# A Deep Learning Approach for Motion Retargeting

**Hanyoung Jang, Byungjun Kwon, Moonwon Yu**
Creative AI Team, NCSOFT
Gyeonggi-do, 13494, Rep. of Korea
jhymail@gmail.com

**Jongmin Kim** [*]
Department of Computer Science
Kangwon National University, Rep. of Korea
jongmin.kim@kangwon.ac.kr

## Abstract

Motion retargeting is a process of copying the motion from one (source) to another (target) character when those body sizes and proportion (e.g, arms, legs, torso, and so on) are different. One of the simplest ways to retarget the human motion is manually modifying its joint angles one at a time, however, it would be a difficult and tedious task to get it done for whole the joints of the given motion sequences. Therefore, the problem of automatic motion retargeting has been studied for several decades, however, the quality of resulting motion is on occasion unrealistic as it can hardly consider the details and nuance of human movements when utilizing the numerical optimization only. To address this issues, we present a novel human motion retargeting system using deep learning framework with motion capture data to produce the high-quality human motion. We make use of the deep autoencoder composed of the convolutional layers and a fully connected layer. Our results show that it adjusts a character to meet the kinematic constraints such as bone lengths and foot placements without noticeable artifacts. Furthermore, the proposed method requires the source motion and bone length ratio as input, and it is more intuitive for users than the previous methods. We believe that our method is desirable enough to be used in the game and VFX productions.

## 1 Introduction

We present a novel motion retargeting system using deep learning framework with an intuitive user input model. The motion retargeting is basically copying the motion from one (source) to another (target) when those body sizes and proportion (e.g, arms, legs, torso, and so on) are different. Several researchers have addressed the automatic motion retargeting problem using some conventional numerical methods in the position and joint space. Gleicher (1998) presents an approach for adapting the human motion to create another with different body proportion and formulated this problem as a constrained optimization with space-time constraints. Lee & Shin (1999) suggests a hierarchical motion retargeting framework while satisfying the constraints by employing a multilevel B-spline representation. This method has an advantage of speed as the computationally demanding space-time optimization process is not considered. Hecker et al. (2008) introduces a system for animating a wide range of characters. Even though those produce normally acceptable human motions from the point view of the numerical optimization, the retargeted motions sometimes look unnatural due to the lack of considering the nuances and details of real human motion, therefore, our goal aims to establish deep learning-based human motion retargeting system while retaining a set of user-defined constraints, which are easily violated when using previous retargeting schemes.

Recently, there are some researches of applying the up-to-date deep learning frameworks to the character animations. Holden et al. (2015) proposes deep autoencoder for denoising and fixing corrupted motion capture data. Based on this work, deep learning-based motion synthesis system is introduced by Holden et al. (2016) and is composed of a set of the convolutional neural networks to map from a set of specified low-level user inputs (e.g, positional constraints, trajectory constraints, and etc.) to high level human movement. Holden et al. (2017) also proposes the phase-functioned neural networks which interpolates the weights of four feed-forward networks to make a single character interactively navigate on uneven terrain without foot skating. Our method is different from those

[*]Corresponding Author

proposed researches pertaining to both the aspect of problem and technical approach. To the extent of our knowledge, human motion retargeting using deep learning framework also has not been previously studied, and we think that our method will be quite handy when retargeting various styles of characters, which is often required in many game and VFX (Visual Effects) companies.

Specifically, the neural network that we establish for retargeting human motion is shown in Figure 1. We train a deep autoencoder to map from source to target motion directly. Our network is fed with two inputs: source motion and a set of bone length ratios for the limbs (right arms/legs, left arms/legs, and spine) of a character. Once the neural network is trained, we are able to produce various sizes of retargeted motions by just adjusting the bone length ratios. However, we found that the predicted motion did not completely satisfy the kinematic constraints (bone lengths and foot placements) formulated as the losses of our network because those are hardly generalized and enforced when using a single network only and easily violated even for subtle changes of the bone lengths. To address this issue, we detach those loss functions in terms of the kinematic constraints from our network and put them into a new kinematic optimizer defined at the prediction phase. For the *kinematic optimization*, we first project the predicted motion back into the latent space and then optimize those latent variables to make a character meet a set of kinematic constraints. Hence, this optimization in the latent space at the prediction phase gives us the resulting motion having no visible and noticeable artifacts such as foot skating, jerkiness, and so on.

