Stochastic simulation with neural networks

Jef Caers

Stanford University,
Department of Geological and Environmental Sciences

Abstract

Extensive outcrop data or photographs of present day depositions or even simple drawings from expert geologists contain precious structural information about spatial continuity that is beyond the present tools of geostatistics essentially limited to two-point statistics (histograms and covariances). A neural net can be learned to collect multiple point statistics from various training images, these statistics are then used to generate stochastic models conditioned to actual data. In petroleum applications, the methodology developed can be a substitute for objects based-algorithms when facies geometry and reservoir continuity are too complex to be modelled by simple object such as channels. The performance of the neural net approach is illustrated using training images of increasing complexity, and attempts at explaining that performance is provided in each case. The neural net builds local probability distributions for the facies types (categorical case) or for petrophysical properties (continuous case). These local probabilities include multiple point statistics learned from training images and are sampled with a Metropolis-Hastings sampler which also ensures reproduction of statistics coming from the actual subsurface data such as locally varying facies proportions.

Contents

1 Introduction

Stochastic simulation is a tool that has been driven by the need for more "representative" *images* than the smoothed *maps* produced by regression techniques such as kriging. Although locally accurate in a minimum error variance sense, kriging maps are poor representations of reality with various artefacts. Kriging produces conditional bias in the sense that through smoothing small values are overestimated and large values are underestimated. Moreover, this smoothing is not uniform since it is minimal near the data locations and increases as estimation is performed further from the data locations. Smoothed maps should not be used where spatial patterns of values is important such as in the assessment of travel times or production rates in flow simulations. Reproduction of the spatial connectivity of extremes is critical in any study involving the determination of risk of a rare event happening.

Most current methods of simulation rely on the modeling of two-point statistics such as covariances and variograms and the subsequent reproduction of these second moments. Yet, many phenomena are more complex in the sense that their spatial patterns cannot be captured by two-point statistics only. In particular, identification of two-point statistics only is not sufficient to reproduce patterns of connectivity of extreme values over long ranges. Strings of extreme values, curvilinear or wedge-shaped clusters of similar values are common occurrences in earth sciences, and their importance in water or oil flow simulations cannot be overstated.

The most common way of performing stochastic simulation is through some form of Gaussian simulation (Deutsch and Journel, 1997, p. 139). This class of simulation algorithms is based on the congenial properties of multi-Gaussian models, which import some severe restrictions not always fully appreciated:

- The marginal distribution is normal which is rather thin-tailed. This is the least restrictive aspect of the multi-Gaussian assumption since we can always "scale" the simulated Gaussian field by employing an appropriate long-tailed marginal back-transformation (program trans in GSLIB, Deutsch and Journel, 1997, p. 132).
- The multivariate and thus all conditional distributions are normal which leads to extremes which in the asymptotic limit, i.e. as the random function is taken to higher threshold levels, become independent. The location of extremes of a stationary isotropic Gaussian field is equivalent to a spatial Poisson process.
- Both the shape and the variance of conditional distributions are homoscedastic, i.e., they are independent of the conditioning data values.

It is sometimes naively thought that a mere univariate back-transformation of a normal distribution into a non-normal one, renders the original Gaussian stochastic simulation a non-Gaussian one, see most major publications on stochastic simulation, except for

Deutsch and Journel (1997, p. 142) and Goovaerts (1997, p. 266). Actually, a univariate transformation merely scales the simulated field back to the target histogram but does not correct for the last two drawbacks cited above.

Improvement in connecting extremes could be obtained by using an indicator simulation method (Deutsch and Journel, 1997, p. 149). In this case, specific transition probabilities of extremes can be imposed but the method is still restricted to two locations only and therefore to two-point statistics. Also, the modeling of multiple indicator covariances can be difficult due to lack of data, particularly at extreme thresholds.

The quest for "better" models or methods is motivated by the fact that in many circumstances additional structural information on the spatial distribution is actually available. This information need not be hard data. It can be exhaustively sampled training images of a deemed similar field, e.g. outcrop photographs, secondary data exhibiting similar spatial variation. Training images can be generated using physical dynamical models or they may be hand-drawn by an expert geologist. Every bit of information gives us more *intelligence* about the actual spatial distribution, particularly that of extreme values. Also, by drawing, producing or making training images, one explicitely states one's prior belief about what the spatial model looks like, without hiding it behind some mathematical formulation such as a multi-Gaussian distribution. Using training images the whole concept of stochastic imaging is turned around: one first selects a realistic image and subsequently tries to construct a model that captures the essence of that image. Thus, instead of imaging an analytical model, one models an image deemed representative.

A framework for incorporating additional multi-point information has been introduced with the concept of multi-point statistics and extended normal equations (Journel and Alabert, 1989, Guardiano and Srivastava, 1992 and Srivastava, 1992). Information is now pooled at more than two locations at the same time, not being restricted to pairs of data such as in the variogram. A general model for incorporating such information is provided by the concept of extended normal equations, which are basically an extension of the normal (kriging) equations to the multi-point case. There are however important aspects that come into play: statistical inference, consistency, CPU and RAM. In the extended normal equations, the kriging matrix should still be positive definite. Even if the scanning of the training image is properly done, singularity can still occur if any two data events retained are fully redundant (in covariance terms) over the specific training image being used or if one data event is never found on the training image, i.e., we are unable to infer the required covariance. The extended normal equations call for all covariances between the various data events (DEV) retained. A DEV is defined as a specific geometric configuration of data locations **and** data values. The major problem lies in the inference of these DEV-covariances since the shear number of possible DEV becomes excessively high. For example consider a central value to be estimated from 14 neighbouring data locations on a square grid. Assume that there are only 10 possible class values for anyone of these 15 values. The possible number of DEV is then 10^{15} ! Probably, not all of 10^{15} DEV's need be inferred to get the

essence of a training image. The question then arises which DEV's should be retained. The answer, unfortunately, can not be found in the extended normal equations. Also this method calls for binning of data. When a phenomenon is continuous, binning or indicator transforms is not a problem for central values. However, if continuous extreme values are binned, tail extrapolation models have to introduced (see Deutsch and Journel, 1997, p. 132) and might have an important influence on the final result.

In literature one can find various other methods for incorporating multi-point statistics

- Incorporating multi-point statistics using simulated annealing: the technique tolerates some amount of inconsistency (non-positive definite situation), however as the number of classes becomes high and the order of the multi-point statistics raises, CPU and RAM quickly become a problem.

- Iterative methods for conditional simulation (Srivastava, 1992 and Wang, 1996). CPU is less of a problem but RAM is, since all conditional distributions (which are multi-point statistics) scanned from the training image must be stored. There is no explicit modeling of these multi-point statistics, so no data-expansion is made. Hence, the training image may have to be very large or there are many of them to allow for enough DEV's to be recognized.

- Markov Random Fields (Besag, 1974 and Tjelmeland, 1996). In this case one explicitely states a model for the multivariate distribution of the values at all spatial locations, typically a non-Gaussian model. The method is internally consistent and not too CPU intensive. Yet, the model contains hundreds of parameters which are difficult to interpret and hence difficult to fit using actual data or a training image. Therefore one typically restricts the model calibration to a few parameters which describe the essence of the training image, the other parameters being set to arbitrary values. Since the interpretation of parameters is difficult, it is difficult to find out from actual data which parameters should be inferred and how.

2 Methodology outline

This paper aims at demonstrating that neural networks applied to stochastic simulation can overcome many of the previously mentioned problems. A neural network is in essence a multi-parameter non-linear regression model that defines a general non-linear mapping from any space into any other space both of arbitrary dimension. In our case, the neural network is trained to determine conditional distributions from a training image, then it proceeds with simulation using these distributions. Conditional distributions relate the value at any location to neighbouring data values within a template centred at that location, see Figure 1. The neighbourhood template can be

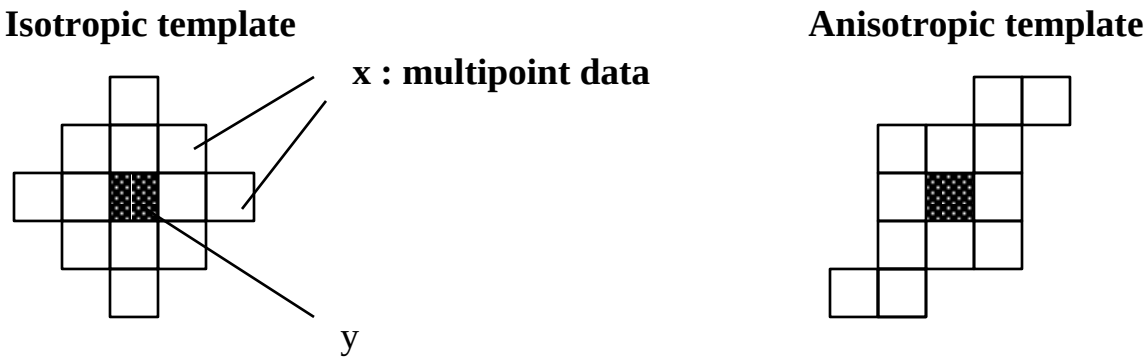

Figure 1: *Defining a neighbourhood or scanning template*

anisotropic (its anisotropy could be associated to variogram ranges) and there could be different templates of different sizes if a multiple grid approach is used (Goodman and Sokal, 1989; Tran, 1994). The limitation to a finite neighbourhood is similar to the screening effect assumption; it is commonly used in kriging. Within such data neighbourhoods, the neural network is learned to determine values y at a any given location given neighbouring data values $\mathbf{x}$.

More precisely, the neural network is trained to model the local conditional probability density function (cpdf)

$$f(y|\mathbf{x})\,\mathrm{d}y = \Pr\{y < Y < y + \mathrm{d}y|\mathbf{x}\}$$

or its integral, the conditional cumulative distribution function (ccdf). The neural network maps the multi-point input (the training image) into a one dimensional output, the previous conditional distribution. In the training phase, we input the conditioning data y and $\mathbf{x}$, that is the training image, and the net determines the model $f(y|\mathbf{x})$. In the generalization phase we input $\mathbf{x}$ and draw values of y from these cpdf's, see Figure 2. The network serves as a non-linear connection between input and output.

In the training phase the network is learned by scanning a training image using a certain template, retrieving multi-point information in the form of conditional distributions. This approach is somewhat different from the classical training phase of a neural network, where the network output is compared with true or target output values, the learning then proceeds by back-propagating the mean square difference between network output and target output through adjusting the network parameters. The output in our case are probability distributions which are not available beforehand. We will show that an error function can still be defined to fine tune the training.

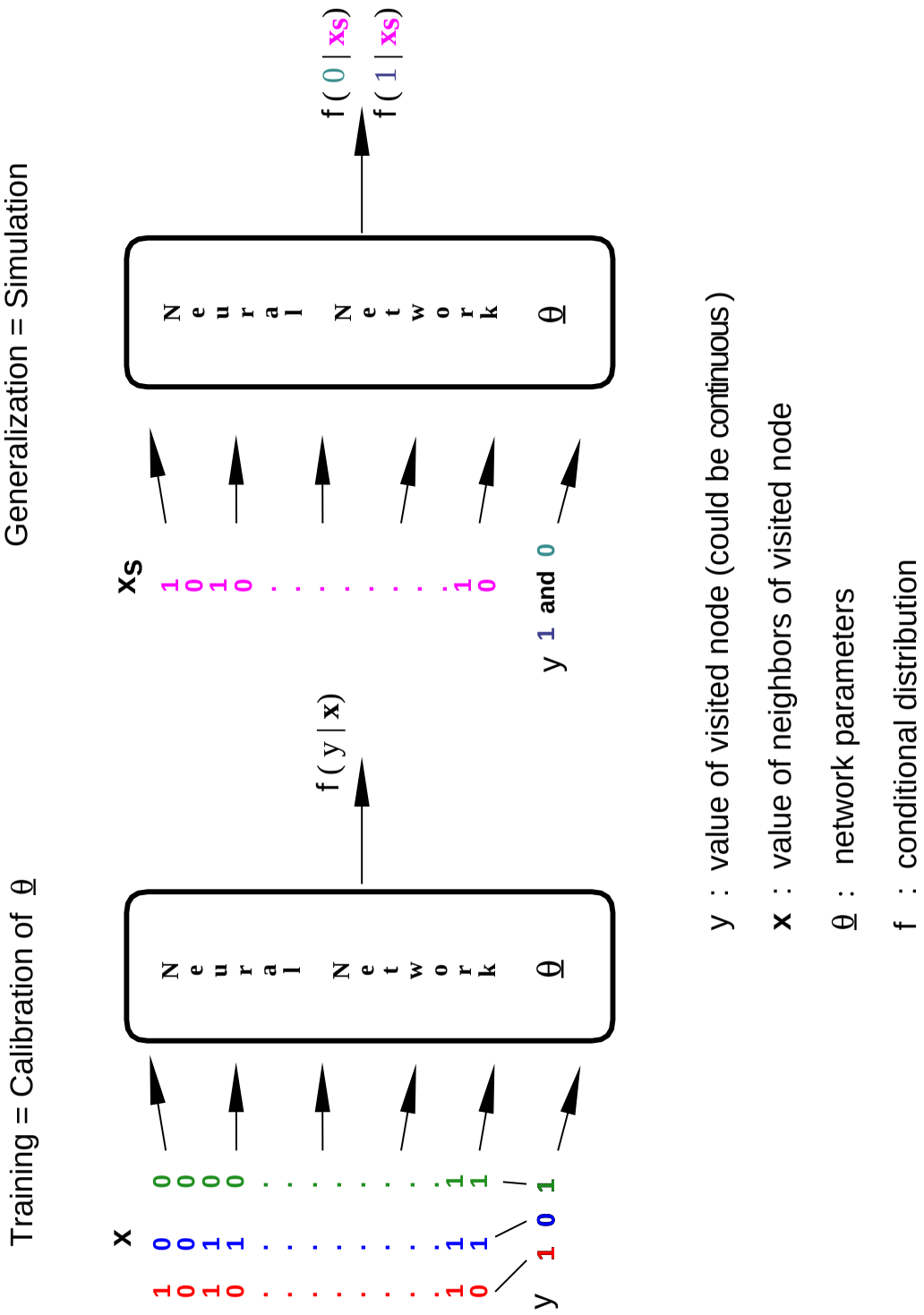

Figure 2: *General network configuration for stochastic simulation for a binary case.*

In the generalization phase, one visits each node i of the N nodes of the simulation grid (except the edges) and draws a simulated value y_i conditioned to the data $\mathbf{x}_i$ found within the prescribed template. The term generalization refers to the fact that, in this phase, the neural network is applied to assess distributions conditioned to point information different from those found in the training data. Generalization thus entails data expansion.

