Abstract

In this thesis, we will propose a stochastic hierarchical neural network, capable of learning and generating complex temporal patterns. Our model is a Deep Belief Net consists from one Restricted Boltzmann Machine and one Conditional RBM. We have chosen human motion as training target because it is a complex pattern with high degrees-of-freedom. The model is trained with motion data captured from actual human motions, and generates same motions as well as new unseen motions. During motion generation, a user can select in real-time which motion to generate next, and the model will generate a motion of naturally shifting from the current motion to the selected next motion. Interestingly, rhythmic motions like walking and running are generated by limit cycles in hidden state space of the model. Also, an unseen gait transition motion between two motions is generated by a bifurcation between corresponding limit cycles. Those dynamical behaviors give robustness and flexibility to the motion generation.

Contents

Chapter 1

Introduction

1.1 Background

Generating a realistic controllable human motion is essential in many applications, such as humanoid robots, character animations, computer games, etc. Unfortunately, human motion is a complex temporal pattern with high degrees of freedom which makes analytic studies difficult. The easiest way to produce a realistic human motion is to use real human motions, which can be acquired by a motion capture system.

The most common way to utilize Human Motion Capture Data (HMCD) in motion generation is motion synthesis where frames from HMCD are directly used in a generated motion. While such data-driven methods produce high quality motions, it is difficult to synthesize a new unseen motion. To synthesize a new motion, small segments from HMCD have to be connected in new order. Finding a right sequence of segments and connecting them smoothly is a time-consuming hard task that requires professional animators and extensive datasets of HMCD. Also those methods works better for off-line applications where whole motion is planned before generation.

A more indirect way to employ HMCD in motion generation is to build a statistical learning model. In this case, HMCD will be used as training data in learning, and motions will be generated by the model alone. The advantage of model-based approach is that generated motions are not restricted to the training motions. If the generalization of the model is sufficient, the model should be able to generate not only the trained motions, but also new unseen motions. Recently, Hidden Markov Model (HMM) [1] and Restricted Boltzmann Machine (CRBM) [2, 3] were used in human motion generation. But those models were more concentrated on regenerating the learned motions, or generating a new motion by changing the style of the learned motions.

1.2 Purpose of the thesis

In this thesis, we will propose a generative model capable of learning temporal patterns. The model is a Deep Belief Net (DBN) with conditional layers for temporal learning. DBN is a deep generative graphical model first introduced by Hinton in 2006 [4], and since then it received a lot of attentions from the machine learning community for its easy unsupervised learning and hierarchical deep architecture. Our main goal is to take advantage of the deep architecture of DBN to generate natural human motions with an on-line control. We have chosen human motion as training target because it is a complex pattern with high degrees-of-freedom. Also, it has shown that human motion can be generated by RBM [2], which is the building block of DBN.

The model consists from two components: RBM and Conditional RBM (CRBM). The RBM is for encoding joint angles in a motion frame into a more compact hidden state. The CRBM is for predicting the next hidden state from previous hidden states. In other words, the RBM learns the spatial constraints and the CRBM learns the temporal constraints in the motion data. This is similar to HMM, where transition probabilities predict the next hidden state from the previous hidden state. Then, hidden states are projected into visible states via observation probabilities. However, the main difference between our model and HMM is that, the number of hidden states in HMM are fixed and each hidden state is represented by single unit. On the other hand, a hidden state in RBM is represented by the values of all hidden units, which gives exponentially many hidden states.

One major aspect of our approach is that the model try to mimic not only the look of human motions, it also try to exhibit dynamical behaviors of human motions. For example, rhythmic motions like walking and running can be considered as a stable limit cycle because those motions are stable and robust against outside perturbations. Also, a transition between two different gaits can be considered as a bifurcation between corresponding limit cycles. Those dynamical behaviors are vital to generating natural human motions and our one purpose is to show those properties on a neural network trained by HMCD.

1.3 Outline of the rest of the thesis

The thesis is organized as follow. Chapter 2 will present work related to this thesis, covering data-driven motion synthesize methods, robot controls inspired Central Pattern Generator and model based deep learning methods. In Chapter 3, Boltzmann Machine (BM) and Restricted Boltzmann Machine (RBM) will be reviewed. Since our model is constructed from them, this chapter will give theoretical basis for the learning and generation in the model. Chapter 5 will define the proposed model and explain learning and generation of motion data in detail. Interesting results from experiments of motion generation will be presented in Chapter 5. Finally, Chapter 6 will conclude this thesis.

Chapter 2

Related work

In this chapter, work related to this thesis will be introduced. First, work done in motion synthesis will be discussed. Next, we will present an approach of robot control inspired Central Pattern Generator. Finally, work about Deep Belief Net and RBM will reviewed.

2.1 Motion synthesis

In recent years, motion capture systems have become a vital tool for character animations, and even for robot controls [5]. While human motion capture data (HMCD) produces a very high quality motion, it is difficult to edit or to generate a new motion. Another weakness such data-driven method is time-consuming work of choosing a right motion sequence from a large dataset to produce the desired motion. There are clever methods like Motion Graph [6] and Interpolation [7] that make a motion synthesis easy and better.

In Motion Graph method, a graph is made from a motion dataset by making motion segments vertexes, and connecting similar motion segments with edges. Then, one have to search for an appropriate path on the graph to produce a desired motion. This method is more efficient than simply searching through whole dataset. Interpolation makes it possible to smoothly connect two different motions by blending the two motions in varying proportion. But if the two motions are correlated, then arbitrary blending will produce an unnatural motion. For example, one cannot interpolate walking and running motion without considering the correlation between two motions such as feet contacts with the ground.

Another set of methods that are used frequently in a motion synthesis are statistical learning models. Those methods are fundamentally differ from data-driven methods. While data-driven methods directly use frames from motion capture data in generated motions, learning models use motion capture data only in a learning process as training data to optimize parameters and then use those learned parameters in motion generation. So after the training, the model doesn't need motion capture data and learning models are likely better at generalization.

Hidden Markov Model is one of probabilistic models used for temporal analysis in many successful applications including a human action analysis [1]. To model human motion by HMM, one need to assign each hidden state to a Gaussian mixture in the state space of joint angles and have to learn transition probabilities between those hidden states. Then, by sampling from HMM, we will get a trajectory in joint angle state space, which is equivalent to a motion. The number of hidden states must be set before learning and too many states could overfit to training data and too few states could be not enough. Also, when there is no transition motion between two different motions in the training data, then HMM will never generate the transition motion. Besides from motion generation, HMM can used in clustering [8] and segmenting [9] of motion capture data.

Stochastic learning model called Conditional Restricted Boltzmann Machine was used in learning and generation of human motion [2]. Actually our model is based on this model and has many similarities. Unlike HMM, CRBM is capable of generating multiple gait motion by a single model and transitions between those gaits are possible by adding noise to the hidden layer. But the transition was stochastic and uncontrollable in CRBM. Also, little more complicated model called Factored CRBM [3], where connections are factored by a style layer, was able to separate the style from motions. There is also HMM based model called Style Machines [10] that do the same thing. Those are the proof of the generalization power of model-based motion generation methods.

2.2 CPG inspired robot control

Central Pattern Generator (CPG) is a small neural network found in the spinal cord of vertebrates. It is proven that CPG has a crucial role in the motion generation of animals and humans. There was an explanation that complex motions in animals are a chain of simple reflexes responding to sensory feedbacks. But it became clear that rhythmic motions do not require sensory input by the discovery of the autonomous motion generation by CPG [11] where an isolated spinal cord of the lamprey still produced patterns of activity of locomotion. Also, it was found that high level nervous processing is not vital in locomotion when walking motion is observed in a decerebrated cat [12]. The simple electrical stimulation of the specific region in a brain stem was enough to induce walking motion at various speeds, and even gait transition by increasing the stimulation level. However, both bottom-up sensory feedbacks and top-down control signals are important in the perfection of movements. It is discovered that the rhythmic activity of CPG could synchronize to a rhythmic sensory input [13].

