding phase, the decoded frame f'_i serves as the input for decoding f'_{i-1}. Similarly, the predicted frame f'_j serves as the input for predicting f'_{j+1}. The forward pass of TEncoder is defined as:

$$\Phi_e(X) = LSTM_e(W_e X + b_e) \tag{1}$$

where $X = \{f_0, f_1, ..., f_{k-1}\}$ is a motion sequence with k frames, (W_e, b_e) are are the weights and biases of the LSTM cells in TEncoder. Similarly, we use Φ_d, Φ_p, (W_d, b_d) and (W_p, b_p) to represent TDecoder and TPredictor respectively.

Although TBTN can learn the temporal dependencies, the representation of the frame itself also requires a modeling strategy. The simplest solution is to feed in raw data such as joint positions and joint angles. However, human motions are highly coordinated [43] while the raw representation ignores the correlations. To learn the correlations directly from raw data, it requires a huge amount of data with large variety. In addition, even if the data is available, the variance can easily be overestimated or underestimated by the RNN networks [6].

To capture the correlations, we observe that the Dofs often move as relatively independent groups based on the semantics of the motion. For example, waving a hand mainly involves the Dofs on one arm. While this kind of group independence may not be universal in all motions, grouping Dofs provides better modelling of the correlations between groups and the overall body in general [27]. Based on this observation, we propose to perform hierarchical spatial modeling on each set of body part(s), as shown in Figure 4. At L1, each group-wise component generates a *summary* for the whole group. For instance, the L1 component for the left arm encodes the variances of all the left arm Dofs in different motions. When the summaries at L1 are combined at L2, the cross-group correlations are modeled. The same principle applies up to L4. Overall, to encode a frame, we group DoFs based on body parts, and then merge them hierarchically until we have a latent encoding. The decoding/prediction is done reversely. By grouping Dofs, group-wise posture variances and cross-group correlations are explicitly modeled.

The Spatial Encoder (SEncoder) decomposes a human body into seven parts: $X = \{X_{root}, X_{torso}, X_{lLeg}, X_{rLeg}, X_{lArm}, X_{rArm}, X_{fp}\}$. X_{root} includes root positions, velocity and angular velocity around the Y axis. X_{fp} is foot contact information. X_{torso} includes spine, neck and head. X_{lLeg} and X_{rLeg} include the hip, knee and foot of the left/right side. X_{lArm} and X_{rArm} include the shoulder, elbow and hand of the left/right side. We define $X_{body} = \{X_{torso}, X_{lLeg}, X_{rLeg}, X_{lArm}, X_{rArm}\}$. Each blue box in the four groups (L_1-L_4) is a fully-connected layer. The layers in the same group have the same number of hidden units. From L_1 to L_4, the number hidden units is 64, 128, 256 and 512 respectively. In L_1, individual body groups are mapped to a latent space. L_2 merges the $lArm$ with $torso$, $rArm$ with $torso$, $lLeg$ with $torso$ and $rLeg$ with $torso$ respectively. L_3 merges the body parts into the upper body and the lower body. Finally, L_4 merges both the upper and lower bodies into the whole body. Besides, the top component is the global velocity on the x-z plane and the rotation around the y-axis. The bottom component is a 4D binary vector on foot contact. Both components are directly merged into L_4, too. We then append a dropout layer [44] and a batch normalization [45] layer to $L_1 - L_4$. The network structure explicitly models the spatial correlations between different Dof groups. The Spatial Decoder (SDecoder) and Spatial Predictor (SPredictor) have the same network structure except that the information is propagated in the reversed direction. We denote the spatial encoder as below:

$$\Sigma_e(X) = \sigma_{L_4}(\{\sigma_{L_3}(\sigma_{L_2}(\sigma_{L_1}(X_{body})), Xroot, X_{fp}\}) \quad (2)$$

$$\sigma_{L_x} = BN(drop(tanh(W_s merge(X_{in}) + b_s))) \quad (3)$$

where σ_{L_x}, $x = 1, 2, 3, 4$ are the activation on L_1 to L_4 layers. W_s and b_s are the weights and the biases of the fully-connected layer. SDecoder and SPredictor decode the latent representation into the original data. They are both denoted by Σ_e^{-1}. Note our spatial encoder is frame-based, different from [27]. While their aim is to learn the temporal patterns of individual joints, we aim to learn spatial correlations.

We find that the spatial and temporal networks together alone can model motion multi-modality (refer to the accompany video). However, we observe periodic jumps, which is a problem in iterative motion prediction that is also observed in other systems [25]. We formulate the removal of this high frequency noise as a learning problem, i.e. to learn a signal to cancel the noise. To this end, we use a residual component. Our residual component consists of four fully-connected layers with ELU activation functions [46] and 512 hidden neurons, except for the last layer where it maps the data back to the original dimension. Both the input and output of the residual component are motion data (decode_len + predict_len, d) where d is the dimension of a single posture, decode_len and predict_len are the number frames that are decoded and predicted. The input is a concatenation of the output of the spatial and temporal networks and the output is the filtered results. SDecoder and SPredictor share weights. This is inspired by [1] that a few full-connected layers can learn a good latent representation. However, note that our solution is very different in the sense that STRNN does not rely on other networks to disambiguate motion modes [1] or to learn a velocity profile [25]. Instead, the residual component learns the distributions of noises and acts as a filter.

4.2 Loss Function and Model Training

Our loss function includes a reconstruction error term. Besides, it contains new terms that are inspired by the past animation research in non deep-learning areas. The first one is the long-horizon costs that governs the smoothness of the generated motions. We found that this seemingly simple cost is the key to harness the motion variances. The second is a group of motion control costs such as control signals and bone-length constraints. Here, we give details of each of the cost terms. We define the full loss function:

$$Cost(X, \Omega) = w_r C_r + w_s C_s \quad (4)$$

where C_r and C_s are reconstruction and smoothness, with weights w_r and w_s.

