Combining Recurrent Neural Networks and Adversarial Training for Human Motion Synthesis and Control

Zhiyong Wang, Jinxiang Chai, and Shihong Xia

Abstract—This paper introduces a new generative deep learning network for human motion synthesis and control. Our key idea is to combine recurrent neural networks (RNNs) and adversarial training for human motion modeling. We first describe an efficient method for training an RNN model from prerecorded motion data. We implement RNNs with long short-term memory (LSTM) cells because they are capable of addressing the nonlinear dynamics and long term temporal dependencies present in human motions. Next, we train a refiner network using an adversarial loss, similar to generative adversarial networks (GANs), such that refined motion sequences are indistinguishable from real mocap data using a discriminative network. The resulting model is appealing for motion synthesis and control because it is compact, contact-aware, and can generate an infinite number of naturally looking motions with infinite lengths. Our experiments show that motions generated by our deep learning model are always highly realistic and comparable to high-quality motion capture data. We demonstrate the power and effectiveness of our models by exploring a variety of applications, ranging from random motion synthesis, online/offline motion control, and motion filtering. We show the superiority of our generative model by comparison against baseline models.

Index Terms—Deep learning, adversarial training, human motion modeling, synthesis and control

1 Introduction

This paper focuses on constructing a generative model for human motion generation and control. Thus far, one of the most successful solutions to this problem is to build generative models from prerecorded motion data. Generative models are appealing for motion generation because they are often compact, have a strong generalization ability to create motions that are not in prerecorded motion data, and can generate an infinite number of motion variations with a small number of hidden variables. Despite the progress made over the last decade, creating appropriate generative models for human motion generation remains challenging because it requires handling the nonlinear dynamics and long-term temporal dependencies of human motions.

In this paper, we introduce an efficient generative model for human motion modeling, generation and control. Our key idea is to combine the power of RNNs and adversarial training for human motion generation, in which synthetic motions are generated from the generator using RNNs and the generated motion is refined using an adversarial neural network, which we call the "refiner network". Fig. 2 gives an overview of our method: a motion sequence X_{RNN} is generated with the generator G and is refined using the refiner network R. To add realism, we train our refiner network using an adversarial loss, similar to GANs [1], such that the refined motion sequences X_{refine} are indistinguishable from real motion capture sequences X_{real} using a discriminative network D. In addition, we embed contact information into the generative model to further improve the quality of the generated motions.

We construct the generator G based on RNNs. RNNs are connectionist models that capture the dynamics of sequences via cycles in the network of nodes. Recurrent neural networks, however, have traditionally been difficult to train because they often contain millions of parameters and have vanishing gradient problems when they are applied to handle long term temporal dependencies. We address the challenge by using a LSTM architecture [2], which has recently demonstrated impressive performance on tasks as varied as speech recognition [3], [4], language translation [5], [6], [7], and image generation [8].

Our refiner network, similar to GANs [1], is built upon the concept of game theory, where two models are used to solve a minimax game: a generator that samples synthetic data from the model and a discriminator that classifies the data as real or synthetic. Thus far, GANs have achieved state-of-the-art results on a variety of generative tasks, such as style transfer [9], 3D object generation [10], image super-resolution [11], image translation [12] and image generation [13], [14], [15].

Our final generative model is appealing for human motion generation. In our experiments, we show that motions

- *Z. Wang and S. Xia are with the Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy Of Sciences, Beijing 100190, China, and also with the University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: {wangzhiyong, xsh}@ict.ac.cn.*
- *J. Chai is with the Texas A&M University, Uvalde, TX 78801 USA. E-mail: jxchai@gmail.com.*

Manuscript received 7 Sept. 2018; revised 20 June 2019; accepted 11 Aug. 2019. Date of publication 5 Sept. 2019; date of current version 24 Nov. 2020. (Corresponding authors: Jinxiang Chai and Shihong Xia.)
Recommended for acceptance by J. Lee.
Digital Object Identifier no. 10.1109/TVCG.2019.2938520

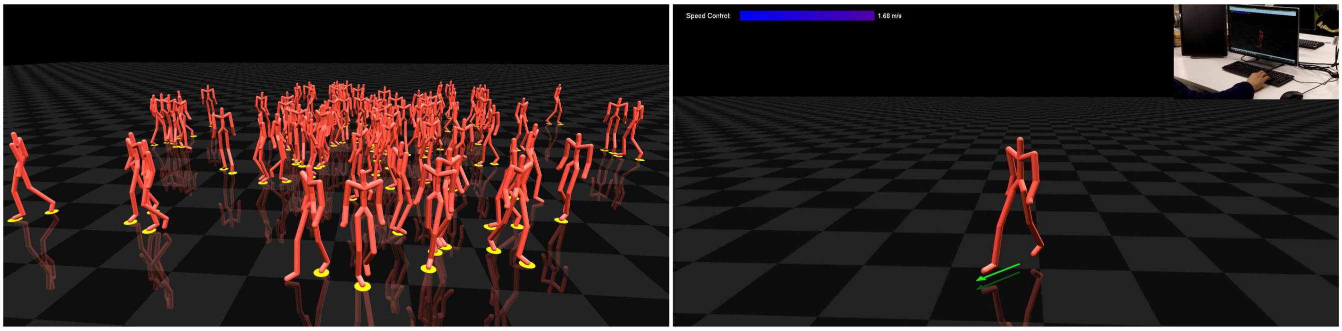


Fig. 1. Human motion generation and control with our model. (left) random generation of high-quality human motions; (right) realtime synthesis and control of human motions.

generated by our model are always highly realistic. Our model is also compact because we do not need to preserve the original training data once the model is trained. In addition, when we acquire new training data, we do not always need to train the model from scratch. If the total amount of data does not exceed the capacity of the network, we can utilize the previous networks as a pretrained model and fine-tune the model with the new training data instead.

We have demonstrated the power and effectiveness of our model by exploring a variety of applications, including motion generation, motion control and motion filtering. With our model, we can sample an infinite number of natural-looking motions with infinite lengths, create a desired animation with various forms of control input, such as the direction and speed of a "running motion", and transform unlabeled noisy input motion into high-quality output motion. We show the superiority of our model through comparison against a baseline generative RNN model. We show that our method achieves a more accurate motion synthesis result than an alternative method with given root trajectory constraints.

1.1 Contributions

Our work is made possible by a number of technical contributions:

- We present a new contact-aware deep learning model for data-driven human motion modeling, which combines the power of recurrent neural networks and adversarial training.
- We train an adversarial refiner network to add realism to the motions generated by RNNs with LSTM cells, using a combination of an adversarial loss and a self-regularization loss.
- We introduce methods for applying the trained deep learning model to motion synthesis, control and filtering.

2 Background

Our approach constructs a generative deep learning model from a large set of prerecorded motion data and uses it to create realistic animation that satisfies various forms of input constraints. Therefore, we will focus our discussion on generative motion models and their application in human motion generation and control.

Our work builds upon a significant body of previous work on constructing generative statistical models for human motion analysis and synthesis. Generative statistical motion models are often represented as a set of mathematical functions that describe human movement using a small number of hidden parameters and their associated

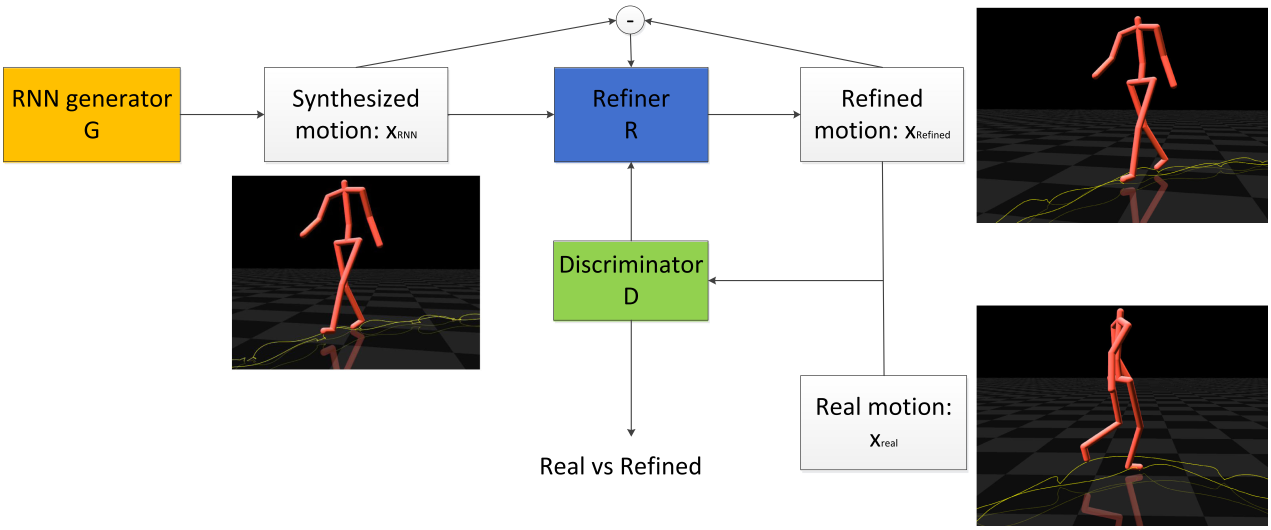


Fig. 2. The Pipeline of our system. We first generate a motion sequence from the RNN generator, G, and then refine the output of the generator with a refiner neural network, R, that minimizes the combination of an adversarial loss and a "self-regularization" term. The adversarial loss "fools" a discriminator network, D, which classifies the motion as real or refined. The self-regularization term minimizes the difference between the synthetic motion and the refined motion.

probability distributions. Previous generative statistical models include Hidden Markov Models (HMMs) [16], variants of statistical dynamic models for modeling spatial-temporal variations within a temporal window [17], [18], [19], [20], [21], and concatenating statistical motion models into finite graphs of deformable motion models [22].

