Learning Nonlinear Constraints with Contrastive Backpropagation

Andriy Mnih and Geoffrey Hinton
Department of Computer Science
University of Toronto

***Abstract*— Certain datasets can be efficiently modelled in terms of constraints that are usually satisfied but sometimes are strongly violated. We propose using energy-based density models (EBMs) implementing products of frequently approximately satisfied nonlinear constraints for modelling such datasets.**

We demonstrate the feasibility of this approach by training an EBM using contrastive backpropagation on a dataset of idealized trajectories of two balls bouncing in a box and showing that the model learns an accurate and efficient representation of the dataset, taking advantage of the approximate independence between subsets of variables.

I. Introduction

Some datasets are well characterized by representing the ways in which the data can vary. For such datasets, models that use hidden causes are often appropriate [Pearl, 1988]. Other datasets are better characterized by representing constraints that correspond to ways in which the data does not vary [Williams and Agakov, 2002]. Many datasets contain both types of structure [Welling et al., 2004]. A particularly challenging type of data for density modeling contains multiple constraints that are usually satisfied fairly accurately but occasionally violated by a large amount. In this paper we describe a type of density model that can deal with such "frequently approximately satisfied" (FAS) constraints [Hinton and Teh, 2001] and apply it to the task of modeling the trajectories of two balls bouncing inside a box. When a ball is in free flight, there are simple constraints on its positions at three adjacent times and these constraints are satisfied very accurately. When a ball bounces off a wall, the free flight constraints are wildly violated, but other constraints come into play, and these other constraints are relatively easy to model if the walls remain in fixed locations. When the two balls collide with each other, there are still constraints on their joint momentum and kinetic energy but these constraints are harder to discover in the sequence of x and y coordinates of the two balls.

A good density model of the sequences of x and y coordinates needs to capture the independent tight constraints that apply in free flight despite their violation during bounces and collisions. It also needs to capture the different independent constraints that apply during bounces and the strong interaction that occurs during collisions. There is no efficient way to do this with commonly used density models like Gaussian mixture models. If all the constraints ceased to apply at the same time it would be easy to split the dataset into two subsets that could be modeled separately, but this is not what happens. We need a method that can handle large sets of FAS constraints that are often independent but occasionally coupled. We propose an energy-based model that implements a product of nonlinear constraints for modelling this type of data.

II. Learning in energy-based models

A deterministic energy-based model (EBM) defines a probability distribution over data vectors by specifying a parametric energy function $E(\mathbf{x}|\mathbf{w})$ and defining the probability of a data vector as

$$p(\mathbf{x}|\mathbf{w}) = \frac{e^{-E(\mathbf{x}|\mathbf{w})}}{\int e^{-E(\mathbf{y}|\mathbf{w})} d\mathbf{y}}, \tag{1}$$

where the integral is over all vectors in the data space. In order for inference and learning in the model to be tractable, we require the energy function and its derivative w.r.t. its parameters to be easy to compute. In addition to that, for the distribution to be proper, the integral must converge.

Energy-based models can be trained by maximizing the log likelihood of the training set under the model using gradient ascent. The derivative of the log likelihood w.r.t. a model parameter w_i, averaged over the training set, is given by

$$\left\langle \frac{\partial \log p(\mathbf{x}|\mathbf{w})}{\partial w_i} \right\rangle_{D^0} = -\left\langle \frac{\partial E(\mathbf{x}|\mathbf{w})}{\partial w_i} \right\rangle_{D^0} + \left\langle \frac{\partial E(\mathbf{x}|\mathbf{w})}{\partial w_i} \right\rangle_{D^\infty}. \tag{2}$$

Here the angle brackets denote expectation w.r.t. a distribution, and D^0 and D^∞ refer to the data distribution and the model distribution respectively. Since we require the energy function to have a tractable derivative, the first term is easy to compute. Computing the second term is much harder, because it requires evaluating an integral over the space of all possible data vectors, which cannot be done exactly for most interesting energy-based models. We can approximate it by sampling from the model distribution using Markov chain Monte Carlo and averaging $\partial E(\mathbf{x}|\mathbf{w})/\partial w_i$ over the samples. However, MCMC can take a long time to reach the equilibrium distribution making the process of generating samples very slow. Since the expectation over the model distribution has to be approximated at least once per parameter update, maximum likelihood (ML) learning in EBMs is impractically slow. To make matters worse, estimates computed using MCMC can have high variance, which can lead to unreliable learning [Hinton, 2002].