4.2.1 *Reconstruction Error*

We use Mean Squared Error (MSE) for C_r to force STRNN to reconstruct motions. C_r = C_d + C_p:

$$C_d = \frac{1}{m}\sum\|(\Sigma_e^{-1}(\Phi_d(\Phi_e(\Sigma_e(X_e)))) - X_e)\|^2 \quad (5)$$

$$C_p = \frac{1}{n}\sum\|(\Sigma_e^{-1}(\Phi_p(\Phi_e(\Sigma_e(X_e)))) - X_p)\|^2 \quad (6)$$

where C_d and C_p are the reconstruction loss of the decoded and predicted motions. m and n are the decoding and prediction lengths, $X = \{X_e, X_p\}$, X_e and X_p are the ground-truth decoding/predicted motions, and Σ_e^{-1} is simply the inverse function of Σ_e. Minimizing C_r results in a tight approximation of the motion manifold. We will denote this cost as *MSE* in Section 5.

4.2.2 *Long-horizon (LH) Cost*

C_s in Equation 4 is smoothness cost:

$$\begin{aligned} C_s = \; & \tfrac{1}{m+n}\sum\|\hat{X}^{t+1}_{body} - 2\hat{X}^{t}_{body} + \hat{X}^{t-1}_{body}\|^2 \\ & + \sum\|\hat{X}^{t}_{root} - \hat{X}^{t-1}_{root}\|^2 \end{aligned} \quad (7)$$

where m and n are the decoding and prediction lengths, $\hat{X}$ is the concatenation (along the time axis) of the decoded and predicted motions, C_s governs the smoothness of the motions and has been widely used in many optimization-based character animation approaches. Note this constraint essentially penalizes big accelerations only and does not overly dampen the motion dynamics. We will denote it as *long horizon constraint (LH)* in Section 5.

4.2.3 *Run-time Costs*

Since the root velocity, root angular velocity around Y-axis and foot planting information are also included in the training data, control can be imposed in run time by penalizing the deviations of the decoded and predicted motion from the desired motion signals. These penalties can be added to Equation 4 with a weight equal to $(1 - w_r - w_s)$:

$$C_{con} = H_{ctr} + H_{bone} + H_{fp} \quad (8)$$

where:

$$H_{ctr} = \sum\|\hat{X}_{root} - \Gamma\|^2 \quad (9)$$

$$H_{fp} = \sum f_{cta}\|\hat{X}'_{foot}\|^2 \quad (10)$$

$$H_{bone} = \sum\|\|\hat{X}^{i}_{body} - \hat{X}^{j}_{body}\| - l^{ij}\|^2 \quad (11)$$

H_{ctr}, H_{bone} and H_{fp} are costs on control signals, bone length and foot contact respectively. H_{ctr} ensures the control is satisfied. Γ is the given control signal specifying root velocity and root angular velocity around the y-axis, allowing us to safely recover the global translation and rotation. H_{fp}

fixs the foot sliding by penalizing the foot velocity $\hat{X}'_{root}$ where f_{cta} is the stepping patterns. $f_{cta} = 1$ if there is foot contact and 0 otherwise. It can be extracted from the motion or manually specified. We use a simple heuristic to automatically detect foot contact states by thresholding the heights and the world speeds of the toes and heels. H_{bone} enforces the bone-length constraints with l^{ij} being the bone length between joint i and j.

4.2.4 Training

As long as Equation 4 is differentiable, we can use stochastic gradient decent for optimization. In our experiments, AdaDelta [47] is used with random weight initialization. We set learning rate = 1, ρ = 0.95, ϵ = 1e-08 and decay = 0.0.

To accelerate training and prevent overfitting, first, we regularize Ws and bs on their L2 norms in Equation 4, which forces the model to use as few activations and biases as possible, both weight coefficients are set to 0.01. Second, we use Dropout [44] with 10% rate and batch normalization [45] after each fully-connect layers in the spatial network.

Next, we use corrupted inputs to train STRNN. Randomly corrupted inputs have been found effective in training [6]. Here, we design a special mechanism to corrupt the inputs for different iterations. We use a Gaussian noise with zero mean and 0.1 standard deviation to corrupt the input in the first iteration. We then gradually decrease the deviation by 0.001 in each of the following iterations until the value becomes 0.0. We find this training mechanism beneficial because it first tries to bring the optimizer into a large area centered around the ground truth, and then gradually brings it to the ground truth. On the way, it keeps the memory of the corrupted data, essentially mapping a small area around the ground truth to the ground truth. This helps to reduce error accumulation over time. Unlike [6] where the noise level gradually increases over time, decreasing the noise level avoids the shortcoming of unable to reliably choose the right model for testing [25]. This is because in the later stage of training, there is no noise in the ground truth, and thus the reported learning and validation errors can be reliably used for selecting the best model.

Finally, we use a hybrid training strategy. We divide the model into the spatio-temporal network (full model without the residual network) and residual network. We first train our spatio-temporal and residual network separately, using LH + MSE and MSE respectively, then compose them together and fine-tune the residual network using only MSE. The spatio-temporal network trained by LH + MSE can generate good motions but with high-frequency noises. Then, the residual network aims to learn to cancel high-frequency noises and the rest aims to learn the motion dynamics. Separate training gives them both good initialization. When fine-tuning, we fix the spatial and temporal networks and only tune the residual component. The reason is that our pre-trained spatial and temporal networks capture the motion dynamics well, but with high frequency and periodic noises (see Section 5). Removing the noise can be seen as learning residuals to cancel them. It is faster for the residual network to learn these residuals if it is pre-trained on the ground-truth.