Most recent work on generative modeling has been focused on employing deep RNNs to model the dynamic temporal behavior of human motions for motion prediction [23], [24], [25], [26], [27], [28]. For example, Fragkiadaki and colleagues [23] proposed two architectures: LSTM-3LR (3 layers of LSTM cells) and ERD (Encoder-Recurrent-Decoder) to concatenate LSTM units to model the dynamics of human motions. Jain and colleagues [24] introduced structural RNNs (SRNNs) for human motion prediction and generation by combining high-level spatio-temporal graphs with the sequence modeling success of RNNs. RNNs are appealing for human motion modeling because they can handle nonlinear dynamics and long-term temporal dependencies in human motions. However, as observed by other researchers [25], [26], current deep RNN based methods often have difficulty obtaining good performance in long term motion generation. They tend to fail when generating long sequences of motion, as the errors in their prediction are fed back into the input and accumulate. As a result, their long-term results suffer from occasional unrealistic artifacts, such as foot sliding, and gradually converge to a static pose. Zhou et al. [28] successfully trained an auto-conditioned RNN model that is able to produce a long motion sequence. However, as their model does not provide an estimation of the distribution of the next frame, their model cannot be directly applied to motion control applications.

To address the challenge, we estimate both the means and standard deviations of the motion parameters. We refine the motions generated by RNN using a "refiner network" with an adversarial loss such that the refined motion sequences are indistinguishable from real motion capture data using a discriminative network. Adversarial training allows us to construct a generative motion model that can randomly generate an infinite number of high-quality motions with infinite length, a capability that has not been demonstrated in previous work. Our user studies show that the motions generated by our model have comparable quality to high-quality motion capture data and are more realistic than those generated by RNNs. Our goal is also different from theirs because we aim to learn generative models for human motion synthesis and control rather than motion prediction for video-based human motion tracking. In our experiments, we show that the user can create a desired animation with various forms of control input, such as the direction and speed of a running motion.

Our work is relevant to recent efforts on character animation and control using deep learning methods [29], [30]. For instance, Holden et al. [29] trained a convolutional autoencoder on a large motion database and then learned a regression between high level parameters and the character motion using a feedforward convolutional neural network. In their more recent work, Holden et al. [30] constructed a simple three layer neural network to map the previous character pose and the current user control to the current pose, as well as the change in the phase, and applied the learned regression function for realtime motion control. Our model is significantly different from theirs because we combine RNNs and adversarial training to learn a dynamic temporal function that predicts the probability distributions of the current pose given the character poses and hidden variables in the past. Given the initial state of human characters, as well as the learned generative model, we can randomly generate an infinite number of high-quality motion sequences without any user input. Another difference is that we formulate the motion synthesis and control problem in a Maximum A Posteriori (MAP) framework rather than regression framework adopted in their work. The goal of our motion synthesis is also different because we aim to generate high-quality human motion from various forms of animation inputs, including constraints from offline motion design, online motion control and motion denoising while their methods are focused on for realtime motion control based on predefined control inputs.

Lee et al. [31] presented an approach using LSTM which can handle foot-ground contacts and non-cyclic motions. They introduced a loss term to penalize foot sliding in training, and a data augmentation method based on motion graphs. Our model is different from theirs in two aspects. The first and most important difference is that their model takes control signal and motion from previous frame as input, while ours only relies on motion from previous frame. This difference in input probably makes the model difficult to model non-cyclic motions, which had also been found by [23], [32]. The second difference is that the output of their model is the character's motion, while our generator gives the distribution of the character's motion. As the output of our generator is a distribution but not certain motion, we cannot apply the strategy of Lee et al. [31]. Instead, we make use of adversarial training to refine the synthesized motions.

Our idea of using adversarial training to improve the quality of synthesized motion from RNNs is motivated by the success of using an adversarial network to improve the realism of synthetic images using unlabeled real image data [14]. Specifically, Shrivastava and colleagues [14] proposed Simulated+Unsupervised (S+U) learning, where the task is to learn a model to improve the realism of a simulator's output using unlabeled real data, while preserving the annotation information from the simulator. They developed a method for S+U learning that uses an adversarial network similar to GANs, but with synthetic images as inputs instead of random vectors. We significantly extend their idea for S+U learning for image synthesis to human motion synthesis by improving the realism of human motion generated by RNNs with adversarial network using prerecorded human motion capture data.

3 Overview

Our goal herein is to learn generative networks from prerecorded human motion data and utilize them to generate natural-looking human motions consistent with various forms of input constraints. The entire system consists of two main components:

Motion Analysis. We describe a method for learning the deep learning model from preprocessed motion data. To be specific, our first step is to learn a generative model based on RNNs with LSTM cells. Next, we train a refiner network using an adversarial loss such that the refined motion sequences are indistinguishable from real motion capture data using a discriminative network. We embed contact information into the GANs to further improve the performance of the refiner model.

Motion Synthesis. We show how to apply the learned GANs to motion generation and control. We formulate the problem in an MAP framework. Given the initial state of human characters, as well as the learned generative model, we find the most likely human motion sequences that are consistent with control commands specified by the user and environmental constraints. We combine sampling-based methods with gradient-based optimization to find an optimal solution to the MAP problem. We discuss how to utilize the contact information embedded in the generative model to further improve the quality of output animation. We adopt a similar MAP framework to transform unlabeled noisy input motion into high-quality animation.

We describe the details of each component in the next sections.

4 Motion Analysis

Our goal is to develop a generative model that formulates the probabilistic distribution of the character state at the next frame $\mathbf{x}_{t+1}$ given the character state and hidden variables at the current frame, denoted $\mathbf{x}_t$ and $\mathbf{h}_t$, respectively. Mathematically, we want to model the following probabilistic distribution:

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{h}_t). \tag{1}$$

In the following, we explain how to model the probabilistic distribution using RNNs in Section 4.1 and how to train a refiner network using an adversarial loss such that the refined motion sequences are indistinguishable from real motion capture data in Section 4.2. Section 4.3 discusses how to utilize the contact information embedded in the generative model to further improve the quality of output animation.

4.1 Generative RNN Model

In this section, we first explain our motion feature representation $\mathbf{x}_t$. Then, we give a brief introduction to RNNs and LSTM cells and explain how to apply LSTM to generative motion modeling. In addition, we provide implementation details on how to stabilize training of the RNN model.

4.1.1 State Feature Representation

Each motion sequence contains the trajectory for the absolute position and orientation of the root node (pelvis) as well as the relative joint angles of 18 joints. These joints are head, thorax, and left and right clavicle, humerus, radius, hand, femur, tibia, foot and toe. Let $\mathbf{q}_t$ represent the joint angle pose of a human character at frame t, it can be written as:

$$\mathbf{q} = [\, t_x \; t_y \; t_z \; r_x \; r_y \; r_z \; \boldsymbol{\theta}_2 \; \cdots \; \boldsymbol{\theta}_d \,]^T, \tag{2}$$

where (t_x, t_y, t_z) is the 3D position of the root joint, r_x, r_y, and r_z are the joint angles of the root joint, and $\boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_d$ are the joint angles of other joints.

To define the features of the character state, we choose to use the relative rotation between the current frame and the previous frame for root rotation around the y-axis. We denote it $\triangle r_y$, which represents the relative global rotation of the character. Our translation features on the x- and z-axes are defined in the local coordinates of the previous frame. The values of global rotation and translation features do not change when arbitrary rotations are applied to the character or the root position of the character changes, which means that our feature representation is rotation invariant and translation invariant.

The relationship between joint angle pose $\mathbf{q}$ and state feature $\mathbf{x}$ can be described as follows:

$$\mathbf{x} = [\, \triangle t_x \; \triangle t_z \; \triangle r_y \; t_y \; r_x \; r_z \; \boldsymbol{\theta}_2 \; \cdots \; \boldsymbol{\theta}_d \,]^T, \tag{3}$$

where the first three parameters $\triangle t_x$, $\triangle t_z$, and $\triangle r_y$ are global features, and t_y, r_x, and r_z are the other three components of the root joint.

Similarly, we can easily transform the state features $\mathbf{x}$ back to the corresponding joint angle pose $\mathbf{q}$. As our global rotation and translation are related to the previous frame, we use a homogeneous global transformation matrix to maintain the global state of the character and initialize it to the identity matrix. For every frame, we can obtain the local matrix of the frame from features and update the global transformation matrix by:

$$\begin{aligned} M_{t+1} &= M_t M_{t+1,local} \\ M_{t+1,local} &= \begin{bmatrix} & & & \Delta t_x \\ & Rot_{3\times3}(\Delta r_y) & & 0 \\ & & & \Delta t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \end{aligned} \tag{4}$$

where the right column of M_{t+1} contains the global root position on the x-z plane for frame $t + 1$. The rotation of all the joints can be directly recovered from the motion features.