The generalization phase consists of generating stochastic simulations (images). The problem is that, initially, we do not have all the neighbouring values of y; typically only a few conditioning data are available. The idea is to initialize the whole field with random values (or according to some prior marginal distribution) and then update each of these values using the network generated cpdf's $f(y|\mathbf{x})$. A random path is defined that visits each node or location, and one keeps cycling through the **whole** field until some convergence is obtained. We will define later criteria for such convergence. This procedure is tantamount to setting up a Markov Chain (Metropolis-Hastings sampler) sampling a stationary distribution. All original conditioning data values remain frozen. The general outline of our method is as follows:

1. Training (=Modeling the image)
   (a) Define a neighbourhood template with which the training image is scanned
   (b) Train the neural net with the scanned data, extracting only "essential" features of the image. The results are a model for the cpdf $f(y|\mathbf{x})$ known for all values of y and $\mathbf{x}$.
2. Generalization (=Simulating an image)
   (a) Define a random path visiting all non-frozen nodes
   (b) Initialize the simulation by freezing the original data values at their locations and filling-in the remainder nodes with values drawn from the global target histogram (prior cdf)
   (c) Start loop : visit all nodes, at each node do
      - Draw a new value for that node (either at random or according to the prior cdf)
      - Retrieve from the neural net the cpdf of the old and new value
      - Change old value into new value according to some probabilistic criterion (e.g. Metropolis criterion).
   (d) Stop loop when convergence is reached, if not goto (c)
   (e) If another image is required start all over from (a)

First, we will develop the general methodology for stochastic simulation using local conditional distributions, and a Metropolis-Hastings sampler. Next we elaborate on how the neural network models these conditional distributions[1].

3 Conditional simulation with Markov Chains

3.1 Sequential simulation

Sequential conditional simulation methods rely on the conditional distribution of the value at each node to be simulated given the original conditioning data and previously simulated values. Points on a regular grid are visited along some random path and their values are drawn. In the case of Gaussian sequential simulation, the parameters of the normal conditional distributions are identified to the kriging means and variances. The dimension of the kriging system rapidly becomes unwieldy if all previously simulated values are retained. An approximation is obtained by retaining only those conditioning values that are most "consequential". In most cases only the "closest" data are retained, where closest is defined with respect to some variogram distance measure. Retaining the closest data amounts to assume that the RF features some Markov behaviour, i.e. the closest data screen the information provided by further away data.

In the case of Direct Sequential Simulation, only the means and variances of the conditional distributions (not necessarily Gaussian) are identified to the kriging results (Xu and Journel, 1994).

3.2 Metropolis sampling

Suppose now that, through some algorithm, we have obtained the probability distribution of the attribute value at any specific location to be simulated conditional to L neighbouring data values; that distribution being generally non-Gaussian. We also assume a Markov property, i.e. the conditioning is restricted to the L neighbours. We could then implement a sequential simulation method with such conditional distributions.

A powerful yet simple method for enforcing such conditional distributions is the Metropolis-sampler. Denote the set of N pixel locations in the simulation grid by $S = \{1 \dots N\}$. The simulated values over that grid is then the array $\mathbf{y} = \{y_1 \dots y_N\}$. Denote the neighbourhood data of a grid node i by a vector $\mathbf{x}_i$, i.e. the values closest to y_i in some sense. Denote by $f(\mathbf{y})$ the joint multivariate density of all N pixel values.

Conditional simulation with a Metropolis sampler calls for iterative visits of each

[1] Refer to section 14 for the list of notations used

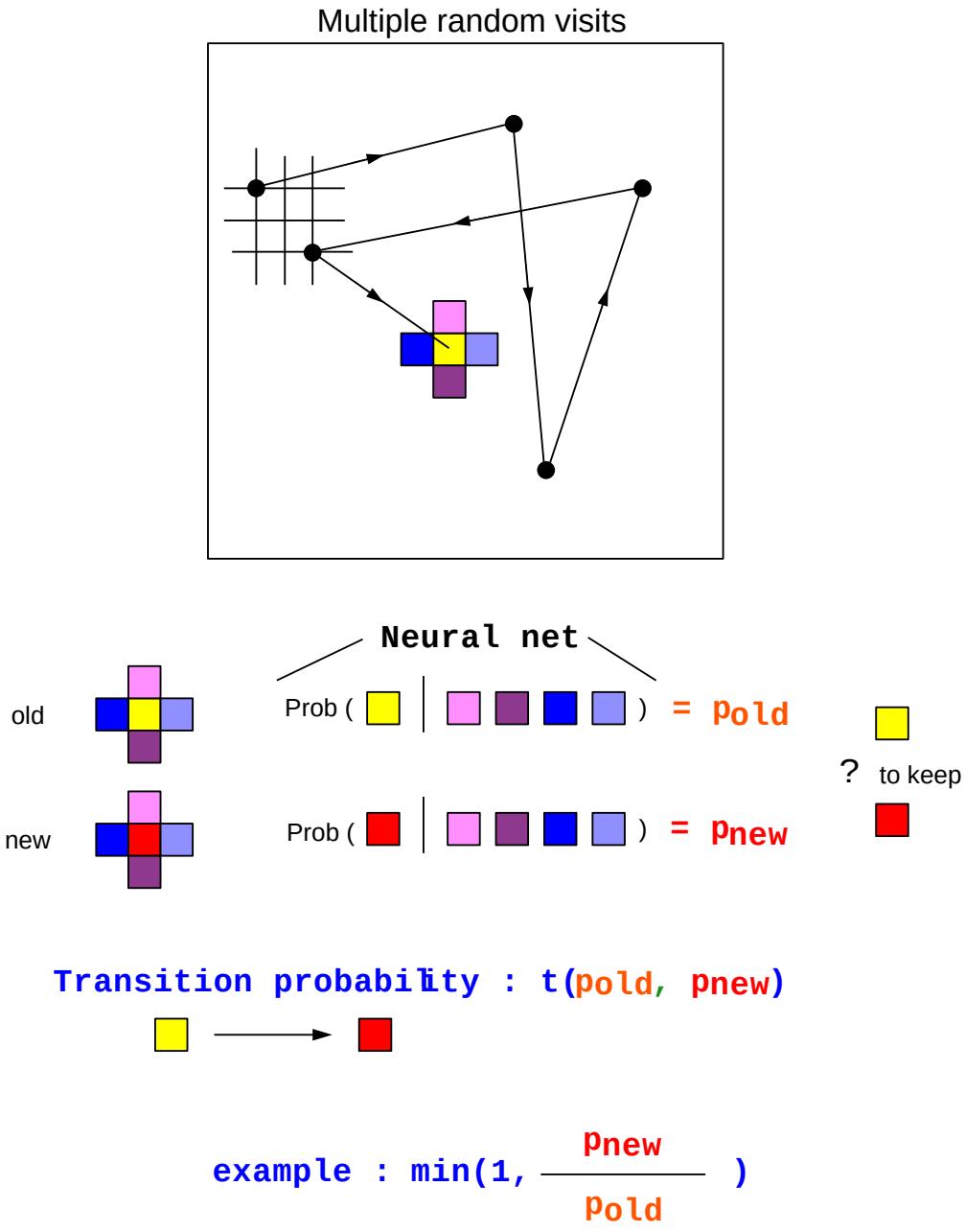

Figure 3: *Metropolis sampling : the field is initially random. Each node is visited and updated according to the ratio* p_{new}/p_{old}.

grid node (Figure 3). The initial grid is filled with random values. At each successive visit of a grid location i with present value y_i, a new value y_i^* is randomly drawn from the set of possible values and the original y_i is replaced by y_i^* with probability

$$\min\left\{1, \frac{f(\mathbf{y}^*)}{f(\mathbf{y})}\right\}$$

where $f(\mathbf{y}^*)$ is the joint distribution of the new pixel grid, where $\mathbf{y}^*$ differs from $\mathbf{y}$ only by the value y_i^* at the current grid location i. Using the conditional probability paradigm this can be rewritten as

$$\min\left\{1, \frac{f(y_i^*|\text{ all } y_j, j \neq i)}{f(y_i|\text{ all } y_j, j \neq i)}\right\}$$

If only conditioning on local neighbouring values is considered, the transition probability from $\mathbf{y}$ to $\mathbf{y}^*$ reduces further to

$$P(\mathbf{y} \to \mathbf{y}^*) = \min\left\{1, \frac{f(y_i^*|\mathbf{x}_i)}{f(y_i|\mathbf{x}_i)}\right\} \tag{1}$$

The local conditional distributions $f(y_i^*|\mathbf{x}_i)$ are required, they were determined by the neural net in the training phase. The original conditioning data always remain frozen at their locations.

The Metropolis sampler thus amounts to a Markov chain with known transition probability matrix $P(\mathbf{y} \to \mathbf{y}^*)$ defined by the expression (1). P is zero whenever $\mathbf{y}$ differs *by* more than one pixel from $\mathbf{y}^*$.

The stationary distribution of the Markov chain is the joint distribution $f(\mathbf{y})$. This Markov chain defines the transition of an image into a new image after one visit of a *single* grid node. Another Markov chain is defined by considering the transition of one image into a new image after *all* grid nodes have been visited, see Figure 4. As the image is initially random, it takes many iterations of the Markov chain before it starts sampling the stationary distribution $f(\mathbf{y})$. This period is called the burn-in period, see Figure 4. We can use the properties of Markov chains to get independent samples of $f(\mathbf{y})$. In a later section, we will discuss the mathematical representation of this joint distribution $f(\mathbf{y})$ and how it is uniquely defined from the knowledge of the local conditional distributions $f(y_i|\mathbf{x}_i)$. It now remains to see how the neural net actually determines these local conditional distributions.

3.3 Honouring global statistics with the Metropolis-Hastings sampler

Many geostatistical simulation methods (see Deutsch and Journel, 1997) honour global statistics such as the histogram and a global variogram. The Metropolis-Hastings sampler, a more general form of the Metropolis sampler leaves the flexibility to explicitely

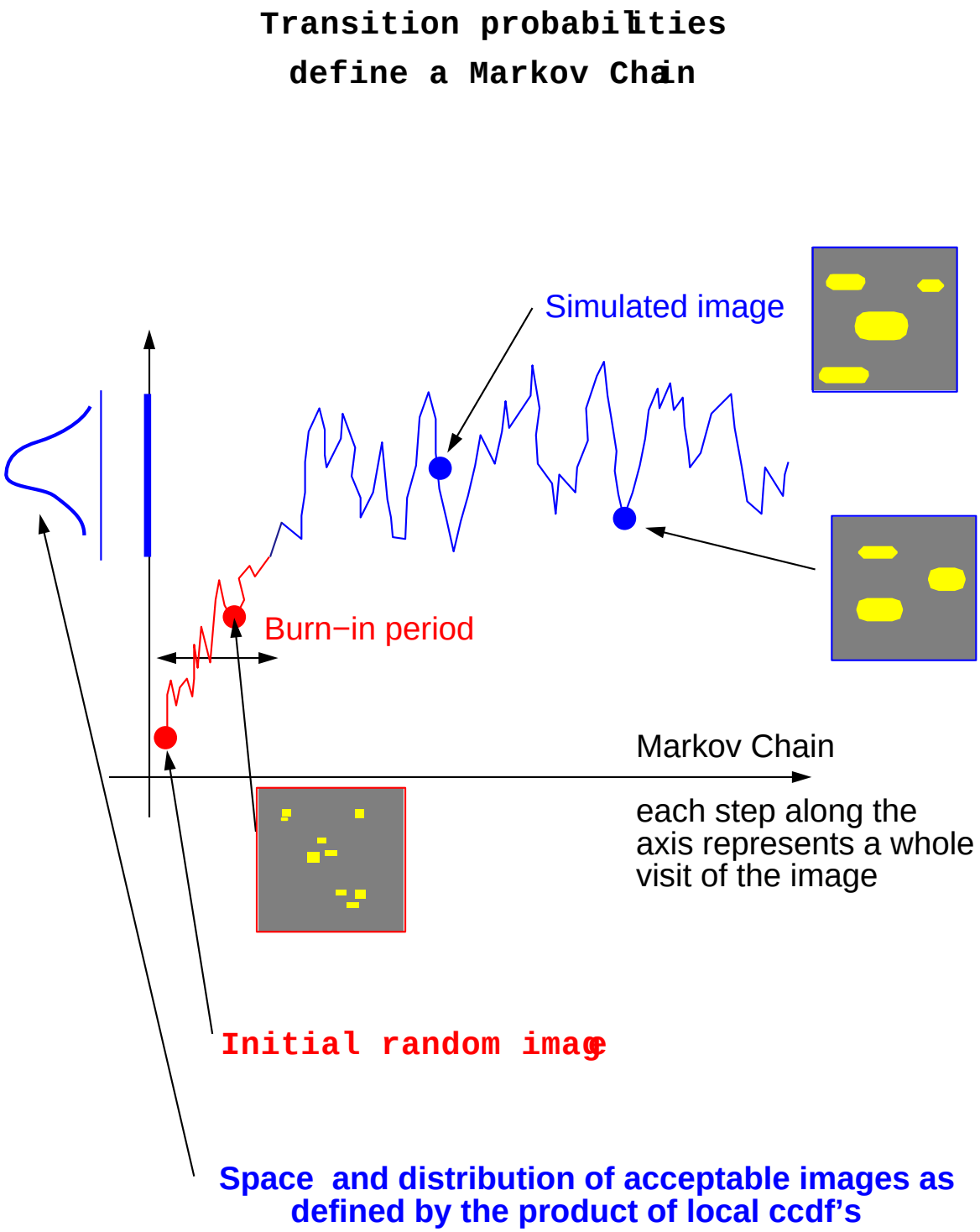

Figure 4: *Markov Chain Monte Carlo: The sampling of images is here depicted as a 1-dimensional Markov Chain. The vertical axis represents the N-dimensional space of all possible images.*

honour a global histogram. The general form of the Metropolis-Hastings probability is written as, see Hastings, 1970; Ripley, 1987, p. 115:

$$P(\mathbf{y} \to \mathbf{y}^*) = \min\left\{1, \frac{f(y_i^*|\mathbf{x}_i)t(y_i^* \to y_i)}{f(y_i|\mathbf{x}_i)t(y_i \to y_i^*)}\right\}$$

where we have now introduced a new but completely arbitrary transition[2] probability t. t describes a transition of one pixel value into another pixel value that is independent of the neighbouring values in $\mathbf{x}_i$. We find back the original Metropolis criterion (1) when t is symmetric, i.e. $t(y_i^* \to y_i) = t(y_i \to y_i^*)$. To honour a specific histogram we will define an objective function that measures the difference between the target histogram and the histogram at the current iteration in the Markov chain. If the global histogram is described by C quantiles q_c then the objective function becomes

$$O(\mathbf{y}) = \sum_{c=1}^{C}(q_c - q_c^{(s)}(\mathbf{y}))^2$$

where $q_c^{(s)}$ are quantiles of the global histogram at the current simulation step, depending on the current set of pixels $\mathbf{y}$. The objective function is then used to define the ratio of transition probabilities as follows

$$\frac{t(y_i^* \to y_i)}{t(y_i \to y_i^*)} = \exp(-(O(\mathbf{y}^*) - O(\mathbf{y}))/k_B)$$

where k_B is a positive constant defined such that the previous ratio is of the same order of magnitude than $f(y_i^*|\mathbf{x}_i)/f(y_i|\mathbf{x}_i)$. The Metropolis-Hastings sampling criterion then becomes

$$P(\mathbf{y} \to \mathbf{y}^*) = \min\left\{1, \frac{f(y_i^*|\mathbf{x}_i)}{f(y_i|\mathbf{x}_i)}\exp(-(O(\mathbf{y}^*) - O(\mathbf{y})/k_B)\right\} \quad (2)$$