5 Experimental Results

In this section, we will first evaluate STRNN from several aspects. Next, we will demonstrate three applications, including long sequence synthesis, motion denoising and controlled motion synthesis.

5.1 Evaluation

STRNN is a family of neural networks with variations in the structure. One variable is the duration of the temporal network, decided by TEncoder, TDecoder and TPredictor, denoted by *encode_len*, *decode_len* and *predict_len* respectively. We vary the three lengths while enforcing encode_len = decode_len. We first evaluate various settings on data consisting of motion segments of *40* frames, denoted by $l = 40$ where l is the length of the segments. Then, we empirically find a good setting and test it on data where $l = 10$ and $l = 20$.

To show the added benefits of different components (spatial, temporal and residual), we first show STRNN with only the temporal network, denoted by *Temporal*. Then, we combine the spatial network and the temporal network denoted by *SpatioTemp*. Finally, we show the full model (spatial+temporal+residual) denoted by *Composite*. All evaluations are accompanied with quantitative (errors) and qualitative results (videos) on the task of long-horizon prediction.

We also show the effects of the terms in loss function (Equation 4) and the hybrid training strategy. We use MSE, LH and HY to denote three different settings. MSE means C_r only, LH means C_r and C_s. HY is the LH loss with the hybrid training strategy, explained in Section 4.2.4.

In the rest of the section, we use the following model notations. Composite_20_20_HY means we use the full model and hybrid training strategy with 20-frame encoding (i.e. 20-frame decoding) and 20-frame prediction, while Temporal_39_1_MSE means we only use temporal network and MSE loss with 39-frame encoding (i.e. 39-frame decoding) and 1-frame prediction. In addition, all experimental results can be found in the video, in which all the demos (except one for comparison purposes) are not post-processed to show the true performance of our model. We also give one example (Section 5.5) to show that our model can benefit from incorporating post-processing to remove issues such as foot sliding.

5.1.1 Encoding vs Predicting

Figure 5 shows the decoding and prediction errors for both training and testing phases. We choose encode_len = 39, 35, 30, 20 for comparison. We emphasize that Figure 5 only shows that how well different models can fit data on 40-frame motion segments. It is not directly related to the visual quality of generated motions, especially of those longer than 40 frames, explained in Section 5.1.2. In term of the data-fitting ability, we find that larger predict_len generally gives better overall results given l is fixed. The only exception is Temporal_39_1_MSE but visually it generates worse results. This indicates that there should be a balance between the encode_len and predict_len. After experiments with different settings, we found that a balanced encoding_len/predict_len pair tends to give better results.

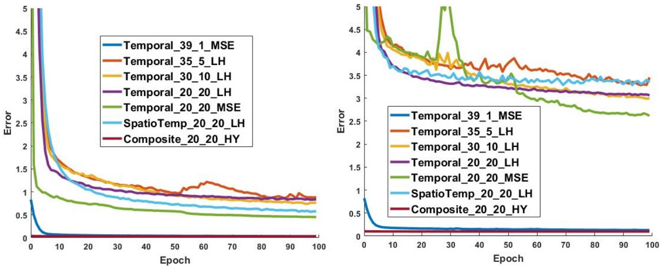


Fig. 5. From left to right: training error and validation error. SpatioTemp_x_y is the model where x is the encode_len and y is the predict_len. Temporal_x_y is the model *without* the spatial encoder/decoder/predictor and the residual network. Composite_x_y is the full model. Also, MSE means only C_r is used. LH means both C_r and C_s are used. HY means the hybrid training results.

So, we empirically set encode_len = predict_len = 20 for our experiments unless specified otherwise.

5.1.2 Motion Prediction Quality

Next, we do both qualitative and quantitative comparisons on prediction tasks with different network settings. The qualitative visual comparisons can be found in the video. In the quantitative comparison, we found that although Figure 5 shows how well different models can fit data, the results are not directly correlated to the visual quality. Since designing a metric that directly reflects visual quality is difficult, we use a metric that evaluates how close the generated motions are from the motion manifold, which has been previously used to measure motion style similarity [20]. Also, we assume that the existing data are representative samples and are dense enough to represent the ’ground-truth’ motion manifold. In practice, we found that this metric is largely consistent with the visual assessment:

$$D_{1nn} = min(dist(D_g, D_m)) \tag{12}$$

$$dist(D_g, D_m) = \frac{1}{n}\sum_{i=1}^{n}(||\hat{f}_i - f_i||_2^2) \tag{13}$$

where D_g and D_m are generated and ground truth motions respectively, each with n frames. $\hat{f}_i$ and f_i are in ith frame in D_g and D_m. $dist(D_g, D_m)$ is essentially the per-frame l_2 norm of two motions. D_{1nn} essentially is the smallest distance between D_g and the ground truth data, which is a measure of how close the generated motion is from the manifold. In practice, D_g and D_m could have different duration. An exhaustive comparison is impractical due to the exponential complexity. As a solution, we use a 10-frame overlapping sliding window to chop both D_g and D_m into segments with the same length (40 frames). Then, we compute D_{1nn} by exhaustively comparing all segments in D_g to all in D_m. To test the generalizability, we use disjoint training and testing datasets. We use 20-frame motion prefixes randomly selected from the testing dataset to generate 200 frames. Then, we compute the metric D_{1nn}.