However, it has been demonstrated that running the Markov chain until the equilibrium distribution is reached is unnecessary [Hinton, 2002] for training EBMs. Starting the chain at the data vectors from the training set and running it for only a few steps produces *confabulations*, which can be used in place of the true equilibrium samples in Eq. 2 to approximate the derivative. This approach results in the *contrastive divergence learning rule* for learning model parameters:

$$\Delta w_i \propto -\left\langle \frac{\partial E(\mathbf{x}|\mathbf{w})}{\partial w_i} \right\rangle_{D^0} + \left\langle \frac{\partial E(\mathbf{x}|\mathbf{w})}{\partial w_i} \right\rangle_{D^n}. \quad (3)$$

Here D^n stands for the distribution of confabulations produced by running MCMC for n steps starting at the data, where n is small. Confabulations can be viewed as data vectors slightly perturbed by the model in such a way that, on average, the model finds them more probable than the original vectors. The learning rule in Eq. 3 tries to make the training cases more probable, on average, than their confabulations by lowering the energy of the training cases while raising the energy of the confabulations.

The main advantage of CD learning is that it is much faster than ML learning. This is a consequence of not having to wait for the Markov chain to reach its equilibrium distribution. Also, it has been argued [Hinton, 2002] that, for the same number of samples, CD learning often is more reliable than ML learning since the variance of the gradient estimate in CD learning is lower because the confabulations are typically much closer to the corresponding data vectors than the equilibrium samples are.

In this paper, we use a feed-forward neural network with a single output unit as a parametric function for assigning energy to data vectors. Neural networks are suitable for this task because they can represent a large class of parametric mappings and the derivatives required to fit them can be efficiently computed with the backpropagation algorithm. When the gradient of the energy function with respect to the input vector is available, as it is when a neural network is used for energy assignment, confabulations can be generated efficiently using the Hybrid Monte Carlo algorithm [Neal, 1993], which simulates the dynamics of a particle that represents a data vector $\mathbf{x}$, on a frictionless surface the height of which is given by $E(\mathbf{x}|\mathbf{w})$. Because the dynamics is simulated in discretized time, the simulation results are not exact and the total energy of the particle is not conserved. To avoid introducing a bias into the simulation, the endpoint of the particle's trajectory is accepted with probability $\min(1, \exp(-\Delta E))$, where ΔE in the change in the combined potential and kinetic energy between the starting point and the endpoint of the trajectory. We use the "leapfrog" discretization that eliminates the numerical errors up to the second order [Neal, 1993]. A confabulation is generated by a making small number of leapfrog steps, starting at a data vector, and accepting or rejecting the trajectory endpoint. If the endpoint is rejected, the particle is returned to the position given by the data vector. The confabulation is given by the final position of the particle. Contrastive divergence learning that uses backpropagation to compute the gradient of the energy function and HMC to generate confabulations is called *contrastive backpropagation* (CBP) [Teh et al., 2003].

III. Experimental results

A. The problem

We applied contrastive backpropagation to the problem of modelling the distribution of trajectories of two interacting balls bouncing in a two-dimensional box. The trajectories are generated by simulating the idealized billiard-ball dynamics in continuous time, with no friction, air resistance, or gravity, treating all ball-ball collisions as perfectly elastic, and recording the coordinates of ball centers at evenly-spaced times. The box is an axis-aligned square of side length 2, centered at the origin. The first ball has a radius of 0.2 and half the mass of the second ball, which is of radius 0.283. The initial coordinates of the balls are chosen uniformly at random so that both balls are entirely inside the box and are not in contact with each other. Ball velocity in each dimension is initialized from the uniform distribution on the interval $[0.2, 0.45]$ and is made negative with probability $\frac{1}{2}$.

Each trajectory is represented by a finite sequence $(x_1^1, y_1^1, x_1^2, y_1^2, ..., x_n^1, y_n^1, x_n^2, y_n^2)$ of the balls' coordinates, where the subscript indicates the time when the coordinate was recorded and the superscript indicates the ball to which the coordinate belongs. We will refer to the system state $(x_t^1, y_t^1, x_t^2, y_t^2)$ recorded at time t as the *frame* t and to the number of frames in a trajectory as its *length*.