Note that the objective function $O(\mathbf{y})$ is easily updatable after each iteration, since only one grid node value changes, hence the honouring of a target histogram can be obtained at virtually no extra cost. Experience has shown that the easiest histogram to honour is the uniform histogram. It is therefore advisable to make a uniform score transform of the training image.

3.4 Multiple grid approach

Many images have large scale features that cannot be represented with a template of limited size. Large scale trends may be present or some structures such as channels may run over the entire image. In such case we can define multiple grids, see Figure 5: simulation starts on the coarser grid allowing to capture the large scale structure, then proceeds onto the finer grids to depict the smaller scale details. The values simulated

[2]in all rigour: it must form an irreducible Markov chain on the same space as the probability P

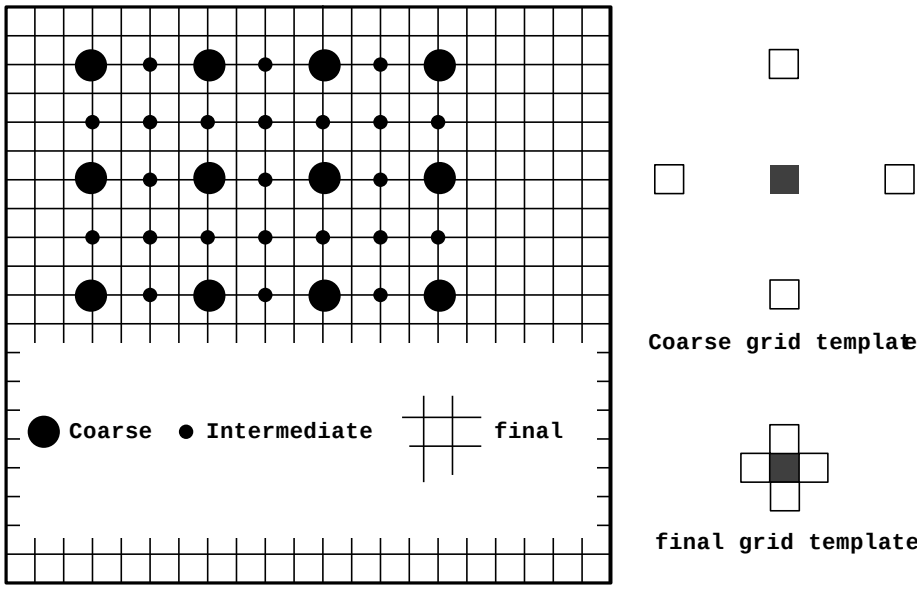

Figure 5: *Using multiple grids in stochastic simulation.*

on the coarse grid are frozen as conditioning data in the subsequent finer grids. In our approach, we need a set of cpdf's for each nested grid, which means that we have to train different neural nets for different grids, with scanned data using templates of different sizes, see Figure 5. It is advisable to use multiple grids when the number of simulation nodes exceeds vastly the original sample size (number of conditioning data).

4 A network architecture for building local conditional distributions

4.1 A one layer network

The neural network is used to model all local conditional distributions needed in the generalization phase of the conditional simulation method. Markov chain algorithms ask for the conditional distribution $f(y_i|\mathbf{x}_i)$ at the visited grid location i given any current realization $\mathbf{x}_i$ of the neighbouring data. The neural network utilizes a general non-Gaussian cpdf model, which is "fitted" or "trained" on the training set. There is no close form mathematical representation for the corresponding multivariate distribution model, the net delivers the table of all required conditional distributions, once their parameters are established. Before describing the parameter fitting algorithm, we must define the network architecture. We first consider a network with one hidden layer (Bishop, 1995).

Consider again gridded values which take continuous outcomes, the y's in Figure 2. We assume a Markov screening to hold in the 2- or 3-dimensional space. *Target value* is typical neural network language for the value y_i for which the conditional distribution $f(y_i|\mathbf{x}_i)$ has to be derived.

A neural network consists of neural nodes which contain mathematical operations and branches which direct the results of these operations to other neural nodes[3]. Most common operations consist of summation, identity functions (input=output) and non-linear tranforms using some sigmoidal functions. The neural nodes are arranged in layers which are connected. The simplest neural network is a network that connects the input to a middle layer and then to the output layer. The middle layer is called the *hidden layer*. A single hidden layer feed forward neural network consists therefore of three layers. If the hidden layer has K internal nodes the ccdf is modeled by mapping the input $y|\mathbf{x}$ into the output, see Figure 6:

$$F(y|\mathbf{x}) = \sum_{k=1}^{K} o_k \, T\left(\sum_{l=1}^{L} \widetilde{w}_{k,l} x_l + v_k y\right) \tag{3}$$

where o_k are K output weights and $\widetilde{w}_{k,j}$ are $K \times L$ input weights linking the hidden layer, v_k are weights between the input target value y and the hidden layer. K is the number of neural nodes in the hidden layer and L is the number of data values x_l in the template $\mathbf{x}$. T is a non-linear transfer function, typically a sigmoid with some known mathematical expression. In our case, there is a further restriction on the function T in that it should be a licit cdf. Since a linear combination of cdf's also is a cdf, if $o_k \geq 0$, $\sum_{k=1}^{K} o_k = 1$, the resulting network output (3) is a legitimate cdf.

[3]to avoid confusion with nodes of the simulation grid we will use the terms *neural nodes* and *grid nodes* to differentiate wherever necessary

For convenience, without changing its architecture, we can rewrite the network as

$$F(y|\mathbf{x}) = \sum_{k=1}^{K} o_k \, T\left(v_k \left(y - \sum_{l=1}^{L} w_{k,l} x_l\right)\right), \quad with \; \widetilde{w}_{k,j} = -v_k w_{k,j} \tag{4}$$

The network architecture is drawn in Figure 6.

From expression (4), it is clear that the ccdf appears as a linear combination of "kernel" cdf's. One can calculate the mean of this ccdf, i.e. the conditional mean as

$$\begin{aligned} \mathrm{E}[Y|\mathbf{x}] &= \int_{-\infty}^{\infty} y \, \mathrm{d}F(y|\mathbf{x}) \\ &= \sum_{k=1}^{K} o_k \int_{-\infty}^{\infty} y \, \mathrm{d}T\left(v_k \left(y - \sum_{l=1}^{L} w_{k,l} x_l\right)\right) \end{aligned}$$

if $T(y)$ is a cdf with zero mean and unit variance then

$$\begin{aligned} \int_{-\infty}^{\infty} y \, \mathrm{d}T\left(v_k \left(y - \sum_{l=1}^{L} w_{k,l} x_l\right)\right) &= \sum_{l=1}^{L} w_{k,l} x_l \\ &= m_k(\mathbf{x}) \end{aligned}$$

and the conditional mean becomes

$$\mathrm{E}[Y|\mathbf{x}] = \sum_{k=1}^{K} o_k m_k(\mathbf{x}) = m(\mathbf{x}) \tag{5}$$

Similarly, the conditional variance is derived as:

$$\begin{aligned} \mathrm{Var}[Y|\mathbf{x}] &= \int_{-\infty}^{\infty} (y - m(\mathbf{x}))^2 \, \mathrm{d}F(y|\mathbf{x}) \\ &= \sum_{k=1}^{K} o_k \int_{-\infty}^{\infty} (y - m(\mathbf{x}))^2 \, \mathrm{d}T\left(v_k \left(y - \sum_{l=1}^{L} w_{k,l} x_l\right)\right) \\ &= \sum_{k=1}^{K} o_k \left(\frac{1}{v_k^2} + (m_k(\mathbf{x}) - m(\mathbf{x}))^2\right) \end{aligned} \tag{6}$$

which is a classical result from variance analysis.

In appendix A, we derive all other moments of the ccdf when T is a logistic distribution. Thus the one hidden layer network architecture results in a ccdf which is a mixture of K cdf kernels of the same type (i.e. a cdf T with zero mean and unit variance) but centred at different values $m_k(\mathbf{x})$. For this one layer network the parameter vector is: $\underline{\theta} = \{o_k, v_k, w_{k,l}, \; k = 1, \ldots K, \, l = 1, \ldots, L\}$. The conditional variance (6) is heteroscedastic in that it is dependent of the data values $\mathbf{x}$. The conditional mean (5) is a linear function of the data $\mathbf{x}$ in that each of the K kernel means $m_k(\mathbf{x})$ is linear in $\mathbf{x}$. This linear relation of $m(\mathbf{x})$ with $\mathbf{x}$ may be too restrictive to model a general image, hence a second hidden layer should be added to obtain a more general non-linear approximator to the conditional distribution function. Correspondingly, the computational effort will increase considerably and, as will be shown, the learning process becomes more difficult.

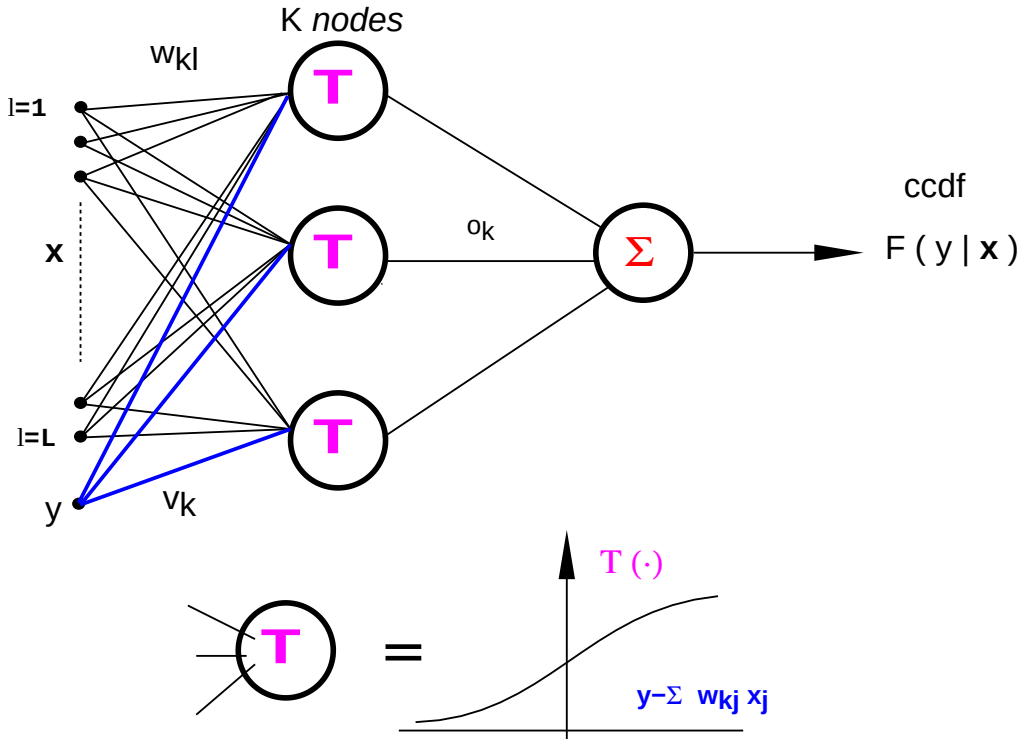

Figure 6: *Single hidden layer network.*

4.2 A two layer network

The fact that the conditional mean in Eq. (5) is linearly dependent on the conditioning data values is a restriction to the flexible approximation capabilities of the neural network. To remove this restriction, we will add a second hidden layer of neural nodes (Bishop, 1995). A first hidden layer containing K_1 nodes is inserted between the input and the original hidden layer with K_2, see Figure 7. A preliminary non-linear transform S of the data $\mathbf{x}$ is applied prior to applying the T transform. In neural net jargon, the S transform constitutes the first layer and the cdf-type transform is then called the second hidden layer, see Figure 7. Thus, the first hidden layer containing K_1 neural nodes is inserted between the input and the original hidden layer (now called second layer) with K_2 neural nodes. This results in the following conditional distribution:

$$F(y|\mathbf{x}) = \sum_{k_2=1}^{K_2} o_{k_2}\, T\left(v_{k_2}\left(y - \sum_{k_1=1}^{K_1} w_{k_2,k_1}\, S\left(\sum_{l=1}^{L} u_{k_1,l} x_l\right)\right)\right) \tag{7}$$

where $u_{k_1,l}$ are weights associated to the first layer and S is any non-decreasing function. Based on the same notation for the one layer neural network, we can simplify Eq. (7).