The results are shown in Figure 6. Our full model (Composite_20_20_HY) outperforms a baseline model (Temporal_39_1_MSE) where no batch prediction is used and no LH loss nor HY training was used (ablation test 1). We also found that LH loss performs better than MSE (ablation test 2). The LH loss only regulates the acceleration, not the position nor velocity, so not overly smoothes the motions. MSE on the other hand has been known to lead to a mean posture as it is an easy local minimum for the optimization. Our speculation is that the combination of both actually allows the optimizer to find a solution that keeps variances so that it does not get stuck in the mean posture local minimum. In addition, by testing the network with different components, we also found that batch prediction produces better results in different settings (Temporal_39_1_MSE vs Temporal_30_10_LH vs Temporal_20_20_MSE in Figure 6). For the sake of simplicity, a visual comparison on more exhaustive experiments can be found in the video (ablation test 3).

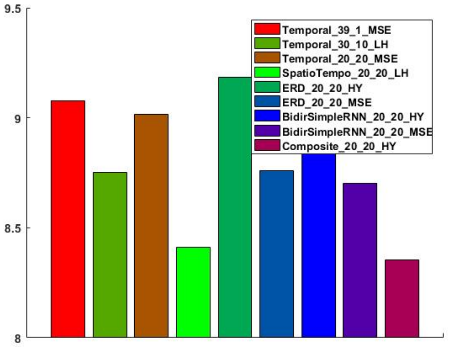


Fig. 6. The D_{1nn} error of different models on a 200 frame motion prediction task.

Next, adding the spatial network retains the motion variances better. Sometimes batch prediction can still lead to underestimating the motion variances, observed in Temporal_20_20_LH (ablation test 4). In contrast, SpatioTemp_20_20_LH retains the variances and prevents the mean pose problem, which is a result of group-wise variances being explicitly modeled.

Further, although SpatioTemp_20_20_LH gives good results (Figure 6), it generates periodic jumps that is due to the under-constrained variances near the boundary of predicted motion segments (ablation test 5). Similar observations were also made in [25]. While in [25] the model learns the velocity instead of the posture to eliminate the problem, the side-effect is that it can still converge to the mean pose in a long run. Our residual network combined with our hybrid training strategy not only improves the quality but also retains the variances. Please see the comparison between Composite_20_20_HY and SpatioTemp_20_20_LH (ablation test 5).

5.1.3 Different Motion Lengths

The added value of different components have been evaluated above. But they are all done with 40-frame motion segments. We therefore also vary the whole motion segment duration to see if it generalizes well. Given the limit of our graphics card, the longest motion segment we experiment is 40 frames. We therefore experimented with 20-frame and 10-frame data under the setting of Composite_10_10_HY and Composite_5_5_HY (ablation test 6 & 7 on two different motion prefixes). We found that our network generalizes well. In the video, all of them generate motions with similar visual qualities. This also suggests that strategy of a balanced encoding_len and predict_len generalizes well too.

5.2 Motion Prediction and Extrapolation

Prediction power is one effective way to test motion manifold models. It is often presented as an initial value problem in which if a prefix of data is given, the model should be able to extrapolate/predict into the future while keeping the predictions on the manifold.

After training our network on the CMU dataset, we take 20 frames as input and let STRNN predict 20 frames into the future. STRNN can reliably predict the next 20 frames. Then, we push STRNN further to extrapolate for even longer sequences to test the stability and quality of the motion manifold. We predict a long sequence of motions by iterations in an open-loop setup, in which there is no subsystem nor feedback variables to moderate the cumulative prediction error. In each iteration, STRNN takes the last 20 frames as input and predict the next 20 frames.

We first show results of 1000-frame prediction in the video. We also push the model further by generating longer sequences. We include motions with **20000** frames. Figure 1 shows the first 2000 frames of a boxing motion. According to our experiments, STRNN can predict far longer than 20000 frames in practice, but we believe that it is already long enough to demonstrate its stability. Also, the length of prediction supersedes all other work as far as we know. The closest one is [26] but our prediction is still much longer. Also, unlike [26], our model is trained on all the motions at once.

5.3 Motion Denoising

Motion capture data can be easily corrupted due to tracking errors. As a motion manifold model, STRNN can fix the data by projecting the corrupted input onto the manifold. We show the results in a controlled experiment where we have high-quality ground truth motions. It allows us to measure reconstruction errors. We employ the CMU and Edinburgh dataset. Since they contained a wide range of motions with different styles and dynamics, we randomly sampled 512 motions. Then, we randomly perturb the motion with a Gaussian noise on each dimension independently. Figure 7 and the video show our high-quality denoising results. Numerical results such as reconstruction errors are given in Section 6, in comparison with other methods.

5.4 Controlled Motion Synthesis

Controlled motion synthesis is another application of STRNN. Given an initial motion and control signal, not only can we use STRNN to impose control on the given motion, the control signal can also be applied to the predicted motion too, which makes it an open-ended prediction and control. Note that although controlled motion synthesis can be done in other works ([1], [5]), our task here is much more challenging because we are applying control signals on predicted motions, which are unknown in the beginning.

Given the initial 20 frames (or randomly chosen from the dataset), we perform prediction and control alternatively. In every iteration, there are two steps: prediction and correction. We first predict the next 20 frames. Then, we project the motion back onto the motion manifold while following the control signals explained in Section 4.2.3 as much as possible. This is done by running the optimization for a number of iterations, i.e. fine-tuning the network for the current iteration. Finally, we move on to the next iteration.

The fine-tuning is a local operation because it adapts the network only for the current iteration. Then, alternating on the open-loop prediction and the local fine-tuning can gradually drag the network out of the natural motion regions. We propose a simple solution. We back up the pre-trained weights. In run time, we use the pre-trained weights for prediction and then fine-tune the weights to satisfy the constraints. Finally, we restore the weights before the next iteration. In this way, the motion prediction is always done on the pre-trained network. In the experiments, we find that the fine-tuning only requires several iterations and thus is very quick. Since STRNN captures the motion multimodality, the generated motion can contain multiple types of movements. Detailed results can be found in the video (long motion sequence of 1000 frames).

