Selected Graphical Models and Their Applications in Data Analysis

Iulian Vlad Serban
Department of Mathematical Sciences,
University of Copenhagen, Copenhagen, Denmark

***Abstract*—This is the final paper for the course Advanced Topics in Data Analysis, University of Copenhagen, 2012. A broad range of graphical models are described and implemented for practical applications. First, ancestral and Markov blanket sampling for Bayesian networks are described. Next, Gaussian mixture models for texture synthesis are described, implemented and tested. A suggestion to improve the texture synthesis is proposed and implemented. Then, Restricted Boltzmann Machines are presented and the Contrastive Divergence (CD) training procedure is described. An altered version of CD is suggested and compared empirically to the original CD. Lastly, object tracking with a Kalman filter is described. This is implemented with a standard particle filter approximating it. Furthermore, a mixed-state particle filter is suggested and implemented. The Kalman filter is compared with the two particle filters empirically.**

***Keywords*-Bayesian networks; Markov blanket sampling; Gibbs sampling; Gaussian mixture models; texture synthesis; Restricted Boltzmann Machines; object tracking; Kalman filter; particle filter;**

I. Bayesian Networks

A. Ancestral Sampling

We start by explaining the ancestral sampling method for Bayesian networks. The goal of ancestral sampling is to provide samples for the entire distribution of the Bayesian network, see [1, p. 365]. We first consider the the joint probability distribution defining a Bayesian network, [1, p. 362]:

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\text{pa}_k) \tag{1}$$

The joint probability distribution can be written as a product of distributions, for each random variable conditioned on the parents of that random variable. Suppose we order the nodes in such a way that any node only has links going out to higher numbered nodes. Then, A node x_k will never have as parent x_m if $m \geq k$. Therefore we start by sampling the lowest numbered node, x_0, which by definition has no parents. The probability of a certain sample x_0 will then be $p(x_0)$. We then go on to sample the next node in the order and continue until we have sampled all the nodes. Every time we sample a node, we use the conditional distribution since we have already its parents. Each such node will then contribute with a multiplying factor $p(x_k|\text{pa}_k)$ to the joint distribution, yielding equation (1).

This procedure presupposes that we have no observed nodes. If we have observed nodes, which have no parents or only parents that are also observed, we may use the same procedure and simply set the observed variables to the observed values instead of sampling them. If we have observed nodes, which have unobserved parents, the procedure breaks down. We would then have to condition the parents of the observed nodes on the observed values to obtain equation (1), but this is not a part of the ancestral sampling procedure.

An alternative is the method of rejection. Here we perform ancestral sampling as if all observed nodes, with parents were unobserved. Once we obtain a sample, we check if the observed nodes in the sample are equal to their observed values. If they are equal we accept the sample, and if not we reject the sample and repeat the procedure. This will ensure that the produced samples are sampled from the distribution of the Bayesian network, conditioned upon the observed variables taking their observed values. See [1, p. 528]. However, this method is only applicable to discrete variables. In particular, it is only tractable for small Bayesian networks, since the number of rejected samples increases with the size of the network.

B. Markov Blanket Sampling

The Markov blanket sampling method samples a single node, conditioned on the other nodes in the Bayesian network. The method can be applied when wishing to sample a single node in a Bayesian network, where all other nodes are observed. It can also be applied as a part of the Gibbs sampling procedure, as discussed later. To understand Markov blanket sampling, we have to understand what the Markov blanket is. The Markov blanket for a variable x_i, denoted by M, is the set of other nodes in the Bayesian network, such that for other nodes $x_j \notin \mathbf{x}_{M\cup\{i\}} \Rightarrow x_i \perp x_j \mid \mathbf{x}_M$. The conditional distribution for a node x_i given all other nodes becomes, [1, p. 382]:

$$\begin{aligned} p(\mathbf{x}|\mathbf{x}_{i\neq j}) &= \frac{p(\mathbf{x}_1,\ldots,\mathbf{x}_D)}{\int p(\mathbf{x}_1,\ldots,\mathbf{x}_D)d\mathbf{x}_i} \\ &= \frac{\prod_k p(\mathbf{x}_k|\text{pa}_k)}{\int \prod_k p(\mathbf{x}_k|\text{pa}_k)d\mathbf{x}_i} \\ &= \frac{\prod_{k\in\{i\cup M\}} p(\mathbf{x}_k|\text{pa}_k)}{\int \prod_{k\in\{i\cup M\}} p(\mathbf{x}_k|\text{pa}_k)d\mathbf{x}_i} \end{aligned} \tag{2}$$

By equation (2), we see that the Markov blanket M consists of the parents of x_i, the children of x_i and finally the parents of the children of x_i. Indeed, the conditional probabilities for the parents of x_i and the parents of the children of x_i, will also cancel out, but they will still remain as conditioned variables in the remaining conditional probabilities, and therefore have to be included in M. Thus, Markov blanket sampling is the method of sampling a single node x_i, conditioned on its Markov blanket M.