$$F(y|\mathbf{x}) = \sum_{k_2=1}^{K_2} o_{k_2}\, T\left(v_{k_2}\left(y - m_{k_2}(\mathbf{x})\right)\right)$$

where we replace the linear regression $m_k(\mathbf{x})$ of expression (5) by a combination of K_1 **non-linear** functions of type S

$$m_{k_2}(\mathbf{x}) = \sum_{k_1=1}^{K_1} w_{k_2,k_1}\, S(m_{k_1}(\mathbf{x})) = \sum_{k_1=1}^{K_1} w_{k_2,k_1}\, S\left(\sum_{l=1}^{L} u_{k_1,l} x_l\right) \tag{8}$$

The following comments can be made about this two layer network

- For $F(y|\mathbf{x})$ to be a permissible ccdf it suffices that $\sum_{k_2=1}^{K_2} o_{k_2} = 1$, $o_{k_2} \geq 0$ and T is a permissible cdf.
- S is valued within the probability range of the cdf T, which in this case is the whole real line $[-\infty, +\infty]$.
- The two layer network appears as a mixture of K_2 kernel cdf's of type T, each with zero mean and unit variance, and centred on

$$m_{k_2}(\mathbf{x}) = m_{k_2}(\mathbf{x}, \mathbf{w}, \mathbf{u}) = \sum_{k_1=1}^{K_1} w_{k_2,k_1}\, S\left(\sum_{l=1}^{L} u_{k_1,l} x_l\right)$$

  Note that $m_{k_2}(\mathbf{x})$ is non-linear in $\mathbf{x}$. Finally:

$$\begin{aligned} \mathrm{E}[Y|\mathbf{x}] &= \sum_{k_2=1}^{K_2} o_{k_2} m_{k_2}(\mathbf{x}, \mathbf{w}, \mathbf{u}) = m(\mathbf{x}) \\ \mathrm{Var}[Y|\mathbf{x}] &= \sum_{k_2=1}^{K_2} o_{k_2} \left(\frac{1}{v_{k_2}^2} + (m_{k_2}(\mathbf{x}, \mathbf{w}, \mathbf{u}) - m(\mathbf{x}))^2\right) \end{aligned}$$

  It appears that the sole function of the added first layer of K_1 nodes, is to make the regression $\mathrm{E}[Y|\mathbf{x}]$ non-linear.

4.3 Understanding the network parameters

It is essential for the implementation of a fast learning neural net to have a better insight into the approximation capabilities of the network and the meaning of each network parameter. We proceed as follows. From the cumulative conditional distribution in expression 7, the conditional density is calculated as the derivative

$$f(y|\mathbf{x}) = \sum_{k_2=1}^{K_2} o_{k_2}(v_{k_2} T'(v_{k_2}(y - m_{k_2}(\mathbf{x})))) \tag{9}$$

where T' is the derivative of T with respect to its argument. If one introduces the function h as

$$h(y - m_{k_2}(\mathbf{x}), v_{k_2}) = v_{k_2} T'(v_{k_2}(y - m_{k_2}(\mathbf{x}))$$

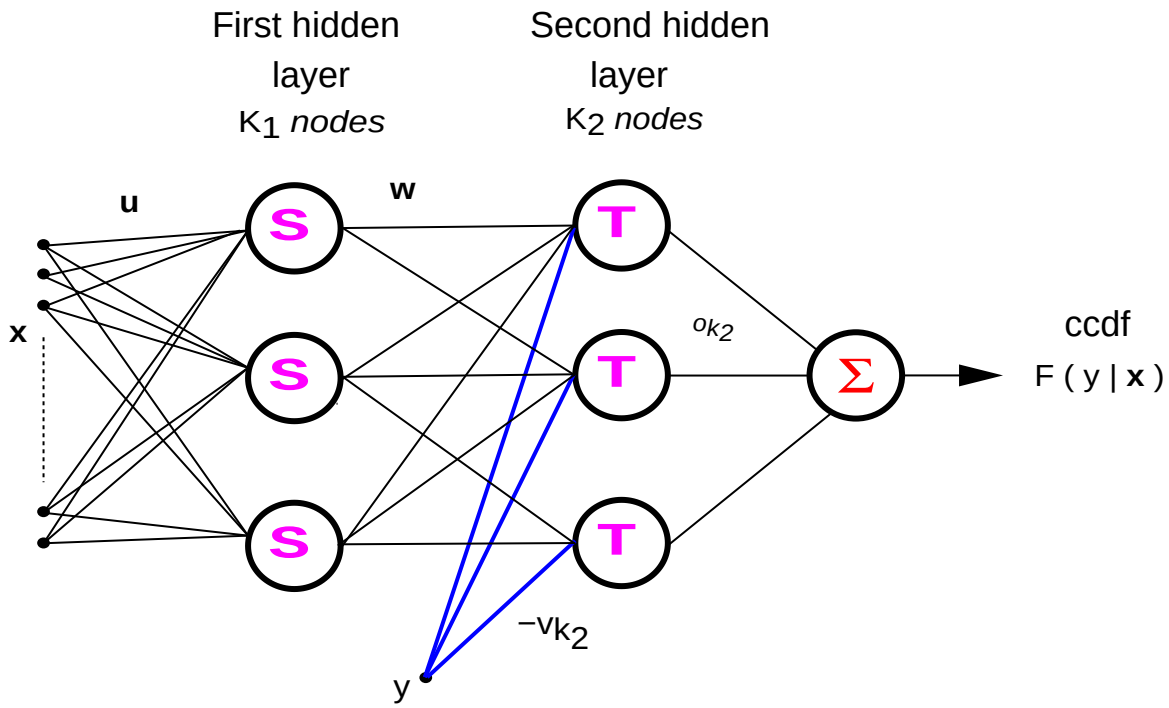

Figure 7: *Adding a second layer.*

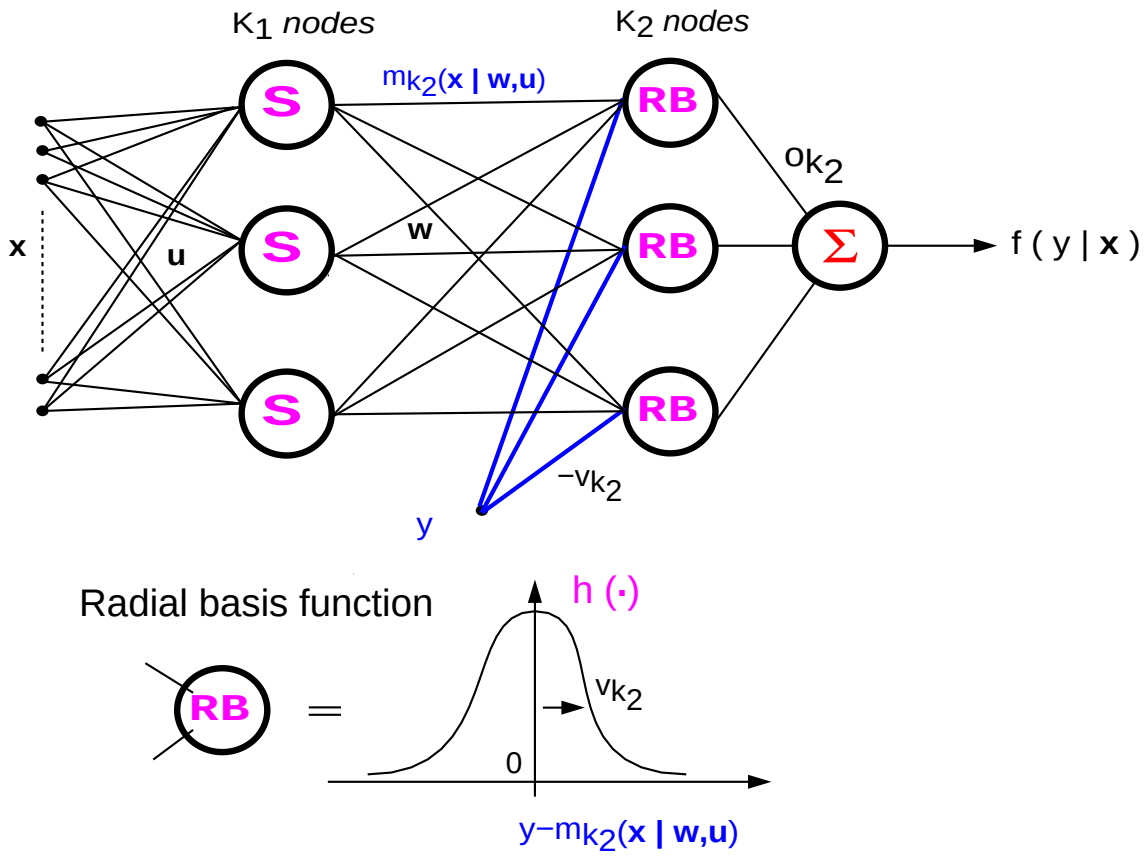

Figure 8: *Neural network that outputs a density instead of a cdf.*

then

$$f(y|\mathbf{x}) = \sum_{k_2=1}^{K_2} o_{k_2}(h(y - m_{k_2}(\mathbf{x}), v_{k_2})) \tag{10}$$

The function h is a bell-shaped pdf if T is a sigmoidal cdf. For example, if T is Gaussian cdf, h is a Gaussian pdf.

The parameter v_{k_2} is a shape parameter controlling the width of the bell. Expression (10) can be interpreted as a decomposition of the output density into various "kernel densities" or "nodal densities" h. The neural network outputting the conditional density is drawn in Figure 8.

The function h is seen as the conditional density of the target y given the conditioning data $\mathbf{x}$ *and* node k_2 in the neural network

$$h(y - m_{k_2}(\mathbf{x}, \mathbf{w}, \mathbf{u}), v_{k_2}) = h(y|\mathbf{x}, \mathbf{k}_2) \tag{11}$$

Since the set $\{o_{k_2}\}$ is a valid probability mass, the nodal densities $h(y|\mathbf{x}, \mathbf{k}_2)$ are com-

pounded with the weights o_{k_2} into the final cpdf:

$$f(y|\mathbf{x}) = \sum_{k_2=1}^{K_2} o_{k_2} h(y|\mathbf{x}, \mathbf{k}_2) \tag{12}$$

$\mathbf{k}_2$ are all the network parameters that define the node k_2, it thus includes all the parameters of $\underline{\theta}$ except the mixing parameters o_{k_2}. In statistical jargon, expression (12) is termed a Gaussian mixture. The parameters o_{k_2} thus appear as prior probabilities or relative contributions of each of the K_2 nodes of the second hidden layer. The parameters v_{k_2} are scale parameters controlling the width of the radial basis functions h and the parameters $\mathbf{w}$, $\mathbf{u}$ define the non-linear connection between the means of these radial basis functions and the input data $\mathbf{x}$.

5 Determining the parameters of the neural net

To determine the values of the various parameters $\underline{\theta} = \{o_{k_2},\ v_{k_2},\ \mathbf{w},\ \mathbf{u}, k_2 = 1, \ldots, K_2\}$ the network must be learned from training data. The training data can be obtained by scanning a training image for all available combinations $\{y, \mathbf{x}\}$. Edge effects are ignored. The classical method of training neural nets, where some mean square deviation of present versus desired output is iteratively mininimized, cannot be used since the output here is a conditional probability which is not known. However, we can use the principle of maximum likelihood to obtain the optimal parameters $\underline{\theta}$. In the maximum likelihood paradigm, one uses the joint density of all grid node values of the training image given the network parameters as the conditional likelihood for the training image given the model. This joint density depends on the network parameters and can be maximized, yielding a maximum likelihood estimate for $\underline{\theta}$.

Application of the likelihood principle can intuitively be justified as follows and is symbolically depicted in Figure 9. We assume that our training image is one sample of an ensemble of images that is generated by a theoretical model defined by the set of parameters $\underline{\theta}$. This assumption need not be true, yet it is the basis of many methods of statistical inference. Such a model can be a Metropolis-Hastings sampler using a neural network to model the conditional distribution. The question now arises about where the training image should fit into the distribution of images defined by the model, see Figure 9. It is argued that the training image should be situated somewhere in the middle of that distribution since among all images it is the most likely to occur, hence the maximum likelihood principle. Note that in the likelihood principle we do not use the actual conditioning data, we do use though the training image.

