# Automatically Learning an Intuitive Animation Interface from a Collection of Human Motion Clips

Marcel Lüdi
*ETH Zürich*

Martin Guay
*Disney Research*

Brian McWilliams
*Disney Research*

Robert W. Sumner
*Disney Research*

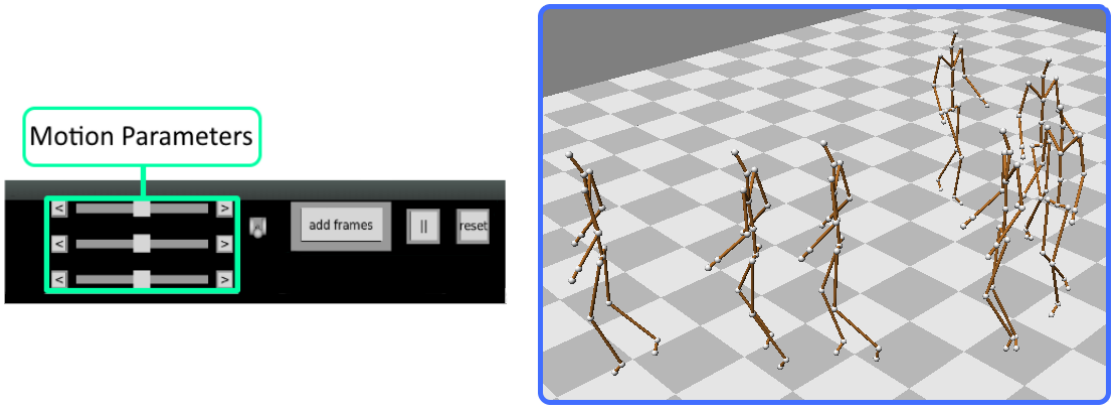


Figure 1. From a disorganized set of human walking motions, we automatically learn an intuitive animation interface with 3 sliders for synthesizing new human walking motions, similar to the ones in the data set. Our algorithm works without any alignment or pre-processing of the data set, and outputs parameters that make sense to humans. By modeling the space of motion as a deep neural network, and constraining the latent variables (our 3 controllers) to be as mutually independent as possible to each other, as well as to cover as wide variations as possible, we obtain parameters that are sensible to human interpretation. The first two dimensions control the left and right step length respectively, while the 3rd parameter controls turning left to right. The user presses "add frames" to generate and append a 1 second motion, to then manipulate the sliders again in a back-and-forth manner to generate longer walking motion sequences.

**ABSTRACT**

In this paper, we automatically learn interpretable low dimensional generative representations of human walking motions using a variational autoencoder. By modeling the latent space of an autoencoder as a low dimensional multi-variate gaussian distribution, we can optimize for an encoding that produces disentangled, independent components which explain most of the variation in the data. The latent variables our model learns are intuitive to humans and can be directly manipulated in a graphical user interface (GUI) via sliders, to generate new walking motions in real time.

**KEYWORDS**

Intuitive interface, character animation, motion manifold.

# 1. INTRODUCTION

Controlling the motion of a digital character is a challenging task as the dimensionality of the character's digital representation is generally high. A long lasting goal of computer animation is to provide intuitive high-level controls for character's shape and motion (e.g. low dimensional parameterizations such as rigs). Data-driven techniques allow generating various new motions similar to the examples in the data-set. However, the learned parameters are either too many, or not interpretable by humans for direct manipulation. For example, applying linear dimensionality reduction on human motion data does not lead to interpretable components, and is not fully automatic as it requires aligning the motion clips.

Deep neural networks, and more specifically convolutional autoencoders, have been recently used to automatically learn low dimensional representations of human motion from disorganized motion clips [Holden et al. (2016)]. However, the learned latent space could not be manipulated directly by humans, mainly due to three reasons. Firstly, the dimensionality required for successful reconstruction remains too high for human manipulation (256 dimensions). Secondly, the granularity of the latent variables can cause small changes in latent space to map to large changes in full space. And thirdly, entanglement in the network can cause the effect of individual latent variables to change when manipulating other variables. For example, a slider may control a foot position, while moving another slider changes the effect of the initial controller from moving the foot to moving a hand; as the network mixes the latent variables to reconstruct the motion.