Such autonomous, robust motion control is desirable for robotics and there are many studies trying to build an artificial CPG [14]. However, most of them studied CPG in more abstract level using mathematical oscillators [15, 16, 17]. It is intriguing how coupling between simple oscillators could produce complex motion patterns. In some cases, CPG based models were successfully mimicked the motion of real animals very well [18]. Even the balancing of a biped robot in unpredictable environment was achieved in a carefully adjusted CPG model [15]. The robustness of CPG based models comes from a stable limit cycle behavior produced by the mathematical oscillators. Coordinated motions became possible because of the phase coupling between those limit cycles and a gait transition was occurred when there is a bifurcation in the system state.

While locomotion based motion controls are robust and manageable, there is a complexity limitation on carefully hand-crafted models. Current models are often restricted to salamanders, snakes, insects, fishes, lower body of a biped robot. Therefore, learning is needed for building a large scale CPG model for human motions. However, there is no such learning model that could successfully mimic the dynamical behaviors of CPG.

2.3 Deep learning

In machine learning, deep learning refers to the learning of high level structures in training data rather than just learning of data itself, which is referred as swallow learning. To capture the deep structure in data, the learning model itself have to have a deep hierarchical structure consists from many layers.

An advantage of the hierarchical architecture is that lower layers learn useful simple features and higher layers combine those features to produce more complex features with little effort. Such compact representation is useful in many machine learning applications such as computer vision. Actually many deep vision models take inspiration from the visual processing in the brain. In the brain, visual processing starts with simple cells that have a small receptive field and respond to simple stimulations, like light points or edges. In the next step, more complex cells emerge that don't have a clear definite receptive field and respond to a very specific stimulation in a complicated way. Deep vision models [19, 20] successfully mimicked this hierarchical architecture with convolutional connections and max pooling.

Deep architectures can be built by stacking up existing learning models like neural network[19], HMM, etc. Recently, Hinton et al. [4] introduced a deep generative model called Deep Belief Net (DBN). DBN is constructed by staking up Restricted Boltzmann Machines which has an efficient unsupervised learning algorithm called Contrastive Divergence [4]. DBN can trained by training each RBM layer-by-layer bottom-up greedy way. Details about DBN can be found in Chapter 3.

DBN can be used in learning useful features for classification [20], nonlinear dimension reduction [21], auto-encoder, etc. In a classification problem like handwritten digit recognition, unsupervised DBN training of a neural network before fine-tuning by backpropagation produces a better classification result [21]. It is presumed that unsupervised learning brings the weights to sensible values and backpropagation only performs a local steepest descent search. Another advantage here is that unlabeled data can be used in unsupervised learning stage.

Building a deep auto-encoder with DBN is very straightforward. Actually, each RBM in DBN is itself an auto-encoder. RBM is trained to reconstruct visible unit values from hidden unit values which are inferred from the visible layer in the first place. The hidden layer will become a compact representation of the visible layer because the common features in data are encoded into the edge weights and variable features are encoded into a hidden layer. In a deep auto-encoder, data is first encoded to the hidden layer of the lowest RBM, then it is subsequently encoded by higher RBMs.

Chapter 3

Temporal Deep Belief Net

Temporal Deep Belief Net (TDBN) is a slightly modified version of Deep Belief Net to handle temporal patterns. In the following sections, building blocks of TDBN will be introduced.

3.1 Boltzmann Machine

Boltzmann Machine (BM) is a stochastic graphical model that has broad applications from machine learning to convex optimizations. BM is a graphical model and it consists from units and edges (see fig 3.1). Units take binary value of 0 or 1 depending on their total input from other nodes. Each edge has a fixed weight value, which is learned in the training process. Unlike most graphical models, edges in BM are undirected so that connected two units depend on each other.

Stochastic dynamics of Boltzmann Machine

BM is an energy model which means that there is a total energy defined and the network evolves towards a lower energy region in the same way as physical systems. However, this trend is a completely stochastic process where lower energy states have higher probability and higher energy states have little probability. This stochastic behavior of BM is defined by Boltzmann distribution.

$$P(\mathbf{v}) \propto e^{-E(\mathbf{v})}$$

Here we used vector $\mathbf{v} = \begin{bmatrix} s_1 & s_2 & \dots & s_n \end{bmatrix}^T$ to represent the state of the network, which includes values of all units in the network. This in-deterministic activity gives BM a unique exploratory

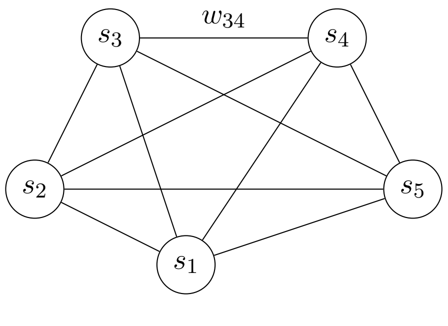


Figure 3.1: *Simple Boltzmann Machine*

behavior where the network has a chance of exploring a vast range of state space. Furthermore, this uncertainty can be controlled by introducing a temperature parameter to the above formula.

$$P(\mathbf{v}) \propto e^{-E(\mathbf{v})/T}$$

In most applications, the temperature is usually set to 1 in the start to widen the search space. Then, the temperature is gradually decreased to refine the search and get a more specific solution. This process is called Simulated Annealing (annealing is metallurgy process to strengthen a metal where the metal is melted in high temperature, and then gradually cooled).

The energy in BM is a function of the values of units s_i, the weights of edges w_{ij} and the biases of units b_i.

$$E(\mathbf{v}) = -\sum_{i<j} s_i s_j w_{ij} - \sum_i s_i b_i$$

The intention behind this equation is that the energy is lower when there are more couples of ON units connected by a positive weight. Each unit's value in the network is sampled to lower this energy. The probability distribution of this sampling process can be calculated from the energy.

$$P(s_i = 1) = \frac{1}{1 + e^{-\sum_{j \neq i} w_{ij} s_j + b_i}} = sigmoid(\sum_{j \neq i} w_{ij} s_j + b_i)$$

So whether the unit i being ON or OFF is only depends on its net input $\sum_{j \neq i} w_{ij} s_j + b_i$. For example, if the unit has positive input, then it's likely to be ON. If the input is exactly zero, then the unit has equal chance of being ON or OFF. To get the solution, this sampling process have to be continued until the network reaches to an equilibrium state.

Learning in Boltzmann Machine

Weights and biases have to be set to right values in order to BM work properly. Machine learning method can help when there is training data. Training data is a collection of states where we want the energy to be lower so that samples from the model would be close to them. In other words, our goal is to minimize the expected energy of states of training data while keeping the energy of the other states at relative high value. To put this in equation

$$minimize \rightarrow \langle E \rangle_{data} - \langle E \rangle_{model}$$

We can solve this problem by the gradient descent algorithm by calculating the negative gradient of the target function with respect to learning parameters: weights and biases.

$$-\frac{\delta(\langle E \rangle_{data} - \langle E \rangle_{model})}{\delta w_{ij}} = \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model}$$

$$-\frac{\delta(\langle E \rangle_{data} - \langle E \rangle_{model})}{\delta b_i} = \langle s_i \rangle_{data} - \langle s_i \rangle_{model}$$

So in the learning step, weights and biases will be updated in the following way.

$$\Delta w_{ij} = \alpha \cdot (\langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model})$$

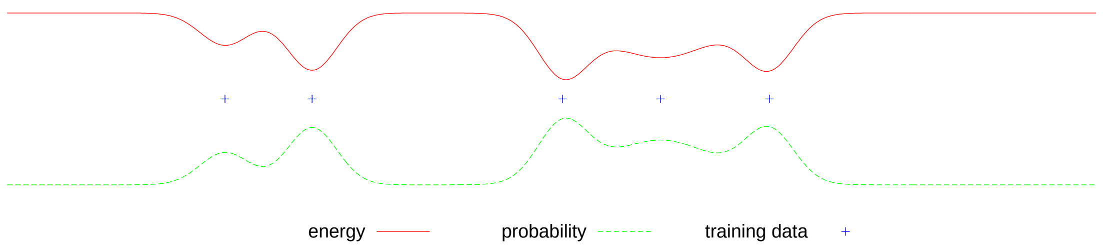


Figure 3.2: *A simple energy surface in BM. As the result of the training, the energy will decrease at the points corresponding to the training data. In contrast, probability at neighbor of those points will increase.*

$$\Delta b_i = \alpha \cdot (\langle s_i \rangle_{data} - \langle s_i \rangle_{model})$$

Here, α is the learning rate which is usually set to a very small value. Expected values can be approximated by the average of few random samples without degrading the learning speed. Usually, training data is divided into small batches of 10 to 100 samples and update is done for each batch to speed up the learning.