C. Markov Blanket Sampling for Gibbs Sampling

The reader can find a description of Gibbs sampling in [1, p. 542]. We proceed directly to relate Markov blanket sampling to Gibbs sampling. By the Gibbs sampling method, after picking a site i in the Bayesian network we will sample it, conditioned on all other nodes in the Bayesian network. This conditional probability is the same as in equation (2). With this in mind, we can perform Gibbs sampling as follows. Initialize all nodes in the Bayesian network, from a distribution with strictly positive probability for all configurations of the nodes. For each iteration of the Gibbs sampler, either pick a site i at random or pick at site i according to some predefined order. Then, identify its Markov blanket and sample the variable x_i according to equation (2). We will call this a Gibbs update. Repeat this procedure until either a predefined number of sites have been samples, or until all sites have been sampled once.

If there are any observed variables, these are set during the initialization to the respective observed values. During the procedure, whenever we pick a variable x_i, which has been observed, we simply leave the variable to the observed value. Therefore, the Gibbs sampler is effectively running only a subset of the variables in the Bayesian network. For this subset of the variables all the properties of the Gibbs sampler, described in [1, chapter 11.3], still holds. Thus, as the number of Gibbs updates go to infinity, the distribution of the produced sample is:

$$p(\mathbf{x}_{\text{not observed}}|\mathbf{x}_{\text{observed}}) \tag{3}$$

This is exactly what we want. Therefore, Gibbs sampling can be applied to Bayesian networks, which have observed variables anywhere in the network.

D. Implementation

Ancestral sampling and Markov blanket sampling have been implemented for [1, p. 362, Figure 8.2] in C++. The nodes have been assumed to take only the values $\{0, 1\}$, with probability distributions representable by conditional probability tables.

II. Texture Sampling using Gaussian Mixture Models

In this section, texture sampling will be described. To perform it we will use *Gaussian Mixture Models* (GMMs), see [1, p. 110-113]. The GMM is defined by the probability density function:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k) \tag{4}$$

Here the parameters $\boldsymbol{\pi}$ are the mixing coefficients and each $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$, called a Gaussian component, is a multivariate normal probability density function, with means $\boldsymbol{\mu}_k$ and covariance matrix Σ_k. For the probability density function to be valid, the covariance matrices must all be positive-definite and the mixing coefficients must satisfy:

$$\sum_{k=1}^{K} \pi_k = 1 \qquad 0 \le \pi_k \le 1 \tag{5}$$

The GMM arises from a scenario, where a data vector $\mathbf{x}$ is drawn from distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$ with probability π_k for $k \in \{1, \ldots, K\}$. The GMM can also be represented as a graphical model with two nodes: an observed node $\mathbf{X}$ and, its parent, a hidden node $\mathbf{Z}$. The hidden node $\mathbf{Z}$ can take the discrete values $\{1, \ldots, K\}$ with probabilities π_k. Given $\mathbf{z} = k$, the visible node $\mathbf{X}$ then takes values from $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$.

A. Texture Synthesizing

The Gaussian mixture models, being Bayesian networks, are *generative models*, that is models that define probability distributions for both the data and the model parameters. This enables us to synthesize new data points, given that the underlying model parameters are known. See [1, p. 43 and p.365] for a short description. Therefore, GMM can be used to synthesize textures.

The synthesis is performed as follows. First, N image patches of size DxD are generated from a texture image. This is done in either of two ways: 1) *deterministic sampling* by cutting the texture image, in a top-down-left-to-right order, into at least N disjoint image patches, which will then form the samples, 2) *random sampling* by picking N coordinates in the texture image uniformly and, then, for each coordinate using the pixels in the DxD square below and to the right of this coordinate to form a sample. Then, the structure of the GMM is specified, i.e. the number of Gaussian mixture components and the initial values of the parameters. Now, the parameters of the GMM are estimated using maximum likelihood. This is done by maximizing the log-likelihood:

$$\ln P(\mathbf{X}|\{\boldsymbol{\mu}_k, \sigma_k, \pi_k\}_k) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x_n}|\boldsymbol{\mu}_k, \sigma_k) \right) \tag{6}$$