4.1.2 Motion Modeling with RNNs

RNN is a class of neural networks that has been widely used to model dynamic temporal behaviors. It employs parameter sharing over time by using the same group of parameters for every frame. In our application, it takes the hidden states and current features as input and is trained to predict the probabilistic distribution of the features at the next frame. The hidden states in the RNN model carry the information about the history. This is why RNNs are suitable for handling long term temporal dependence. The derivative of an RNN needs to be computed by Back Propagation Through Time (BPTT) [33] which is similar to normal back propagation methods except for its sequential structure. Similar to many other deep neural networks, RNNs also suffer from the problem of vanishing gradient because the gradient flow must pass through an activation layer in each frame and thus the magnitude of the gradient decreases quickly over time. This prevents the network from taking a relatively long history into account. LSTM cells [2] were introduced to address this challenge. In LSTM cells, the hidden state is divided into two parts: C and h. C carries the

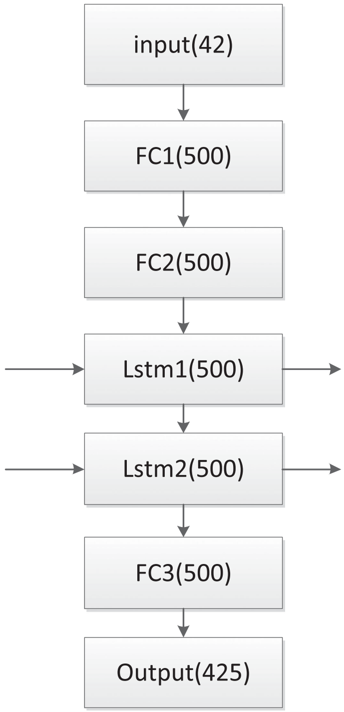


Fig. 3. The structure of our RNN model. It has three fully connected layers (FC1, FC2, and FC3) and two LSTM layers (LSTM1 and LSTM2). The numbers in the brackets are the width of the corresponding layer.

memory of the network, and $\boldsymbol{h}$ is the output of the network. Please refer to the Appendix for more details of LSTM.

LSTM ensures that gradient flow in C_t no longer needs to pass through activation functions. The vanishing gradient problem is therefore significantly reduced. As the LSTM model introduces a new variable C_t that carries the long term memory, the formulation of the problem can be rewritten as follows:

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{h}_t, \mathbf{C}_t). \tag{5}$$

The structure of our RNN motion model is shown in Fig. 3. To predict the distribution of the next frame, our output should not be the state features themselves. Similar to [34], we model the probabilistic distribution of the features in the next frame using a Gaussian Mixture Model (GMM). The distribution of the GMM can be written as follows:

$$p(\mathbf{x}_{t+1}) = \sum_{i=1}^{M} w_i N(\mathbf{x}_{t+1}|\mu_i, \sigma_i), \tag{6}$$

where the ith vector component is characterized by a normal distributions with weights w_i, means μ_i and standard deviations σ_i. To simplify the problem, we assume that the covariance matrix of every Gaussian is a diagonal matrix. The GMM model requires $\sum_{i=1}^{M} w_i = 1$, $w_i > 0$, and $\sigma_i > 0$. However, the output of our network can be in $(-\infty, +\infty)$. To bridge this gap, we define a transformation between the network output $\widehat{w_j}$, $\widehat{\mu}_{i,j}$, and $\widehat{\sigma_{i,j}}$ and the GMM parameters $w_i, \mu_{i,j}$, and $\sigma_{i,j}$ as follows:

$$w_i = \frac{e^{\widehat{w_i}}}{\sum_{j=1}^{M} e^{\widehat{w_j}}}, \mu_{i,j} = \widehat{\mu}_{i,j}, \sigma_{i,j} = e^{\widehat{\sigma_{i,j}}}, \tag{7}$$

where $\sigma_{i,j}$ is the standard deviation of the j-th dimension in the ith Gaussian. By this transformation, we ensure that the weights w_i and the standard deviations of the Gaussian distribution $\sigma_{i,j}$ are positive, and $\sum_{i=1}^{M} w_i = 1$. M is set to 5 in our experiment.

The loss function for every frame can be written as:

$$\arg\min E = -logp(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{h}_t, \mathbf{C}_t)). \tag{8}$$

During training, we compute $\frac{\partial E}{\partial p}$ from the GMM transformation and backpropagate the derivative to the top layers to obtain all the derivatives of the parameters. Our network is trained with RMSProp [35]. In our implementation, the derivative is clipped in the range [-5, 5].

4.1.3 Model Training

As observed by other researchers [25], [26], RNN based generative models often have difficulty obtaining good performance in long term motion generation. They tend to fail when generating long sequences of motion, as the errors in their prediction are fed back into the input and accumulate. As a result, their long-term results suffer from occasional unrealistic artifacts, such as foot sliding, and gradually converge to a static pose. In the following, we summarize our strategies for training RNNs.

Adding Noise into the Training Process. Because the prediction of every frame often has small error, there is a risk that the error might accumulate over time and cause the model to crash at some point. Similar to [34], [36], we introduce independent identically distributed Gaussian noise to the state features $\mathbf{x}_t$ during training in order to improve the robustness of our model prediction. By adding noise to the network input, the network to be learned becomes more robust to noise in the prediction process. As a result, the learned model becomes more likely to recover from small error in the prediction process. In our experiment, we set the mean and standard deviation of Gaussian noise to 0 and 0.05, respectively.

Downsampling the Training Data. We find that downsampling the training data from 120*fps* to 30*fps* can improve the robustness of the prediction model. We suspect that downsampling allows the RNN to handle longer term temporal dependence in human motions.

Optimization Method. Optimization is critical to the performance of the learned model. We have found that RMSProp [35] performs better than SGD method adopted in [23].

The Size of the Training Datasets. Training an RNN model from scratch requires considerable training data. Insufficient training data might result in poor convergence. We find that training an initial model with large diverse datasets, *(e.g., the CMU dataset)* and then refining the model parameters using a smaller set of high-quality motion data can reduce the convergence error and lead to a better motion synthesis result.

The batch size is set to 20, and the time window size is set to 50. Our learning rate is initialized to be 0.001, it is multiplied by 0.95 after every epoch. We train for 300 epochs. We find that the loss converges after approximately 150 epochs, which is approximately 30000 iterations.

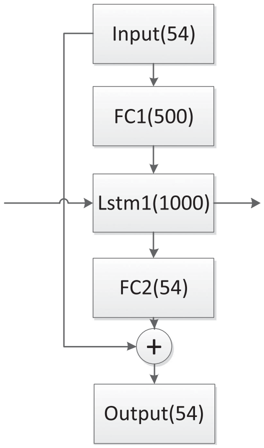


Fig. 4. The structure of the GAN generator network. It consists of two fully connected layers (FC1 and FC2) and one LSTM layer (Lstm1). The numbers in the brackets are the width of the corresponding layer.

4.1.4 Motion Generation with RNNs

We now describe how to generate a motion instance with the learned RNN model. Given the character poses of the initial frames, $\mathbf{q}_0$ and $\mathbf{q}_1$, we first transform them into the features space $\mathbf{x}_0$ and $\mathbf{x}_1$ as we described in Section 4.1.1. For every frame, given the current features $\mathbf{x}_t$ and the current hidden states $\mathbf{h}_t$ and $\mathbf{C}_t$, we apply them as the input of the RNN model to obtain a Gaussian Mixture Model for the probabilistic distribution of the features at the next frame (see Fig. 3), the hidden states are updated at the same time. Next, we sample from the Gaussian Mixture Model to obtain an instance for the features at the next frame $\mathbf{x}_{t+1}$. We repeat the process to generate a sequence of state features over time: $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2, \ldots,$ $\mathbf{x}_n$. Finally, we transform the state features back to the corresponding joint angle poses to create a motion sequence: $\mathbf{q}_0$, $\mathbf{q}_1$, $\mathbf{q}_2, \ldots, \mathbf{q}_n$.

4.2 Adversarial Refiner Training

Starting from the motion features $\mathbf{x}_{RNN}$ generated by the generative RNN model G, our goal herein is to train a refiner network R by using an adversarial loss such that the refined motion data $R_\theta(\mathbf{x}_{RNN})$ are indistinguishable from the real motion data $\mathbf{x}_{real}$ using a discriminative network D, where θ is the parameters of the refiner network.

To add realism to the motion generated by RNNs, we need to bridge the gap between the distributions of synthesized motion data and real motion data. An ideal refiner will make it impossible to classify a given motion sequence as real or refined with high confidence. This need motivates the use of an adversarial discriminator network, D_ϕ, which is trained to classify motions as real vs refined, where ϕ is the parameters of the discriminator network. The adversarial loss used in training the refiner network is responsible for "fooling" the network into classifying the refined motions as real. Following the GAN approach, we model this as a two-player minimax game and update the refiner network R_θ and the discriminator network D_ϕ alternately.

4.2.1 Generative Adversarial Networks

The adversarial framework learns two networks (a generative network and a discriminative network) with competing losses. The generative model (R) (*i.e.*, the refiner network) transforms the motion generated by the RNN ($\mathbf{x}_{RNN}$) into the refined motion $R_\theta(\mathbf{x}_{RNN})$ and tries to "fool" the discriminative model D. The loss function for the generative model is described as follows:

$$\arg\min_\theta \; -log(D_\phi(R_\theta(\mathbf{x}_{RNN}))), \tag{9}$$

where $D_\phi(\mathbf{x})$ is the probability of $\mathbf{x}$ being classified by the discriminative network as real data. This loss function wants to "fool" the discriminative model, so that the refined motion $R_\theta(\mathbf{x}_{RNN})$ should be indistinguishable from real motion data. Here, we use the -logD loss rather than the original log(1-D) loss to avoid the early gradient vanishing problem, especially when the discriminator is too strong.

We define the loss function of the discriminative model as follows:

$$\arg\min_\phi -log(1 - D_\phi(R_\theta(\mathbf{x}_{RNN}))) - log(D_\phi(\mathbf{x}_{real})). \tag{10}$$

This loss function ensures that the learned discriminative model is capable of distinguishing "real" or "refined" data.

In our implementation, we focus the refiner network on the features that are ignored by the generator model. The most important features lost in the RNN model are the positions and velocities of the end effectors. Therefore, we compute the velocities and positions of the end effectors from the input $\mathbf{x}$, denoted $\mathbf{p}_{end}(\mathbf{x})$ and $\mathbf{v}_{end}(\mathbf{x})$, respectively, and include them in the input for both the generative and discriminative models.