After successful training, the energy surface of BM will have steep basins at training data (see fig 3.2). At the same time, the probability of those states will increase, ensuring that sampled states would be close to the training data.

True random sampling in Boltzmann Machine is almost impossible because sampling of each unit depends on thr current values of all other units. Thus, the next sampled state will always be biased by the previous state in some extent. Approximate unbiased sampling is possible if one keeps sampling until the correlation of the current sampling with the previous sampling becomes ignorable.

Conditional Boltzmann Machine

If there are clamped units whose values are fixed to given values, then the other units in Boltzmann Machine will become conditioned on them. Fortunately, clamped units will not interfere with learning because the input from the clamped units can be thought as a bias. Units holding known parameters or control variables are usually clamped during sampling to get a conditioned sample from the network. Connections from clamped units are drawn as directed edges to represent one-way dependency between nodes (see fig 3.3).

Real valued Gaussian unit

If a unit is allowed to take real value in Boltzmann Machine, the sampled value will be infinity. To fix this issue, unit's priori can be set to Gaussian Distribution by an additional term in the energy function.

$$E(\mathbf{v}) = \sum_i \frac{s_i^2}{2} - \sum_{i<j} s_i s_j w_{ij} - \sum_i s_i b_i$$

The sampling distribution of a Gaussian unit is a unit variance Gaussian distribution with a mean equal to the total input to the unit. But, there are two shortcomings in using a Gaussian unit as

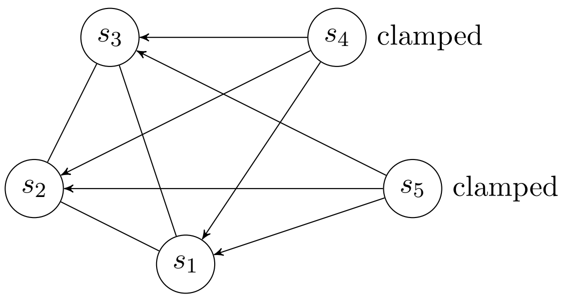


Figure 3.3: *Conditional Boltzmann Machine*

a real valued unit. First, it cannot correctly represent a variable with different distribution. For example, a joint angle in motion data has highest probability density at its both ends, which is very different from the Gaussian distributions. Second, learning is unstable in a network with Gaussian units.

3.2 Restricted Boltzmann Machine

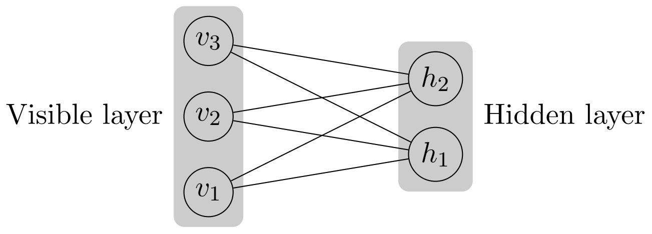


Figure 3.4: *Restricted Boltzmann Machine with three visible units and two hidden units. Edges are restricted only to visible-hidden.*

In Restricted Boltzmann Machine [22] units are divided into two layers: visible layer and hidden layer (see fig 3.4). Visible units represent directly observable variables and hidden units represent the hidden state of those variables. Connections in RBM is restricted that two nodes in a same layer cannot be connected. In this way, visible units are conditionally independent of one other given the hidden state, and vice versa. So it is possible to get an unbiased sampling of $\langle v_i h_j \rangle_{data}$ with a single sampling step. Here, v_i is the i-th visible unit, h_j is the j-th hidden unit. However, sampling of $\langle s_i s_j \rangle_{model}$ is still requires infinitely many sampling steps, but there is an approximate learning algorithm called Contrastive Divergence [23] where $\langle v_i h_j \rangle_{model}$ is approximated by $\langle v_i h_j \rangle_{recon}$ which can be easily sampled. The exact procedure of the learning algorithm is following:

1. Set training data to visible units
2. Sample hidden units conditioned on the visible layer to get $\langle v_i h_j \rangle_{data}$
3. Sample visible units conditioned on the hidden layer to get reconstruction of the training data
4. Sample again hidden units conditioned on the reconstructed visible layer to get $\langle v_i h_j \rangle_{recon}$
5. Update weights and biases by

$$\Delta w_{ij} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}$$

$$\Delta b_{v_i} = \langle v_i \rangle_{data} - \langle v_i \rangle_{recon} \quad \Delta b_{h_j} = \langle h_j \rangle_{data} - \langle h_j \rangle_{recon}$$

This approximate learning of gradient descent works very well and very fast. As explained before, learning in Boltzmann Machine has two effects on its energy surface: (1) the energy lowers at the states included in training data which corresponds to the term $\langle v_i h_j \rangle_{data}$, and (2) the energy increases at sampled states from the model which corresponds to the term $\langle v_i h_j \rangle_{model}$. Since a state with low energy has high sampling probability, the second effect actually flattens the energy surface by increasing the energy at low energy states. In Contrastive Divergence however, the second effect only flattens the states close to the training data because of single step sampling. But this works because this energy wall around the training data keeps model in a desired region.

After successful training, there must be a hidden state for each training data in which the total energy is at its lowest. We can find such hidden state by simply performing a deterministic sampling from the training data. Reversely, we also can make an approximate reconstruction of the training data from the hidden state. This correspondence between the hidden state and the training data can be used as auto-encoder where information in the training data is compressed into a compact hidden state.

Conditional Restricted Boltzmann Machine for temporal patterns

In temporal patterns, the next state depends on previous states. This temporal correlation can be modeled by Conditional Restricted Boltzmann Machine (CRBM) where RBM is conditioned on previous visible states. In figure 3.5, the current visible state is conditioned on the past three visible states, but the hidden state depend on only the current visible state. Since past values are currently known, those past units are clamped during sampling and we can think them as dynamic bias for the visible units.

To train this CRBM, we need a sequence of visible states as training data. The sampling of the hidden layer is no different from a normal RBM. However, the past units have to be fixed to previous visible states during the sampling of the visible state.