However, this poses a serious problem. There can, in general, be multiple local optima in the log-likelihood, as noted in [1, p. 438]. This means an analytical closed form solution does not exist, and instead an iterative scheme must be applied. There are a number of ways to do this, but one particular method has had considerable success in maximizing the log-likelihood. This is the *expectation maximization algorithm* (EM algorithm). A comprehensive description of it can be found in [1, p. 438-439]. When using the EM algorithm, one has to be careful from both a theoretical and practical point of view, because the procedure can produce degenerate solutions. This would for example be the case if one of the Gaussian components had variance and covariances going to zero, where its mean was fixed on a single data point. Then the log-likelihood would go towards minus infinity, because all other data points would have a probability going towards zero, which means that the terms in the inner sum goes towards zero in equation (6). There is another problem, which is more specific to the EM algorithm. The algorithm requires computing the inverse and determinant of the covariance matrices of each Gaussian component. Not only is this computationally very expensive, but it can also result in numerical errors, since the normalization constant in the multivariate normal distribution grows exponentially with the number of dimensions. To avoid this, two precautions are taken. First, if either the entry-wise norm of a covariance matrix becomes too small, or the rank of the covariance matrix does not equal D^2, the covariance matrix is reset. Second, all probabilities calculated in the program are constrained to be above some very small constant, because they are theoretically always strictly positive. Lastly, the each pixel intensity is scaled linearly to be within $[0,1]$.

B. Transforming the data

So far, we have assumed that the data, in its scaled format, fits perfectly with the GMM. Although the GMM is a very rich model, this is not necessarily the case or it might require an immense number of components to model the data well. We will therefore take a statistical approach to transforming the data, and apply the bijective transformation

$$\mathbf{y} = \mathbf{x}^T \mathbf{x}, \tag{7}$$

where $\mathbf{x}$ is an image patch. If $\mathbf{x}_i$ was close to a uniform distribution on $[0,1]$, then the transformation would make the probability mass of $\mathbf{y}_i$ skewed to the left, which potentially would be better modeled by a normal distribution.

C. Implementation

A Gaussian mixture model was implemented in C++ with the EM algorithm for finding the maximum likelihood parameters, exact sampling using the Cholesky decomposition and sampling using the MCMC metropolis-hastings algorithm, see [1, p. 541], with the isotropic multivariate normal distribution as proposal distribution. The precautions for the EM algorithm and the transformation, discussed earlier, were also implemented. An interface for loading images, training and testing the Gaussian mixture model was also developed. The implementation was based on the C++ linear algebra library Armadillo [11].

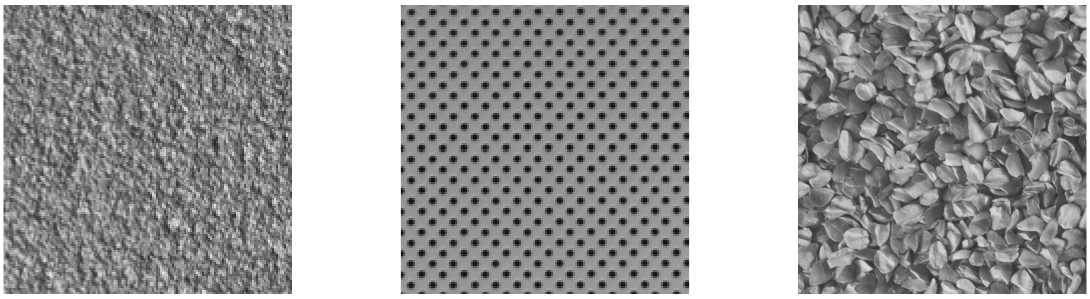
D. Empirical Results

A number of experiments were carried out on a set of textures. First, the standard GMM was trained on 100 $8x8$ texture patches. The GMM was trained using 50 trials, where the means were initialized uniformly on $[0,1]$, the covariance matrices were set to standard multivariate isotropic covariance matrix, where the parameter solution with the highest log-likelihood was choosen as the solution. Having a large number og Gaussian components did not improve the solutions and often resulted in singular covariance matrices. Therefore, 5, 6 and 10 Gaussian components were used. Tests were carried out, using both sampling methods described above. First, the means were analyzed from graphical plots, but did not appear to convey sufficient information for the naked eye to relate it to the exact textures. They did however provide some indication, of the variance of colors in the textures.

Second, the produced samples were analyzed. For most image textures, the random sampling method seemed to yield the best solution. However, for a downscaled version of the ceiling texture, shown below, the deterministic sampling method seems to have slightly worked better. Although the ceiling texture has perfectly aligned black spots, the GMM produced a texture with a higher variance of color and more sporadic black spots. The reader should keep in mind that the ceiling texture for which the GMM is downscaled.