We visualize the trajectory of each ball by drawing a point showing its location for each frame and connecting the points corresponding to pairs of consecutive frames with line segments. A line segment shows the true path taken by the ball between the frames if no bounce occurred during that time. If a bounce did occur, the line segment indicates only that the points connected by it correspond to consecutive frames. We indicate which frame a point corresponds to by showing the time index t of the frame next to the point. The trajectory of the first ball is drawn using a solid line, while a broken line is used for the trajectory of the second ball. To make it easier to evaluate the accuracy of a trajectory visually, we also draw two squares inside the box, which mark the valid range of coordinates for the ball centers. The line style of a square is the same as the line style of the ball trajectory to which it applies.

B. Network architecture

We used a deterministic feed-forward neural network with two hidden layers and a single output unit as an energy-based model. Since a network has a fixed number of input units, it can be used to model the density of only fixed-length trajectories directly. We chose to model trajectories of length 5 because it is sufficient to capture the essential structure of trajectories without making the model unnecessarily cumbersome. As a result, the network has 20 input units, representing the four coordinates of the two balls in each of the five frames.

We found that the network worked well when the first and the second hidden layers contained 200 and 80 units respectively. The activation function of the units in the hidden layers is $f(x) = \log(1 + x^2)$, which is the energy function corresponding to the Cauchy distribution, while the output unit is linear.

In a network with this architecture, each unit in the first hidden layer implements a linear constraint, with the input to each unit being the amount by which the constraint is violated by the input vector. The output of such a unit is the penalty for violating this constraint. The units in the second hidden layer implement nonlinear constraints on the input vector by imposing linear constraints on the outputs of the units in the first hidden layer, which allows the network to capture the dependencies between the violations of the constraints implemented by the first hidden layer. It can, for example, make the cost of violating several constraints at the same time be much less than the sum of the individual violation costs. Since the output unit is linear and the activation function of each unit in the second hidden layer is an energy function for a heavy-tailed distribution, the EBM implements a product of nonlinear FAS constraints.

Even though the network can directly model trajectories of length 5 only, it can be used to induce probability distributions over trajectories of any fixed length greater than 4. We define the energy of a trajectory of length n to be be the sum of the energies of the $n-4$ subtrajectories of length 5 contained in it. When the energy of a trajectory is converted to probability using Eq. 1, the normalization is carried out over the space of trajectories of the same length, leading to a separate probability distribution for each trajectory length.[1]

C. Training

The network was trained on a dataset consisting of 10000 trajectories of length 5. Training took 2000 passes through the training set with parameters updated after each 100 trajectories. Each confabulation was generated by performing one step of HMC consisting of 10 leapfrog steps, with the leapfrog step size dynamically adjusted to keep the acceptance rate around 90%.

The synthetic noise-free nature of the dataset requires very tight constraints to model it well. Such constraints pose a problem for HMC-based learning because they force HMC to use a very small leapfrog step size to keep the acceptance rate reasonable, which makes learning unacceptably slow. The problem is made worse by the tendency of CBP to learn some constraints much faster than others, leading to models that never manage to learn most of the constraints. We worked around the problem by adding Gaussian noise of standard deviation 0.01 to the original training set before each pass and using the noisy dataset during that pass. This technique prevents HMC from being trapped by very tight constraints, since the noise effectively imposes a limit on their tightness.

[1]This approach does not have the desirable property that $\int p_n(f_1,...,f_n)df_n = p_{n-1}(f_1,...,f_{n-1})$, where f_i is the i^{th} frame. That is, the marginal distribution resulting from marginalizing out the last frame in the distribution of trajectories of length n, p_n, in general will not be identical to the distribution of trajectories of length $n-1$, p_{n-1}, as defined by the model. However, this is not a flaw of EBMs, but simply a consequence of defining the probability of a long trajectory to be proportional to the product of probabilities of its (overlapping) subtrajectories.

In order for the distribution defined by the network to be normalizable, the weights to the output unit have to be non-negative. We did not enforce this requirement explicitly because initializing these weights to 1 was sufficient to prevent them from becoming negative during training. All other weights were initialized from a zero-mean Gaussian distribution of small variance.