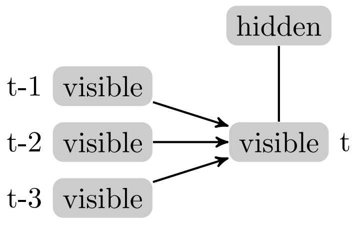


Figure 3.5: *Conditional Restricted Boltzmann Machine for temporal patterns. The difference from a normal RBM is that the visible layer is conditioned on its past values. In this particular case, past three steps are connected to the visible layer.*

3.3 Deep Belief Net

The representative power of RBM is very limited because of restricted connections. Fortunately, by stacking up RBMs, we can easily build a hierarchical structure called Deep Belief Net with rich representative power (see fig 3.6). In DBN, the hidden layer of the bottom RBM becomes the

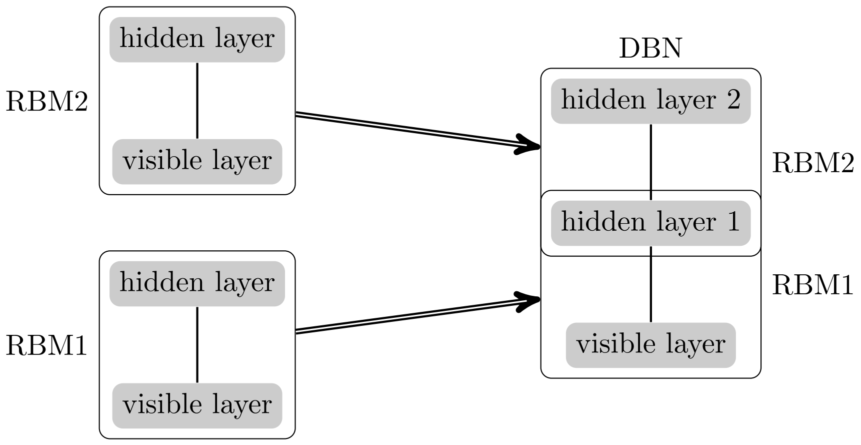


Figure 3.6: *Construction of the Deep Belief Net by stacking two RBMs up. The hidden layer of RBM1 and the visible layer of RBM2 merges into single layer. Resulting DBN will have single visible layer and two hidden layers.*

visible layer of the next RBM. There is no limit on how many RBMs could be in a DBN. But usually, the number of layers is kept under four.

DBN is attractive because its training is no more difficult than RBM despite of a complex structure. Training of DBN is simply greedy layer-by-layer training of the component RBMs. Let's take the DBN in figure 3.6 as example. First, we have to train RBM1 by training data. After that, we collect hidden states corresponding to each training data in RBM1. That collection is then used as training data for RBM2. In other words, RBM1 learns patterns in the training data and RBM2 learns patterns in the hidden state of RBM1.

Another explanation is that RBM1 learns many simple features of the training data. Then, RBM2 learns to combine those features to construct more complex features. Such hierarchical feature recognition is very useful in image processing and similar feature recognition was observed in human visual processing [24]. By using max-pooling and convolutional connections in DBN, very powerful image recognition system can be built [20]. Since each RBM in DBN encodes its visible state into its hidden state, DBN also can be used as a deep auto-encoder where training data is encoded in several steps to more and more compact representations.

Chapter 4

Outline of the proposed model

We want our model to be able to learn and generate motion data. There must be temporal elements in the model to learn human motions. Also, the units in the lowest layer have to be Gaussian units because motion data is given as sequence of 56 real valued joint angles. Finally, we have to embed control parameters in the highest layer of the model. To meet those requirements, we have chosen a Temporal Deep Belief Net (TDBN) architecture consisting from CRBM and RBM.

Actually, single CRBM can suffice all those conditions in theory. However, the quality of generated motions in TDBN is much better than a CRBM. Besides, the learning in CRBM is very instable if visible units are Gaussian units because weights between Gaussian units tend to become very large during the learning.

4.1 Deep architecture of the model

The proposed model is built by merging the hidden layer of a RBM with the visible layer of a CRBM (see fig 4.1). Therefore, in total it has three layers: the visible layer, the hidden layer and the 2nd hidden layer. Since the 1st hidden layer was originally the visible layer of CRBM, it has directed connections from its past states. In the model, the current hidden state is conditioned on past three hidden states. Joint angles will be represented by the visible layer in the model and control parameters will be included in the 2nd hidden layer. Let's see each layer in more detail.

Layers in the RBM

The lowest layer in the model is the visible layer. Joint angles in motion data will be represented by values of the visible units. Since motion capture data consists from 56 joint angles in a skeleton, the visible layer will have 56 Gaussian units.

The next layer in the model is the hidden layer. This hidden layer and the visible layer will constitute a RBM (see fig 4.1). We sparsely connected this RBM using the human body structure. Joint angles are divided into five groups based on rough correlation: joint angles of left leg, right leg, left arm, right arm and back. The visible and hidden layers are also divided into five parts and each part corresponds to a single group of joint angles (see fig 4.2). The sparse connection in the RBM greatly reduces the number of parameters without degrading learning capacity. We do not have to worry about the synchronization between those separate parts because they are connected through the CRBM. Each part has 30 hidden units so the entire hidden layer has 150 binary units.

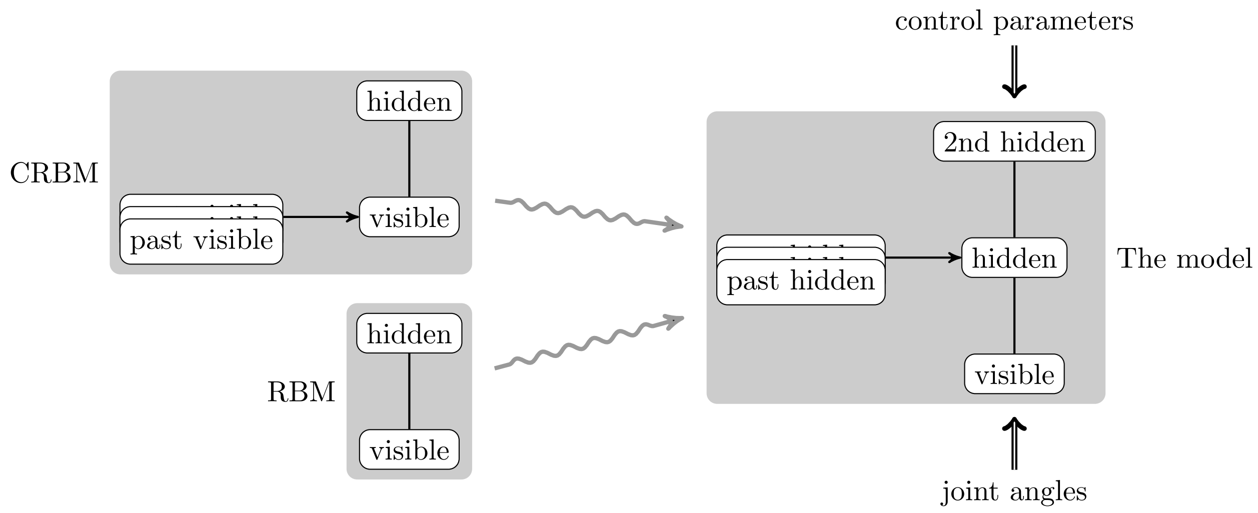


Figure 4.1: *The model is built by merging CRBM with RBM to make temporal Deep Belief Net. Only the past values of the 1st hidden layer are kept as history in the model. Joint angles are represented by the visible units and control parameters are included in the 2nd hidden layer as clamped unit.*

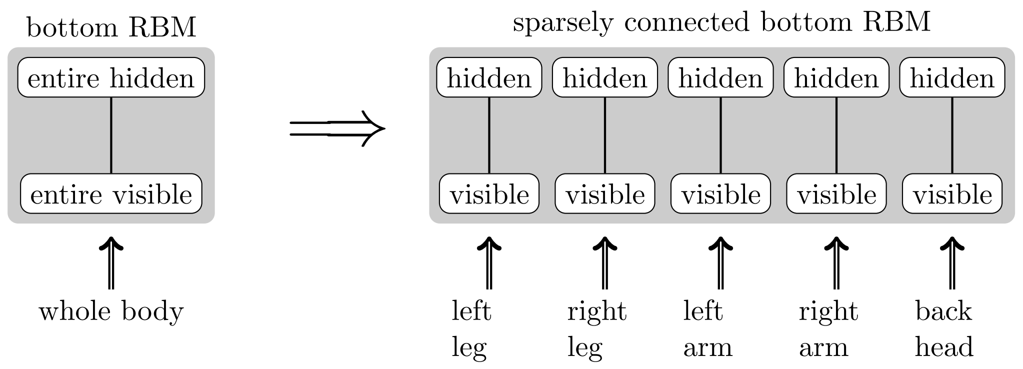


Figure 4.2: *Sparsely connected version of the bottom RBM. The visible and hidden layer are divided into five parts and those parts are separately connected.*

During the training of this RBM, randomly selected single frame from the motion data will be set to the visible layer. There is no temporal element in this RBM, so the RBM learns static human postures rather than a dynamic motion. After trained by the training data, the RBM must have a capability to encode each posture in the motion data to a hidden state. Those hidden states will be used in the training of the CRBM. During motion generation, the role of the RBM is to decode a hidden state back to joint angles.