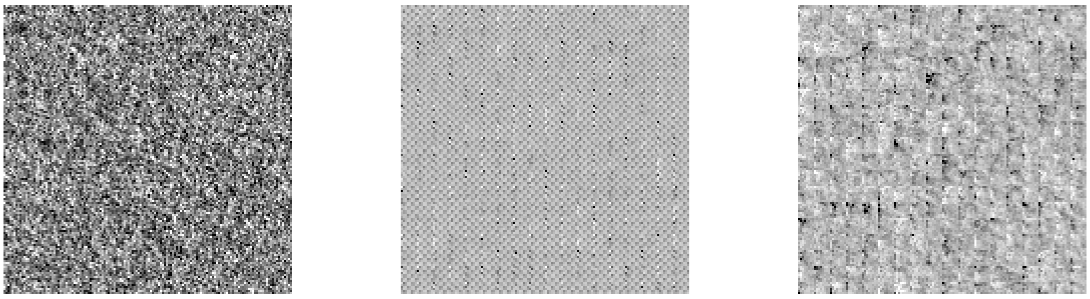
Third, the proposed transformation was analyzed in depth by plotting densities of the color values for each pixel. This was done for the wall texture, which from the previous testings, seemed a good candidate for sampling using GMMs. A standard GMM and a GMM with the transformation was used, both with only 3 Gaussian components to ensure that the EM solutions was close to the maximum likelihood solution. From the density plots, it appears that the wall texture already resembles a normal distribution, but that standard GMM models a quite differently distribution. The GMM applying the transformation appears to model much the same distribution as the Standard GMM, but seems to have removed some of the noise in the tails. From the experiment, it is impossible to say whether or not the GMM with transformation works better. However, the experiment shows that transformation of the data can change the model and should therefore be investigated further, in an attempt to improve the GMM.

Overall, the GMM models were found to have only a partial success at sampling image textures. In particular, they failed to capture a number of coherent structures in the textures. There are a number of possible reasons for this. It could be that the image patches needed to be larger, such as for example $10x10$ image patches, to properly model the textures. This would be computationally intractable and would require an alternative learning method. Similarly the number of Gaussian components could be increased, but this would also quickly lead to an intractable learning. It is also likely that the texture structures are difficult to capture using a GMM, for example because of the correlation structure and large number of pixels. It seems likely that a GMM would be better suited at capturing the underlying color distribution, and not the exact textures defined by pixels.



(a) Wall (b) Ceiling (c) Seeds

Figure 1: Texture Images



(a) Wall (b) Ceiling (c) Seeds

Figure 2: Sampled textures from GMM with data transformations. These are based on downscaled textures of the above. Wall) 8x8 patches sampled from a 5-component GMM, trained on 40 random sampled patches. Ceiling) 8x8 patches sampled from a 6-component GMM trained on 30 image patches, which were sampled deterministically. Seeds) 8x8 patches sampled from a 5-component GMM, trained on 50 random sampled patches.

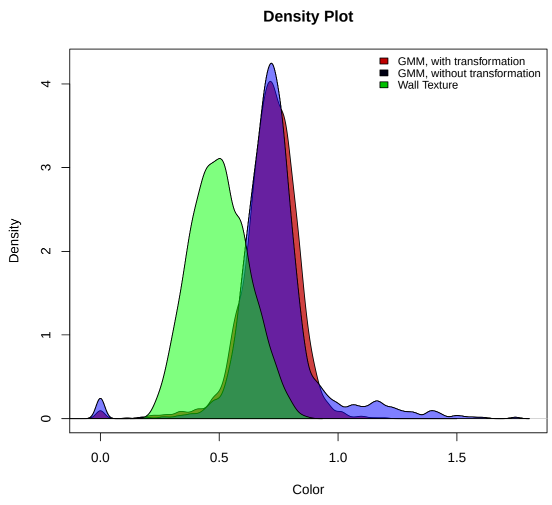


Figure 3: Density plot of the color values scaled to fit $[0, 1]$. The GMMs had 3 Gaussian components, and were trained on $7x7$ image patches to make the computations tractable. 512 Gaussian kernels were used for the density plot.

III. Restricted Boltzmann Machines

In [2, p. 29], the Restricted Boltzmann Machines (RBMs) are introduced as *energy-based models*, with the energy

$$\text{Energy}(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T\mathcal{W}\mathbf{v}, \tag{8}$$

where $\mathbf{v}$ and $\mathbf{h}$ are vectors representing the states of the visible and hidden nodes, respectively, of the graphical model. The vectors $\mathbf{b}$ and $\mathbf{c}$ are the bias parameters. The parameter $\mathcal{W}$ is the interaction matrix, connecting the visible nodes to the hidden nodes. Assume that the nodes only take discrete binary values. Then the probability distribution function would be

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z}\exp(\text{Energy}(\mathbf{v}, \mathbf{h})), \tag{9}$$

where Z is a normalization constant, called the partition function. This construction implies that $h_i \perp\!\!\!\perp h_j|\mathbf{V}$ for $i \neq j$ and $v_i \perp\!\!\!\perp v_j|\mathbf{H}$ for $i \neq j$. See [2, p. 30-32] for a comprehensive description of the RBMs.

One of the main goals of RBMs is to *sample*, i.e. generate visible nodes $\mathbf{v}$ using the distribution in equation (9) marginalized over $\mathbf{H}$. However, this requires that the parameters $\mathbf{b}, \mathbf{c}, \mathbf{W}$ are known. Let $\mathbf{v} = m$ and $\mathbf{h} = n$. By definition, Z is a sum of exponential functions over every configuration $\{\mathbf{v}, \mathbf{h}\}$. This leads to an algorithmic complexity of $O(2^{n+m})$, which is exponential in the number of nodes, and therefore makes the model intractable for anything but small models. Therefore, exact maximum likelihood learning of the parameters is intractable. Instead

Markov chain Monte Carlo (MCMC) procedures are used to train the RBMs.