We used a learning rate of 0.01 and a momentum of 0.9 for all weights. Each weight had an adaptive gain that was additively increased if the current weight increment had the same sign as the last one and multiplicatively decreased otherwise. L_1 weight decay[2] of 10^{-3} was applied to the weights from the input layer to the first hidden layer and from the first to the second hidden layer. Such a high value of weight decay delays the appearance of high energy barriers that prevent HMC from exploring the energy landscape efficiently.

D. Inspecting the weights

After training the network, we inspected its weights. Almost every unit in the first hidden layer implements a (linear) constraint on coordinates of the same type (x or y). The majority of the constraints apply to coordinates of both balls, but some of them involve coordinates of only one of the balls. Five of the constraints involve both coordinate types for both balls and appear to be used for modelling collisions between the balls. An inspection of the weights between the hidden layers reveals that most of the input to each unit in the second hidden layer comes from the units in the first hidden layer that implement constraints on coordinates of the same type. This means that the nonlinear constraints implemented by the units in the second hidden layer apply mostly to coordinates of the same type, but for both balls. Some of the second hidden layer units specialize further by implementing nonlinear constraints on the coordinates of the same type for a single ball. This means that the network takes advantage of the fact that parts of trajectories of different balls are often independent of each other.

E. Sampling from the model

We used HMC to generate 200 samples from the model to see what it had learned. Each sample was obtained by performing 1000 steps of HMC with each step consisting of a sequence of 50 leapfrog steps. Simulated annealing was used to reduce the time required to reach the equilibrium distribution. The leapfrog step size was adjusted after each step of HMC to keep the acceptance rate around 90%.

Since the radius of the first ball is 0.2, the range of its x and y coordinates in perfect trajectories is $[-0.8, 0.8]$. The corresponding range for the coordinates of the second ball is $[-0.717, 0.717]$. Comparing these ranges to the ones computed from the samples reveals that the largest violation of the range constraint is about 0.09. However, only 8 of the 200 samples violate the range constraints by 0.05 or more, which means that significant range constraint violations are rare.

[2]Using L_1 weight decay of c corresponds to adding a penalty term $c\sum_i |w_i|$ to the objective function being minimized.

We inspected the samples visually to determine how well different trajectory segment types were modelled. The wall bounces and free-flight segments were handled quite well. The ball-ball collisions, however, were modelled with lower accuracy. The balls bounced off each other almost every time they should have, but the direction and the magnitude of the resulting velocities were sometimes clearly inaccurate. Also, most of the significant range constraint violations occurred around the time of a ball-ball bounce. It appears that the model captured the essence of ball-ball bounces but had some difficulty modelling them with high accuracy.

F. Super-resolution

To determine how well the EBM models the distribution of trajectories, we used it to compute the super-resolution versions of test trajectories. We generated a test set of 100 trajectories of length 9 and made a low-resolution version of it by setting the coordinates in the even-numbered frames to zero. A super-resolution version of each low-resolution trajectory was computed by performing gradient descent in energy w.r.t. the missing coordinates until the norm of the gradient fell below 10^{-3}. To avoid poor local minima, the procedure was repeated 10 times, starting with different initializations of the even-numbered frames, and the solution with the lowest energy was kept. Each time the missing coordinates were initialized from a uniform distribution on the interval [-0.8, 0.8]. Figure 1 shows an example of a test trajectory, its low-resolution version, and the corresponding super-resolution version found using the EBM.

As a baseline for comparison, we used a simple linear interpolation method that fills each missing even-numbered frame f_t of a high-resolution trajectory by the average of the two neighboring odd-numbered frames f_{t-1} and f_{t+1}. Note that if there is no bounce between time $t-1$ and time $t+1$, this approach computes the correct value for f_t. However, if there is a bounce in that time interval, f_t computed by the method is incorrect.

The mean squared distance from a test trajectory to the super-resolution reconstruction of its low-resolution version computed using the EBM is 0.0928. The corresponding distance for the super-resolution trajectories computed using the linear interpolation method is 0.4361. These numbers clearly show that using the EBM for doubling the time resolution of trajectories produces much better results than simple frame averaging. This indicates that the EBM models trajectories both with and without bounces quite accurately.