Layers in the CRBM

The hidden layer and the 2nd hidden layer constitutes a CRBM conditioned on three past hidden states (see fig 4.1). Those three past hidden layers have 450 units in total and there are $150 \times 450 = 67500$ directed edges between them and the hidden layer. This large number of edges are dedicated to the learning of the temporal constraints in the pattern of the hidden layer's activity. To train the CRBM, continues four frames from the motion data have to be encoded into hidden states by the RBM and set to the hidden layer and its past layers. During sampling, the input from the past states to the hidden units can be thought as dynamic bias which changes with time. The rest of the training is the same as a normal RBM. In a motion generation process, a sequence of hidden states will be generated by the CRBM. This can be done by sampling of the current hidden state conditioned on past hidden states and shifting the sampled hidden state to the past hidden layers to get the sampling of the next hidden state and so on. Finally, those hidden states will be decoded into joint angles by the RBM to get motion data.

The highest layer in the model is the 2nd hidden layer. It has small number of units. Actually, the model still can work even if the 2nd hidden layer had zero units (Interestingly, the learning process of CRBM without the hidden layer is very similar to the backpropagation algorithm). But units in this layer will be used in different purposes. For example, in the gait transition experiment, those units will become control units holding control parameters for gait types. Since, gait type is set by a user, those control units will be always clamped except for the training step where a hidden layer is sampled from a reconstructed visible layer.

Chapter 5

Experiments & Results

In the following experiments, we train the proposed model using human motion capture data and generate various motions from the model. Expected values are used instead of sampled binary values during motion generation to reduce the noise in motion.

5.1 Training data description

Human Motion Capture Data (HMCD) is used as training data in all experiments. HMCD used in this thesis is acquired from the CMU motion capture database [25]. Each motion data was consisted from two files: (1) Acclaim Skeleton File for basic information about a skeleton and coordinate systems, and (2) Acclaim Motion Capture data for actual motion information. Each frame in data is composed from 56 real-valued angles for the joints in the skeleton and three body position coordinates.

Throughout all experiments we used only three motions as training data: walking, running and bending. The walking motion was three cycles long and the running motion was only 1.5 cycles long. The bending was a short motion of a standing person bending over and picking something with an one hand and standing up. There was no motion of transition between those motions in the training data.

In preprocessing, the frame rate of the HMCD is reduced from 120fps down to 40fps, which is enough for our purpose. Also, position information is discarded from the training data because it is not part of the motion. Therefore, motions generated from the model are consist only from joint angles at frame rate of 40fps without position information. Finally, each variable in the data is normalized to have zero mean and unit variance so it can be correctly expressed by Gaussian units in the model.

Data visualization

There are two ways to visualize an output motion: (1) format an output as motion data file and play it as an animation or (2) project an output motion into two or three dimensional space by Principal Component Analysis, so it can be plotted as a trajectory. We wrote a program that can visualize an output from the model as animation. Animations help to subjectively judge the naturalness of generated motions because people are very sensitive to unnatural motions. To analyze an output more objectively, we used Principal Component Analysis to reduce the output dimension so that

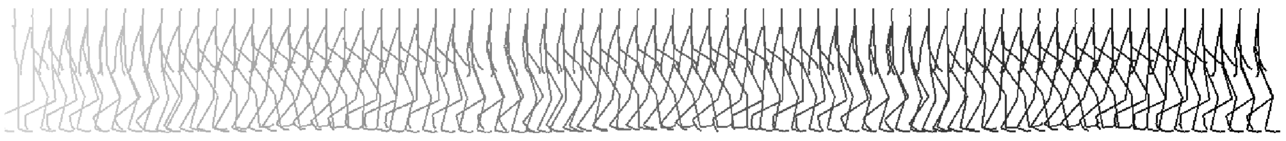


Figure 5.1: *The walking motion generated from the model at 40fps. Each motion frames is placed with constant distance from its previous frame to show the change.*

motion can be plotted as a continues trajectory in space. In this way, generated motions can be easily compared to its original motion.

5.2 Stable locomotion with limit cycle

First, we will train the model by a short walking motion to examine its stability. Human motion capture data of short walking motion (3 cycles long) is used as training data. We have to train the RBM before training the CRBM. Each motion frame in the training data was set to the visible layer of the RBM in random order to make learning unbiased. The training was stopped after 1000 epochs.

Because the RBM now can encode each motion frame into a hidden state, the training motion can be thought as a sequence of hidden states. This sequence then will be used as training data in the CRBM training. The hidden layer will be set to a hidden state randomly selected from the sequence. At the same time, the past hidden layers will be set to the previous hidden states from the sequence. This will make sure that the CRBM learns the correct motion order. The training of the CRBM was continued 100 epochs. The total training time was about three minutes on a notebook with core-2-duo processor and 2GB memory.

To start motion generation in the model, the past hidden layers have to be set from the training sequence. Once it is done, the value of the hidden layer can be generated by Gibbs sampling in the CRBM conditioned on those past hidden states. To eliminate unnecessary noise, Gibbs sampling is done deterministically by using an expected value as unit's value rather than using a sampled binary value. After sampling done, the hidden state will be shifted to the past hidden layers to get ready for the sampling of the next hidden state. The CRBM will generate a sequence of hidden states by repeating this procedure. Then, this sequence will be decoded into a motion by the RBM. A walking motion generated in this way is presented in figure 5.1.

Limit cycle behavior

Let's investigate the model as a dynamic system. Repetitive motions like walking can be considered as a limit cycle in the state space and the system state continuously circles this limit cycle during motion generation. The purpose of setting the past hidden layers to sampled values from the training data in the start of motion generation was to make sure that the system's initial state is on the walking limit cycle. But if the model succeeded in producing a stable walking limit cycle, any motion should eventually converge to the walking motion even if the initial state was outside the limit cycle. To test this hypothesis, we generated 100 independent short walking motions from the model. But this time, we added a large Gaussian noise to the past hidden states in the start of each motion to simulate a perturbation. The result is shown in figure 5.2 where each motion is

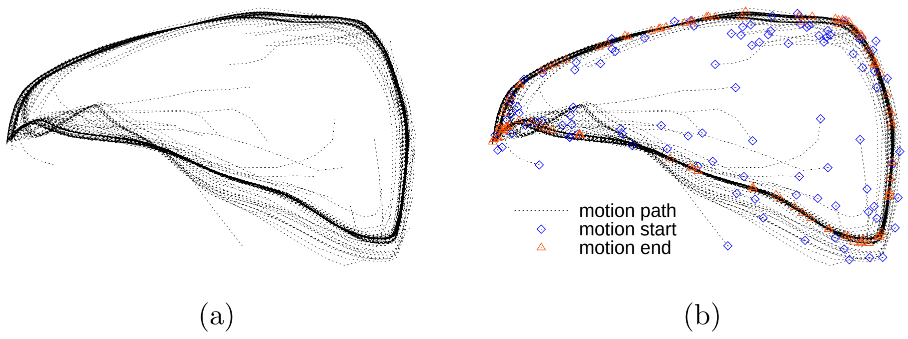


Figure 5.2: *(a) Limit cycle in walking motions. Motions which are started from random hidden states converge into a single limit cycle. One second long 100 motions were generated and projected into 2 dimensional space by PCA. (b) shows where the motions are started and where they ended.*

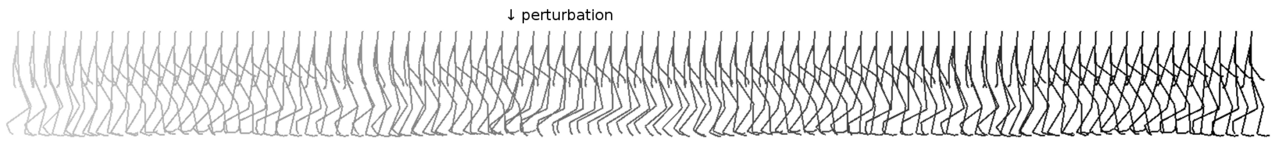


Figure 5.3: *A motion naturally returns to a normal walking motion after a perturbation. The perturbation is simulated by forcibly shifting the phase of the left leg by a half cycle.*

represented by a dotted trajectory. We used PCA to reduce the motion data dimension from 56 to 2. Initial and final states of the each motion is marked in (b). We can see that the motions started from different perturbed hidden states are all ended on a single dark loop, which is the normal walking motion.