In this work, we automatically learn a disentangled latent space with low granularity that is interpretable by humans and that can be manipulated directly via GUI sliders to generate new motions in real time. We draw upon the concept of variational autoencoders (VAEs) [Kingma & Welling (2014)], and represent the latent space as a multi-variate gaussian distribution, allowing us to seek a large coverage for each dimension, as well as independence between the dimensions during the training process. This is achieved by not only minimizing the traditional reconstruction cost w.r.t. network parameters, but also by penalizing deviations from a canonical multi-variate normal distribution, i.e. $H \sim N(0,I)$ in latent space, where each dimension should have standard deviation as close as possible to one. This prevents from learning a latent variable representation with extremely fine granularity, and also provides more independence between dimensions—thereby favoring disentanglement. As a result, each latent variable encodes a consistent portion of the motion space and the network preserves this consistency when changing the values of the other latent variables.

While optimizing for independence in the VAE latent space has been successful with various data-sets before, such as 2D face images and digits [Kingma & Welling (2014)], its application to 3D human motion is new and leads to challenges of its own. There is a tradeoff between the quality of the encoding and the meaningfulness provided by the disentanglement. Training directly for 3 dimensions that could be manipulated by a user leads to failures in the training process: the latent space is too small to successfully reconstruct the motion. By training a larger intermediate space of 20 dimensions, we obtain similar reconstruction quality, while at the same time obtaining independent latent variables that are interpretable by humans. However, 20 dimensions still remains too many for intuitive use, and the latent variables are not ordered according to importance. Hence we further reduce the dimensionality to 3 using linear dimensionality reduction, which leads to a compact set of variables the user can manipulate directly to generate new motions—as shown in Fig. 1.

In short, we reduce the space of one second motion clips (61 frames) for a 66 d.o.f. skeleton (i.e. 4026 dimensions) to 20 dimensions with the VAE, and then reduce furthermore to 3 using a PCA. The first two components control the left and right step length respectively, and the third dimension controls turning left to right motions. Note that the effect of each slider remains consistent when modifying the other sliders. In our accompanying video, we show various examples of walking motions synthesized by sequentially appending one second motion clips generated by controlling our three sliders alone.

# 2. RELATED WORK

**Learning generative models of motion**. A large body of work utilizes statistical dimensionality reduction, both at the individual pose and animation clip levels, as an integral part of animation tools. In contrast to our work, the parameters of the learned models are not used by humans directly, but instead used to represent or constrain the character in various optimization-based tasks such as inverse kinematics [Grochow et al. (2004)],

[Wei & Chai (2011)], [Holden et al. (2016)] or space-time optimization [Safonova et al. (2004)], [Chai & Hodgins (2007)], [Min et al. (2009)]. Additionally, training the model is not always fully automatic, or does not always succeed. For example, linear dimensionality reduction applied to motion clips—such as in [Safonova et al. (2004)], [Chai & Hodgins (2007)], [Min et al. (2009)]—requires first aligning the motion clips. Also, non-linear dimensionality reduction methods that can use fewer latent variables as the Gaussian Process Latent Variable Model (GPLVM) used in [Grochow et al. (2004)], does not scale to large data-sets. In contrast, our learning process is fully automatic, successful when trained over large data-sets, and produces model parameters that are intuitive enough for human users to manipulate directly.

Similar to our work, Holden et. al [Holden et al. (2015)], [Holden et al. (2016)] use an autoencoder to automatically reduce a large disorganized data-set. In their work, they reduced down to 256 dimensions and the user cannot manipulate the latent variables directly as the granularity is high and the dimensions are entangled. In contrast, we reduce the latent space down to 3, and the effect of our latent variables on the motion remain consistent during editing—regardless of the configuration of the other dimensions.

**Manually parameterizing motion data-sets**. Another line of work lets the user parameterize motions w.r.t. to meaningful dimensions. Rose et al. [Rose et al. (1998)], in their paper *Verbs and Adverbs*, allow the user to label motion clips of a given category (e.g. walks, referred to as *verbs*) along stylistic dimensions (referred to as *adverbs*), and interpolate between them using radial basis functions. State machines with blend trees are often used in games and interactive applications to parameterize the character's motion w.r.t. a direction. While there has been works on automatically building motion parameterizations (e.g. [Kovar et al. (2002)] [Heck & Gleicher (2007)]), in practice they are built manually by taking perfectly aligned motion clips.