Refiner Network. Our generative model has two fully connected layers and one LSTM layer (Fig. 4). Mathematically, the refiner network learns a regression function R_θ that maps the input motion $\mathbf{x}$ as well as the positions and velocities of the end effectors to the refined motion $R_\theta(\mathbf{x})$:

$$\begin{aligned}\mathbf{R}_{input} &= \begin{bmatrix} \mathbf{x} \\ \mathbf{p}_{end}(\mathbf{x}) \\ \mathbf{v}_{end}(\mathbf{x}) \end{bmatrix} \\ R_\theta(\mathbf{x}) &= (\mathbf{w}_{r2} \cdot lstm(relu(\mathbf{w}_{r1} \cdot \mathbf{R}_{input} + \mathbf{b}_{r1})) + \mathbf{b}_{r2}) + \mathbf{x}.\end{aligned} \tag{11}$$

where the network parameters θ include the weights of the LSTM layer (Lstm1) and the weights and biases of two fully connected layers, including $\mathbf{w}_{r_1}$, $\mathbf{w}_{r_2}$, $\mathbf{b}_{r_1}$, and $\mathbf{b}_{r_2}$.

Discriminative Model. For the discriminator, we aim to build a classifier to distinguish the refined motions from the real motions. We apply a bidirectional LSTM [37], which have been demonstrated to be more powerful [37] than an LSTM, to model the discriminative model. The structure of the network is shown in Fig. 5. Mathematically, it can be written as follows:

$$\begin{aligned}\mathbf{d}_{input} &= \begin{bmatrix} \mathbf{p}_{end}(\mathbf{x}) \\ \mathbf{v}_{end}(\mathbf{x}) \end{bmatrix} \\ D_\phi(\mathbf{x}) &= \mathbf{w}_{d2} \cdot bilstm^d(relu(\mathbf{w}_{d1} \cdot \mathbf{d}_{input} + \mathbf{b}_{d1})) + \mathbf{b}_{d2},\end{aligned} \tag{12}$$

where the network parameters ϕ include the weights of the bidirectional LSTM layer and the weights and biases of two fully connected layers, including $\mathbf{w}_{d_1}$, $\mathbf{w}_{d_2}$, $\mathbf{b}_{d_1}$, and $\mathbf{b}_{d_2}$.

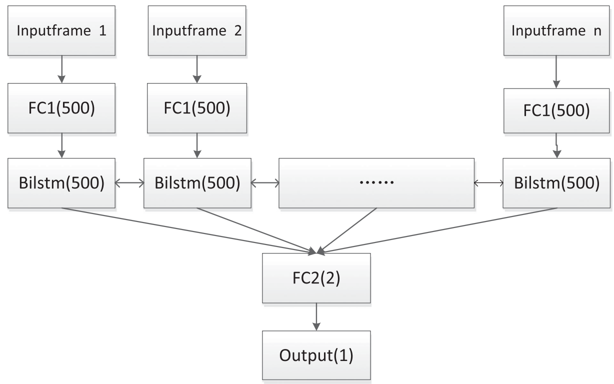


Fig. 5. The structure of the GAN discriminator network. It consists of two fully connected layers (FC1 and FC2) and one bidirectional LSTM layer (Bilstm). The numbers in the brackets are the width of the corresponding layer.

Motion Regularization. We add a regularization term to the generative loss function to ensure that the difference between the input motion and refined motion is as small as possible. This leads to the following generative loss function:

$$\arg\min_{\theta} E = -log(D_{\phi}(R_{\theta}(\mathbf{x}_{RNN}))) + \lambda||root(\mathbf{x}) - root(R_{\theta}(\mathbf{x}))||^2, \tag{13}$$

where root($\mathbf{x}$) and $root(R_{\theta}(\mathbf{x}))$ are the root positions of the input motion $\mathbf{x}$ and the refined motion $R_{\theta}(\mathbf{x})$, respectively. In addition, the weight λ controls the importance of the regularization term. In our experiment, λ is set to 20.

4.2.2 Adversarial Training Details

Adversarial training is hard because of the competition between the generative and discriminative networks. It deteriorates fairly easily when one of the two models is too strong. Our training strategies for adversarial training are summarized as follows:

Training the Generative Model More. We have found that if the two networks are trained equally in a cycle, the discriminator often dominates the generator, leading to crashing of the training. In our practice, the generative model and discriminative model are each updated 75 times in the beginning. After that, we update the generative model five times and the discriminative model once at every step.

Using a History of Refined Motions. Another problem of adversarial training is that the discriminator network only focuses on the latest refined motions. The lack of memory may cause (i) false divergence of the adversarial training, and (ii) the refiner network reintroducing the artifacts that the discriminator has forgotten. To solve this problem, we update the discriminator using a history of refined motion rather than only the ones generated by the current network. To this end, similar to [14], we improve the stability of adversarial training by updating the discriminator using a history of refined motions, rather than only the ones in the current mini-batch. We slightly modify the algorithm to have a buffer of refined motions generated by previous networks. Let B be the buffer size, and b be the batch size. We generate B fake data in the very beginning. At each iteration of discriminator training, we sample $\frac{b}{2}$ motions from the current refiner network, and sample an additional $\frac{b}{2}$ motions from the buffer to update the parameters of the discriminator. After each training iteration, we randomly replace $\frac{b}{2}$ samples in the buffer with the newly generated refined motions. In practice, B is set to 320, and b is set to 32.

Adjusting the Training Strategy when One of the Models is Too Strong. During training, we find that sometimes the discriminative model is easily trained to be too strong such that the generative model is nearly broken. To balance the training, we multiply the iteration times of the generative model by 2 when the discriminator's softmax loss < 0.01. In contrast, we divide the iteration times of the generative model by 2 when the discriminator's softmax loss > 1. We find this to be a useful strategy to avoid unstable GAN training. This approach is similar to that of [15].

Both the generative and discriminative models are trained by RMSProp [35]. We set the learning rate of the refiner to be 0.002 and the learning rate of the discriminator to be 0.005. The decay rate of RMSProp is set to 0.9.

4.2.3 Motion Generation with Adversarial Training

After the refiner network is trained, we can combine it with the generative RNN model for motion synthesis. To achieve this goal, we first choose an initial state that we then use along with the RNN generative model (G) to randomly generate a motion sequence as described in Section 4.1.4. Next, we compute the velocities and positions of the end effectors for every frame of the generated motion sequence. We augment the motion features with the velocities and positions of the end effectors and input them into the refiner network (R) to obtain the refined motion features. In the final step, we transform the refined motion features back to the corresponding joint angle poses to form the output animation.

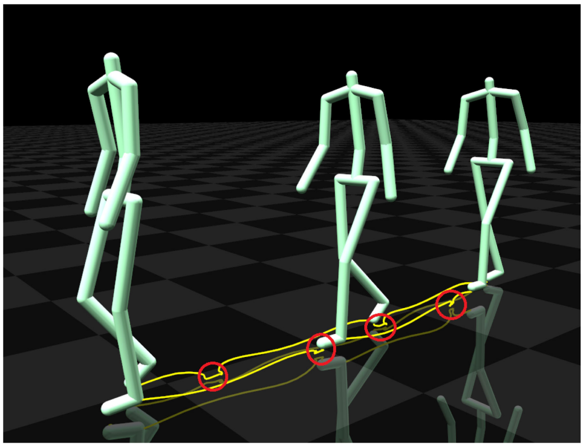
4.3 Contact-Aware Motion Model

In this section, we describe how to embed the contact information into our generative model to further improve the quality of the generated motion. To this end, we first describe our semi-automatic contact labeling. We start from computing the speed of the left and right toes for all the training motion clips. Then, we select a speed threshold (0.45 m/s in practice) for judging the foot contact. For the frames in which the speed of the left/right toe is below the threshold, we label the left/right foot contact state as 1 for the current frame; otherwise, the foot contact state is labeled as 0. After the automatic labeling, we manually check the result to avoid label mistakes caused by noisy motion data.

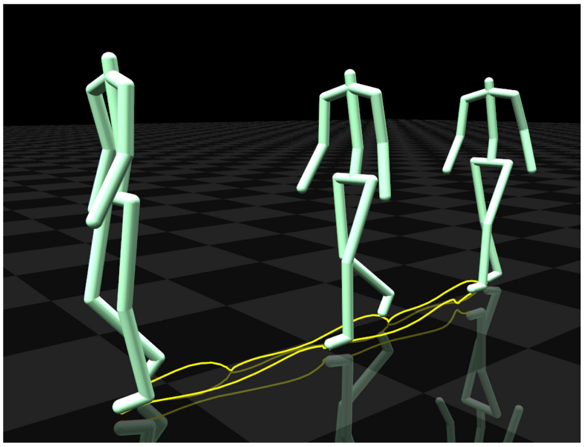
In our application, we encode the contact information into a 2x1 binary vector $\mathbf{c} = [c_l, c_r]^T$. The first bit represents if the left foot of the character is on the ground, and the second bit represents if the right foot is on the ground. We augment the motion features with contact information for RNN motion modeling. Our motion feature can be written as:

$$\mathbf{q} = [\, t_x \; t_y \; t_z \; r_x \; r_y \; r_z \; \boldsymbol{\theta}_2 \; \cdots \; \boldsymbol{\theta}_d \; c_l \; c_r \,]^T. \tag{14}$$

In addition, we also augment both the network input and the network output with the contact vector in adversarial training. Contact awareness further improve the quality of the generated motion. Another advantage of contact-awareness is

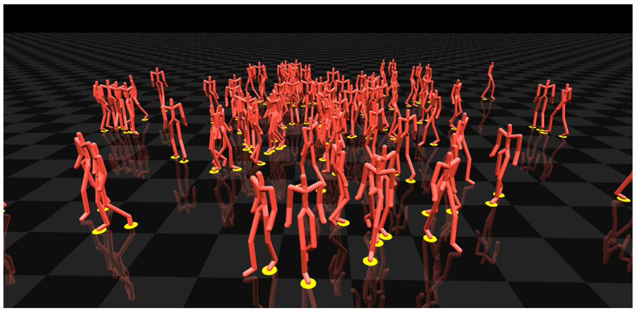


(a)