Suppose, we have a training image consisting of the set of pixel values $\mathbf{y}^{(o)} = \{y_i^{(o)}, i = 1, \ldots, N\}$. Maximizing the likelihood of drawing the specific training image $\mathbf{y}^{(o)}$ requires knowledge of the joint density $f(\mathbf{y}|\underline{\theta})$ for any set $\mathbf{y}$ of all N pixel values given any choice for the parameters $\underline{\theta}$. Although this joint density can in theory be

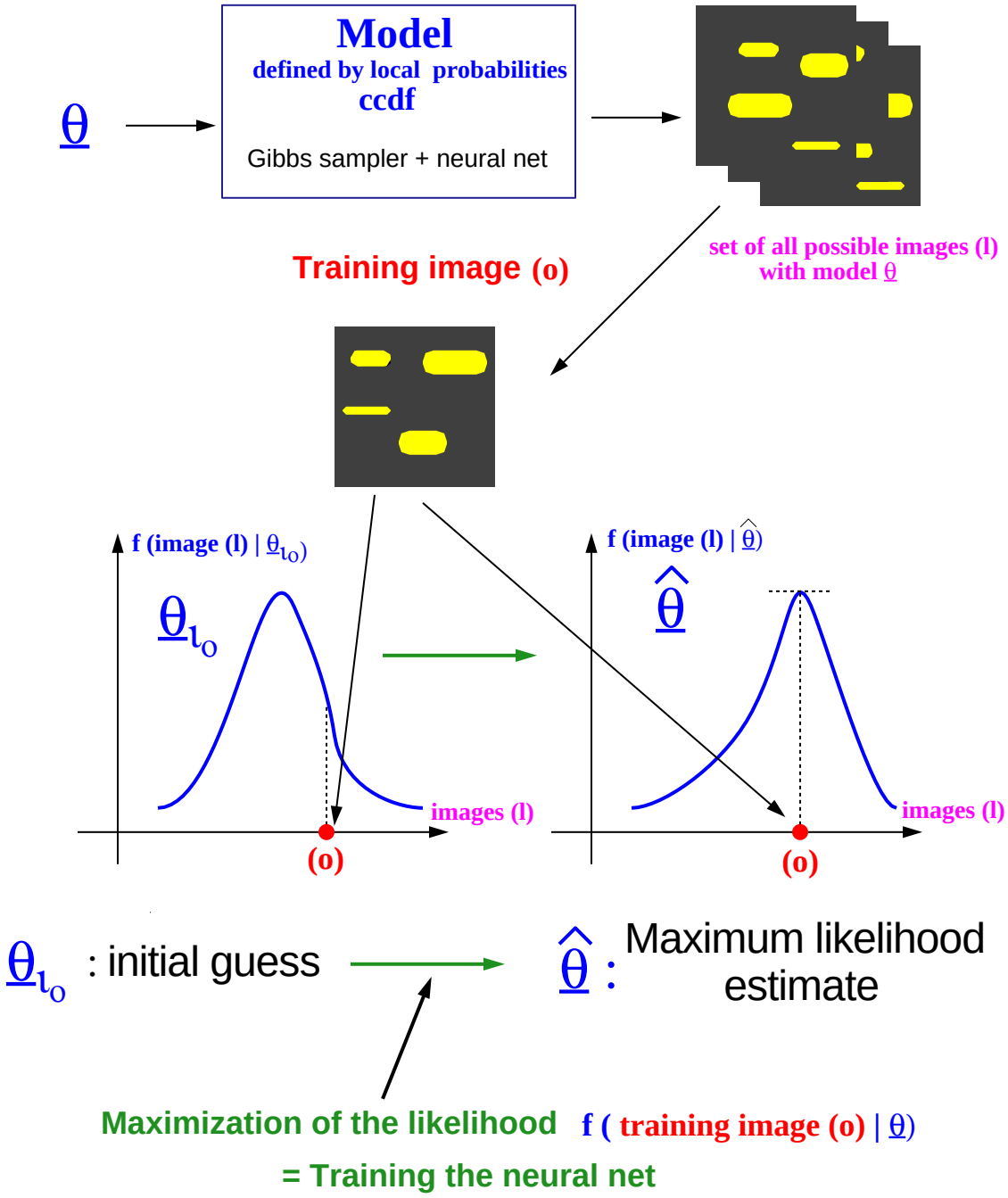

Figure 9: *Likelihood principle in spatial statistics.*

calculated from knowledge of the local cpdf's $f(y_i|\mathbf{x}_i, \underline{\theta})$, see appendix B, that calculation is not practical. Thus we need to compromise and replace the joint density by, for example, the product of all N known local cpdf's $f(y_i|\mathbf{x}_i, \underline{\theta})$. More precisely, the set of parameters $\widehat{\underline{\theta}}$ retained is that which maximizes the pseudo-likelihood defined as (see Besag, 1975; Gidas, 1993)

$$\ell(\underline{\theta}, \mathbf{D}) = \prod_{i=1}^{N} f(y_i^{(o)}|\underline{\theta}, \mathbf{x}_i^{(o)}) \tag{13}$$

Expression (13) is called the "likeliness" of the training image $\mathbf{D}$ constituted by the N observed pixel values $\{y_i^{(o)}, i = 1, \ldots, N\}$. It can be shown that in many cases, maximizing the likeliness yields a consistent[4] estimate of the network parameters. Actually, instead of maximizing the likeliness, we will minimize minus the log-likeliness, defined as the error function:

$$Er(\underline{\theta}, \mathbf{D}) = -\log(\ell(\underline{\theta}, \mathbf{D})) = -\sum_{i=1}^{N} \log(f(y_i^{(o)}|\underline{\theta},\ \mathbf{x}_i^{(o)})), \tag{14}$$

where Er stands for error function.

Many methods could be considered to minimize this error-function, such as steepest descent or conjugated gradient methods. However, these usually take a lot of iterations, rendering the training procedure slow and tedious. Also there is a risk to end up in a local minimum. In the next paragraph, we will present the EM-algorithm (Expectation-Maximization) as an alternative for optimization building on an interpretation of the network as a Gaussian mixture distributions.

It is important to note that the numbers K_1 and K_2 are fixed a priori, they are not determined through our optimization procedure. There exists methods available in the neural network literature to adapt the number of neural nodes during the training phase (so-called growing and pruning algorithms, see Bishop, 1995).

6 Fast learning with the EM algorithm

6.1 The EM-algorithm in general

The Expectation-Maximization algorithm (EM-algorithm) is a general iterative technique for maximum likelihood estimation, see Dempster et al. (1977) and for applications to neural networks see Jordan and Jacobs (1994). Each iteration consists of two steps: the Estimation (E) and Maximization (M) step. The M-step is a maximization

[4] Consistency should be understood here in the classical statistical sense, where as more realizations of the same model are available, the fitted model parameters converge in probability to the "true" model parameters.

of a likelihood function that is redefined by each E-step. The EM-algorithm is used for maximum likelihood estimation in cases where the likelihood in terms of joint density or conditional density is a difficult function to obtain and to maximize. To overcome this difficulty, one simplifies the likelihood by introducing additional variables, termed "hidden" (neural jargon) or "missing" (statistical jargon) variables. The observed data is then called the *incomplete data set* which is completed with the missing variables to form a *complete data set*.

Denote in general the observed random variables as $\mathbf{Z}_o$ which have the joint density function $f(\mathbf{z}_o, \underline{\theta})$. The unobserved or missing variables are denoted as $\mathbf{Z}_u$ with the joint density $f(\mathbf{z}_o, \mathbf{z}_u, \underline{\theta})$. The main result of the introduction of $\mathbf{Z}_u$ is that the latter joint density is a much simpler function than the original density $f(\mathbf{z}_o, \underline{\theta})$. That original density is then obtained through integration over all possible $\mathbf{z}_u$ values as

$$f(\mathbf{z}_o, \underline{\theta}) = \int_{all\ z_u} f(\mathbf{z}_o, \mathbf{z}_u, \underline{\theta})\, \mathrm{d}\mathbf{z}_u \tag{15}$$

Because of the integration (or summation), $f(\mathbf{z}_o, \underline{\theta})$ is indeed a difficult function to optimize in terms of $\underline{\theta}$. We will see hereafter how to sidestep this difficulty. In the following we will seek maximization of the log-likelihood, thus we will work on $\log f(\mathbf{z}_o, \underline{\theta})$

We will first define the two steps of the EM-algorithm and then proceed with a intuitive explanation. Consider the following steps

Make an initial guess for the parameters $\underline{\theta}$ and call it the current maximum $\underline{\theta}^{(p)}$

Iterate: for $p = 1, \ldots, n_{max}$ set $\underline{\alpha} = \underline{\theta}^{(p)}$ and do an

E-step: Calculate the expectation

$$\begin{aligned} Q(\underline{\theta}, \underline{\alpha}) &= \int_{all\ \mathbf{z}_u} \log f(\mathbf{z}_o, \mathbf{z}_u, \underline{\theta})\, f(\mathbf{z}_u | \mathbf{z}_o, \underline{\alpha})\, \mathrm{d}\mathbf{z}_u \\ &= \mathrm{E}[\log f(\mathbf{z}_o, \mathbf{Z}_u, \underline{\theta}) | \mathbf{z}_o, \underline{\alpha}] \end{aligned} \tag{16}$$

$Q(\underline{\theta}, \underline{\alpha})$ appears as a function of both sets of parameters $\underline{\theta}$ and $\underline{\alpha}$ seperately.

M-step: Perform maximization of

$$\underline{\theta}^{(p+1)} = \arg \max_{\underline{\theta}} Q(\underline{\theta}, \underline{\alpha})$$

That is find $\underline{\theta}$ which maximizes $Q(\underline{\theta}, \alpha)$.

Thus, at each E-step we redefine Q as a function of the current best estimate $\underline{\alpha} = \underline{\theta}^{(p)}$ and try to find a new set of parameters $\underline{\theta}$ better in the maximum likelihood sense. This better set $\underline{\theta}^{(p+1)}$ is found through the M-step. The next iteration $(p+1)$ is made by setting $\underline{\alpha} = \underline{\theta}^{(p+1)}$.

The missing variables $\mathbf{Z}_u$ are introduced to simplify the maximization of the likelihood, in this case $\log f(\mathbf{z}_o, \underline{\theta})$. Since there is actually no data on the missing variables, we have to define a probability model that relates the missing variables to the actual variables: this is the density $f(\mathbf{z}_u|\mathbf{z}_o, \underline{\alpha})$ used in (16). We will now show that an increase in the function $Q(\underline{\theta}, \underline{\alpha})$ also entails an increase in the likelihood $\log f(\mathbf{z}_o, \underline{\theta})$. Consider Bayes's relation

$$\log f(\mathbf{z}_o, \underline{\theta}) = \log \frac{f(\mathbf{z}_o, \mathbf{z}_u, \underline{\theta})}{f(\mathbf{z}_u|\mathbf{z}_o, \underline{\theta})} = \log f(\mathbf{z}_o, \mathbf{z}_u, \underline{\theta}) - \log f(\mathbf{z}_u|\mathbf{z}_o, \underline{\theta})$$

By taking the expectation with regard to $f(\mathbf{z}_u|\mathbf{z}_o, \underline{\alpha})$ of both left and right side of this relation, one finds

$$\begin{aligned} \log f(\mathbf{z}_o, \underline{\theta}) &= Q(\underline{\theta}, \underline{\alpha}) - \int_{all\ \mathbf{z}_u} \log f(\mathbf{z}_u|\mathbf{z}_o, \underline{\theta})\, f(\mathbf{z}_u|\mathbf{z}_o, \underline{\alpha})\, \mathrm{d}\mathbf{z}_u \\ &= Q(\underline{\theta}, \underline{\alpha}) - \mathrm{E}[\log f(\mathbf{Z}_u|\mathbf{z}_o, \underline{\theta})|\mathbf{z}_o, \underline{\alpha}] \end{aligned} \tag{17}$$

Now, since $\log a \leq a - 1$, $\forall a > 0$,

$$\begin{aligned} \mathrm{E}\left[\log \frac{f(\mathbf{Z}_u|\mathbf{z}_o, \underline{\theta})}{f(\mathbf{Z}_u|\mathbf{z}_o, \underline{\alpha})}|\mathbf{z}_o, \underline{\theta}\right] &\leq \mathrm{E}\left[\frac{f(\mathbf{Z}_u|\mathbf{z}_o, \underline{\theta})}{f(\mathbf{Z}_u|\mathbf{z}_o, \underline{\alpha})}|\mathbf{z}_o, \underline{\theta}\right] - 1 \\ &= \int_{all\ \mathbf{z}_u} \frac{f(\mathbf{z}_u|\mathbf{z}_o, \underline{\theta})}{f(\mathbf{z}_u|\mathbf{z}_o, \underline{\alpha})} f(\mathbf{z}_u|\mathbf{z}_o, \underline{\alpha})\, \mathrm{d}\mathbf{z}_u - 1 = 0 \end{aligned}$$

we find that in expectation with regard to $f(\mathbf{z}_u|\mathbf{z}_o, \underline{\alpha})$

$$\mathrm{E}[\log f(\mathbf{Z}_u|\mathbf{z}_o, \underline{\theta})|\mathbf{z}_o, \underline{\alpha}] \leq \mathrm{E}[\log f(\mathbf{Z}_u|\mathbf{z}_o, \underline{\alpha})|\mathbf{z}_o, \underline{\alpha}] \tag{18}$$

If $Q(\underline{\theta}^{(p+1)}, \underline{\alpha}) > Q(\underline{\alpha}, \underline{\alpha})$ and we have the equality (17), then relation (18) entails

$$\log f(\mathbf{z}_o, \underline{\theta}^{(p+1)}) > \log f(\mathbf{z}_o, \underline{\alpha})$$

Each step insures the increase of $\log f(\mathbf{z}_o, \underline{\theta})$ without having to perform the possibly difficult integration in (15), provided the function Q is easy to minimize. More precisely, the successive minimization of the expected values (16) should be simpler than the minimization of $\log f(\mathbf{z}_o, \underline{\theta})$. In the next section we will show what the missing variables $\mathbf{Z}_u$ are in the context of our neural network problem and what the function Q is.

6.2 EM-algorithm applied to neural networks

To accelerate the maximum likeliness estimation through the EM-algorithm we define the missing variables as $K_2 \times N$ indicator variables $J_{k_2}(i)$, $i = 1, \ldots N, k_2 = 1, \ldots, K_2$; recall N is the number of grid locations, K_2 is the number of neural nodes in the second hidden layer. The indicators can be 0 or 1 with the constraint that within any of the N sets $\{J_{k_2}(i), k_2 = 1, \ldots, K_2\}$ only one of the K_2 indicators $J_{k_2}(i)$ is 1, the remaining

being 0. The interpretation of these missing indicator variables may proceed as follows. If we knew which node in the second hidden layer, equivalently which component in the Gaussian mixture, is responsible for generating a certain data point $(y_i^{(o)}|\mathbf{x}_i^{(o)})$, we would be able to decompose the likeliness into independent terms, one for each neural node (or component in the mixture). Note that the same neural node (k_2) may generate several data of the same "sort" or "class", that is one could have $j_{k_2}(i) = j_{k_2}(i') = 1$ with $i \neq i'$. In such case the two data i and i' are said to be of the same "class".

Accounting for the expression (12) of the mixture cpdf model, the error function in Eq. (14) is written as

$$Er(\underline{\theta}, \mathbf{D}) = -\sum_{i=1}^{N} \log \left(\sum_{k_2=1}^{K_2} o_{k_2} h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \mathbf{k}_2) \right) = -\sum_{i=1}^{N} \log \left(\sum_{k_2=1}^{K_2} o_{k_2} h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\theta}_{k_2}) \right) \tag{19}$$

$\underline{\theta}_{k_2} = \{v_{k_2}, \mathbf{w}, \mathbf{u}, k_2 = 1, \ldots, K_2\}$ are all the network parameters that define the node $\mathbf{k}_2$, that is, $\{\underline{\theta}_{k_2}, k_2 = 1, \ldots, K_2\}$ includes all the parameters of $\underline{\theta}$ except the mixing parameters o_{k_2}. The probability distribution for $J_{k_2}(i)$ given the network parameters $\underline{\theta}$ and $\mathbf{x}_i$ is then modelled by the mixing coefficients o_{k_2} as

$$\Pr(J_{k_2}(i) = 1|\mathbf{x}_i^{(o)}, \underline{\theta}) = o_{k_2}$$

There is a probability of o_{k_2} that node $\mathbf{k}_2$ has generated datum $(y_i^{(o)}|\mathbf{x}_i^{(o)})$ using the network parameters $\underline{\theta}$.