**Deep Learning** has proven very successful at finding intricate structures in high-dimensional data across various domains [LeCun et al. (2015)]. Deep learning algorithms are state-of-the art in object recognition [Krizhevsky et al. (2012)] [Ciregan et al. (2012)] and have been successful in video classification [Karpathy et al. (2014)], [Ji et al. (2013)] and speech recognition [Graves et al. (2013)]. Recently, there has been an interest in using deep convolutional network architectures as generative models to produce novel data from the network [Goodfellow et al. (2014)], [Vincent et al. (2010)]. The strength of deep conv. nets lies in their ability to automatically learn appropriate features in data-sets. For example, in image recognition, the first layers often produce filters similar to edge detection filters while deeper layers hold more complex filters corresponding to different objects [Zeiler & Fergus (2014)]. Our work applies deep conv. nets to learning a motion manifold of human motion.

Du et al. use a hierarchical recurrent neural network trained on a large motion capture data-set to classify different motions [Du et al. (2015)] and achieve state-of-the-art action recognition performance. Taylor et al. [Taylor & Hinton (2009)], [Taylor et al. (2011)] use conditional Restricted Boltzmann Machines (RBMs) to learn a time-series predictor which can predict the next pose of a motion given several previous frames; with improvements made using the spike-and-slab version of the recurrent temporal RBM [Mittelman et al. (2014)]. Holden et al. [Holden et al. (2015)], [Holden et al. (2016)] use a convolutional autoencoder to automatically compute a low-dimensional latent representation of human motion, which is then used inside a path-based motion synthesis interface. It is worth noting that deep conv. nets have been used in other areas of computer animation such as for the control of simulated characters. The network typically represents both a value function and a state-action feedback function within a deep reinforcement learning framework. Levine et al. [Levine & Koltun (2014)] use a neural network to learn optimal control policies for bipedal locomotion and [Peng et al. (2016)] for learning terrain-adaptive locomotion skills.

**Variational Autoencoders**. Variational autoencoders (VAEs) [Kingma & Welling (2014)], [Rezende et al. (2014)] have emerged as a successful way to efficiently learn probabilistic *generative* models of complex data distributions. VAEs compress the data by representing it as a distribution over a low-dimensional latent space whose parameters are learned by an encoding network. The VAE objective aims to simultaneously minimize both the reconstruction error of the decoder network and the *Kullback-Leibler* (KL) divergence between the distribution over the latent space and a standard normal distribution. New data examples can then be obtained by decoding samples taken from the latent distribution.

One well known problem with VAEs is that the latent variables are often uninterpretable. Recently, Higgins et al. [Rezende et al. (2014)] showed that with a minor modification – scaling the KL divergence by a factor, $\beta$ – encourages the independence between the latent variables, resulting in *disentangled* representations which allows the latent variable to be directly interpretable.

In this work, we apply the idea of optimizing for independence in latent space to the case of human walking motion. As a result, the disentangled latent variables are interpretable by humans and the user can directly manipulate them to generate new motion sequences.

## 3. DATA ACQUISITION AND PRE-PROCESSING

Motion capture data often have different skeleton sizes (bone lengths) and dimensionality (number of joints). We first describe how we convert different skeletons into a uniform skeleton. The motion representation fed into the network can impact the learning performance. We wish an encoding that is invariant to global translations and rotations. Hence we convert the motions into a local body frame coordinate system, with the origin located on the ground where the root position is projected onto. By representing each joint position relative to this body frame, we can also easily measure the dissimilarity between poses by measuring the Euclidian distance between joint positions.

***Animation Data***

The data-set we used is mainly composed of walking motions over flat terrain with various left and right turns. It holds a few backward walks, runs, and uneven terrain, but holds no side stepping. The captured motion clips are sampled at 120 frames per second and the final data-set contains approximately 1.5 million frames of walking motion. We gathered motion clips both from the *CMU* data-set [CMU (n.d.)], as well as our in-house produced motions—performed using a *Perception Neuron* system [Ltd (n.d.)].