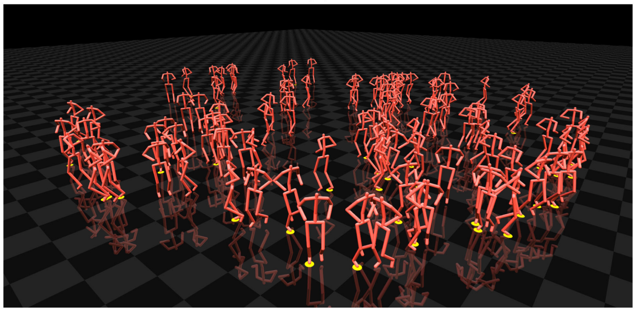


(b)

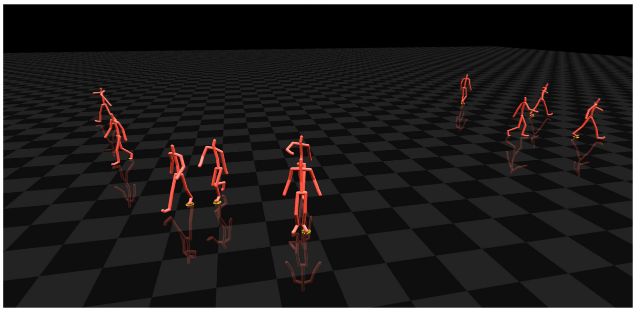
Fig. 6. A comparison between (a) motion generated by the RNN and (b) motion generated by the refinement network. The GAN model eliminates the foot sliding problem and obtains a more robust performance.



(a)



(b)



(c)

Fig. 7. The random motion generation results of (a) walking motion, (b) running motion. (c) motion with different styles from the same initial frame. The yellow circles show the foot contact label.

to automatically label every frame of the generated motion with contact information. This allows us to enforce environmental contact constraints in a motion generalization process, thereby eliminating noticeable visual artifacts, such as foot sliding and ground penetration in the output animation. Fig. 6 shows a comparison before and after the refinement.

5 Motion Synthesis and Control

In this section, we demonstrate the power of our GAN motion model for various applications, including random motion generation, offline motion design, online motion control and motion denoising.

5.1 Random Motion Generation

To achieve accurate motion control, we require a generative model that can generate various motions. We first show our motion's ability to generate different motions. As mentioned in Sections 4.1.4 and 4.2.3, our model can sample various motions from any initial frame. Similar to our approach in training, we allow the input of the network to have a small noise. Experiment shows that this can expand the generative ability of the model. This means that we can have network noise $\mathbf{d}_1$, $\mathbf{d}_2$,...,$\mathbf{d}_n$ as another group of variables in addition to motion features. The generated motion is foot contact-aware because of the contact awareness of our model.

As our model is foot contact-aware, we introduce a postprocessing procedure for offline applications after motion generation. We can extract contact information from generated motion features. We apply an inverse kinematic technique to ensure that the speed of the contact points is zero, and all the contact points are located on the corresponding contact plane. This postprocessing is used for the result of the random motion generation and the offline motion design in our demo. If the contact information in the training data is noisy or the model generates wrong contact label, the postprocessing may lead to an unnatural gait type.

We show a random motion generation example for stylized motions in Fig. 7.

5.2 Offline Motion Design

Our generative model is also well suited for generating natural-looking human motion consistent with user-defined input. We formulate the motion control problem in an MAP estimation framework. We treat the given constraints as the observation from an unknown motion sequence. To ensure that we obtain a motion sequence with high quality even when the given constraints are impossible to satisfy, we assume that the observations contain noise. We use the trained motion model as the motion prior. Our goal is to find a sequence of motion vectors $\mathbf{s}$={s_i,i=1,2,...T} that is most likely to appear. This means that we want to maximize the probability $P(\mathbf{s}, \mathbf{d}|\mathbf{s}_0, \mathbf{c}, \mathbf{e})$. According to Bayes' rule, we have:

$$\begin{aligned} &\arg\max_{\mathbf{s},\mathbf{d}} \; P(\mathbf{s}, \mathbf{d}|\mathbf{s_0}, \mathbf{c}, \mathbf{e}) \\ &= \arg\max_{\mathbf{s},\mathbf{d}} \frac{P(\mathbf{s}, \mathbf{d}|\mathbf{s_0}) P(\mathbf{c}, \mathbf{e}|\mathbf{s}_0, \mathbf{s}, \mathbf{d})}{P(\mathbf{c}, \mathbf{e}|\mathbf{s}_0)} \\ &\propto \arg\max_{\mathbf{s},\mathbf{d}} \; P(\mathbf{s}, \mathbf{d}|\mathbf{s}_0) P(\mathbf{c}|\mathbf{s}_0, \mathbf{s}, \mathbf{d}) P(\mathbf{e}|\mathbf{s}_0, \mathbf{s}, \mathbf{d}). \end{aligned} \tag{15}$$

The first term is the prior term, it can be written as:

$$p(\mathbf{s}, \mathbf{d}|\mathbf{s}_0) = P(\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_T) \prod_{i=1}^{T} P(\mathbf{s}_i|\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{i-1}, \mathbf{d}_0, \ldots, \mathbf{d}_{i-1}). \tag{16}$$

The first part of the prior term is Gaussian noise, it obeys a Gaussian distribution, and the prior for the noise is:

$$P(\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n) = \prod_{i=2}^{n} e^{-\frac{\mathbf{d}_i^2}{2\sigma_{noise}^2}}. \tag{17}$$

Here, σ_{noise} is set to 0.05 by experiment. The second part of the prior is the probability from the RNN-based GAN models.

Control Term. The second term of Equation (15) is the control term. Our model enables the user to accurately control a character at the kinematic level. Low-level kinematic control is important because it allows the user to accurately control motion variations of particular actions. Our motion control framework is very flexible and supports any kinematic control inputs. The current system allows the user to control an animation by selecting a point (e.g., root) on the character and specifying a path for the selected point to follow. The user could also direct the character by defining the high-level control knobs, such as turning angles, step sizes, and locomotion speeds. More specifically, we allow the user to control the root path of an animated character. For root joint projection on the ground for every frame $root(\mathbf{s}_i)$, we first find the nearest points $\mathbf{c}_{near,i}$ on the curve. We assume that there is Gaussian noise with a standard deviation of σ_{fit} for the user's control inputs $\mathbf{c}$. Then, we can define the likelihood of fitting as follows:

$$p_{curve} \propto \prod_{i=1}^{T} e^{-\frac{\|root(\mathbf{s}_i) - \mathbf{c}_{near,i}\|^2}{\sigma_{fit}^2}}. \tag{18}$$

The standard deviation of the fitting term σ_{fit} indicates the preference of the user's fitting accuracy. The smaller it is, the more attention the controller pays to the fitting accuracy. When the given curve is not achievable and σ_{fit}^2 is too small, the synthesized motion would be strange. In our experiment, σ_{fit}^2 is set to 0.5.

Contact Awareness Term. The last term is the contact awareness term. Due to the contact awareness of our model, our generated motion is automatically annotated with contact information. We first retrieve contact information from the network output. Then, for each frame, if there is a contact between the character and the environment, we measure two distances: the first one is the distance between the synthesized contact point on the character in the current frame and the previous frame; the second one is the point plane distance between the synthesized contact point on the character and the corresponding contact plane. We assume Gaussian distributions with standard deviations of σ_{con} for the adjacent frames constraint of the contact-awareness term and $\sigma_{con,y}$ for the point-plane constraint of the contact-awareness term. Then, the contact awareness term can be written as:

$$\begin{aligned} P_{contact} = e^{-\frac{\|f(s_{i-1})_{foot} - f(s_i)_{foot}\|^2}{\sigma_{con}^2}} \\ \cdot e^{-\frac{\|\mathbf{n} \cdot (f(s_{i-1})_{foot} - p_{plane})\|^2}{\sigma_{con,y}^2}}. \end{aligned} \tag{19}$$

Because the original probability is hard to evaluate, we transform it to its -log form, so the overall loss function is

$$\min_{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n, \mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n} E = -log(P_{prior} \cdot P_{noise} \cdot P_{curve} \cdot P_{contact}). \tag{20}$$

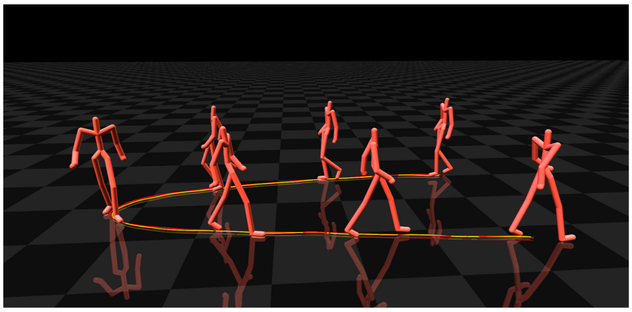
This loss function is nonconvex because of the complexity of the RNN. To solve this optimization problem, we combine the power of sampling and gradient-based optimization to achieve a good result in an acceptable time. Given a starting frame and a user-defined curve, our experiment shows that synthesizing the motion sequence frame-by-frame will not yield good results. This occurs because in an RNN model, the performance in every frame is highly related to previous frames. Therefore, we use spatial temporal optimization to make the generated motion sequence smooth.

In the optimization, the parameters of the neural network are fixed. The optimizer tries to find the solution that best fits the user-defined constraints while satisfying the motion distribution predicted from the neural network and minimizing the noise.