5.5 Contact Violation and Post Processing

In our demos, we do not use post-processing to fix foot sliding. As a result, some motions do violate contact constraints. This is partially due to the existing foot-sliding in data. We show the raw results for evaluation and comparison purposes. However, similar to existing work, post-processing such as Inverse Kinematics can be employed to fix foot sliding issues. We show one example of a controlled motion synthesis with a long and complex control trajectory. The video includes before and after post-processing results using [48] .

6 Comparison with Other Models

6.1 Comparisons with Alternative RNNs

Given there is a large body of deep learning networks in computer graphics and vision, machine learning, etc., it is impractical to do exhaustive comparisons. We, therefore, focus on the relevant area of recurrent neural networks. Some of them are designed for motion prediction such as [6] denoted as ERD, while some of them are for action recognition such as [27] denoted as BidirSimpleRNN. It is hard to do a fair black box comparison because they are designed for different purposes. Therefore, we mainly compare different spatial encoding/decoding mechanism under the same encoding/decoding/predicting setting to see how they fit data and generalize. We use the CMU dataset and keep an 80/20 training and testing ratio. We computed the same D_{1nn} error explained in Section 5.1.2. The results are shown in Figure 6. STRNN in general generate better motions. In addition, we found that motions generated by ERD has smaller variances than BidirSimpleRNN. We speculate that this is due to the posture encoding/decoding components in ERD. However, these components are merely fully-connected layers while STRNN models the skeleton explicitly. SRTNN still produces the best results visually as shown in the video.

Method	80	160	240	320	400	480	560
LSTM_3LR	0.41	0.67	1.15	1.50	1.78	2.02	2.26
CRBMs	0.68	1.13	1.55	2.00	2.45	2.90	3.34
6GRAM	1.67	2.36	2.94	3.43	3.83	4.19	4.53
GPDM	1.76	2.5	3.04	3.52	3.92	4.28	4.61
ERD	0.89	1.39	1.93	2.38	2.76	3.09	3.41
STRNN(ours)	**0.36**	**0.39**	**0.43**	**0.46**	**0.48**	**0.5**	**0.51**

TABLE 1

Predication Error Comparison during the 80, 160, 240, 320, 400, 480 and 560 ms of prediction. Quantitative evaluation for longer temporal horizons is not possible due to stochasticity of human motion [6].

6.2 Prediction Accuracy Analysis

Prediction accuracy has been used to compare predictive models. Although it has been shown that prediction accuracy for short-term predication is not reliable, averaging over a number of motions shows the average predication accuracy. This accuracy might not be a good metric for motion prediction of a specific type but a good indicator to show how quickly errors accumulate. We follow the protocol in [6] and compare our model with others on H3.6M dataset [49]. We randomly choose 8 prefixes from the dataset and compare our prediction errors with some state-of-the-art methods. The error is per-frame Euclidean distance between the generated motion and the ground truth. They are: (a) 3-layer linear LSTM model (LSTM_3LR) [6], (b) Conditional Restricted Boltzmann Machines (CRBMs) [3], (c) a nearest neighbor N-gram model with n=6 (6GRAM) [6], (d) Gaussian Process Dynamic Model (GPDM) [38] and (e) ERD [6]. Also, we only compare the prediction errors within the first 560ms. This is largely due to the randomness in human motions where long predictions are not suitable for such comparisons [6].

We also compare our method with the one in [25] on prediction accuracy using the same per-frame Euclidean metric mentioned above. We downloaded their code and followed their protocol. Based on their code, the training and testing data come from different subjects across the same set of actions. The training data contains Subject 1, 6, 7, 8, 9 and 11. The testing data contains only Subject 5. We trained the model with the best possible setup and ran it for 200k iterations. Finally, we compute errors of the same 4 actions: Walking, Easting, Smoking and Discussion using randomly sampled motion prefixes within each action class [25]. Results are shown in Table 2. Numerically, STRNN performs better in all four classes. One possible reason is that the method in [25] used joint angles as representations. A small error in the joint angle space can cause big errors in the joint position space. This is consistent with our early findings when we experimented with different representations of postures. It is difficult to compare for longer duration due to the stochasticity of human motions [6], [25]. Also, it is difficult to do visual comparison because the duration is too short.

6.3 Denoising Comparisons

We also compare the motion manifold qualities through motion denoising, which can be done by projecting the corrupted motions back to the manifold [50]. As STRNN does it by feeding corrupted motions into the encoders and reconstructing denoised motion in the decoders, the reconstruction error is a good way to evaluate the manifold quality. We compare our method with [50] (SC), using the source code downloaded from the authors' website. The CMU and Edinburgh (CE) dataset are used because of their high quality.We randomly selected 512 motions. To show how our method is far more robust against stepping pattern noises, we use Gaussian noises with zero mean and 0.3 standard deviation for all joint positions, and 3 and 5 standard deviations on the stepping patterns to generate two polluted datasets. Note that the standard deviation of the noise added onto the feet joints is still 0.3 (i.e. the same as any other joints). It is only the stepping patterns, which are four binary variables, are disturbed with noises with higher standard deviations (3 and 5). In both implementations, for the stepping pattern parameter, a value higher than 0.5 is regarded as one or zero otherwise.