A. Contrastive Divergence

The MCMC procedure called *Contrastive Divergence* (CD) has had considerable success, see [2, Section 5.4] and [3]. The CD procedure relies on Gibbs sampling, which is described in [1, Section 11.3]. There are some variants of the CD procedure, but the most common can be described as follows. Let $S = \{\mathbf{v}_l\}_l$ be a training data set. The CD procedure is then an approximation of the gradient of the log-likelihood using k Gibbs sampling steps. For each $\mathbf{v}_l$, set $\mathbf{v}_l^{(0)} = \mathbf{v}_l$ and sample:

$$\begin{aligned} \mathbf{h}_l^{(1)} &\sim P(\mathbf{h}|\mathbf{v}_l^{(0)}) \\ \mathbf{v}_l^{(1)} &\sim P(\mathbf{v}|\mathbf{h}_l^{(1)}) \\ &\dots \\ \mathbf{h}_l^{(k)} &\sim P(\mathbf{h}|\mathbf{v}_l^{(k-1)}) \\ \mathbf{v}_l^{(k)} &\sim P(\mathbf{v}|\mathbf{h}_l^{(k)}) \end{aligned} \tag{10}$$

The approximated log-likelihood w.r.t. w_{ij}, b_j and c_i is then given by, respectively:

$$\sum_l \left(\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}_l) h_i v_j - \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}_l^{(k)}) h_i v_j \right) \tag{11}$$

$$\sum_l v_{l,j} - v_{l,j}^{(k)} \tag{12}$$

$$\sum_l p(h_i = 1|\mathbf{v}^{(0)}) - p(h_i = 1|\mathbf{v}^{(k)}) \tag{13}$$

The parameters are then updated according to a learning rate $\eta > 0$, by using steepest ascent on the approximated log-likelihood.

One proposed change of the CD procedure is to change the sampling procedure, in equation (10), into:

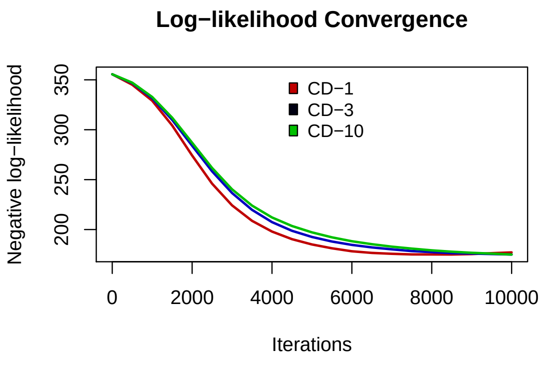
$$\begin{aligned} \mathbf{h}_l^{(1)} &\sim p(\mathbf{h}|\mathbf{v}_l^{(0)}) \\ \mathbf{v}_l^{(1)} &= p(\mathbf{v}|\mathbf{h}_l^{(1)}) \\ &\dots \\ \mathbf{h}_l^{(k)} &\sim p(\mathbf{h}|\mathbf{v}_l^{(k-1)}) \\ \mathbf{v}_l^{(k)} &= p(\mathbf{v}|\mathbf{h}_l^{(k)}) \end{aligned} \tag{14}$$

Except for the first sample, each visible sample is replaced by the vector of probabilities, from which it before was sampled. This will effect the values calculated in equation (11), as we will see later.

B. Empirical Results

To investigate and compare the above proposed CD procedures, a series of simulations were performed. For implementation the Shark Library [9] was used in C++ and its source code modified to fit the altered CD procedure.

The simulations were performed on two problem data sets. First, on the *Bars-And-Stripes* problem, where each sample consists of $4x4$ binary units. To generate a sample, an orientation is first picked, either vertical or horizontal with equal probability. Then for either each row or column,



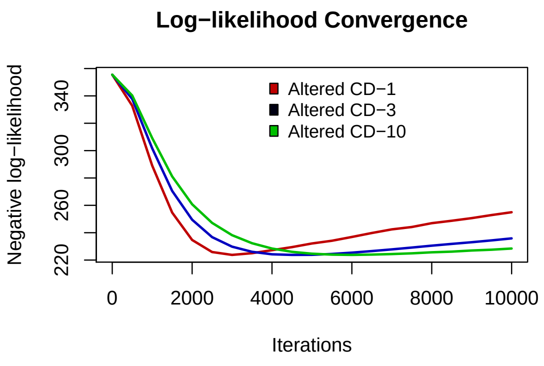


Figure 4: The average log-likelihood of RBMs trained using CD and altered CD on the Bars-And-Stripes problem. The learning rate used was 0.05.

according to the orientation, a single state is sampled from a uniform Bernoulli distribution, which is then assigned to the entire row or column. See also [5]. A lower bound of the log-likelihood is 102.59. For each simulation a total 32 samples were generated, which appears to be sufficient for the RBM to learn a significant part of the underlying problem structure. An RBM with 8 hidden units was trained for 10.000 iterations on the data set. The log-likelihood was recorded at every 500th iteration. The results in each figure were produced using 100 trials. The results show the negative log-likelihood.