A visual comparison of the test trajectories to the corresponding super-resolution trajectories indicates that the network has the most difficulty filling in the frames that are near ball-ball bounces. Frames that are a part of free-flight trajectory segments or wall bounces are usually filled in with high accuracy. It should be noted, however, that the ball-ball collisions are difficult to model, as the position and velocity of the balls after a collision are a complicated and sensitive function of those attributes before the collision. Estimating such a function accurately can require large amounts of training data. Moreover, it is likely that the network used is not powerful enough to model this mapping accurately.

G. Denoising

We tested the model further by using it as a prior for denoising noisy trajectories. We consider trajectories corrupted by additive i.i.d. zero-mean Gaussian noise of standard deviation σ. Let $\mathbf{y}$ be the trajectory obtained by corrupting a noiseless trajectory $\mathbf{x}$ with noise. Then the MAP estimate of the original trajectory $\mathbf{x}$ is given by

$$\begin{aligned} \mathbf{x}^{\mathbf{MAP}} &= \operatorname*{argmin}_{\mathbf{x}} \left(-\log p(\mathbf{y}|\mathbf{x}) - \log p(\mathbf{x})\right) \\ &= \operatorname*{argmin}_{\mathbf{x}} \left(\sum_i \frac{(\mathbf{y}_i - \mathbf{x}_i)^2}{2\sigma^2} + E(\mathbf{x}|\mathbf{w}) \right). \quad (4) \end{aligned}$$

We can estimate the original trajectory by performing gradient descent on the cost function in Eq. 4, where the first term is the energy contribution from the Gaussian noise model while the second term in the energy assigned to $\mathbf{x}$ by the EBM.

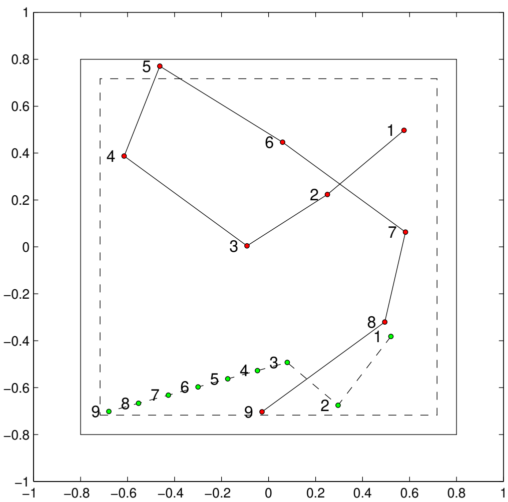
We generated a test set of 200 trajectories of length 10 and corrupted it using i.i.d. Gaussian noise of standard deviation of 0.1. The MAP estimates of the original trajectories were computed by performing gradient descent on the cost function in Eq. 4 until the L_2 gradient norm fell below 10^{-3}. Since our goal was to evaluate the EBM as a model of trajectories, we used the true value of σ in Eq. 4. In practice, σ is usually unknown and has to be estimated from data. An example of a test trajectory along with its noisy version and the corresponding denoised trajectory is shown on Figure 2.

The mean squared distance from a test trajectory to its noisy version is 0.4049. The mean squared distance from a test trajectory to the corresponding denoised trajectory is 0.3352, which is a small but significant improvement. Note that even if a perfect trajectory model was used as a prior for denoising, the denoised trajectory would not, in general, be identical to the trajectory that was corrupted by noise. As a result, it is not clear what value for the mean squared error corresponds to the perfect performance.

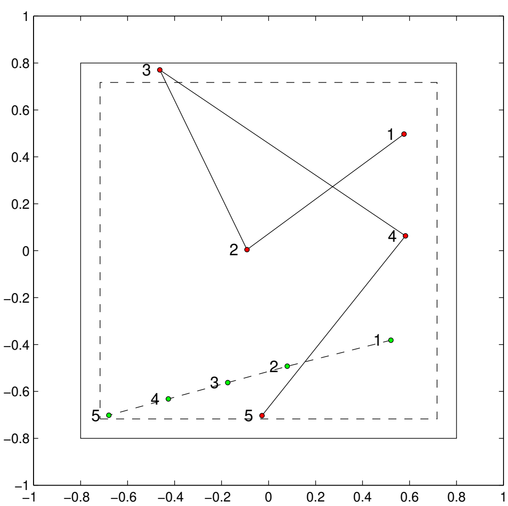
A visual inspection of the original/noisy/denoised trajectory triples showed that the denoised trajectories were significantly closer to the original trajectories than the noisy trajectories were, especially when the trajectories contained no ball-ball collisions. More importantly, the denoised trajectories were much more trajectory-like than than the corresponding noisy trajectories.

