Review 5.4 - A Deep Learning Framework for Character Motion Synthesis and Editing

Shivam Thukral

November 19, 2019

1 Description

Data-driven motion synthesis is being used in computer animation as in this animators only need to supply high level parameters. Such approaches cannot be fully automated as they require a significant amount of manual data preprocessing (segmentation, labeling) by human intervention[1].
In this paper the authors propose a deep learning framework to map high level parameters to an output motion by first learning a motion manifold and then producing a mapping between the user input to output motion.
Main contributions of this paper are:

- High quality synthesis of motion from high level parameters with no manual preprocessing
- A method for motion editing for satisfying user constraints and transforming motion style.
- Editing and synthesis are being done in the same framework
- The technique is fast, procedural and parallel.

Summary of proposed method:
The method can be outlined in three stages : disambiguation using foot contact, synthesis and editing. In order to understand foot contact let's first discuss motion synthesis.

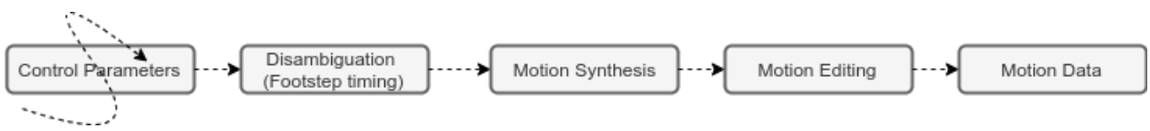

Figure 1: Outline of the method

Synthesis: Convolutional Neural Network has seen a great success in classification and segmentation tasks. Here, the authors use CNN on motion data. In this step, convolutions are being done in the temporal domain. The pose of the character is generated as a weighted sum of the input variables. The resulting motion looks like a character is floating along the ground and is not really walking or following the input curve.
Reason: There is ambiguity in the data. Many different motions have the same trajectory. All of the motions gets averaged by the neural network, this procedures a floating effect where the character is in the average pose.

Disambiguation using foot contact: Introduce extra variable in the input. Most effective variable according to the authors was foot contact times. We can automatically label this using the speed and height of the foot. This process is automated by learning a model which can generate these foot contact times from the trajectories.
Foot Contact: uses a small neural network to map trajectories to contact duration's and frequencies (speed of walk,run). From these two variables we can generate a binary square wave which tells if the feet is on the ground or not. This square wave is also supplied as an input to the feed-forward network.

Editing: Till this point if we try to generate the motion, then this motion will have foot sliding problem and in some cases the trajectory would not be followed properly. Authors can post process this motion but this may not look natural as this might introduce jerkiness of the body. So, they edit the motion manifold learned by a convolutional auto encoder network. They use a large database of motion to find what should be a natural motion or natural pose [Holden et al 2015][2].
Autoencoder: This network tries to reproduce the input you give it. The network learns the projection operator of motion manifold.
In this autoencoder, the hidden unit values parameterize the manifold surface. They are coordinates on the subspace in the high dimensional space. If we adjust the coordinates then we can move on the surface manifold and always keep our motion natural but change our motion.

Constraint Satisfaction Problem: Motion editing is a constraint satisfying problem over the hidden units. Here, they find the hidden unit values such that the motion produced by them satisfies the constraints

e.g. feet should not slide, follow a particular trajectory.

Positional Constraint:

- Local foot velocity must be equal to global foot velocity.
- Global velocity of the character must be equal to the given trajectory.

$$Pos(H) = \sum_j \left\| v_r^H + \omega^H * p_j^H + v_j^H - v_j^H - v_j' \right\|_2^2$$

where $v_j' \in \mathrm{R}^{nx3}$ is the target velocity of joint j in the body coordinate system, and v_r^H , p_j^H , $v_j^H \in \mathrm{R}^{nx3}$ and $\omega^H \in \mathrm{R}^n$ are the root velocity, joint j's local position and velocity, and the body's angular velocity around the Y axis, respectively.