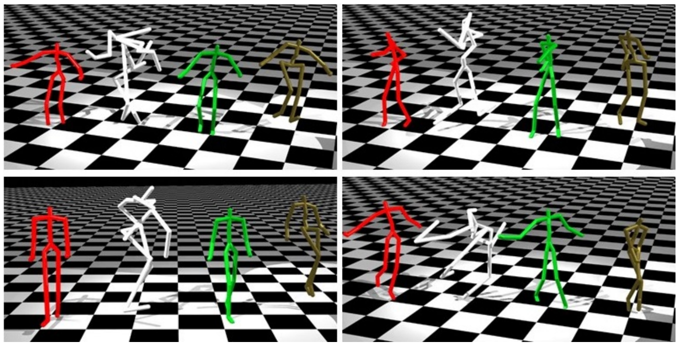


Fig. 7. Denoising comparisons. Red: original, White: corrupted, Green: STRNN, Yellow: SC

Four sets of screenshots of the corresponding frames in the original, corrupted and denoised motions are shown in Figure 7 and the video. The method proposed in [50] tends to overly smooth motions. We speculate that this is mainly due to the under-estimate of motion variances as their do convolutions and max-pooling along time. In contrast, STRNN captures the dynamics of the motions well. Also, SC depends heavily on precise clear stepping patterns. For motions that involve unclear stepping patterns, such as dancing with intentional foot skate, or motions with corrupted stepping patterns, their method may not be able to produce reasonable results.

For numerical comparison, we consider the sum of squared errors of the joint positions of all motions between the original and denoised motions, as summarized in Figure 8. The reconstructed motions by both STRNN and SC are denoted as STRNN_0.3_3, SC_0.3_3, STRNN_0.3_5 and SC_0.3_5 respectively. Given the same noise level, STRNN's reconstruction error is smaller than SC's. Also, when the noise level becomes bigger on the stepping patterns, STRNN is less influenced than SC is.

7 Performances

The code, implemented with Theano and Keras, runs on a computer with Intel(R) Xeon(R) CPU E5-1650 v4 3.6GHz, 64GB memory and an Nvidia GTX 1080 graphics card. For motion manifold learning, the pre-training on CMU and Edinburgh dataset takes approximately 16 hours. Once the network is trained, the motion generation is fast. For

	Walking				Eating				Smoking				Discussion			
milliseconds	80	160	320	400	80	160	320	400	80	160	320	400	80	160	320	400
[25]	5.30	6.15	6.74	7.29	4.96	5.99	7.16	8.29	8.08	9.36	10.53	11.63	8.98	10.09	11.12	12.02
STRNN(ours)	2.85	2.83	2.77	2.82	2.66	2.60	2.49	2.53	3.61	3.65	3.67	3.69	2.59	2.72	2.86	2.64

TABLE 2
Predication Error Comparison during the 80, 160, 320 and 400ms of prediction for 4 actions: Walking, Eating, Smoking and Discussion.

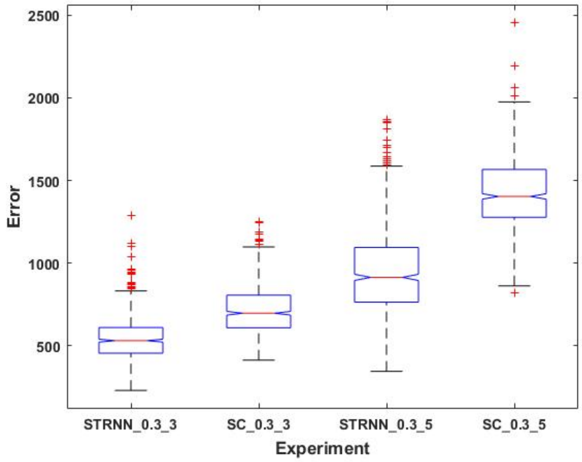


Fig. 8. X Axis: Experiments. Y Axis: Reconstruction error of STRNN_a_b and SC_a_b [50]. a and b are the standard deviations of the Gaussian noise patterns added on the body motion and stepping patterns.

long-horizon extrapolation, the system generates motions at 80000 FPS on average. For motion synthesis, we run 20 steps to satisfy various constraints for every 40-frame window. The system still generates motions at 1200 FPS on average.

8 Discussion

8.1 Alternative Architecture of STRNN

Firstly, both the TDecoder and TPredictor can be *conditional* or *unconditional*. During training, an unconditional TDecoder takes the ground truth X_t for decoding X_{t-1} while a conditional decoder takes the reconstructed $\hat{X}_t$ for decoding X_{t-1}. The same logic applies to TPredictor. The conditional scheme allows the model to learn multiple modes in the data distribution, without which, the model might average multiple modes. However, if data correlation is big (i.e. consecutive frames are too similar to one another), the conditional scheme forces the model to learn the correlation so that the temporal motion variance is underestimated. Furthermore, the gradient for fixing the error is small and it requires long-term knowledge about the input sequence. In our context, all these concerns are not existent. We choose the conditional scheme because human motions are multi-modal. The design of STRNN enables us to use the long-term knowledge of the input sequence so that if high correlation does appear, there will be a gradient to fix it. However, if required, it is possible for STRNN to generalize further by using either an unconditional scheme or a hybrid approach (e.g. unconditional TDecoder and conditional TPredictor or vice versa).

Next, the temporal memory mechanism can be modeled by other neural cells such as Gated Recurrent Unit (GRU) [51]. [52] shows a detailed comparison between LSTM and GRU on many datasets, and finds no conclusive evidence to prefer one to the other. We found that LSTM performed better by different margins on the datasets used. We speculate that this is due to the memory unit in LSTM that keeps the mid/long horizon dependencies.