We have to note that the motions are started from random hidden states, which is different random visible state. There are visible states that cannot be produced from any hidden state. That's why random initial hidden states (blue dots) in figure 5.2-b are not looking completely random. So we can't say that a motion started from any pose would smoothly converge to the walking motion (but we can say that any motion will converge to the walking motion possibly with a big discontinues leap).

Synchronization of limbs

We conducted another experiment to show the limit cycle behavior in the model with a more practical example. We simulated a perturbation to a walking motion by forcibly shifting the phase of a left leg. This is done by setting the hidden units corresponding to the left leg to values from a previously generated normal walking motion with a slight delay. The other hidden units are sampled with Gibbs sampling as the same way as the previous experiments. In other words, the motion of only the left leg is replayed from a previously recorded walking motion. The left leg motion will have a big discontinuity when we start setting it's hidden state from a different motion. This leap can be clearly seen from a generated motion (see fig 5.3). In figure 5.4, femur joint angles from both legs are plotted. A red line representing the left leg has shifted its phase at frame 100 due to the forced perturbation. At that time, a blue line representing the right leg has lost its

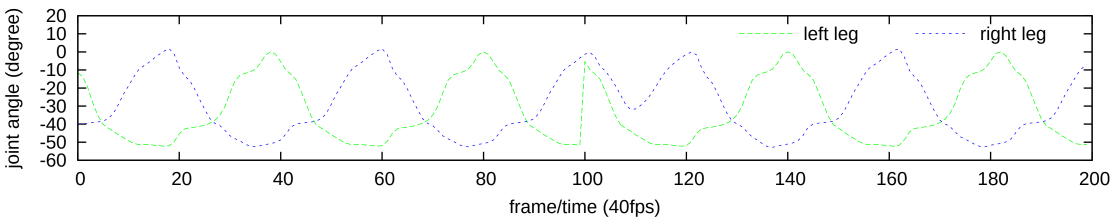


Figure 5.4: *Femur joint angles from each leg is plotted. When the left leg is perturbed at frame 100, the right leg motion is smoothly synchronizing with the left leg motion.*

synchrony with the left leg. However, those two motions are deeply correlated by their past states and such out-of-sync motion must have relatively high energy. Therefore, the motion of the right leg will eventually synchronize with the left leg and the motion will return to the walking limit cycle.

5.3 Gait transition by bifurcation

In this experiment, we will try to train the model on two different gait styles: walking and running. The goal of this experiment is two-fold: (1) to show that it is possible to learn and generate two different motions by a single model, and (2) to demonstrate that the model can generate a natural gait transition motion. An advantage of having a single model for all motions is that a transition between gaits will be much easier since we do not have to switch models during motion generation.

We used the whole 2nd hidden layer as a control layer to control which gait should be generated by the model. Each unit in this layer will have a same fixed value: 0 for walking and 1 for running. We will call this value as a gait parameter. The control layer (or the 2nd hidden layer) will always be clamped except for the sampling from a reconstruction during training. Two separate motion capture data will be used as training data, but there is no gait transition motion in neither of them.

The training of the RBM is the same as the previous experiment. But walking and running frames have to be learned in random order to prevent a bias to one gait. The training of the CRBM is slightly easier than before because we can calculate $\langle v_i h_j \rangle_{data}$ directly because h_j is fixed to the gait parameter. During a reconstruction step however, we have to unclamp the control layer to calculate correctly $\langle v_i h_j \rangle_{recon}$.

After the training, we successfully generated separate walking and running motions by setting the gait parameter to 0 and 1. However, when we changed this gait parameter during a motion generation, the model generated a gait transition motion. In figure 5.5, we showed one such example where the gait parameter was changed to 1 during a walking motion. From that moment, the motion smoothly transferred from walking to running in less than half second.

We observed a smooth gait transition even if the control parameter changes abruptly. This smoothness can be explained by the possibility that directed connections between the hidden layer and its past states are learned the continuity constraint of the human motion from training data. As a result, even though top-down control changes abruptly, the lower level will try to generate a pattern that meets the continuity constraint as well as the top-down control constraint.

The number of units in the 2nd hidden layer was set to 20 in this experiment. More units will boost the effect of the gait parameter on the hidden layer, making transition more sudden and

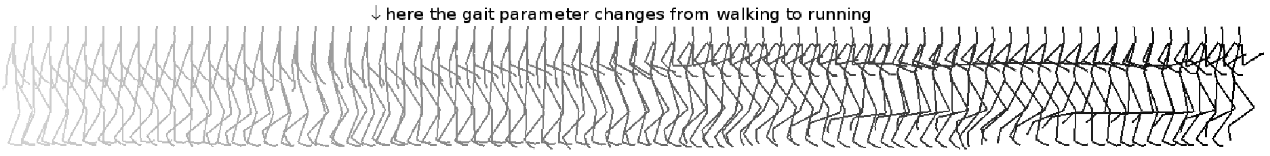


Figure 5.5: *A gait transition motion generated from the model. The gait control parameter is changed from walking to running during motion generation. The result is a natural smooth transition motion despite of abrupt control change. Complete transition will take no more than half second (which is equal to 20 frames in the figure).*

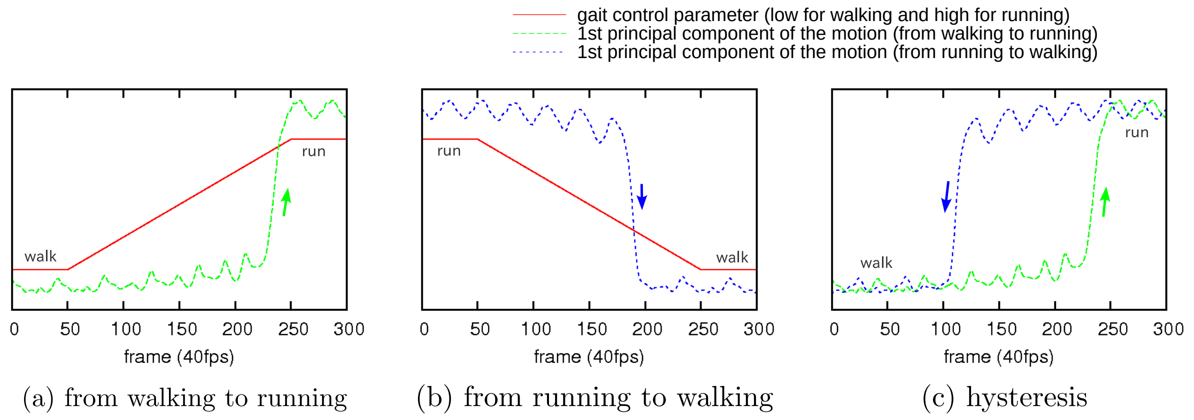


Figure 5.6: *In (a),(b), gait transition motions are plotted with the gait parameter value. Motions are transformed into one dimension by PCA. In (c), two transitions in (a),(b) are compared at the same gait parameter value which shows clear hysteresis.*