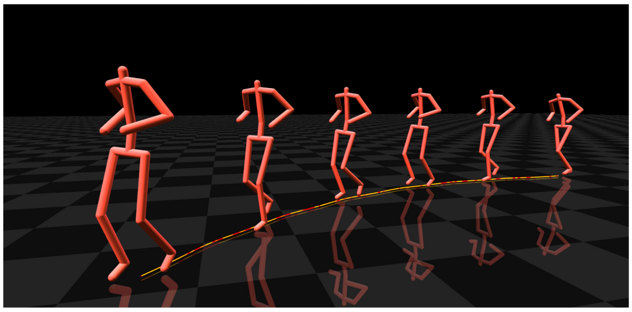
We solve the problem through the sliding windows method. We optimize for a sequence of B frames each time, which we call a "window". For adjacent windows, they share an overlap of b frames. For the initialization of the optimization, the overlapped part with the last window is set to the optimization result from last window, and the other part is sampled by our motion model. We set B and b to 34 and 17, respectively.

Because of the complexity of the loss function, the gradient descent method can be very slow to converge and sometimes leads to local minima. We combine the sampling-based method and gradient-based method to solve the problem. We first apply particle swarm optimization [38] for a few iterations to find a good initialization and then switch to gradient-based optimization to obtain a more precise result. We compute the gradient $\frac{\partial E}{\partial \mathbf{s}}$ and $\frac{\partial E}{\partial \mathbf{d}}$ by the chain rule in the optimization.

As our motion features are based on the global motion states of the previous frame (we use $\triangle r_y$, $\triangle t_x$, $\triangle t_z$ as part of the features), our jacobian matrix for the constraint term



(a)



(b)

Fig. 8. The result of offline motion control. The red line on the ground is the user input constraint, the yellow line is the root trajectory for the synthesized motion. We draw a few keyframe of the synthesized motion here. (a): A walking result, (b): A running result. Our results are best seen in video form.

and foot contact term is more complex than in previous approaches. Therefore, we analytically evaluate it by $BPTT$. Because of the RNN prior term, our problem is not in the least squares form. We apply the LBFGS [39] method to solve the problem.

5.3 Online Motion Control

In addition to offline motion design, our model is also suited for online motion control. We allow the user to control the speed and direction of the character (see Fig. 9). Similar to the offline case, we also model the problem in an MAP framework.

$$\arg\max_{\mathbf{s},\mathbf{d}} P(\mathbf{s},\mathbf{d}|\mathbf{s}_0)P(\mathbf{c}|\mathbf{s}_0,\mathbf{s},\mathbf{d})P(e|\mathbf{s}_0,\mathbf{s},\mathbf{d}). \tag{21}$$

We still have the RNN prior term, the control term and the contact awareness term. One difference lies in the control term, where we need to address direction and speed constraints from the user. We also perform the optimization in a sequence of n frames each time. As a result, the control response has a latency. Here, we set n to 5. As the control term is not the same, we formulate the corresponding loss function for the new control terms. Similarly, our realtime motion control system allows for accurate and precise motion control at the kinematic level. Our current implementation allows the user to control the speed and direction of locomotion, such as walking and running. The online constraints includes speed control and direction control, which can be written as:

$$P(\mathbf{c}|\mathbf{s}_0,\mathbf{s},\mathbf{d}) = P(\mathbf{c}_{speed}|\mathbf{s}_0,\mathbf{s},\mathbf{d})P(\mathbf{c}_{direction}|\mathbf{s}_0,\mathbf{s},\mathbf{d}). \tag{22}$$

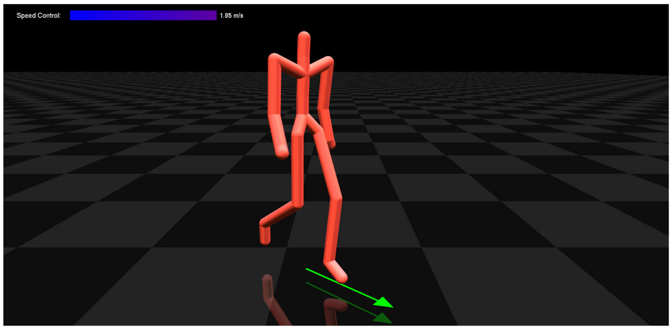


Fig. 9. Online motion control. We have speed and direction control here. The control speed is shown on the left-top of the screen, and the direction is shown by the arrow on the ground. Our results are best seen in video form.

We address speed and orientation online control in the following.

Speed Control. We allow the user to give a speed control to the character. As the speed of the character naturally changes with time, we constrain the character's average speed over the window as close as possible to the given speed control. Assuming that the control result is a Gaussian distribution around the control input with standard deviation σ_{speed}, we have:

$$P(\mathbf{c}_{speed}|\mathbf{s}_0,\mathbf{s},\mathbf{d}) \propto e^{\frac{\|\frac{1}{T}\sum_{i=1}^{T} speed_i - \mathbf{c}_{speed}\|^2}{\sigma^2_{speed}}}. \tag{23}$$

Direction Control. The user can also control the moving direction of the character. We define the facing direction of the character as the angle around the y-axis of the root joint. We measure the difference between the direction of the last frame and the control input. We assume a Gaussian distribution for the control input:

$$P(\mathbf{c}_{direction}|\mathbf{s}_0,\mathbf{s},\mathbf{d}) \propto e^{\|direct_T - \mathbf{c}_{direction}\|^2}, \tag{24}$$

where $direct_T$ is the direction of the last frame in the batch, and $\mathbf{c}_{direction}$ is the direction control input. The optimization problem is also solved by gradient-based optimization. Using GPU acceleration by a GTX970 graphics card, we have an average frame rate of 30 fps.

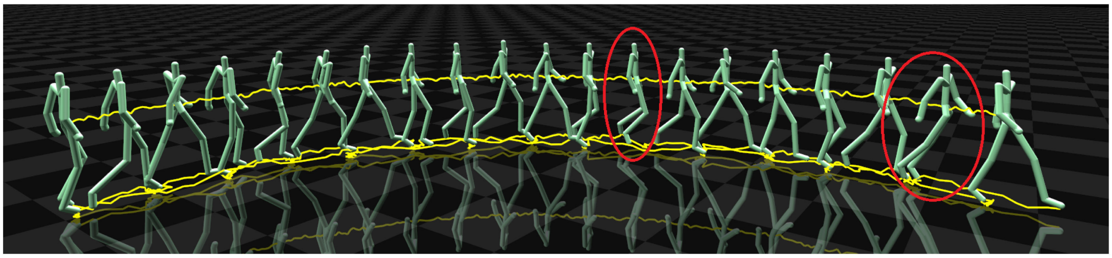
5.4 Motion Denoising

Motion denoising takes a noisy and unlabeled motion sequence as input and generates a natural-looking output motion that is consistent with the input motion (see Fig. 10). We formulate the problem as an MAP problem:

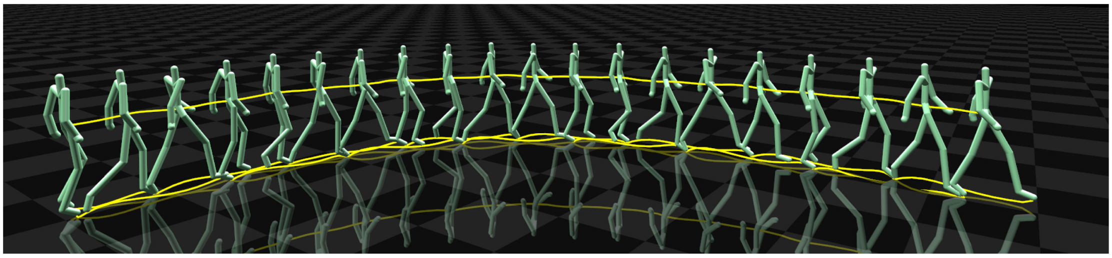
$$\arg\max_{\mathbf{s},\mathbf{d}} P(\mathbf{s},\mathbf{d}|\mathbf{s}_0)P(\mathbf{c}|\mathbf{s}_0,\mathbf{s},\mathbf{d})P(\mathbf{e}|\mathbf{s}_0,\mathbf{s},\mathbf{d}). \tag{25}$$

The first prior term and the third environment contact constraint term is the same as those in Section 5.2. We model the second constraint term in the following.

We assume that the noise consists of two parts: the noise of the root position and the noise of the joint angles. We model both types of noise in the original motion as Gaussian noise. Therefore, we have:



(a)



(b)

Fig. 10. A comparison of motion before and after filtering. The trajectories of the root joint and foot joints are shown in yellow. (a) the original noisy motion. (b) the motion after applying filtering. The joint trajectory of the motion before filtering is jerky and outlier poses are circled in red. This result is best seen in video form.

$$P(\mathbf{c}|\mathbf{s}_0, \mathbf{s}, \mathbf{d}) \propto e^{\frac{(\mathbf{c}_{root} - root(\mathbf{s}_i))^2}{2\sigma^2_{root}}} \cdot e^{\frac{\sum_{j=1}^{n} ||angle_j(\mathbf{c}) - angle_j(\mathbf{s}_i)||^2}{2\sigma^2_{angle}}}, \tag{26}$$

where $\mathbf{c}_{root}$ and $\mathbf{c}_{angle}$ are the root position and joint angles of the original motion, and $root(\mathbf{s}_i)$ and $angle(\mathbf{s}_i)$ are the root position and joint angles of the filtered motion. This optimization problem is also solved by LBFGS [39].