There are other strategies regarding decoding/predicting mechanism in RNNs for different tasks. The Residual Network structure [25] facilitates the learning because all the decoder/predictor needs to learn is the first-order motion information. The Attention mechanism [53] makes use of the whole prefix as a context for decoding, such that the decoder/predictor would be more context-aware. We experimented with a model similar to [25] gave lower training errors but generated results similar to LSTM_3LR (i.e. converging to a mean pose in long-horizon prediction). We also tried introducing the Attention mechanism, which gave worse training/testing errors and generated highly jittering motions in long-horizon prediction. We speculate that this is because all the frames in the motion prefix are taken as a context while some of them should not have influence on later frames at all.

The multi-objective control (MOC) method in [30] can also generate high-quality motions under user controls. However, our method is orthogonal to MOC in several aspects. It is therefore not very meaningful to do a direct numerical comparison. STRNN aims to learn a global natural motion manifold by learning all unlabeled motions together while MOC learns to control specific labelled actions. Also, MOC learns controlled motion transitions through manually crafted motion grammars while STRNN learns transitions purely from data and randomly generates them in an open-loop setting.

STRNN uses a spatial encoder to project the frames onto a higher-dimensional space (512 hidden units), similar to [6], [25], which is somewhat counter-intuitive and different from previous findings that human motions can be embedded into a lower-dimensional space using techniques such as Principle Component Analysis [16]. We tried several different settings to map motions onto a lower-dimensional space, such as reducing the cell number is L1-L4 in Figure 4. We found that they are all unable to learn the motion manifold well. While this does not mean deep neural networks in lower-dimensional space is unsuitable for the problem, our empirical evidence shows that it could be challenging for them to capture the temporal multi-modality.

Lastly, the temporal network can consist of more than one layer. Stacking multiple layers of LSTM can model deeper temporal non-linearity in the data. In our dataset, it is not required since joint trajectories can be well approximated as piece-wise linear functions. However, it could be needed for other data types such as texts and videos.

8.2 Limitations

There are some limitations with STRNN. The transitions in the training data are important. During extrapolation, it is easier for STRNN to transit from one motion to another when there is at least one motion coming next. Essentially it has to do with the connectivity of the data. Given the sheer volume of the training data and that nearly all captured motions end somewhere close to a neutral standing posture, it is almost always the case. More training data can also improve the situation.

Second, when multiple next motions are possible, the transition can take a number of frames to transit into the next motion, which creates a bit of "hesitation" in the motion. It is caused by the roughly equal probabilities of several candidate motions. The equilibrium is very soon broken by the accumulated error that acts as a perturbance that brings the system into favoring one candidate motion. This can also be caused by the rest poses in the training data where the subject does not move much for a period of time, as noticed as "dead-times" in [26].

In addition, since our model does not take the environment information into consideration, with training data such as climbing a ladder or stepping on/over some obstacles, our model sometimes generates those motions too. In the future, we wish to incorporate environmental information into the network.

Lastly, since there is no motion labelling such as "jumping" or "walking", there is no global control in the action level. A direction to be pursued is to incorporate action labels. One possible approach is to convert STRNN into a conditional network where the motion generation is conditioned on action labels. A related direction is a global control mechanism that enables the users to specify actions at particular timings.

9 Conclusion

In this paper, we propose a new deep-learning framework to learn a motion manifold. Comparing to existing systems, our framework creates better results in various applications. The success is built upon two innovative solutions to maintain motion variances. We construct a hierarchical spatial encoder by dividing the skeletal structure into parts with strong spatial proximity, thereby preserving motion variances. We also construct a batch prediction network that allows long-horizon optimization, thereby harnessing the motion variances.

Acknowledgments

The project is partially supported by EPSRC (Ref:EP/R031193/1). The authors wish to gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. The project was supported in part by the Royal Society (Ref: IES\R2\181024).

References

[1] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 138:1–138:11, Jul. 2016.

[2] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 473–482, Jul. 2002.

[3] G. W. Taylor, G. E. Hinton, and S. Roweis, "Modeling human motion using binary latent variables," in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS'06. Cambridge, MA, USA: MIT Press, 2006, pp. 1345–1352.

[4] J. Min and J. Chai, "Motion graphs++: a compact generative model for semantic motion analysis and synthesis," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, p. 153, 2012.

[5] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 42:1–42:13, Jul. 2017.

[6] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 4346–4354.

[7] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 491–500, Jul. 2002.

[8] M. G. Choi, J. Lee, and S. Y. Shin, "Planning biped locomotion using motion capture data and probabilistic roadmaps," *ACM Trans. Graph.*, vol. 22, no. 2, pp. 182–203, Apr. 2003.

[9] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 522–531, Aug. 2004.

[10] X. Wu, M. Tournier, and L. Reveret, "Natural character posing from a large motion database," *IEEE Computer Graphics and Applications*, vol. 31, no. 3, pp. 69–77, May 2011.

[11] M. E. Yumer and N. J. Mitra, "Spectral style transfer for human motion between independent actions," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 137:1–137:8, Jul. 2016.

[12] S. Xia, C. Wang, J. Chai, and J. Hodgins, "Realtime style transfer for unlabeled heterogeneous human motion," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 119:1–119:10, Jul. 2015.

[13] S. Takahashi, K. Yoshida, T. Kwon, K. H. Lee, J. Lee, and S. Y. Shin, "Spectral-based group formation control," *Computer Graphics Forum*, vol. 28, no. 2, pp. 639–648, 2009.

[14] K. Yang, K. Youn, K. Lee, and J. Lee, "Controllable data sampling in the space of humanposes," *Comput. Animat. Virtual Worlds*, vol. 26, no. 3-4, pp. 457–467, May 2015.

[15] H. Liu, X. Wei, J. Chai, I. Ha, and T. Rhee, "Realtime human motion control with a small number of inertial sensors," in *Symposium on Interactive 3D Graphics and Games*, ser. I3D '11. New York, NY, USA: ACM, 2011, pp. 133–140.