## 2 Data Processing

We use the CMU motion capture database for training the neural network (CMU (2013)). Similar to the motion data pre-processing from Holden et al. (2015; 2016), each character consists of $j = 21$ joints, and each motion clip to be used for the training neural network has $n = 240$ postures. Each posture is represented as a set of joint positions that is relative to the root position. We also take into account the root rotational velocity and translational velocity along the X and Z axis as the input training features. As a result, the total dimension of input training features for a single training data is 16,560 (69 $\times$ 240). We also annotate the contact state per posture about left and right foot and those are used to enforce foot constraints during the kinematic optimization process.

To establish the training data, we start with two steps: normalizing whole training data (motion capture data) for $\mathbf{X}$ in Figure 1 and retargeting each motion to several motions having different bone lengths for $\mathbf{Y}$. At the first step, we retarget all motion capture data to normalized one. To do so, we need to find a scaling factor, which is generally the ratio between the height of reference and non-normalized T-pose, however, it is at times inaccurate when the length of one's arms are significantly different. To address this issue, we compute the optimal scaling factor $\alpha \in \mathbb{R}$ between the reference and non-normalized T-pose using Procrustes' method (see more details in Sorkine (2009)). First, we can compute the optimal rigid rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and translation $\mathbf{t} \in \mathbb{R}^3$ to transform the non-normalized to reference T-pose by minimizing $\sum_{j=1} w_j \|(\mathbf{R}\mathbf{p}_j + \mathbf{t}) - \mathbf{q}_j\|_2^2$ where $\{\mathbf{p}_j\}$ and $\{\mathbf{q}_j\}$ are a set of joint positions of the non-normalized and reference T-pose, respectively and $w_j$ is the weight value for the $j$-th joint. Then, we are able to obtain the optimal scaling factor $\alpha$ by minimizing $\sum_{j=1} w_j \|\alpha\mathbf{R}(\mathbf{p}_j + \mathbf{t}) - \mathbf{q}_j\|_2^2$ with respect to $\alpha$. We apply this process to each posture and then solve inverse kinematics (IK) (Buss (2004)) to produce the normalized postures.

At the second step, we manually resize the bone lengths based on the user-defined ratio in terms of legs, arms, and spine across whole normalized motions and solve the IK once again. In our experiments, the bone length ratio ranges from 0.8 to 1.8 for the original one and it is uniformly spaced of 0.5, therefore, the 27 (3 $\times$ 3 $\times$ 3) retargeted motion clips are generated from a single motion clip because the bone lengths for the left/right arms or left/right legs should be the exactly same each other. The main reason why we augment the retargeting motions instead of using the original ones only is because those are insufficient to train the deep autoencoder. We use about 10K of the retargeted motion data for training the proposed neural network.

## 3 Neural Network

Figure 1 shows our network. The input of deep autoencoder is composed of joints positions $\mathbf{X} \in \mathbb{R}^{240\times69}$ and the corresponding bone length ratio $\mathbf{Z} \in \mathbb{R}^5$ while the output is the retargeted