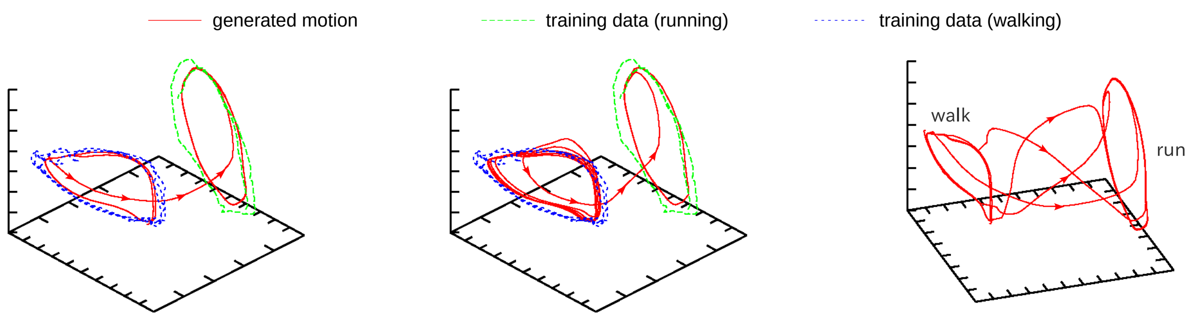
short. On the contrary, less units will boost the effect of the past hidden states on the hidden layer, making transition slower and even impossible in some cases.

Emergence of hysteresis

We did another experiment where the gait parameter was gradually changed from walking to running in five seconds (fig 5.6-a). Also, a same experiment is done in reverse direction: from running to walking (fig 5.6-b). One would expect the gait transitions to occur at the same gait parameter threshold. However, if we overlap the two transitions to have same gait parameter value in figure 5.6-c, we can see that the threshold for a gait transition is dependent on a direction. Such a direction dependency of a dynamic system is called a hysteresis.

Humans exhibit a similar hysteresis during a gait transition [26]. In [26], this hysteresis is explained in dynamic theory where walking and running is viewed as a limit cycle (LC). When the locomotion speed is slow, a walking LC is stable and a running LC is unstable. As the speed increases, the running LC also becomes stable. However, since the walking LC is still stable, the system will not leave the walking LC. Further increase in the speed will make the walking LC unstable and the system will transfer to the stable running LC. Such a quality change in a system is called bifurcation. So a gait transition can be thought as a bifurcation.

Same explanation can be applied to our model. In the walking experiment, we showed that



(a) abrupt control change (b) slow continuous control change (c) variety in bifurcation

Figure 5.7: *Bifurcation in state space when the gait changes from walking to running. There are two distinct limit cycles for walking and running. (a) As soon as the control parameter changes, the walking limit cycle (LC) becomes unstable and running LC becomes stable. The system state will travel from the walking LC to the running LC without a big leap. (b) Even the control parameter changes slowly, the bifurcation occurs in the same way. (c) There is no definite path between two LCs. The system travels different paths in each gait transition.*

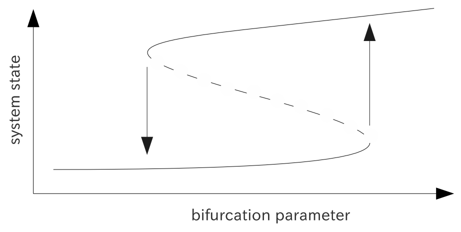


Figure 5.8: *Saddle node bifurcation diagram*

there is a walking limit cycle in the model. But in this experiment, we trained the model on two different motions, so there must be two separate limit cycles. A gait transition happened when we changed the gait parameter. Therefore, the gait parameter is controlling the stability of those limit cycles. In figure 5.7-a, we showed a bifurcation from a walking LC to a running LC when the gait parameter changed suddenly. In figure 5.7-b however, the gait parameter changed slowly. Interestingly, the system state kept circling the walking LC until the gait parameter almost reaches to the running value. Then suddenly, the system state leaves the walking LC and transfers to the running LC without losing continuity (there are actually 5-10 frames in the transition path). Such a bifurcation between two limit cycles is called a saddle node bifurcation. In figure 5.8, we showed the bifurcation diagram of the saddle node bifurcation where the system state shifts between two stable attractors depending on the bifurcation parameter. But there is a region where both attractors are stable and this produces a hysteresis in the system.

Figure 5.7-c demonstrates one of the advantages of our model: a stochastic behavior. Even though four motions of a gait transition are generated in a same condition, each motion path differs from one other because of the inherent stochastic behavior of Boltzmann Machine. Such variety in same motion is important in some applications such as computer games where same repeated motions give an artificial look to characters.

5.4 Nonrhythmic motion and interruption

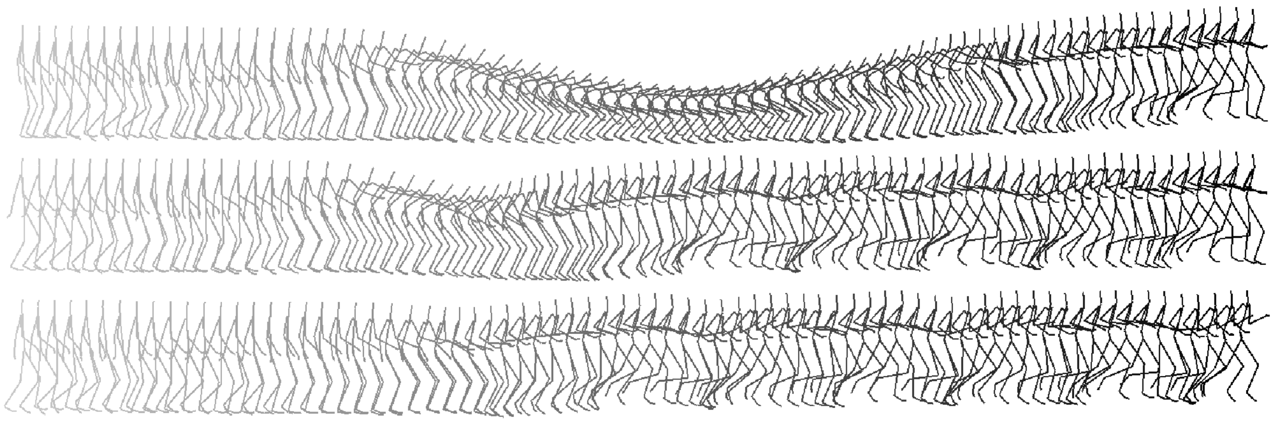


Figure 5.9: *Continues generation of three motions: walking, bending over to pick up something and running. In the top motion, running motion is started after the completion of the bending motion. In the middle motion, the bending motion is interrupted by the running motion. In the bottom figure, the interruption occurs in more early stage.*

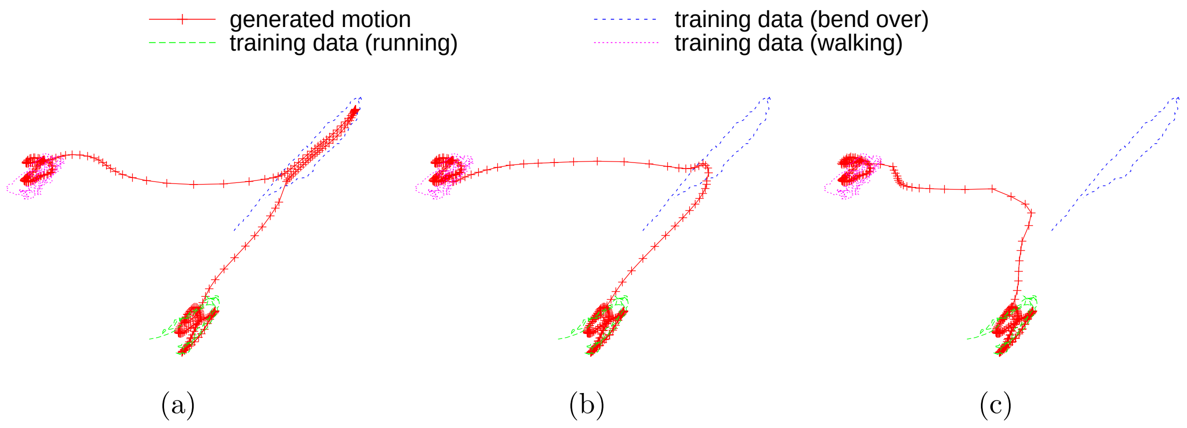


Figure 5.10: *Two dimensional representations of the motions in figure 5.9 and original training motions. The motion starts from the walking region (green) and smoothly transfer to the bending region (blue) and ends at the running region (pink). In (b) and (c) however, the bending motion is interrupted before its completion.*