***Data Formatting***

We now convert the motion clips into a format suitable for training. First we reduce the temporal dimension by sub-sampling the clips at 60 frames per second by interpolating the joint angles for any missing poses. Then we convert the different skeletons into a common skeleton featuring 21 joints. We manually assign the correspondence between joints for the common skeleton. Given a joint correspondence, we transfer the joint angles and then scale the bone lengths to the target skeleton bone length. The final motion may still hold discrepancies, and thus we adjust the joint angles as to match the target skeleton as much as possible; using a full body inverse kinematics solve.

To convert the skeleton into a *body-local* coordinate system invariant to global translation and rotation, we start by computing the global 3D joint positions of the character. Then we compute the forward direction of the character using the vectors between left and right shoulders, as well as left and right hips, averaging them and taking the cross product with the vertical up axis **y**. We compute the relative positional and rotational velocity of each frame w.r.t. its body frame (origin on the ground where the root is projected onto) and store the values. We emphasize that the relative positions of the joints are relative to the body frame and not to a world origin.

Hence, the final global position of the animation is recovered by integrating the velocities of each frame over time. Finally, we compute the mean pose and standard deviation of the whole data set and normalize the each frame by removing the mean and dividing by the standard deviation—same is applied to the horizontal and rotational velocities.

# 4. VARIATIONAL AUTOENCODER

We model and train both an encoder $\Phi$ and decoder $\Phi^{\dagger}$, but note that only the decoder is used for motion synthesis. The encoder $\Phi(X)$ takes an input motion clip $X \in R^{k \times d}$ of d joints by $k$ frames, and encodes it into a c-dimensional latent variable $H \in R^{c}$. From the latent space, a decoder $\Phi^{\dagger}(H)$ is used to compute a motion clip $\hat{X}$ of k frames. The full autoencoder architecture is shown in Fig. 2, and we start by describing the encoder and then the decoder.

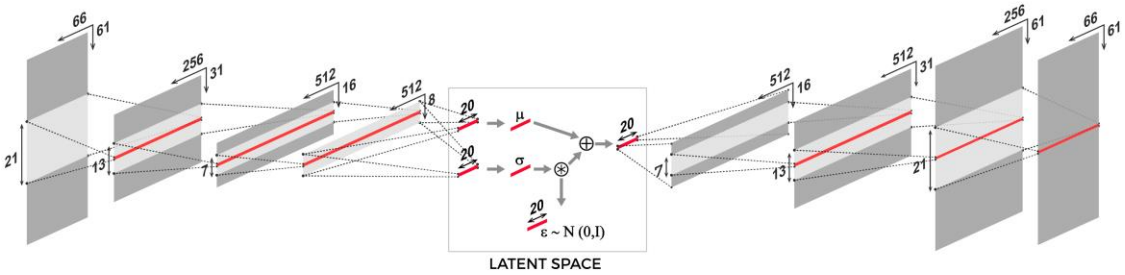


Figure 2: The structure of our variational autoencoder. The encoder represents the latent space as a multivariate gaussian distribution. During training, the mean and standard deviations are combined with a random sample to generate the latent sample corresponding to the input motion clip. See Training section for more details.

***Encoder***

The encoder takes an input animation X and processes it through a traditional convolutional network, with the difference that its last layer outputs a *mean* $\Phi_{\mu}(X)$ and standard deviation $\Phi_{\log(\sigma)}(X)$ to represent the final latent space (as a multi-variate gaussian distribution).

Hence, the input animation first goes through three convolutional layers each followed by a max pooling layer to reduce the temporal resolution:

$$\Phi(X) = ReLU(\Psi(ReLU(\Psi(ReLU(\Psi(X * W_1 + b_1)) * W_2 + b_2)) * W_3 + b_3)) \quad (1)$$

where $\Psi$ is a max pooling layer in the temporal axis with a reduction factor of 2 and $ReLU(x)$ is the nonlinear rectifying operation defined as $ReLU(x) = max(x, 0)$. The operator * denotes a convolution using weight matrices $W_i \in R\hat{}(m_{i-1} \times m_i \times w_i)$ ($m_{i-1}$ is the number of units in the previous layer), $w_i$ the convolution filter width, and $b_i \in R^{m_i}$ are the biases.