The complete data error function (derived from the complete data likeliness) involves the indicator variables defined such that we *supposedly* know that node $\mathbf{k}_2$ is responsible for generating a given data point $(y_i^{(o)}|\mathbf{x}_i^{(o)})$ if $J_{k_2}(i) = 1$. The missing variables are modeled through the conditional density

$$\begin{aligned} f(y_i^{(o)}, j_{k_2}(i) = 1|\mathbf{x}_i^{(o)}, \underline{\theta}) &= \Pr(J_{k_2}(i) = 1|\mathbf{x}_i^{(o)}, \underline{\theta})\, f(y_i^{(o)}|\mathbf{x}_i^{(o)}, j_{k_2}(i) = 1, \underline{\theta}) \\ &= o_{k_2} h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\theta}_{k_2}) \\ &= \prod_{k_2=1}^{K_2} (o_{k_2} h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\theta}_{k_2}))^{j_{k_2}(i)} \end{aligned}$$

where we assume *conditional* independence of $j_{k_2}(i)$ and $y_i^{(o)}$ with respect to $\mathbf{x}_i^{(o)}, \underline{\theta}$. The error function associated with the complete data likeliness is then

$$\begin{aligned} Er_c(\underline{\theta}, \mathbf{S}) &= -\sum_{i=1}^{N} \log f(y_i^{(o)}, j_{k_2}(i) = 1|\mathbf{x}_i^{(o)}, \underline{\theta}) \\ &= -\sum_{i=1}^{N} \sum_{k_2=1}^{K_2} j_{k_2}(i)[\log o_{k_2} + \log h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\theta}_{k_2})] \end{aligned} \tag{20}$$

where $\mathbf{S}$ is the complete data set including both observed and missing variables

$$\mathbf{S} = \{y_i^{(o)}|\mathbf{x}_i^{(o)}, j_{k_2}(i),\ i = 1, \ldots, N,\ k_2 = 1 \ldots, K_2\}$$

Note that the use of the indicators has allowed the logarithm to be brought inside the summation (20) over the K_2 nodes (which was not possible in the incomplete likeliness expression (19)).

At the E-step of the EM-algorithm, the expectation of the complete data likeliness (20) has to be taken with respect to $\Pr(J_{k_2}(i)|y_i, \mathbf{x}_i, \underline{\alpha})$, where $\underline{\alpha}$ are the current network parameters. This expectation is written as

$$\begin{aligned} Q(\underline{\theta}, \underline{\alpha}) &= -\sum_{i=1}^{N} \sum_{k_2=1}^{K_2} \mathrm{E}[J_{k_2}(i)|y_i^{(o)}, \mathbf{x}_i^{(o)}, \underline{\alpha}] \left(\log o_{k_2} + \log h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\theta}_{k_2})\right) \\ &= -\sum_{i=1}^{N} \sum_{k_2=1}^{K_2} \mathrm{E}^{(p)}[J_{k_2}(i)] \left(\log o_{k_2} + \log h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\theta}_{k_2})\right) \end{aligned} \tag{21}$$

with $\mathrm{E}^{(p)}[J_{k_2}(i)] = \mathrm{E}[J_{k_2}(i)|y_i^{(o)}, \mathbf{x}_i^{(o)}, \underline{\alpha}]$ and (p) denotes the iteration level in the EM-algorithm.

The expectation of $J_{k_2}(i)$ will depend on the current network parameters $\underline{\alpha}$. The new network parameters are determined by optimization of this expectation $Q(\underline{\theta}, \underline{\alpha})$. This decomposes the optimization problem into a much simpler one as follows.

We start with an initial guess for the parameters $\underline{\theta}$ by giving them random values, yet satisfying the constraints $\sum_{k_2=1}^{K_2} o_{k_2} = 1$, $o_{k_2} \geq 0$ and $v_{k_2} \geq 0$. At the iteration step (p) the current network parameters are denoted as

$$\underline{\alpha} = \{o_{k_2}^{(p)}, v_{k_2}^{(p)}, \mathbf{w}^{(p)}, \mathbf{u}^{(p)},\ k_2 = 1, \ldots, K_2\} = \underline{\alpha}_{k_2} \cup \{o_{k_2}^{(p)}, k_2 = 1, \ldots, K_2\}$$

The new parameters are found through minimization of (21). For this we need to know the expectations of $J_{k_2}(i)$. The expectation of an indicator is equivalent to a probability $\pi_{k_2}(i)$

$$\pi_{k_2}^{(p)}(i) = \mathrm{E}^{(p)}[J_{k_2}(i)] = \Pr(J_{k_2}(i) = 1|y_i^{(o)}, \mathbf{x}_i^{(o)}, \underline{\alpha})$$

The expectation is then developed to Bayes' theorem:

$$\begin{aligned} \mathrm{E}^{(p)}[J_{k_2}(i)] &= \frac{f(y_i^{(o)}, j_{k_2}(i) = 1|\mathbf{x}_i^{(o)}, \underline{\alpha})}{f(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\alpha})} \\ &= \frac{o_{k_2}^{(p)} h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\alpha}_{k_2})}{\sum_{k=1}^{K_2} o_k^{(p)} h(y_i^{(o)}|\mathbf{x}_i^{(o)}, \underline{\alpha}_k)} \end{aligned} \tag{22}$$

This is the Expectation step.

The calculation of the derivates (see appendix C) with respect to the parameters $\underline{\theta}$ allows the maximization step (zero derivative) which provides the new set of parameters

$$\underline{\theta}^{(p+1)} = \{o_{k_2}^{(p+1)}, v_{k_2}^{(p+1)}, \mathbf{w}^{(p+1)}, \mathbf{u}^{(p+1)},\ k_2 = 1, \ldots, K_2\}$$

as follows

$$o_{k_2}^{(p+1)} = \frac{1}{N}\sum_{i=1}^{N} \pi_{k_2}^{(p)}(i) \tag{23}$$

$$v_{k_2}^{(p+1)} = \frac{\sum_{i=1}^{N} \pi_{k_2}^{(p)}(i)}{\sum_{i=1}^{N} \pi_{k_2}^{(p)}(i)(y_i^{(o)} - m_{k_2}(\mathbf{x}_i^{(o)}, \mathbf{w}^{(p)}, \mathbf{u}^{(p)}))^2} \tag{24}$$

As for the set of new parameters $\mathbf{u}^{(p+1)}$ and $\mathbf{w}^{(p+1)}$ we have to minimize the term

$$\sum_{i=1}^{N}\sum_{k_2=1}^{K_2} \pi_{k_2}^{(p)}(i)(y_i^{(o)} - m_{k_2}(\mathbf{x}_i^{(o)}, \mathbf{w}^{(p)}, \mathbf{u}^{(p)}))^2 v_{k_2}^{(p)} \tag{25}$$

which is done using a steepest descent method. A summary of the EM-algorithm is given in Figure 10

EM-algorithm

- Initialize $\underline{\theta}$, call it the current best estimate $\underline{\theta}^{(p)}$
- For each step in the EM algorithm: $p = 1, \ldots, n_{max}$, set $\underline{\alpha} = \underline{\theta}^{(p)}$
  - **Do an E-step:**
    - Consider $J_{k_2}(i)$ as an indicator such that among any of the N sets

      $$\{J_{k_2}(i), k_2 = 1, \ldots, K_2\}$$

      only one of the K_2 indicators $j_{k_2}(i)$ is 1, the remaining being 0, that is $J_k(i)J_{k'}(i) = 0, \forall k \neq k',\ k, k' = 1, \ldots, K_2$ and $\sum_{k_2=1}^{K_2} J_{k_2}(i) = 1$.
    - Calculate the posterior probabilities $\pi_{k_2}^{(p)}(i) = \mathrm{E}^{(p)}(J_{k_2}(i))$ using $\underline{\alpha}$
  - **Do an M-step:**
    - Maximize the function $Q(\underline{\theta}, \underline{\alpha})$ as defined in (21) by calculating a new set of parameters

      $$\underline{\theta}^{(p+1)} = \{\mathbf{w}^{(p+1)}, \mathbf{u}^{(p+1)}, o_{k_2}^{(p+1)}, v_{k_2}^{(p+1)}, \forall k_2\}$$

      This is done by setting $o_{k_2}^{(p+1)}$ and $v_{k_2}^{(p+1)}$ to their expressions (23) and (24)
      then minimize expression (25) to find $\mathbf{w}^{(p+1)}, \mathbf{u}^{(p+1)}$

Figure 10: *The EM-algorithm for training a neural network.*

Thus, as opposed to the parameters o_{k_2} and v_{k_2} which can be calculated in one step, determination of the weights $\mathbf{w}, \mathbf{u}$ calls for an iterative numerical optimization method, such as steepest descent. Ideally, the weights $\mathbf{w}^{(p+1)}$ and $\mathbf{u}^{(p+1)}$ should also

be found in one step at each iteration of the EM-algorithm. This is unfortunately not the case and results in greater CPU time.

To alleviate this problem, Husmeier and Taylor (1997) propose to use a Random Vector Functional Link (RVFL) net for the first hidden layer in our two-hidden layer neural network. The random functional link net is drawn in Figure 11 and adds direct connections between the input and the second hidden layer. The weights $\mathbf{u}$ between the input and the first hidden layer are fixed, while the weights $\mathbf{w}$ are determined (they are said to be adaptable). Generally, the weights $\mathbf{u}$ are drawn from a Gaussian distibution with zero mean and variance σ_u^2. By fixing the weights $\mathbf{u}$, the term $m_{k_2}(\mathbf{x}_i, \mathbf{w}, \mathbf{u})$ becomes linear in the weights $\mathbf{w}$, and the expression (25) can be optimized wih respect to $\mathbf{w}$ in one step. The first hidden layer then outputs a linear combination (with adaptable weights) of direct inputs and non-linearly transformed inputs determined by the fixed weights $\mathbf{u}$. In a sense we split the input into a linear adaptable part and a non-linear fixed part. Note that the non-linear contribution of the first hidden layer is still weighted with the adaptable weights $\mathbf{w}$. Husmeier and Taylor use a calibration method for the parameter σ_u. For our application we take $\sigma_u^2 = 1$.

Fixing the weights $\mathbf{u}$ entails that they are not changed during the EM-algorithm, the weights $\mathbf{w}$ on the contrary remain adaptable. The original first hidden layer, producing the means $m_{k_2}(\mathbf{x}_i, \mathbf{w}, \mathbf{u})$ is replaced by a different hidden layer, i.e. the RVFL. Igelnik and Pao (1995) showed that the RVFL has the same approximation capabilities as a standard one hidden-layer neural network. This approach, retains the flexibility of a two-layer neural network but provides greater efficiency through rapid EM-steps as follows.

In the maximization step of the EM-algorithm, the weights $\mathbf{w}$ are now determined in one step since

$$m_{k_2}(\mathbf{x}^{(o)}, \mathbf{w}, \mathbf{u}) = \mathbf{w}_{k_2}^T m_{k_2}(\mathbf{x}^{(o)}, \mathbf{u})$$

where we denote by $\mathbf{w}_{k_2}$ the weights of the links feeding into the node $\mathbf{k_2}$ of the second hidden layer, and by $m_{k_2}(\mathbf{x}, \mathbf{u})$ the activities[5] of the nodes in the input and in the first hidden layer. Now the expression (25) becomes quadratic in $\mathbf{w}$. In appendix C, it is shown that the new weights $\mathbf{w}_k^{(p+1)}$ are found by solving the system

$$(\mathbf{G^{(p)}\Pi_{k_2}^{(p)}G^{(p)}}^T)\mathbf{w_{k_2}^{(p+1)}} = \mathbf{G^{(p)}\Pi_{k_2}^{(p)}y} \quad \text{for each } k_2 \tag{26}$$

where $\mathbf{G^{(p)}}$ is the matrix of values $\{m_{k_2}(\mathbf{x}_i, \mathbf{u}^{(p)}),\ i = 1,\ N\}$ and $\mathbf{\Pi_{k_2}^{(p)}}$ is a diagonal matrix with on its diagonal the values $\{\pi_{k_2}^{(p)}(i), i = 1, N\}$

Relations (23), (24) and (26) define the maximization step.