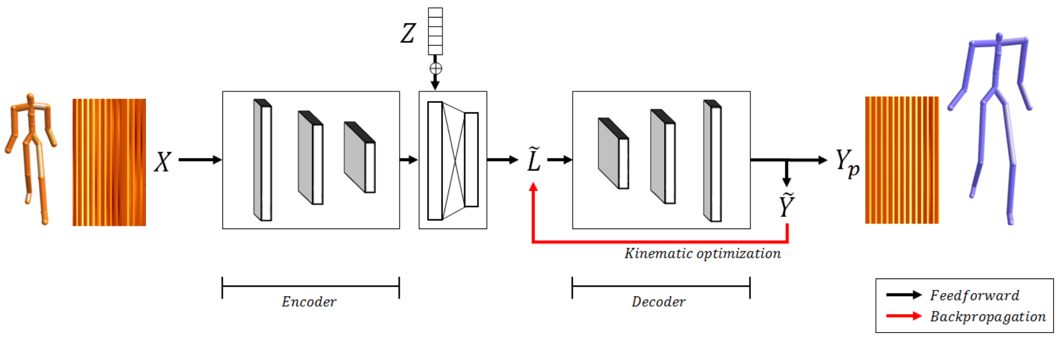

Figure 1: System overview: We train a motion retargeting network to retarget one (source) to another (target) motion and make use of the kinematic optimization process during the prediction phase for making character correctly contact with the ground and follow the desired trajectory while strictly maintaining the bone lengths as well.

Table 1: Configuration of our network

| Layer Type | Kernel | Stride | Outputs Size | Activation |
|---|---|---|---|---|
| Input | | | 240*69 | |
| Convolution | (7,1) | (1,1) | 240*64*1 | RELU |
| MaxPooling | (2,1) | (2,1) | 120*64*1 | |
| Convolution | (7,1) | (1,1) | 120*128*1 | RELU |
| MaxPooling | (2,1) | (2,1) | 60*128*1 | |
| Convolution | (7,1) | (1,1) | 60*256*1 | RELU |
| MaxPooling | (2,1) | (2,1) | 30*256*1 | |
| Fatten + Concat | - | - | 7680+5 | |
| Dense | - | - | 7680 | ELU |
| Convolution | (7,1) | (1,1) | 30*256*1 | RELU |
| Upsampling | (2,1) | (1,1) | 60*256*1 | |
| Convolution | (7,1) | (1,1) | 60*128*1 | RELU |
| Upsampling | (2,1) | (1,1) | 120*128*1 | |
| Convolution | (7,1) | (1,1) | 120*64*1 | RELU |
| Upsampling | (2,1) | (1,1) | 240*64*1 | |
| Convolution | (7,1) | (1,1) | 240*69*1 | |

joint positions $\mathbf{Y} \in \mathbb{R}^{240\times 69}$. The output of the encoder and the bone length ratio $\mathbf{Z}$ are concatenated in the middle of the deep autoencoder and passed to the fully-connected layer. Note that, the network consists of seven 1D convolutional layers and one fully-connected layer between the encoder and decoder. The first three convolutional layers play a role for encoding the motions and also the remaining four convolutional layers are used to decode them. The first convolutional layer has 64 filters and the second convolutional layer has 128 filters and The next convolutional layer has 256 filters.The remaining four convolutional layers have 256, 128, 64, and 69 filters, respectively. Each convolutional layer except the final layer is followed by "RELU" activation function. Table 1 shows the detail of network configuration.

### 3.1 Training

We train the proposed neural network in a way of minimizing the reconstruction and smoothness loss functions based on $\mathcal{L}_2$ norm distance between the predicted and the training motions. As aforementioned, given two types of inputs, $\mathbf{X}$ and $\mathbf{Z}$, our network generates the retargeted motion. The objective loss function that we need to minimize is as follows:

$$\ell_{r,s} = \underbrace{\lambda_r \phi_r(\hat{\mathbf{Y}}, \mathbf{Y})}_{\text{Reconstruction Loss}} + \underbrace{\lambda_s \sum_t \phi_s(\hat{\mathbf{Y}}_t, \hat{\mathbf{Y}}_{t-1})}_{\text{Smoothness Loss}}, \tag{1}$$

where, $\lambda_r$ and $\lambda_s$ are user-defined variables, and $\mathbf{Y}_t$ is the posture at frame $t$ ($1 < t \leq 240$). Here, $\phi(\cdot)$ is $\mathcal{L}_2$ norm distance function. Utilizing a smoothness loss for training the network produces the output motion $\hat{\mathbf{Y}}$ to be smoothly followed the original motion $\mathbf{Y}$. In our experiments, we set $\lambda_r = 1.0$ and $\lambda_s = 0.2$. We do not recommend $\lambda_s$ to be higher than 0.2 because it will have an effect on the output motion to be blurred. We use the Adam adaptive gradient descent method with a learning rate of $1 \times 10^{-4}$ (Kingma & Ba (2014)). Our implementation is based on the Theano (theano:2016) and cuDNN (cudnn:2014) using a single GTX 1080Ti GPU.