H. Comparison to a Gaussian mixture model

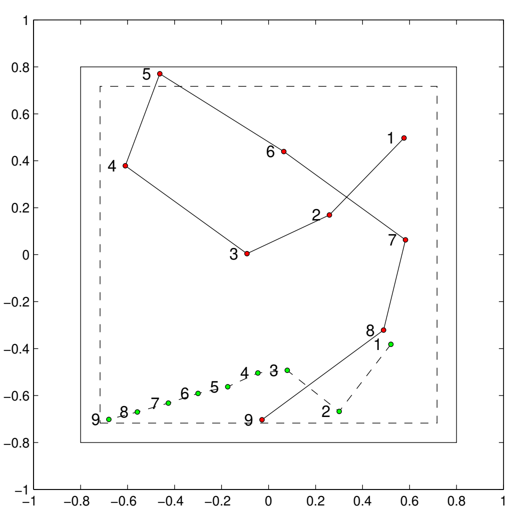
We compared the EBM to a Gaussian mixture model (GMM) trained on the same training set. The mixture models were fitted using the implementation of the EM algorithm from the Netlab toolbox [Nabney and Bishop, 2003]. Before running EM, the mixture components, which had full covariance matrices, were initialized by running the K-means algorithm. To prevent the matrices from becoming singular in the course of training, their eigenvalues were constrained to stay above 10^{-5}.



(a)

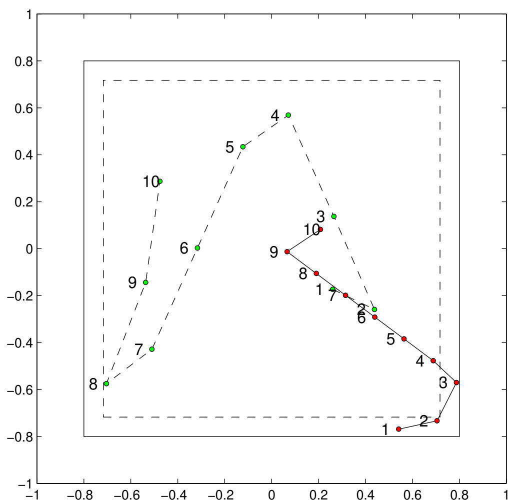


(b)

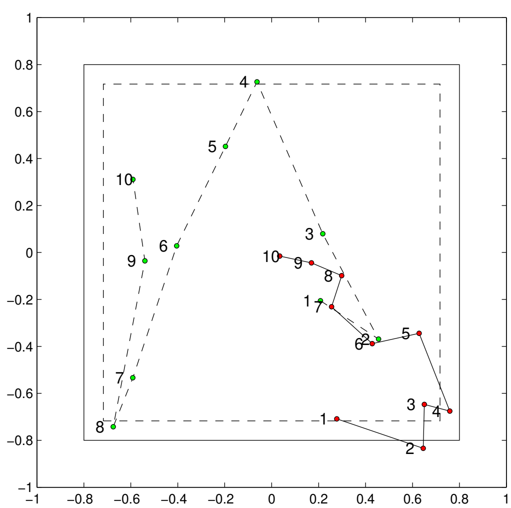


(c)

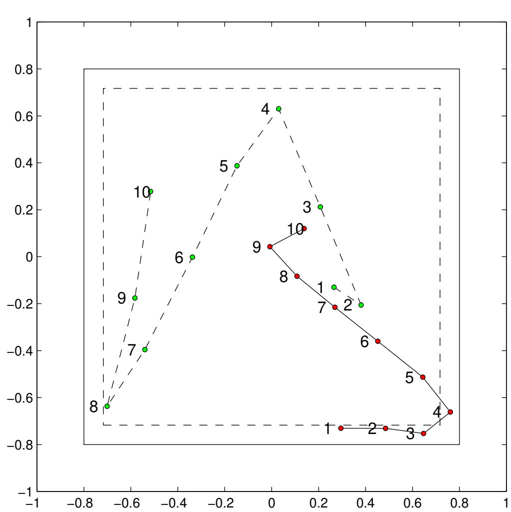
Fig. 1. Super-resolution example: (a) a test trajectory from the test set; (b) its low-resolution version; (c) the super-resolution version of the trajectory in (a).