After the three convolution layers, the sample goes through two separate fully connected layers resulting in the final the *predicted mean* and *predicted standard deviation*:

$$\begin{aligned} \Phi_{\mu}(X) &= flat(\Phi(X)) \cdot W_{\mu} + b_{\mu}, \\ \Phi_{\sigma}(X) &= flat(\Phi(X)) \cdot W_{\sigma} + b_{\sigma}, \end{aligned}$$

where here $\cdot$ denotes a matrix multiplication, with weights $W_{\mu} \in R^{(s \cdot m_3) \times c}$ and $W_{log(\sigma)} \in R^{(s \cdot m_3) \times c}$ (given s dimensions for each neuron at layer 3) and the $flat()$ function reshapes the output to be one dimensional. In this case s is the number of units after convolutions (the before-last layer). The dimensionality of each layer of the encoder can be found in Fig. 2.

***Decoder***

The decoder $\Phi^{\dagger}$ is similar to the encoder but in reverse order. It takes as input a latent space sample $H \in R^{c}$, processes it in a fully connected layer and then gives the result to a convolutional network which increases the temporal resolution with inverse pooling operations; each by a different margin. The inverse max pooling operation chooses the value of the closest hidden unit on the temporal axis. Hence after the inverse pooling, each neuron chooses its value based on the previous neuron that is closest temporally. The factor by witch the temporal dimension is up-sampled is chosen such that the original poses are still present in the new layer. This means that we actually gradually fill the holes in the animation by generating the missing frames. Finally the neurons are activated using a rectified linear operation, resulting in the decoder function:

$$\Phi^{\dagger}(H) = \Psi^{\dagger}(ReLU(\Psi^{\dagger}(ReLU(\Psi^{\dagger}(flat^{\dagger}(H \cdot W_0 + b_0)) * W_1 + b_1)) * W_2 + b_2)) * W_3 + b_3 \quad (2)$$

where $flat^{\dagger}$ reshapes the flat output of the fully connected layer for further processing into the convolutional layers. In this case, the output of the $flat^{\dagger}$ function is a matrix $H' \in R^{s \times m_0}$ where s is the number of frames after down sampling the original input and $m_0$ is the number of hidden units in the first layer. $\Psi^{\dagger}$ denotes an inverse pooling operation. However, for decoding, the expansion factor is fixed to 2 for all layers to reflect the max pooling from the encoder. The output of this network is an animation clip $\hat{X} \in R^{n \times d}$ with $n$ frames.

## 5. TRAINING

Given a collection of motion clips $X$, the autoencoder is trained to encode the motion clips into latent space and then decode them back into full space. We minimize the error between each initial clip and the reconstructed clip. The difference with traditional autoencoders is that we represent the latent variables as a multi-variate gaussian distribution and we maximize for independence (disentanglement) between components by penalizing each standard deviation from a value of one, which also provides large coverage for each dimension and prevents from high granularity. This is realized through a KL-divergence term that penalizes the distance of the gaussian distribution generated by the network $H(\Phi_{\mu}(X), \Phi_{\sigma}(X))$ to a canonical multi-variate gaussian distribution $N(0, I)$ [Kingma & Welling (2014)]. Both terms together with additional sparsity $\gamma||\theta||_1$ encouraging fewer network parameters $\theta$, results in total cost:

$$Cost(X, \theta) = \alpha||X - \Phi^{\dagger}(H(\Phi_{\mu}(X), \Phi_{\sigma}(X)))||_2^2 + \beta KL(\Phi_{\mu}(X), \Phi_{\sigma}(X)) + \gamma||\theta||_1, \quad (3)$$

where

$$KL(\mu, \sigma) = \frac{1}{2}\sum_{i=0}^{c}(\mu_i^2 + \exp(\log\sigma_i)^2 - 1 - 2\log(\sigma)_i) \quad (4)$$

During training, the sample given to the decoder is computed using the generated mean and standard deviation from the encoder: $H(\Phi_{\mu}(X), \Phi_{\sigma}(X)) = \Phi_{\mu}(X) + \epsilon * \exp(\log\Phi_{\sigma}(X))$, where $\epsilon$ is a random sample generated from a standard normal distribution $\epsilon \sim N(0,1)$.

At the start of training the $\beta$ is set to zero such that the network first learns a stable reconstruction before optimizing the distribution. Over the course of the session $\beta$ gets increased using a sigmoid function which lets the network adapt to the additional cost term.