Simulations were also performed on a second problem, the *Labeled Shifter Ensemble* problem. Here each sample consists of 19 binary units. These were generated as follows. First, the first 8 units are drawn independently from a uniform Bernoulli distribution. Then, the states of the next 8 units are cyclically shifted copies of the first 8 units. The shift is either zero, one unit to the left or one unit to the right. The last three states are then set according to the shift used. This data set was generated deterministically, by loop over each possible combination and saving it. This yielded $3 \cdot 2^8 = 768$ samples. If the data is modeled perfectly, the log-likelihood is 5102.43. See [4]. An RBM with 6 hidden units was trained for 1000 iterations on the data set. The log-likelihood was recorded at every 200th iteration. The results in each figure were produced using 100 trials. The results show the negative

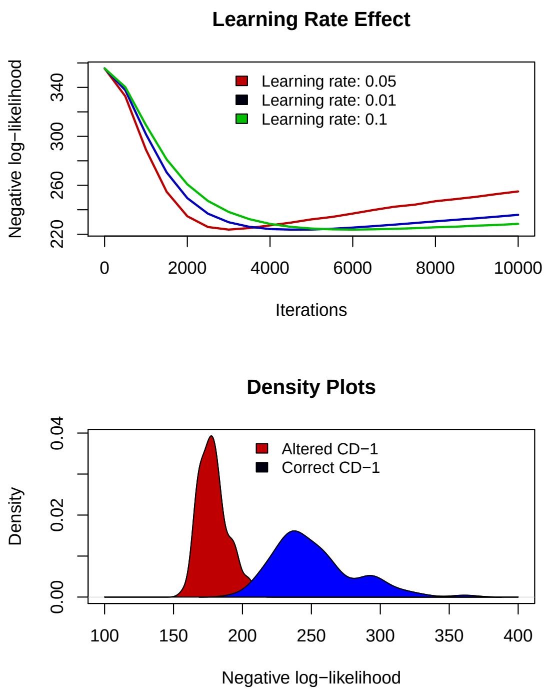


Figure 5: Top) The average log-likelihood of RBMs trained using altered CD, with different learning rates, on the Bars-And-Stripes problem. Bottom) density plots of RBMs trained using CD and altered CD with learning rate 0.05. 512 Gaussian kernels were used for the density plot.

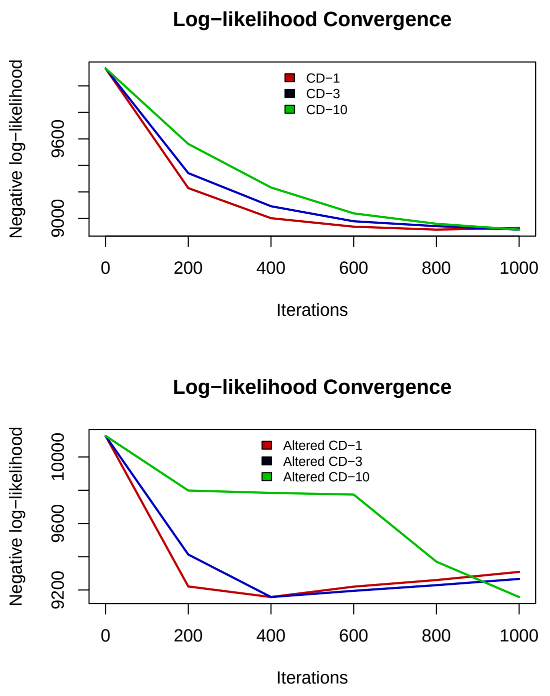


Figure 6: The average log-likelihood of RBMs trained using CD and altered CD on the Labeled Shifter Ensemble problem. The learning rate used is 2.5.

log-likelihood.

For the Bars-And-Stripes problem, the standard CD procedure outperforms the altered CD procedure w.r.t. log-likelihood. The standard CD procedure appears to improve the log-likelihood more consistently and ends up reaching a higher point than the altered CD procedure. It further seems that the log-likelihood of the altered CD starts to decrease after around 2000 iterations, which indicates that this method should not be used for training RBMs. Furthermore, lowering the learning rate appears to worsen the learning outcome.

For the Labeled Shifter Ensemble problem, the results appear to be slightly more positive for the altered CD procedure. Nevertheless. the standard CD procedure still outperforms the altered CD procedure. For some of the altered CD procedures, the log-likelihood decreases after the 200th iteration. A higher number of steps seems to remove this effect, but instead learns much slower.