(a)



(b)



(c)

Fig. 2. Denoising example: (a) a test trajectory from the test set; (b) trajectory (a) corrupted by noise; (c) trajectory (b) after denoising.

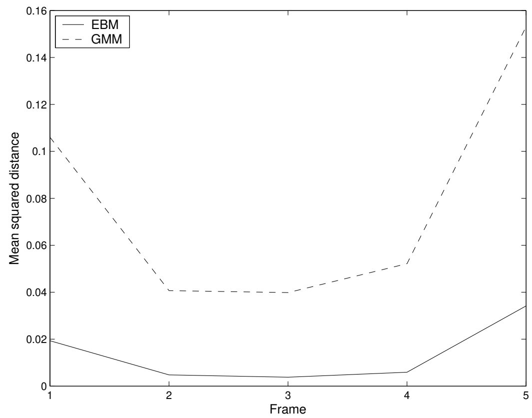


Fig. 3. Mean squared distance over a test set of 2000 trajectories between a test trajectory and the corresponding predicted trajectory for the frame prediction task.

To avoid choosing the number of mixture components explicitly, we considered the 6 possible values $\{20, 30, 40, 50, 60, 70\}$. Since the EM algorithm often gets trapped in poor local minima, we fitted 10 models for each number of components considered. The resulting sixty mixture models were compared based on the probability they assigned to a validation set consisting of 1000 trajectories of length 5. The winning model, which had 30 mixture components, was compared to the EBM.

Since the normalization term in Eq. 1 is hard to compute, we cannot compare the EBM to the mixture based on the probability they assign to a test set. Instead, our comparison of the two models is based on the idea that, given a perfect trajectory with a small number of coordinates missing, a good trajectory model should be able to predict the missing coordinates values accurately. We used the models to predict the coordinate values in a missing frame given all other frames of a trajectory.

The predictions were made by performing local minimization, starting at a test trajectory, of the energy of the trajectory under the model w.r.t. coordinates to be predicted, while keeping the other coordinates fixed. The accuracy of predictions was measured by the squared distance between the test trajectory and the local minimizer. Local minimization was performed using steepest descent with an adaptive step size until the norm of the gradient w.r.t. free coordinates fell below 10^{-4}.

The models were compared on a test set of 2000 trajectories of length 5. The results are shown on Figure 3, where the values plotted are the averages over the test set. The mean squared distance from a test trajectory to the trajectory predicted by the EBM was 4 to 10 times smaller than that the corresponding distance for the GMM. This indicates that the predictions made by the EBM are, on average, significantly more accurate than the ones made by the GMM, which shows that the EBM has a better model of the trajectories.

IV. Discussion

Frequently approximately satisfied constraints provide an efficient way of modelling certain types of datasets that are difficult to model using other methods. This approach to data modelling was introduced in Hinton and Teh [2001], where a model implementing a product of linear FAS constraints was applied to image patches. In this paper, we have shown that the nonlinear version of this approach is also feasible by applying an EBM implementing a product of nonlinear FAS constraints to trajectories of two balls bouncing in a two-dimensional box. The EBM's performance on the denoising and super-resolution tasks demonstrated the high quality of the resulting model. A comparison of the EBM to a Gaussian mixture model of the dataset showed that the EBM modeled trajectories with a significantly higher accuracy than the mixture model did. The fact that some of the nonlinear constraints learned by the EBM specialized on a single ball and/or coordinate type, indicates that the model takes advantage of the near-independence of between x and y coordinates as well as the loose coupling between coordinates of different balls. This suggests that EBMs with similar architecture should be efficient at modelling distributions in which subsets of variables are approximately independent.

References

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1711–1800, 2002.

G. E. Hinton and Y.W. Teh. Discovering multiple constraints that are frequently approximately satisfied. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, volume 17, 2001.

I. Nabney and C. Bishop. Netlab neural network software, 2003.

R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

J. Pearl. *Probabilistic Inference in Intelligent Systems. Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

Y.W. Teh, M. Welling, S. Osindero, and G. E. Hinton. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:1235–1260, Dec 2003.

M. Welling, F. Agakov, and C. K. I. Williams. Extreme components analysis. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

C. K. I. Williams and F. V. Agakov. Products of Gaussians and probabilistic minor component analysis. *Neural Computation*, 14(5), 2002.