[5] *activities* are neural jargon for all values from a previous layer that are inputted to the next layer

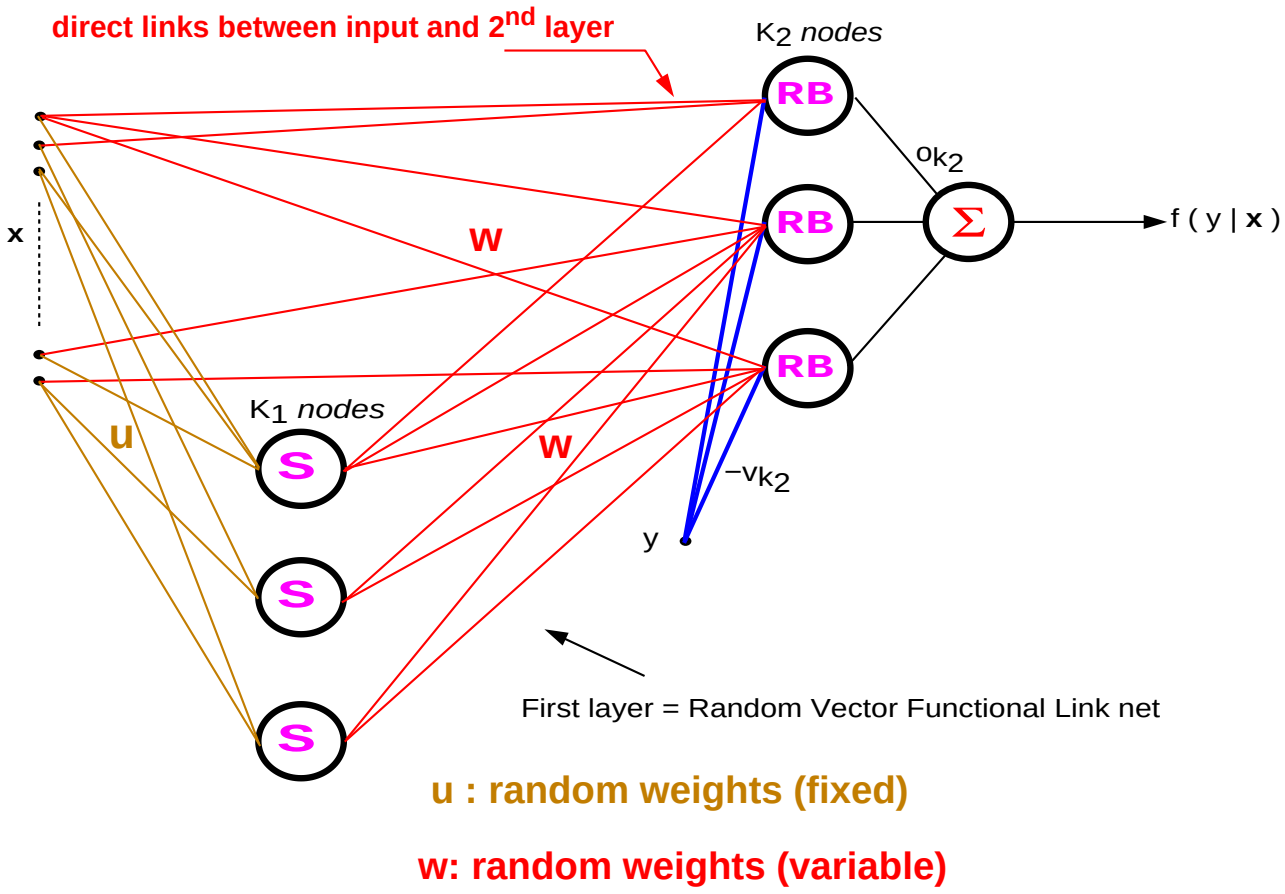

Figure 11: *Architecture of a two layer neural net where first layer is a random vector functional link net.*

7 Obtaining the essential features of an image

The neural network architecture is defined by the numbers (K_1, K_2) of internal nodes, the input, output and how all these are linked. Training on an image is performed through maximization of the error function (14) through iterative EM-steps. One cannot however simply fit the data extracted from an image using the EM-algorithm, since the fitting error will gradually decrease and the neural network may end up fitting the training data exactly beyond their essential and exportable spatial features. Indeed, by using enough nodes (K_1, K_2 large) one could fit exactly all training patterns. A way to overcome this problem is to stop the training early, also denoted in neural language as *early stopping*. However, when to stop is not clear.

Consider a second image, or if not available, we divide up our training image such that an independent set can be obtained. The idea is to evaluate the same likelihood error function on the second cross-validation set using the parameters fitted on the first training set. We define then a second error function which is calculated at each step of the EM-algorithm

$$Er_{cv}(\underline{\theta}_{tr}, \mathbf{D}^{cv}) = -\log(\ell_{cv}(\underline{\theta}_{tr}, \mathbf{D}^{cv})) = -\sum_{i=1}^{N} \log(f(y_i^{cv}|\underline{\theta}_{tr},\, \mathbf{x}_i^{cv})) \tag{27}$$

where y_i^{cv} is the cross-validation data. Note that the cross-validation error is calculated exclusively with the cross-validation data $\mathbf{D}^{cv}$ but using the training parameter values $\underline{\theta}_{tr}$. The cross-validation data set is *not* used to determine the parameters $\underline{\theta}_{tr}$ in the minimization, yet the cross-validation error (27) it is updated at each step of the EM-algorithm. The cross validation error in (27) will tend to have a minimum as the EM-algorithm proceeds, see Figure 12. This can be understood by viewing the training with maximum likelihood as a result of two opposing tendencies. On one hand, the goodness of fit of the training patterns by the neural net will increase as the likelihood is maximized, on the other hand the amount of learning will decrease.

The cross-validation error will tend to decrease at first as the network learns the essential features of the first training image but will increase as the network starts overfitting the first training data modeling features which are not shared by the second cross-validation image. We will show examples of training and cross-validation eror functions in the examples.

8 Implementation notes

The source code is written in FORTRAN90 which gives us flexibility and efficiency for matrix computation, yet keeping the code simple and easily readable by third parties (the complete program including all subroutines comprise less than 1000 lines !). As shown in appendix C, a singular value decomposition has to be performed for each of the

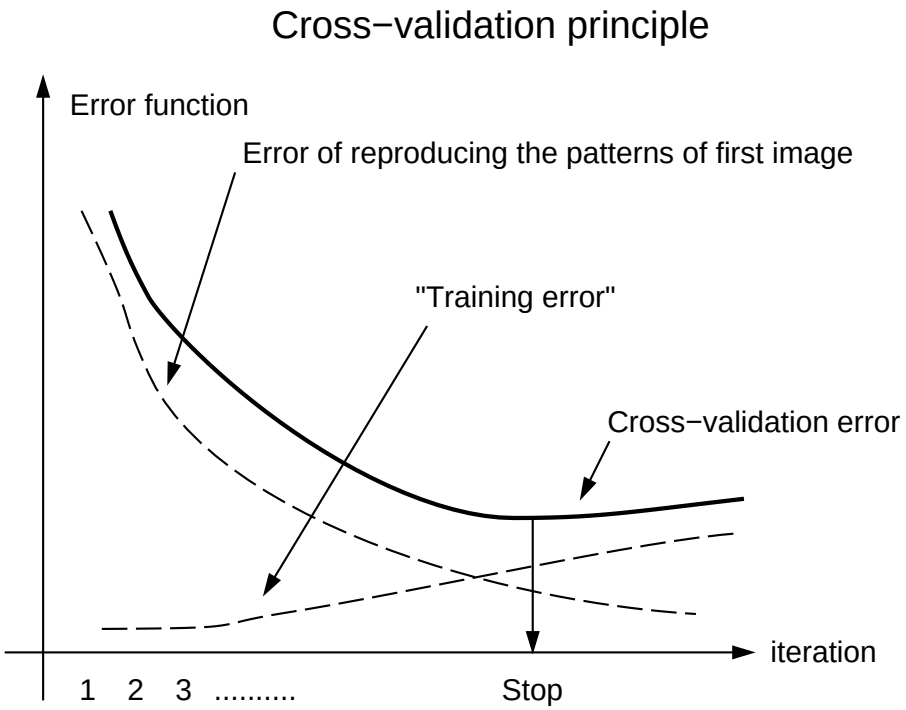

Figure 12: *Training error and cross-validation error.*

K_2 nodes in the second hidden layer of the neural network. We used the decomposition algorithm provided by Press et al. 1996.

The network parameters have to be initialized before training. The following choices were made

$$\begin{aligned} o_{k_2}: &\quad o_{k_2} = \frac{1}{K_2},\ k_2 = 1, \ldots, K_2 \\ v_{k_2}: &\quad v_{k_2} = 0.5,\ k_2 = 1, \ldots, K_2 \\ w_{k_2}: &\quad w_{k_2} \sim N(0,1)\ k_2 = 1, \ldots, K_2 \\ u_{k_1}: &\quad u_{k_1} \sim N(0,1)\ k_1 = 1, \ldots, K_1 \end{aligned}$$

The networks stops training when the cross-validation error reaches its minimum. If multiple grids are used, this should not necessarily result in a multiple increase in computer time. It is easy to see that the training of a net on multiple grids can be parallelized if parallel CPU is available. The simulation of these multiple grids are however performed in series from coarse to fine.

The cost of training a two layer network depends both on the number N of data used (=number of grid nodes) and on its architecture, that is K_2, K_1 and L. The largest cost in CPU time is associated to building and solving the K_2 linear systems (26). The CPU-training formula could be written as

$$\text{training CPU} = O(K_2 \times N^2) + O(K_2 \times (L + K_1)^3)$$

The simulation is less costly with

$$\text{simulation CPU} = O((L + K_1) \times K_2 \times N)$$

Note however that in the training phase scanning, we need not scan every pixel. One can scan every two pixels or ever three pixels to get a total N' of data far less than N. This will result, however, in loss of information.

9 Examples

A range of example training images is shown in order to assess the performance of the stochastic simulation method proposed. The source code to reproduce these examples is available in the GUTLIB directory. The GSLIB-format programs and their parameter files are described in section 16.

Chess-board. A simple pattern is generated by alternating high and low grid node values. The intensities of both high and low values are distributed according to a uniform distribution. A 50×50 training and cross validation image is shown in Figure 13, together with the scanning template used. We opted for a small neural network with only $K_2 = 2$ nodes in the second hidden layer (Gaussian nodes) and $K_1 = 6$ nodes in the first hidden layer. Learning of this neural network took only 8 iterations. Since the border grid nodes are not changed during the Metropolis-Hastings-sampling, the resulting simulation will show edge-effects as shown in Figure 13 even after a large number of iterations (2000) over the whole grid. Indeed, if the border grid nodes are fixed, the image can never converge completely to the pattern read on the training image. This problem could be solved by learning a second neural network specific for border templates. Instead, we opted to wrap around the grid (Wang, 1996) so that the neural network is applied equally to all grid nodes. This results in a perfect reproduction of the training pattern see Figure 14. Note that because this is an unconditional simulation, the simulated image of Figure 14 has a dark pixel at the bottom left pixel while the training image has a light pixel at that position.

Stripes. Another simple periodic pattern is generated by alternating stripes of high and low grid node values. The high and low values are distributed according to a uniform distribution with random spatial pattern. A 50×50 training and cross-validation image is shown in Figure 15. The template used is the template 2 shown in Figure 16. In this case, we opted for a larger network as the template is larger and the image is slightly more complex ($K_1 = 20$ nodes, $K_2 = 5$ in the second hidden layer). The training phase was completed after 25 EM-steps. In Figure 15 we show two resulting unconditional simulations which differ by the shifted pattern of the stripes. The target and simulated (simulation # 1) histogram is shown as well. The simulation was completed after 200 visits of the whole image.

Fractures. A training fracture network is generated using an unconditional simulation method: we simulated a single direction of fractures with average length 11 grid nodes randomly located within a background matrix, see Figure 16. Since the simulated fractures have a distinct anisotropy, we initially considered a simple anisotropic template as shown in Figure 16. Yet the results produced with a large isotropic template are equally good even if we use the same neural architecture of 20 first and 10 second hidden layer nodes in both cases. Training was completed after 22 EM-steps. The amount of fractures in the matrix is 8%. The simulations shown are unconditional.