Initial State Estimation. There is a disadvantage for a traditional RNN-based motion model in the first frame. When we perform motion synthesis, the RNN output would have hopping occur between the first frame and the second frame. For most applications, we can simply discard the first frame to avoid the problem. However, certain scenarios (e.g., motion filtering) require us to solve the problem. The traditional RNN-based sequence synthesis method starts the synthesis with the zero state [3], [23]. However, in training, the input states for most frames are nonzero. The network is trained to fit the next frame with previous frame features and the appropriate hidden state. Therefore, when the hidden state of the network is set to zero in the first frame, the network is not sufficiently trained to address this situation and may produce wrong synthesis result. To solve the problem, we estimate the appropriate hidden state for the first frame by our RNN motion model. The appropriate hidden state makes the motion in the second frame very likely to appear in the estimated motion distribution, and the loss function is:

$$\min_{h_1} E = -log \sum_{i=1}^{M} w_i P(\mathbf{s}_2|\mathbf{s}_1, \mathbf{h}_1), \tag{27}$$

where h_1 is the hidden state for the first frame. We estimate the initial hidden state by gradient-based optimization. We compute the gradient $\frac{\partial E}{\partial h}$ by backpropagation and use the gradient descent method to perform the optimization. Given appropriate initial states, the generated motion would be smooth between the first frame and second frame.

6 Results

We have demonstrated the power and effectiveness of our model in motion generation, motion control and motion denoising. To the best of our knowledge, this is the first generative deep learning model capable of generating an infinite number of high-quality motion sequences with infinite lengths. The user studies show that motions generated by our model are comparable to motion capture data obtained by ten Vicon cameras. In addition, we have demonstrated the superiority of our model over a baseline RNN model. Our results are best seen in video form.

We have captured walking and running data of a single actor for 525 motion sequences. These motion sequences vary in speed, step length, and turning angle. In addition, we use CMU dataset to perform pretraining. Our model is relatively small because we do not need to preserve the original motion sequence once the model is trained. Our generative model (41 Mb) is much smaller than the size of the original training datasets (133 Mb). The original dataset contains 419488 frames (58.26 minutes) of walking and running data that vary in speed, step size, and turning angles. When we acquire new training data, we do not always need to train the model from scratch. If the total amount of data does not exceed the capacity of the network, we can utilize the previous networks as a pretrained model and fine-tune the model with the new training data instead.

Random Motion Generation. This experiment demonstrates that our model can generate an infinite number of high-quality animation sequences with infinite length. We select a certain starting frame and then generate 100 motions following the method we described in Section 4.2.3. The results show that the generated motion varies in step size, speed, and turning angle. The results can been seen in Fig. 1. To demonstrate that our method can be used to model various motions, we also train our model on a stylized motion dataset [40]. The result shows that our model samples various motions from the same initial frame. We show the

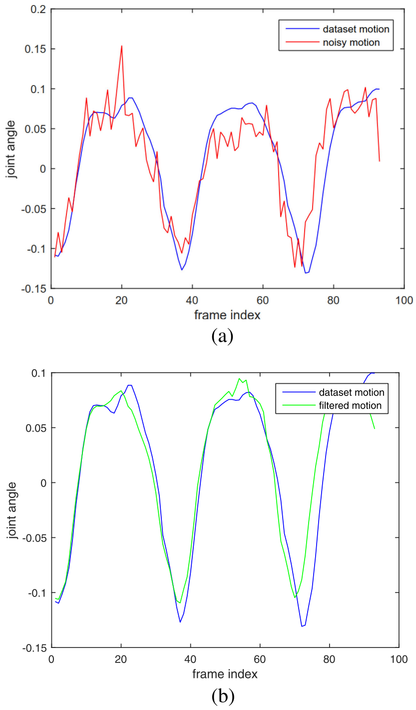


Fig. 11. A comparison of one joint angle for the motion before and after the filtering and a similar motion in the training dataset. The filtered motion is more smooth and similar to the dataset motion.

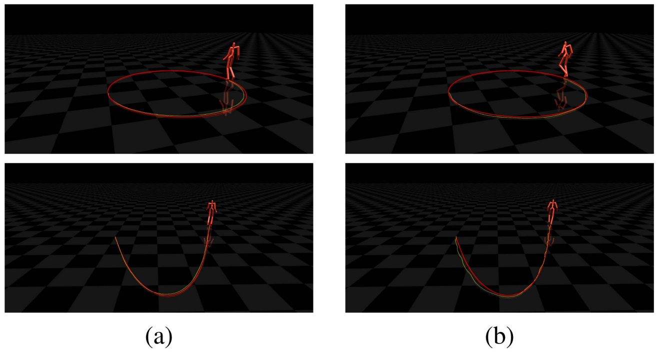


Fig. 12. Comparison against PFNN in offline motion design: (a) result from our method; (b) result from PFNN. In both figures, the given curve is shown in red, and the synthesized root trajectory is in yellow. From top to bottom are the circle curve and sine curve.

result in Fig. 7. This result is best seen in the supplementary video.

Offline Motion Design. The user can generate a desired animation that is consistent with the control input. This experiment shows that our model can generate a desired animation by specifying the projection of the root trajectories on the ground. In the accompanying video, we show the process of creating desired, natural-looking animations with both hand-drawn curves and predefined curves. The results can be seen in Fig. 8.

Online Motion Control. The online motion control system offers precise, realtime control over human characters, including the speed and direction of walking and running. The accompanying video shows that the character can make a sharp turn, e.g., -180 degrees or +180 degrees, or speed up and down with little latency. A result can be seen in Fig. 9.

Motion Denoising. The accompanying video shows a side-by-side comparison between input noisy motion and output motion after denoising for both walking and running. In both cases, the input motion appears very jerky and contains significant foot-sliding artifacts and occasional outlier poses. It is hard to judge when and where foot contact occurs in the input motion. Our motion denoising algorithm automatically identifies the foot contact frames and outputs high-quality motion that closely matches the input motion. One joint angle of the motion before and after the filtering can been seen in Fig. 11. The keyframes of the motion before and after motion filtering can be seen in Fig. 10.

The Advantage of Combining RNNs and Adversarial Training. The accompanying video shows a side-by-side comparison of generated motions from RNNs with and without adversarial training. For both walking and running cases, the results from RNNs alone are occasionally jerky and contain frequent foot-sliding artifacts, while the combined model produces highly realistic motion without any noticeable visual artifacts. This can also be seen in Fig. 6.

Comparison between our Method and PFNN [30]. We evaluate the effectiveness of our method by comparing against PFNN [30]. Our comparison focuses on offline motion design. Although PFNN is designed an online algorithm, the authors also show results of following predefined paths. The purpose of this experiment is to demonstrate that our optimization-based algorithm achieves a more accurate result than online algorithm. As our model is a contact-aware model, our result also has higher quality.

In offline motion design, we select a predefined curve as constraint and then synthesize motions by the two models separately. For our model, the method described in Section 5.2 is used to synthesize a motion that best fits the given constraints while satisfying the motion prior from the neural network. For PFNN, the given constraints are used as the trajectory of the network input. We use the code and trained model from the authors for PFNN [30] in the experiment. We extract the contact information for walking motion clips in the dataset of PFNN, then train our model on these motion clips for the experiment.

We show the given curves and the root trajectories of the synthesized motions in Fig. 12. The difference between them are measured by mean square error (MSE) between the root trajectory of the synthesized motion to the given curve. The result is shown in Table 1 and Fig. 12. The results show that our method achieves a more accurate result.

Motion Quality. We evaluate the quality of synthesized motions via user studies. Specifically, we compare the quality of our synthesized motions against high-quality motion capture data ("Mocap") and those generated from RNNs ("RNN"). We implement the "RNN" synthesis method based on the algorithm described in Section 4.1.4. "Mocap"

TABLE 1
MSE Error in *cm* for our Method and PFNN

method	circle curve	sine curve
ours	1.8428	1.2704
PFNN	2.4764	2.0886

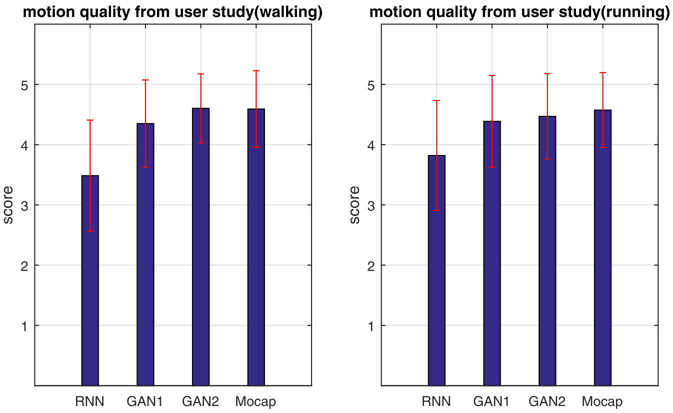


Fig. 13. Comparisons of motion quality. We ask the users to give a score (1-5) of how realistic the synthesized motions are. This graph shows the results of these scores, including the means and standard deviations for motions generated via our methods with and without contact handling ("GAN1" and "GAN2"), a baseline RNNs method ("RNN"), and high-quality motion capture data from ten Vicon cameras ("Mocap").

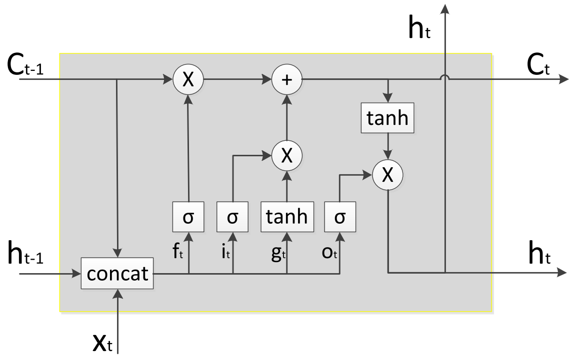


Fig. 14. The structure of LSTM.

represents high-quality motion capture training data captured by ten Vicon [41] cameras. "GAN1" and "GAN2" represent motions generated by a combination of our RNNs and the refiner network with and without automatic foot contact handling, respectively. Note that our contact-aware deep learning model can automatically label every frame of the generated motion with contact information. This information allows us to automatically enforce environmental contact constraints on the output motion.