The training of the network is similar to training a traditional autoencoder. The weights are initialized using the "fan-in" and "fan-out" criteria, while the biases are initialized to zero. The motion clips are randomly drawn from the database and the cost function is minimized using stochastic gradient descent with derivatives computed via Tensorflow [Abadi et al. (2015)], internally using the adaptive gradient descent algorithm Adam. To avoid over fitting to the training data, we used a dropout of 0.2. The variational autoencoder is trained over 200 epochs on two NVIDIA Titan X GPUs.

Figure 3 shows the evolution of the error during training. The autoencoder first learns a hidden representation of the input before being exposed to the KL-divergence term, i.e. before increasing β. Once it reaches a certain value, the reconstruction error increases to minimize the additional KL-divergence cost.

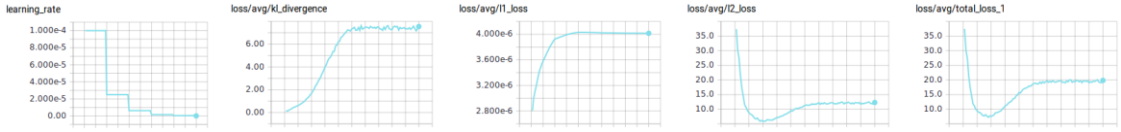

Figure 3: Value of the loss function during the training of our variational autoencoder. The l2-loss is the reconstruction error and the l1-loss is the regularization term.

***Linear Reduction***

We reduced the full space of one second walking motion clips to 20 latent variable spaces. To further reduce the hidden space dimensions, we apply linear dimensionality reduction (principal component analysis) on a subset of the motion clips, and extract the 3 most significant components for the user to manipulate. We first encode the subset of motion clips using the encoder. We compute a linear projection matrix from the encoded motion clips, which allow us to map a 3 dimensional space to the 20 dimensional latent space. As a result, the user can manipulate the 3 sliders shown in Fig. 1 to generate believable motions.

## 6. RESULTS

Our variational autoencoder was trained on the walking motion data-set described in Section 3. We implemented an interactive interface to visualize the reconstructed motions, which is shown both in Fig. 1 and the accompanying video. The user can change either the hidden units (1 of the 20 values), or the final reduced 3 which map to the 20 dimensions. Evaluating the network is fast: from the hidden values, we can synthesize a one second motion clip and render the animation instantly—in real-time.

The initial 20 dimensions are disentangled and some of the variables play a clear role such as turning or stepping length, while others have a smaller influence on the resulting motion. To reduce the number of redundant dimensions, we performed principle component analysis (PCA) on the latent values computed with a subset of the training data (in Section 5). This way, we can retain only the three most important components, leaving out those that had little influence on the motion.

The first two components are associated with the left and right legs. Their magnitudes control the step length: a large value for the first component control step length of the left leg, and the second for the right leg. The sign of the components control the phase of the walk: a negative value for the left leg takes a first step with the left leg, while a positive value shifts the step later in time. In Fig. 4 we show the first pose generated by manipulating these two components.

The third dimension controls the direction in which the character turns. A negative value creates a rotation to the left while a positive value turns in the other direction. Given only these three components, we were able to generate a variety of motions, as shown in the accompanying video. Naturally, further customization and refinement is possible by manipulating some of the individual hidden components (one of the 20 sliders). While the effects on the motions can be subtle, they can provide additional leaning to the right or to the left, or hunching/straightening of the back.

Finally, to generate a longer motion, the user appends several motion clips generated from the three controls sequentially. We blend the synthesized motions over a small temporal window. To facilitate the generation of the next sequence the starting point of the character is set such that the beginning matches the end of the previous animation as closely as possible. To this end the first two dimensions are reversed to change the starting leg of the animation.