C. Discussion

There is no apparent reason to believe that the altered CD procedure should perform well. Indeed, Gibbs sampling works by using conditional probabilities of the true distribution to produce new samples. If instead of producing actual visible samples, the conditional probabilities themselves are used, there is no guarantee that the hidden samples in equation (14) are produced by the true model distribution. For the altered CD procedure it holds that

$$\begin{aligned} &p(h_i^{(r)} = 1|\mathbf{v}^{(r-1)}) \\ &= \sigma\left(\sum_{j=1}^{m} w_{ij}\sigma\left(\sum_{i=1}^{n} w_{ij}h_i^{(r-1)} + b_i\right) + c_j\right). \end{aligned} \tag{15}$$

All the *information* provided by $\mathbf{h}^{(r-1)}$ is reused in finding $\mathbf{h}^{(r)}$, contrary to the standard CD procedure where only the state values are passed on. This could create an *information bottleneck*, which would reduce the CD procedures ability to search the entire configuration space effectively. A short argument is presented in [3, p. 5]. In general, the information reuse might lead the samples $\mathbf{h}^{(r)}$ and $\mathbf{h}^{(r-1)}$ to be highly correlated, which means that $\mathbf{v}_l^{(k)}$ will be correlated with $\mathbf{h}_l^{(1)}$. Therefore, it is likely that the samples produced by the altered CD procedure are not from the true model distribution. This is confirmed by the fact that increased learning rates improve the altered CD procedure for the Bars-And-Stripes problem. If the procedure was sampling from the true model distribution, a lower learning rate should result in a slower, but more stable, learning. Nevertheless, the samples produced by the altered CD procedure still reflect some characteristics of the model. For instance, samples with high probability under the true model are also likely to have a high probability of being sampled according to (15). This would explain the partial success of the method on the Labeled Shifter Ensemble problem

IV. Tracking with Kalman and Particle Filters

An important application of probabilistic graphical models lie in object tracking. Given a sequence of observations of an object, for example images of an object in movement, we would like to infer its position recursively. This is often referred to as estimating the *state of the system*. The problem can be defined as finding the maximum likelihood solution of a *linear dynamical system*. Let $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$ and $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$ be vectors representing the state of the system and the observations at times $1, \ldots, t-1$, respectively, then assume:

$$\mathbf{x}_t = \mathcal{A}\mathbf{x}_{t-1} + \mathbf{w}_t \tag{16}$$

$$\mathbf{z}_t = \mathcal{H}\mathbf{x}_t + \mathbf{v}_t \tag{17}$$

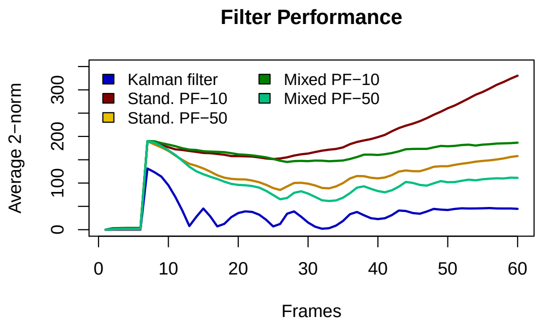
Here $\mathcal{A}$ and $\mathcal{H}$ are matrices representing the model parameters. The vectors $\mathbf{w}_t$ and $\mathbf{v}_t$ are random variables with means zero, which are said to represent the process and sensor noise. Given observations $\mathbf{z}_1, \ldots, \mathbf{z}_t$, our objective is then to infer the states $\mathbf{x}_1, \ldots, \mathbf{x}_t$. This can be done in a number of different ways, with the *Kalman filter* being one of the most popular methods. See [6, p. 20] for a comprehensive description. The Kalman filter assumes that both $\mathbf{w}_t$ and $\mathbf{v}_t$ are multivariate normal distributions, which are independent of each other. This ensures that the probability distribution factorizes, which makes the Kalman filter fast to compute. However, for certain problems the assumption of multivariate normal distributions is quite false, and therefore the Kalman filter is ineffective for these. Instead one can use *particle filters*, which allows computation of a vast number of probability distributions. In brief, they work by sampling *particles* from the conditional probabilities of $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, which are then weighted according to $p(\mathbf{z}_t|\mathbf{x}_t)$. They are, however, approximate methods and can therefore be ineffective and computationally intractable, for certain problems. The purpose of this section is to empirically compare the performance of the Kalman filter with two distinct particle filters for tracking objects, on the problem of tracking a single bouncing from video frames in two dimensions. See [8] for a similar problem.