We have evaluated the quality of motions on 25 users, including males and females. Most of the users have no previous experience with 3D animation. For both running and walking, we randomly generate seven animation sequences from each algorithm along with seven animation sequences randomly chosen from the mocap database. We render animations on a stick figure, similar to Fig. 9. Then, we randomly organize all the animation clips. We ask the users to watch each video and provide a score of how realistic the motion is, ranging from 1 ("least realistic") to 5 ("most realistic"). We report the mean scores and standard deviations for the motions generated by each method, see Fig. 13. The user studies show that the motions generated by our method ("GAN2") are highly realistic and comparable to those from the mocap database ("Mocap"). The evaluation also shows the advantage of combining RNNs and adversarial training for human motion synthesis, as both "GAN1" and "GAN2" produce more realistic results than "RNN". In addition, the score difference between "GAN1" and "GAN2" shows the advantage of embedding contact information into the deep learning model for human motion modeling and synthesis.

7 Conclusion and Future Work

We have introduced a generative deep learning model for human motion synthesis and control. Our model is appealing for human motion synthesis because it is generative and can generate an infinite number of high-fidelity motion sequences to match various forms of user-defined constraints. We have demonstrated the power and effectiveness of our models by exploring a variety of applications, ranging from random motion synthesis to offline and realtime motion control, and motion denoising.

We have shown that motions generated by our model are always highly realistic. One reason is that our generative model can handle both the nonlinear dynamics and long term temporal dependence of human motions. Our model is compact because we do not need to preserve the original training data once the model is trained. Our model is also contact aware and embedded with contact information, thereby removing unpleasant visual artifacts often present in the motion generalization process.

We formulate the motion control problem in a maximum a posteriori (MAP) framework. The MAP framework provides a principled way to balance the tradeoff between user constraints and motion priors. In our experiments, the input constraints from motion control might be noisy and could even be unnatural. When this occurs, the system prefers to generate a "natural-looking" motion that "best" matches the input constraints rather than generating the "best" possible motion that "exactly" matches the user constraints.

We have tested the model on a walking and running dataset. In the future, we plan to test the model on aperiodic motions, such as jumping and dancing. Currently, our model does not achieve a good result for aperiodic motions. One possible solution is combining our model with other systems that can handle complex motions, e.g., [28]. We also plan to test the model on a heterogeneous motion database, such as walking, running, jumping and their transitions. Our system achieves realtime control (30 fps) on a GTX970 card; however, it is still not fast enough for mobile applications. Therefore, one direction for future work is to speed up the system via network structure simplification and model compression.

We show that our generative deep learning motion model can be applied for motion generation, motion control and filtering. In the future, one possibility is to model heterogeneous motions with it. We believe that the model could also be leveraged for many other applications in human motion analysis and processing, such as video-based motion tracking, motion recognition, and motion completion. One of the immediate directions for future work is, therefore, to investigate the applications of the models to human motion analysis and processing.

Appendix

LSTM cells divide hidden states into two parts: h is sensitive to short term memory, while C carries long term memory. Mathematically, it is:

$$
\begin{aligned}
(i, f, o, g)_t &= W_{i,f,o,g} \cdot \begin{bmatrix} C_{t-1} \\ h_{t-1} \\ x_t \end{bmatrix} + b_{i,f,o,g} \\
\widetilde{i_t} &= \sigma(i_t) \\
\widetilde{f_t} &= \sigma(f_t) \\
\widetilde{o_t} &= \sigma(o_t) \\
\widetilde{g_t} &= tanh(g_t) \\
C_t &= \widetilde{f_t} \cdot C_{t-1} + \widetilde{i_t} \cdot \widetilde{g_t} \\
h_t &= \widetilde{o_t} \cdot tanh(C_t),
\end{aligned} \tag{28}
$$

where σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and $tanh(x) = \frac{1-e^{-x}}{1+e^{-x}}$. $\widetilde{f_t}$ is called the forget gate. When $\widetilde{f_t}$ is rather small, all the historical memory of the LSTM cell will be lost. This is useful when we do not want to maintain the memory any more. $\widetilde{i_t}$ is called the input gate. New information is acquired by the long term memory **C** through this gate. o_t is called the output gate. It influences the output of the LSTM cell.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China No. 61772499 and Natural Science Foundation of Beijing Municipality No. L182052.

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Advances Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

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

[3] A. Graves, "Sequence transduction with recurrent neural networks," arXiv:1211.3711, 2012.

[4] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 6645–6649.

[5] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. INTERSPEECH*, 2010, vol. 2, Art. no. 3.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv:1409.0473, 2014.

[8] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra, "Draw: A recurrent neural network for image generation," *CoRR*, vol. abs/1502.04623, 2015. [Online]. Available: http://arxiv.org/abs/1502.04623

[9] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015. [Online]. Available: http://arxiv.org/abs/1508.06576

[10] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3D-R2N2: A unified approach for single and multi-view 3D object reconstruction," *CoRR*, vol. abs/1604.00449, 2016. [Online]. Available: http://arxiv.org/abs/1604.00449

[11] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," *CoRR*, vol. abs/1609.04802, 2016. [Online]. Available: http://arxiv.org/abs/1609.04802

[12] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, July 21-26, 2017, pp. 5967–5976. [Online]. Available: https://doi.org/10.1109/CVPR.2017.632. doi: 10.1109/CVPR.2017.632.

[13] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[14] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2107–2116.

[15] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *CoRR*, vol. abs/1703.10717, 2017. [Online]. Available: http://arxiv.org/abs/1703.10717

[16] M. Brand and A. Hertzmann, "Style machines," in *Proc. 27th Annu. Conf. Comput. Graph. Interactive Techn.*, 2000, 183–192.

[17] Y. Li, T. Wang, and H.-Y. Shum, "Motion texture: A two-level statistical model for character synthesis," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 465–472, 2002.

[18] J. Chai and J. Hodgins, "Constraint-based motion optimization using a statistical dynamic model," *ACM Trans. Graph.*, vol. 26, no. 3, 2007, Art. no.8.

[19] M. Lau, Z. Bar-Joseph, and J. Kuffner, "Modeling spatial and temporal variation in motion data," *ACM Trans. Graph.*, vol. 28, no. 5, 2009, Art. no. 171.

[20] J. Min, Y.-L. Chen, and J. Chai, "Interactive generation of human animation with deformable motion models," *ACM Trans. Graph.*, vol. 29, no. 1, 2009, Art. no. 9.

[21] X. Wei, J. Min, and J. Chai, "Physically valid statistical models for human motion generation," *ACM Trans. Graph.*, vol. 30, pp. 19:1–19:10, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1966394.1966398

[22] J. Min and J. Chai, "Motion graphs++: A compact generative model for semantic motion analysis and synthesis," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 153:1–153:12, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2366145.2366172

[23] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 4346–4354.

[24] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5308–5317.

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

[26] I. Habibie, D. Holden, J. Schwarz, J. Yearsley, and T. Komura, "A recurrent variational autoencoder for human motion synthesis," in *Proc. British Mach. Vis. Conf.*, London, UK, Sept. 4-7 2017. [Online]. Available: https://www.dropbox.com/s/6lqup02kbcpaejz/0414.pdf?dl=1

[27] J. Bütepage, M. J. Black, D. Kragic, and H. Kjellström, "Deep representation learning for human motion prediction and classification," *CoRR*, vol. abs/1702.07486, 2017. [Online]. Available: http://arxiv.org/abs/1702.07486

[28] Y. Zhou, Z. Li, S. Xiao, C. He, Z. Huang, and H. Li, "Auto-conditioned recurrent networks for extended complex human motion synthesis," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r11Q2SlRW

[29] 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. [Online]. Available: http://doi.acm.org/10.1145/2897824.2925975

[30] 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. [Online]. Available: http://doi.acm.org/10.1145/3072959.3073663

[31] K. Lee, S. Lee, and J. Lee, "Interactive character animation by learning multi-objective control," *ACM Transactions on Graphics*, 2018, Art. no. 180.

[32] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5308–5317.

[33] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Netw.*, vol. 1, no. 4, pp. 339–356, 1988.

[34] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[35] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[36] G. W. Taylor and G. E. Hinton, "Factored conditional restricted boltzmann machines for modeling motion style," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1025–1032.
[37] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[38] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*. Berlin, Germany: Springer, 2011, pp. 760–766.
[39] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, no. 1, pp. 503–528, 1989.
[40] 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. [Online]. Available: http://doi.acm.org/10.1145/2766999
[41] Vicon, 2015. [Online]. Available: http://www.vicon.com

Zhiyong Wang received the BSc degree in automation from Tsinghua University (THU), China, in 2011. He is currently working toward the PhD degree in computer science at the University of Chinese Academy of Science, supervised by Prof. Shihong Xia.

Jinxiang Chai received the PhD degree in computer science from Carnegie Mellon University (CMU). He is currently an associate professor with the Department of Computer Science and Engineering, Texas A&M University. His primary research is in the area of computer graphics and vision, with broad applications in other disciplines such as virtual and augmented reality, robotics, human computer interaction, and biomechanics. He received an NSF CAREER award for his work on theory and practice of Bayesian motion synthesis.

Shihong Xia received the PhD degree in computer science from the University of Chinese Academy of Sciences. He is currently a professor of the Institute of Computing Technology, Chinese Academy of Sciences (ICT, CAS), and the director of the human motion laboratory. His primary research is in the area of computer graphics, virtual reality and artificial intelligence.