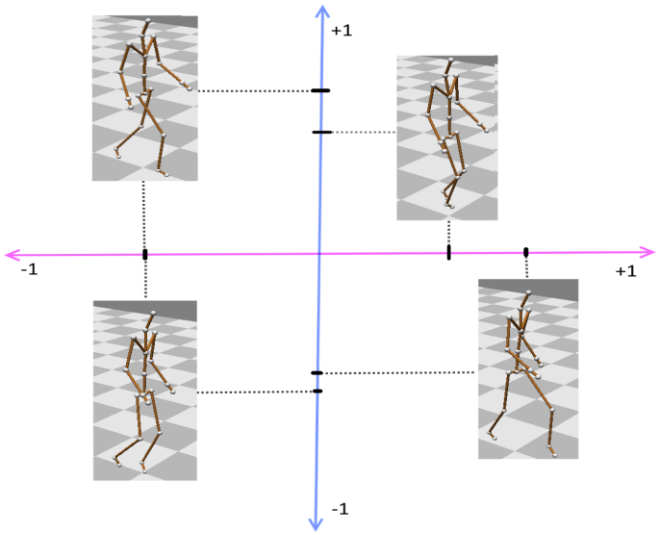


Figure 4: The first two components of the low-dimensional space and their effect on the first pose of the motion. The first dimension (x axis in pink) controls the left leg. Its magnitude controls the length of the step while the sign controls the phase of the step, where a positive value causes the left leg to take a first step. The second dimension (y axis in blue) is analogous to the first but for the right leg.

***Comparing with PCA on Full Space***

In practice non-linear dimensionality reduction techniques such as the GPLVM do not scale well to large data sets. Hence in practice, principal component analysis is used in animation, but this requires aligning the motion clips. We demonstrate here that a PCA cannot be applied directly to the full space to automatically extract meaningful parameters.

We take the normalized training set and subtract the per clip mean. Each motion clip corresponds to a 4026 dimensional point and we reduced it to 5. Unlike with the VAE, none of the first 5 components could capture body movement and they could only capture full body positioning. In other words, the PCA directly applied to the full space cannot compute useful dimensions interpretable by humans. In contrast, we automatically compute roughly disentangled latent dimensions. That is, the effect of a latent variable on the motion remains same—regardless of the values of the other dimensions.

## 7. LIMITATIONS AND FUTURE WORK

Because we optimize for large standard deviations (close to 1), and because our model hold few latent variables—the reconstruction error can be larger than with a traditional autoencoder. On the other hand, each dimension is more independent and better suited for human interpretation.

An important parameter in our model is the latent space dimensionality. In practice there is a trade-off between having few dimensions and the quality of the reconstruction. Our goal is to obtain few dimensions, but in our experience less than 20 resulted in substantially low reconstruction errors. With too few latent components, the model cannot capture enough variation to generate believable results. We were successful by training an intermediate disentangled latent space and then further reducing to 3 using linear dimensionality reduction. In the future we will investigate ways of ordering the dimensions with respect to their importance during the optimization process directly.

Related to the number of latent variables is the length of the input motion sequence. Longer clips require the model to be more expressive meaning the number of latent variables required is higher for accurate reconstruction. After experimentation, we chose 61 frames covering a range of 1 second motions. In this time, a normal walking motion takes about two steps which lends itself well as a cutting point. The filter widths are chosen such that they cover the whole clip. Increasing this value too much results in smoothed out reconstructed motions, or sometimes even failure to converge. Setting too small a width will produce noisy outputs, as the system is not designed to learn the smoothness from the data alone.

Finally, we trained our VAE mainly on forward walking motions with turns, which means we cannot generate backward or side stepping motions. This could be addressed by augmenting the data-set and giving equal importance to all the different types of motions. Learning transitions between different motion types, e.g. between walking and running remains an open question. We believe this may be possible to achieve in the future by interpolating in the appropriate latent space.

## 8. CONCLUSION

We applied the concept of a variational autoencoder to learn a low dimensional, generative representation of human walking motion. Within a fully automated process, we were able to reduce the space of one second human walking motions, down to 3 dimensions, that the user can directly use to generate novel walking sequences (as shown in Fig. 1). The main benefit of our approach is that the learning process automatically identifies disentangled latent variables, whose effect remains consistent throughout the editing process. Hence, the variation induced by each dimension on the generated motion is consistent and interpretable by a human. As a consequence, the latent variables can be directly manipulated via sliders to create believable animations; including walking around corners as well as taking long or short steps, as shown in our accompanying video. In the future, we plan on extending this approach to other types of motions such as running, as well as for automatically learning stylistic attributes.

## ACKNOWLEDGEMENT

A brief acknowledgement section may be included here.