Style Constraint: This constraint can be divided into two parts:

- Content Term (C) : hidden unit values must be similar to the hidden unit values of one input clip.
- Style Term (S) : Gram matrix of hidden unit values must be similar to the gram matrix of the hidden unit values for some style clip.
- Gram Matrix (G): is what encodes the style. This is co-activation of hidden units saying when one hidden unit is active, how active other units should be.

$$Style(H) = s \left\| G(\Phi(S)) - G(H) \right\|_2^2 + c \left\| \Phi(C) - H \right\|_2^2$$

2 Analysis

Results:

- Apart from showing animation results of animating character movements using high level parameters, the authors also evaluate the autoencoder representation by comparing its performance with other comparable networks.
- Authors show another example where the motion character is governed by other control signals example, position of feet as input and training is done on a database of fighting motion to generate kicking and punching controls.
- Finally, the authors show results of before and applying constraints to the character movement. They also transform the style of the motion.

Strengths:

- This system has a fast execution at runtime. This allows us to create motion for many characters at once and hence is suitable for creating animation of large crowds.
  - This approach can compute every pose in the timeline in parallel
- This method is a procedural method. In this, we can jump to an arbitrary point on the timeline and ask for what the pose is for that point in the timeline. This makes it a good fit for animation softwares like Maya.
- Previous procedural approaches require motion to be segmented, aligned and labelled before the data can be included in the model. On the other hand, this model automatically learns from large datasets.
- We can use other control signals for motions as well. For example, the positions of the feet as the input control signal. Here, we train it on a database of fighting motion to generate kicking and punching motions.
- On comparison with motion edited in the hidden unit space with those edited in the Cartesian space by inverse kinematics. The former produces much smoother results as the motion is edited on the learned manifold.
- In this paper, the authors propose to edit motion data in the space of the motion manifold.
- The unsupervised nonlinear manifold learning process does not require any motion segmentation or alignment which makes the process significantly easier than previous approaches.
- Positional constraints can be used in a variety of tasks like picking up a ball etc.

Weakness:

- This paper does not give a general solution to the ambiguity problem. They propose a kind of adhoc solution for locomotion which is a very common task. This is still an open problem in the field of machine learning.
- The input parameters of the feedback forward network framework need to be carefully selected such that there is little ambiguity between the high level parameters and the output motion.

  - These parameters include filter widths of the human motion and those of the trajectories for the feedforward network.
  - These values are initially set through intuition and fine-tuned through visual analysis.

- The current implementation of autoencoder has only single layer. If we want to use more high level features then we need to have deeper network. To handle this we can add more and layers but in this case output tends to get more blurry.
  - One solution for this problem is to use depooling techniques such as hypercolumns [Hariharan et al 2014]

- This approach is not good for interactive applications where the output is computed frame by frame in series.

Impact of the paper:
According to Semantic Scholar this paper has 120 citations. In this, 24 are highly influenced papers and 56 of them cite methods.

3 Clarity of Exposition and Reproducibility

The author clearly sets a background for this paper. In the related work section, the author highlights the following methods:

- Motion Graphs [Heck et al 2007] : This methods involves a lot of human effort. After capturing the motion, we need to cut it into strides, and then we need to align these different strides. After this we need to do classification (left turn, right turn, jog, run etc)

- Methods which interpolate motion or interpolate poses have scalability issues. Such methods need to store the whole data in the memory, For example, using RBF and Gaussian processes this grows to $O(n^2)$ with the number of data points. Hence, such methods are limited to some 1000 data points only. Local methods like KNN require expensive acceleration structures like kd-trees. These structures are very hard to produce as well. [Lee et al 2010] [Park et al 2002].

- RNN methods try to predict the next pose using some number of previous poses. Results generated from RNN methods can be unstable in the sense that they can have high frequency noise and dying out problem [Taylor et al 2011] [Levine et al 2012].

All the details of the network architecture are provided in sufficient depth. However, for learning a manifold of human motion data using Convolutional Autoencoders we have look back into one of the previous work of Holden from 2015[2]. This approach is capable of learning a manifold from CMU database and this manifold is treated as a prior probability distribution over the human motion data. Since most of the algorithm details are well specified and some code is available online, I think we can reproduce this work.

References

[1] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph.*, 35(4):138:1–138:11, July 2016.

[2] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, SA '15, pages 18:1–18:4, New York, NY, USA, 2015. ACM.