Motions can be learned by the model is not restricted to rhythmic motions. However, motions with too many still frames are difficult to learn because the model predicts the next frame based on only past three frames. In this experiment, we added a new motion of “bend over, scoop up and rise” to the training data. Since there are three motions to learn, we will divide the control layer into three equal parts each corresponding to one motion. To generate a specific motion, all we have to do is to set the units in corresponding part to 1 and the other units in the control layer to 0. Learning algorithm is the same as previous experiments.

In figure 5.9, the model generated a bending motion between walking and running. This is done by changing the control units from walking to bending, then from bending to running. We generated three versions of this motion to show the flexibility of the model. In the top figure, the bending motion is completed before transforming into the running. This can be seen from the motion trajectory in figure 5.10-a where each training motion is plotted. However, if we set the control layer to running before the bending motion completion, the resulting motion will look like the middle figure where the bending is interrupted and naturally transferred into running motion. If we see the corresponding motion trajectory in figure 5.10-b, we understand that the motion is actually reached the bending region before turning to the running region without executing the bending motion. If we change the control to running earlier, the motion trajectory will change the direction before reaching the bending region (see fig 5.10-c). A corresponding motion is shown in the bottom of figure 5.9 where the bending motion is barely observable. Such flexibility is convenient in on-line applications such as computer games, robot control, etc. The transition shown in figure 5.10-c cannot produced by traditional methods where each transition between two motions is synthesized in advance.

Unlike the walking and running motions, the bending motion has fixed point attractor rather than a limit cycle because it has ending. The result from this experiment shows that a bifurcation between a limit cycle and a fixed point attractor is possible in the model.

Chapter 6

Conclusion

In this thesis, we introduced a stochastic hierarchical model capable of learning and generating human motions. Human motion capture data was used as training data. The results of the experiments show that our model not only replays the trained motions, but it also can smoothly switch between those motions. It is impossible to generate a natural motion of gait transition without being aware of the similarities of walking and running motion. Therefore, it can be said that the model learned the common features in the two motion patterns. Also, a transition between motions are generated by a bifurcation in the model, rather than an interpolation where each transition is prepared in advance. The bifurcation is triggered by a control parameter and it is more flexible and manageable. For example, a transition from A motion to B motion can be canceled in the middle and change the direction to C motion in natural way. Further, each gait motion is generated by a limit cycle in the hidden space. This gives robustness to the model that a perturbed motion smoothly converges to a normal motion. Those dynamical behaviors are vital to the motion generation of both humans and robots, and it is novel to produce them on a neural network trained on real human motions.

Even though Conditional RBM, which is a single layer DBN had used previously in modeling of the human motion [2], there was no concrete example of using multilayer DBN for temporal pattern. We presented one possible use of DBN in temporal learning in the similar way as HMM: visible states are encoded to hidden states and temporal constraints are learned between those hidden states. This approach is more efficient than learning temporal constraints between visible states because hidden states are more compact than visible states.

The proposed model can be useful many applications from a motion generation of game characters to a humanoid robot control. Moreover, other temporal patterns such as speech and music score can be generated by the model. It would be interesting to model a human voice because spoken words have clear hierarchical structure suitable for the Deep Belief Net. Our model is only one way of utilizing DBN in temporal pattern learning and there are many ways to construct deep architecture with arbitrary many layers.

Future work

We showed the existence dynamical behaviors in our model. It is intriguing that a neural network simply trained on rhythmic continues patterns is capable of producing limit cycles and bifurcations. However, it is not clear the mechanism behind this and mathematical analysis is necessary to get full understanding.

Acknowledgements

I am grateful to Professor Takashi Chikayama and Associate Professor Kenjiro Tauro for their valuable advises and helpful comments on earlier drafts of this thesis. I also thank to the all member of Chikayama-Taura laboratory for their insightful discussions.

Bibliography

[1] J. Yang, Y. Xu, and CS Chen. Human action learning via hidden Markov model. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 27(1):34–44, 2002.

[2] G.W. Taylor, G.E. Hinton, and S.T. Roweis. Modeling human motion using binary latent variables. *Advances in neural information processing systems*, 19:1345, 2007.

[3] G.W. Taylor and G.E. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1025–1032. ACM, New York, NY, USA, 2009.

[4] G.E. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[5] S. Kajita, M. Morisawa, K. Miura, S. Nakaoka, K. Harada, K. Kaneko, F. Kanehiro, and K. Yokoi. Biped walking stabilization based on linear inverted pendulum tracking. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4489–4496. IEEE, 2010.

[6] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *ACM SIGGRAPH 2008 classes*, pages 1–10. ACM, 2008.

[7] L. Zhao and A. Safonova. Achieving good connectivity in motion graphs. *Graphical Models*, 71(4):139–152, 2009.

[8] D. Kulić, W. Takano, and Y. Nakamura. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains. *The International Journal of Robotics Research*, 27(7):761, 2008.

[9] W. Takano and Y. Nakamura. Humanoid robots autonomous acquisition of proto-symbols through motion segmentation. In *IEEE-RAS International Conference on Humanoid Robots*, pages 425–431, 2006.

[10] M. Brand and A. Hertzmann. Style machines. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192. ACM Press/Addison-Wesley Publishing Co., 2000.

[11] S. Grillner. Neurobiological bases of rhythmic motor acts in vertebrates. *Science*, 228(4696):143, 1985.

[12] M.L. Severin F.V. Orlovsky G.N. Shik. Control of walking by means of electrical stimulation of the mid-brain. *Biophysics*, 11:756–765, 1966.

[13] S. Grillner and P. Wallén. On peripheral control mechanisms acting on the central pattern generators for swimming in the dogfish. *The Journal of experimental biology*, 98:1, 1982.

[14] A.J. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653, 2008.

[15] G. Taga, Y. Yamaguchi, and H. Shimizu. Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, 65(3):147–159, 1991.

[16] JJ Collins and IN Stewart. Symmetry-breaking bifurcation: a possible mechanism for 2: 1 frequency-locking in animal locomotion. *Journal of mathematical biology*, 30(8):827–838, 1992.

[17] G. Schöner, WY Jiang, and JAS Kelso. A synergetic theory of quadrupedal gaits and gait transitions. *Journal of theoretical Biology*, 142(3):359–391, 1990.

[18] A.J. Ijspeert, A. Crespi, D. Ryczko, and J.M. Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416, 2007.

[19] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[20] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[21] G.E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504, 2006.

[22] G.E. Hinton. Boltzmann machine. `http://www.scholarpedia.org/article/Boltzmann_machine`, 2007.

[23] Guy Mayraz and G.E. Hinton. Recognizing handwritten digits using hierarchical products of experts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(2):189–197, 2002.

[24] D.H. Hubel, J. Wensveen, and B. Wick. *Eye, brain, and vision.* Scientific American Library New York, 1988.

[25] Carnegie Mellon Univ. Motion capture dataset. `http://mocap.cs.cmu.edu`.

[26] F.J. Diedrich and W.H. Warren Jr. Why change gaits? Dynamics of the walk run transition. *Journal of Experimental Psychology*, 21(1):183–202, 1995.