## REFERENCES

Abadi, M. et al, 2015. *TensorFlow: Large-scale machine learning on heterogeneous systems*. http://tensorflow.org/

Chai, J. & Hodgins, J.K, 2007. Constraint-based motion optimization using a statistical dynamic model. *ACM Transactions on Graphics (TOG)* Vol. 26.

Ciregan, D. et al, 2012. Multi-column deep neural networks for image classification. *In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE*. pp. 3642–3649.

CMU (n.d.). *Carnegie Mellon University Mocap Database*. http://mocap.cs.cmu.edu/

Du, Y. et al, 2015. Hierarchical recurrent neural network for skeleton based action recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 1110–1118.

Goodfellow, I. et al, 2014. Generative adversarial nets. *In Advances in Neural Information Processing Systems*. pp. 2672–2680.

Graves, A. et al, 2013. Speech recognition with deep recurrent neural networks. *In 2013 IEEE international conference on acoustics, speech and signal processing, IEEE*. pp. 6645–6649.

Grochow, K. et al, 2004. Style-based inverse kinematics. *ÁCM Trans. Graph.* Vol. 23, No. 3, pp. 522–531.

Heck, R. & Gleicher, M, 2007. Parametric motion graphs. *In Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, I3D '07, ACM*. New York, NY, USA, pp. 129–136.

Holden, D. et al, 2016. A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph.* Vol. 35, No. 4, pp. 138:1–138:11.

Holden, D. et al, 2015. Learning motion manifolds with convolutional autoencoders. *In SIGGRAPH Asia 2015 Technical Briefs, SA '15, ACM*. New York, NY, USA, pp. 18:1–18:4.

Ji, S. et al, 2013. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*. Vol. 35, No. 1, pp. 221–231.

Karpathy, A. et al, 2014. Large-scale video classification with convolutional neural networks. *In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. pp. 1725–1732.

Kingma, D. P. & Welling, M, 2014. Auto-encoding variational bayes. *International Conference on Learning Representations, ICLR*.

Kovar, L. et al, 2002. Motion graphs. *In Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02, ACM*. New York, NY, USA, pp. 473–482.

Krizhevsky, A. et al, 2012. Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems*. pp. 1097–1105.

LeCun, Y. et al, 2015. Deep learning. *Nature*. Vol. 521, No. 7553, pp. 436–444.

Levine, S. & Koltun, V, 2014. Learning complex neural network policies with trajectory optimization. *In ICML*. pp. 829–837.

Ltd, N. (n.d.). *Perception Neuron*. https://neuronmocap.com/content/axis-neuron-software

Min, J. et al, 2009. Interactive generation of human animation with deformable motion models. *ACM Transactions on Graphics (TOG)*. Vol. 29.

Mittelman, R. et al, 2014. Structured recurrent temporal restricted boltzmann machines. *In Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. pp. 1647–1655.

Peng, X. B., 2016. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)*. Vol. 35, No. 4.

Rezende, D. J. et al, 2014. Stochastic backpropagation and approximate inference in deep generative models. *In Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. pp. 1278–1286.

Rose, C. et al, 1998. Verbs and adverbs: Multidimensional motion interpolation using radial basis functions. *IEEE Computer Graphics and Applications*. Vol. 18, pp. 32–40.

Safonova, A. et al, 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *In ACM SIGGRAPH 2004 Papers, SIGGRAPH '04, ACM*. New York, NY, USA. pp. 514–521.

Taylor, G. W. & Hinton, G. E, 2009. Factored conditional restricted boltzmann machines for modeling motion style. *In Proceedings of the 26th annual international conference on machine learning, ACM*. pp. 1025–1032.

Taylor, G. W. et al, 2011. Two distributed-state models for generating high-dimensional time series. *Journal of Machine Learning Research*. pp. 1025–1068.

Vincent, P. et al, 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*. pp. 3371–3408.

Wei, X. & Chai, J, 2011. Intuitive interactive human-character posing with millions of example poses. *IEEE Comput. Graph. Appl.* Vol. 31, No. 4, pp. 78–88.

Zeiler, M. D. & Fergus, R, 2014. Visualizing and understanding convolutional networks. *In European Conference on Computer Vision*. pp. 818–833.