## 3.2 Prediction

An initial $\tilde{\mathbf{Y}}$, which is obtained by given $\mathbf{X}$ and $\mathbf{Z}$ in Figure 1, often looks unnatural since kinematic constraints are easy to be violated. To address this issues, we iteratively update $\tilde{\mathbf{L}}$ via kinematic optimization. In the process, the network weights are fixed and the latent variables are adjusted by performing back-propagation, a natural process of neural networks, in terms of foot contacts, trajectory, and bone length loss functions. After the optimization process we obtain the newly updated latent variable $\mathbf{L}_p$, and $\mathbf{Y}_p$ as well. The proposed kinematic optimization performs different loss functions as follows:

$$\mathbf{L}_p = \underset{\tilde{\mathbf{L}}}{\operatorname{argmin}} \; \underbrace{\lambda_b f_b(\tilde{\mathbf{L}}, \mathbf{Z})}_{\text{Bone Length Loss}} + \underbrace{\lambda_f f_f(\tilde{\mathbf{L}}, \mathbf{F})}_{\text{Foot Contact Loss}} + \underbrace{\lambda_t f_t(\tilde{\mathbf{L}})}_{\text{Trajectory Loss}} \;, \tag{2}$$

where $\lambda_b$, $\lambda_f$, and $\lambda_t$ are user-defined variables and $\mathbf{F}$ is the foot contact and toe contacts information that are annotated on the source motion data. In our experiments, we set $\lambda_b = 5.0$, $\lambda_t = 0.5$, and $\lambda_f = 5.0$. Here, $f_b(\tilde{\mathbf{L}}, \mathbf{Z})$ is bone length loss function. The length of a link between two joints could not be strictly preserved during the retargeting process. The length constraints maintain the specified length of a link while preserving the current direction of the link. The foot contact loss function $f_f(\tilde{\mathbf{L}}, \mathbf{F})$ is for getting rid of foot skating that often happens in the predicted motion $\tilde{\mathbf{Y}}$. For forcing the animation to follow the predicted trajectory from the network, we extract the root rotational velocity and translational velocity along the X and Z axis from $\tilde{\mathbf{Y}}$ and minimize the errors arising from them.

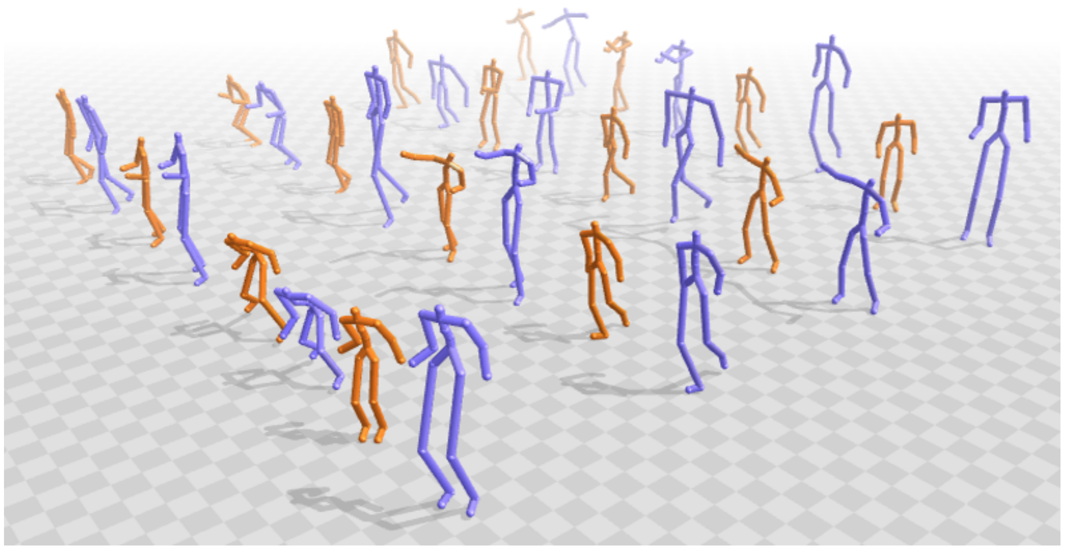
Figure 2: In this example, our system retargets the normalized source motions into the dozens of different types and sizes of output motions while preserving kinematic constraints.