[16] A. Safonova, J. K. Hodgins, and N. S. Pollard, "Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 514–521, Aug. 2004.

[17] H. J. Shin and J. Lee, "Motion synthesis and editing in low-dimensional spaces: Research articles," *Comput. Animat. Virtual Worlds*, vol. 17, no. 3-4, pp. 219–227, Jul. 2006.

[18] J. Chai and J. K. Hodgins, "Performance animation from low-dimensional control signals," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 686–696, Jul. 2005.

[19] J. Tautges, A. Zinke, B. Krüger, J. Baumann, A. Weber, T. Helten, M. Müller, H.-P. Seidel, and B. Eberhardt, "Motion reconstruction using sparse accelerometer data," *ACM Trans. Graph.*, vol. 30, no. 3, pp. 18:1–18:12, May 2011.

[20] H. P. H. Shum, E. S. L. Ho, Y. Jiang, and S. Takagi, "Real-time posture reconstruction for microsoft kinect," *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1357–1369, 2013.

[21] Z. Liu, L. Zhou, H. Leung, and H. P. H. Shum, "Kinect posture reconstruction based on a local mixture of gaussian process models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 11, pp. 2437–2450, Nov 2016.

[22] H. Lou and J. Chai, "Example-based human motion denoising," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 5, pp. 870–879, Sept 2010.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105.

[24] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement

learning," *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, vol. 36, no. 4, 2017.

[25] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," *CoRR*, vol. abs/1705.02445, 2017.

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

[27] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1110–1118.

[28] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 677–691, April 2017.

[29] A. Haque, A. Alahi, and L. Fei-Fei, "Recurrent attention models for depth-based person identification," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 1229–1238.

[30] K. Lee, S. Lee, and J. Lee, "Interactive character animation by learning multi-objective control," in *To appear in SIGGRAPH Asia 2018*, Dec 2018.

[31] Y. Li and R. Nevatia, *Key Object Driven Multi-category Object Recognition, Localization and Tracking Using Spatio-temporal Context*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 409–422.

[32] H. S. Koppula and A. Saxena, "Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13. JMLR.org, 2013, pp. III–792–III–800.

[33] CMU. (2016) Cargnie mellon university motion database. [Online]. Available: http://mocap.cs.cmu.edu/

[34] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber, "Documentation mocap database hdm05," Universität Bonn, Tech. Rep. CG-2007-2, June 2007.

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

[36] W. Li, Z. Zhang, and Z. Liu, "Action recognition based on a bag of 3d points," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, June 2010, pp. 9–14.

[37] M. Lau, Z. Bar-Joseph, and J. Kuffner, "Modeling spatial and temporal variation in motion data," *ACM Transactions on Graphics*, vol. 28, no. 5, 12 2009.

[38] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 283–298, Feb 2008.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[40] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, C. Laurent, Y. Bengio, and A. C. Courville, "Towards end-to-end speech recognition with deep convolutional neural networks," *CoRR*, vol. abs/1701.02720, 2017.

[41] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[42] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," *CoRR*, vol. abs/1502.04681, 2015.

[43] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 522–531, Aug. 2004.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[46] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015.

[47] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[48] H. P. H. Shum and E. S. L. Ho, "Real-time physical modelling of character movements with microsoft kinect," in *Proceedings of the 18th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '12. New York, NY, USA: ACM, Dec 2012, pp. 17–24.

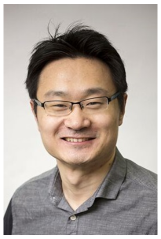
[49] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[50] D. Holden, J. Saito, T. Komura, and T. Joyce, "Learning motion manifolds with convolutional autoencoders," in *SIGGRAPH Asia 2015 Technical Briefs*, ser. SA '15. New York, NY, USA: ACM, 2015, pp. 18:1–18:4.

[51] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014.

[52] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[53] C. S. Catalin Ionescu, Fuxin Li, "Latent structured models for human pose estimation," in *International Conference on Computer Vision*, 2011.



He Wang is an Assistant Professor (Lecturer in UK) at School of Computing, University of Leeds, UK. His research interest is computer graphics, vision and machine learning and applications. Previously he was a Postdoctoral Associate at Disney Research Los Angeles. He received my PhD in 2012 and did a post-doc afterwards both in School of Informatics, University of Edinburgh. Before his PhD, he worked in industry for 4 years after graduating from Zhejiang University, China.

Edmond S. L. Ho is a Senior Lecturer in the Department of Computer and Information Sciences at Northumbria University, Newcastle, UK. Before joining Northumbria University in 2016, he was a Research Assistant Professor in the Department of Computer Science at Hong Kong Baptist University. He received the BSc degree from the Hong Kong Baptist University, the MPhil degree from the City University of Hong Kong, and the PhD degree from Edinburgh University.

Hubert P. H. Shum is an Associate Professor (Reader) in Computer Science at Northumbria University and the Director of Research and Innovation of the Computer and Information Sciences Department. Before that, he was a Senior Lecturer at Northumbria University, a Lecturer at the University of Worcester and a postdoctoral researcher at RIKEN Japan. He received his PhD degree from the University of Edinburgh, his Master and Bachelor degrees from the City University of Hong Kong.

Zhanxing Zhu is an Assistant Professor at Peking University and Beijing Institute of Big Data Research. He is closely affiliated with Deep Learning Lab of Peking University. Previously he obtained his PhD in machine learning from School of Informatics of University of Edinburgh, UK. His research interests cover methodology/theory of machine learning and artificial intelligence and their applications in various areas.