A. Particle filters

Two particle filters were implemented in MATLAB [10]. The first was a standard particle filter, using sampling importance resampling, as described in [7, p. 8]. The distribution used was the same as defined by the Kalman filter in a MATLAB implementation, provided by Cristina Manfredotti. Let $\mathbf{x} = (x_1, x_2, x_3, x_4)$ be the state of the ball, representing position (in width and height) and the velocity (in width and height) respectively. Then

$$\mathcal{A} = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathcal{H} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{18}$$

where $dt = 1$ is a constant representing the acceleration. The covariance matrices were assumed isotropic and can



s

Figure 7: Average 2-norm difference between filtered position and the observed object position. The graph was produced using 50 trials for each filter.

be found in the attached software.

The second particle filter was a *mixed-state* particle filter, based on the work of [8]. In particular, the iterative algorithm presented in [8, p. 2] was used, which implied that the particles sampled for one frame was used in sampling the particles for the next frame. The ball was assumed to be in either of two states: 1) in movement, or 2) bouncing, i.e. in contact with the ground. When the ball is in movement, the particles are sampled according to $\mathcal{A}$ above. When the ball is bouncing each particle is sampled according to

$$\mathcal{A} = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & \tau & 0 \\ 0 & 0 & 0 & \tau \end{pmatrix}, \tag{19}$$

where $\tau \sim \text{Uniform}[0, 1]$. Transitions between the two states were defined such that the ball was in movement most of the time, and only for a single frame bouncing. Given a sufficient number of particles, the model construction with the random variable τ, should allow the ball to accelerate and deaccelerate, thus capturing the bouncing effect more effectively.

B. Empirical Results

The the Kalman filter was compared to the two particle filters, described above, w.r.t. finding the balls observed position. Since the observation noise is quite low in the experiment, the closer the filters are to the observed position of the ball, the better we would expect them to perform. The particle filters were tested on 50 trials with 10 and 50 particles for each. The 2-norms averaged over these trials are shown in Figure 7.

The results indicate that the mixed-state particle filter is significantly better than the standard particle filter, if they are both given the same number of particles. The standard particle filter also seems to diverge, i.e. consistently move away from the ball, at the last few frames if the number of particles is low. This is not observed for the mixed-state particle filter.

However, both particle filters are significantly worse than the Kalman filter at finding the observed ball position. Because the standard particle filter is an approximation of the Kalman filter, we would indeed expect it to perform this way. Furthermore, as the results strongly indicate, the higher the number of particles, the better the performance. This supports the theoretical results for sampling importance resampling algorithm, see [1, p. 534], which implies that the particles will converge in distribution to the true model distribution, as the number of particles go to infinity. Therefore, taking expectation over these samples, will converge towards the Kalman filtered state, because it is defined as the expectation of multivariate normal distributions. The mixed-state particle filter resembles the standard particle filter closely, which would explain why this also performs worse than the Kalman filter, but improves as the number of particles increase.

V. Conclusion

In this paper, a broad range of graphical models were described and implemented for practical applications. Ancestral and Markov blanket sampling were first described from a theoretical viewpoint. Then, Gaussian mixture models were described, implemented and tested for synthesizing image textures. A method of data transformation was proposed. The results of these were mixed, and indicated that the Gaussian mixtures have difficulty capturing patterns with certain coherent structures. Then, Restricted Boltzmann Machines were presented with the Contrastive Divergence (CD) training procedure. An altered version of the learning procedure was suggested and compared empirically to the original CD. The results indicated that the altered version performed poorly and that it should not be used instead of the original procedure. Lastly, object tracking with a Kalman filter was described. Two particle filters, a standard and a mixed-state model, were implemented for tracking a bouncing ball. These were empirically compared to the Kalman filter. The results favored the Kalman filter, but indicated that the mixed-state particle filter was a significant improvement over the standard particle filter.

References

[1] C. M. Bishop, *Pattern Recognition And Machine Learning*, 1st edition. Springer, 2006.

[2] Y. Bengio, Learning deep architectures for AI, Foundations and Trends in Machine Learning, 2(1), 2009.

[3] G. E. Hinton, A Practical Guide to Training Restricted Boltzmann Machines, Department of Computer Science, University of Toronto, 2010.

[4] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*, p. 282317. MIT Press, 1986.

[5] D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, 2002.

[6] G. Welch and G. Bishop, *An Introduction to the Kalman Filter*, ACM, 2001

[7] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, A Tutorial on Particle Filters for OnlineNonlinear/Non-Gaussian Bayesian Tracking, IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 50, NO. 2, 2002

[8] M. Isard and A. Blake, A mixed-state condensation tracker with automatic model-switching, ICCV '98 Proceedings of the Sixth International Conference on Computer Vision, p. 107-112, IEEE Computer Society, 1998

[9] C. Igel, T. Glasmachers, and V. Heidrich-Meisner. Shark. *Journal of Ma- chine Learning Research*, 9:993996, 2008.

[10] MATLAB 7.12.0 R2011a, The MathWorks Inc., Natick, Massachusetts, 2011

[11] C. Sanderson, Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical Report, NICTA, 2010.