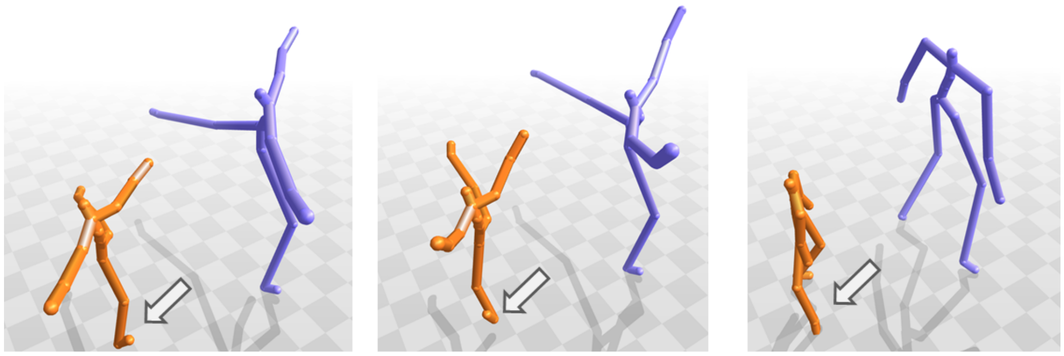
Figure 3: Results of dancing motion. In this example, the results obtained from our method (right) show that ours has an ability to automatically fix the corrupted source motion (left) and retargets it while preserving kinematic constraints. Note that each arrow indicates the problematic joints produced from baseline retargeting method.

## 4 Results

Figure 2 shows the results of our network with several different styles of source motions produced from random bone length ratios ranging from 0.8 to 1.8. Our method produces realistic retargeted motions even for extremely dynamic motions such as jumping and dancing. This reveals that the network potentially and smoothly navigates the manifold of the character motions corresponding to the bone length ratios. Figure 3 shows that the proposed method is also quantitatively and visually appealing about the retargeted characters when compared to the baseline retargeting method. In this figure, the source motion (left) is normalized using the previous method (Buss (2004)) which sometimes gives us the corrupted motion because it optimizes whole joint angles using a non-linear programming rather than manifold so that our method inherently has an advantage of correcting those undesirable source motions to be natural ones.

## 5 Conclusion

We have presented a novel motion retargeting method based on the deep learning framework and believe that resulting motions are desirable enough to be used in the game and VFX productions. We achieve robustness compared to the traditional motion retargeting methods and is also able to fix the corrupted motion that is occasionally obtained when using previous ones. For the future work, we plan to collect more training motion data as much as we can and incorporate our network with the motion style transfer network proposed by Holden et al. (2016) to build more versatile motion retargeting framework. There is also room for accelerating the kinematic optimization by generating more retargeted motions through our method and reusing them as training data.

## References

Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16, 2004.

CMU. Carnegie-mellon motion capture database. http://mocap.cs.cmu.edu/, 2013.

Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 33–42. ACM, 1998.

Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph.*, 27(3):27:1–27:11, 2008.

Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, pp. 18. ACM, 2015.

Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, 35(4):138, 2016.

Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 36(4):42, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 39–48. ACM, 1999.

Olga Sorkine. Least-squares rigid motion using svd. *Technical notes*, 120(3):52, 2009.