Anisotropic Gaussian field. An unconditional Gaussian simulation was produced

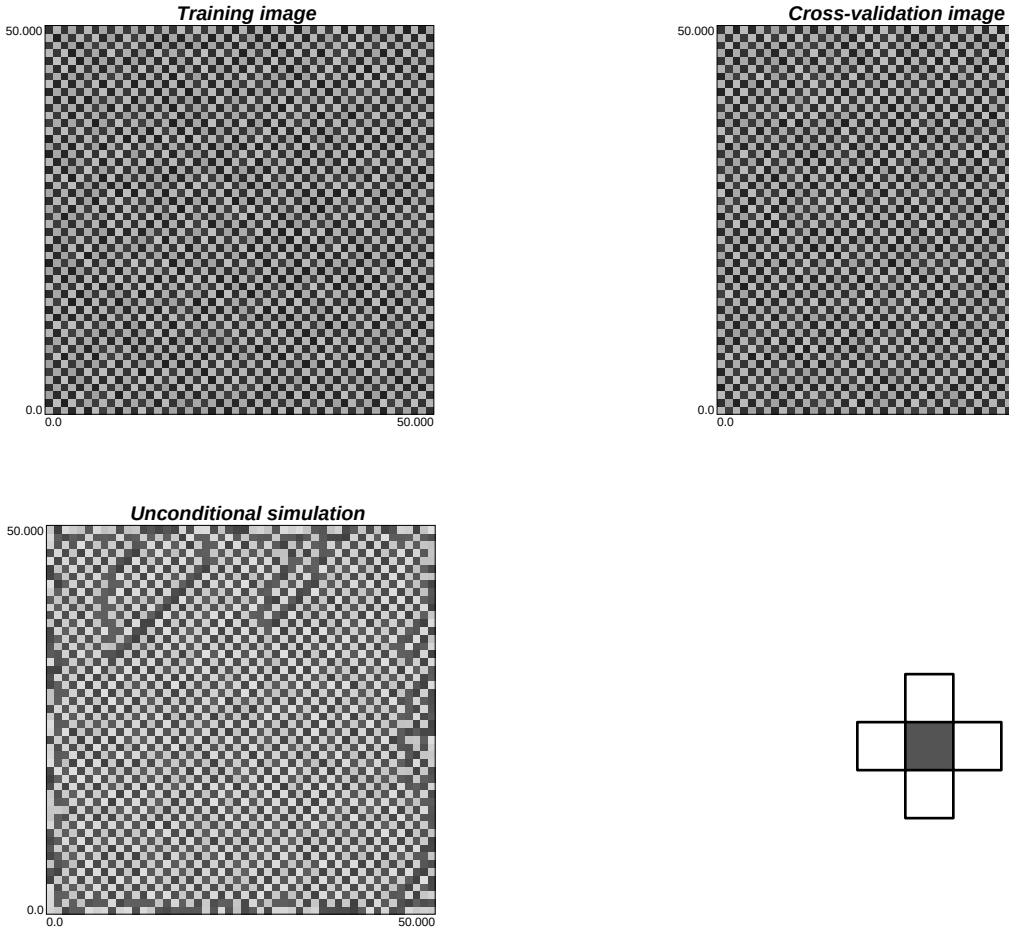

Figure 13: *Chess-board pattern. Note the border effects.*

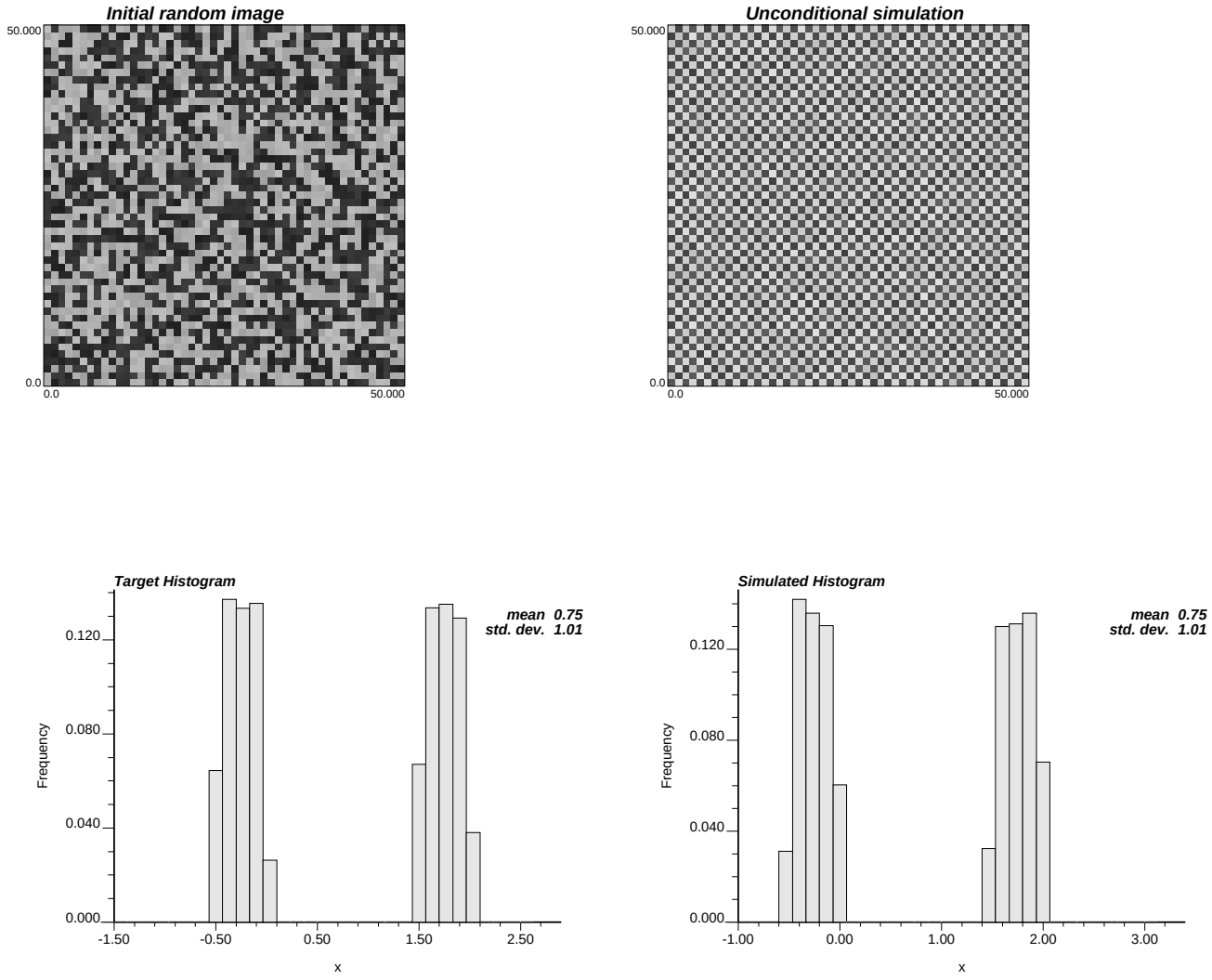

Figure 14: *Chess-board pattern. Border effects are ignored by wrapping the simulation grid around itself.*

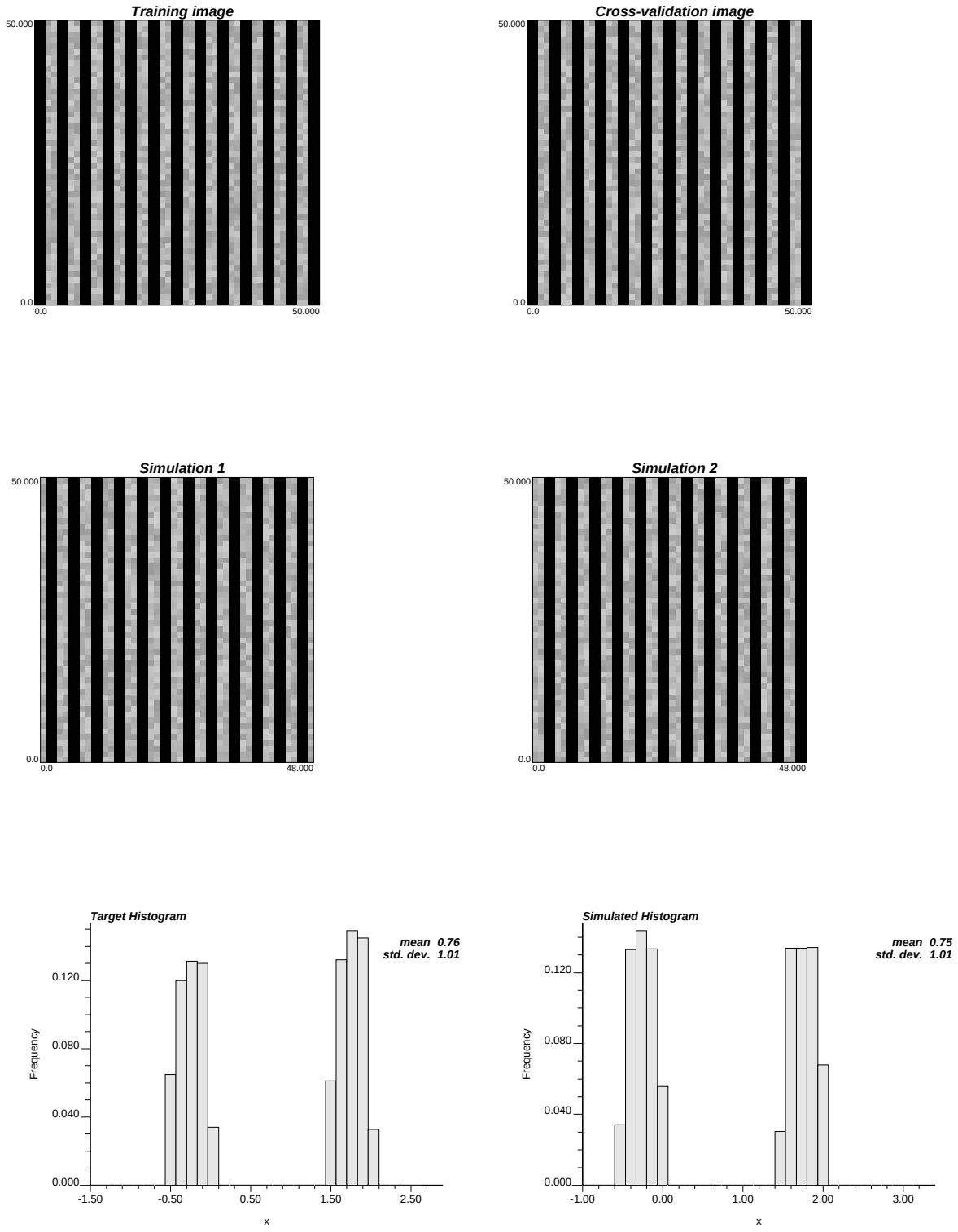

Figure 15: *Pin-striped suit pattern.*

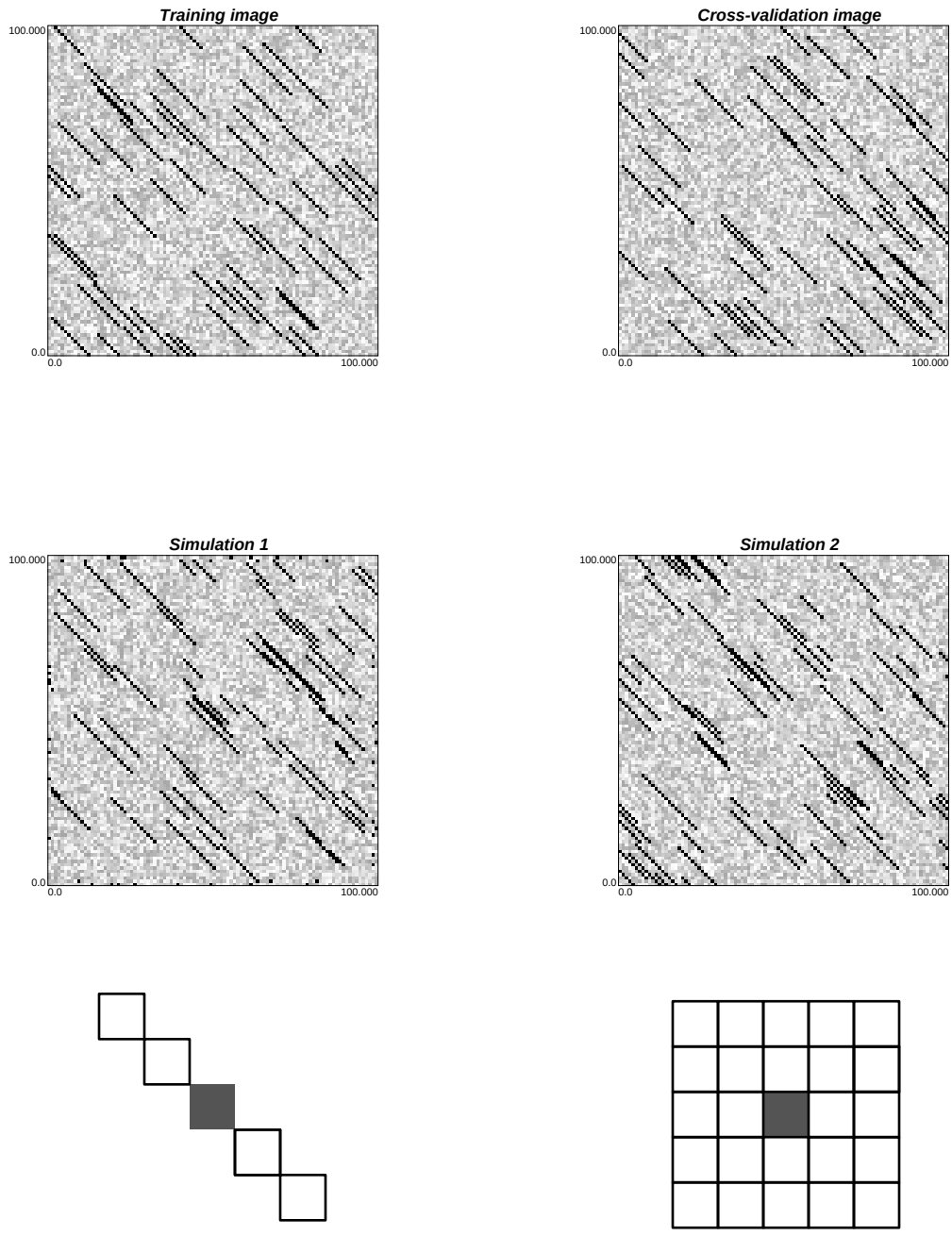

Figure 16: *Single set of fractures.*

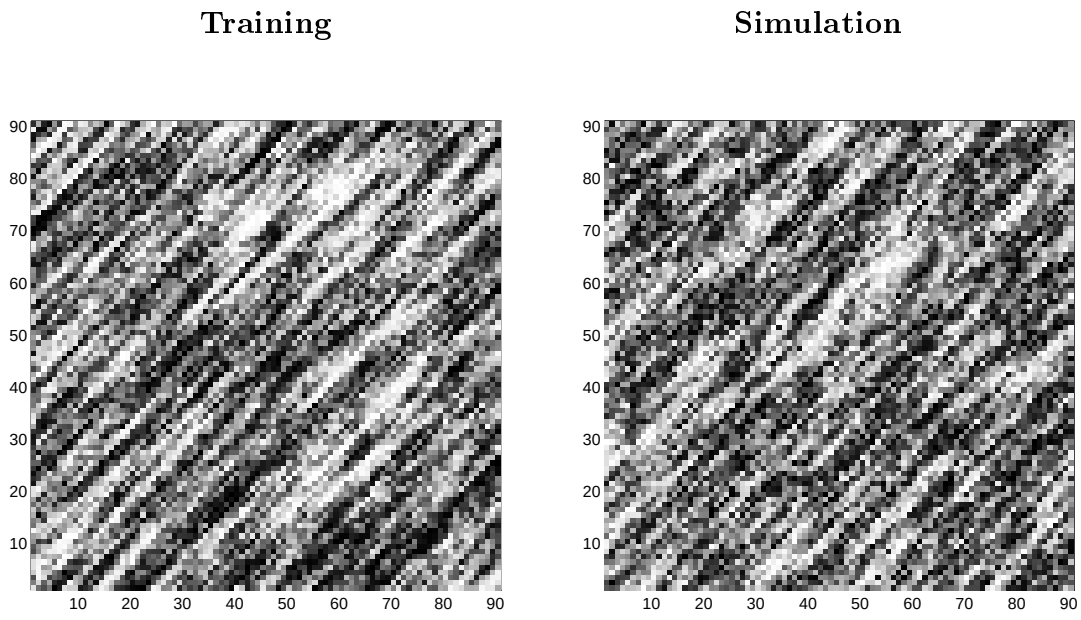

Figure 17: *Gaussian anisotropy.*

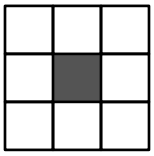

Figure 18: *Template used for anisotropic Gaussian training image.*

with program sgsim (Deutsch and Journel, 1997, p. 170) to generate a training and cross-validation image with strong anisotropy in the NE direction. An isotropic template is used to scan the training image, see Figure 22. The isotropy of the template does not affect the reproduction of the anisotropy as displayed by the unconditional simulation in Figure 17. This result confirms findings by Wang (1996) where the shape of the template has little effect on reproduction of anisotropies.

Walker Lake data set. The Walker lake data set is a set of 78000 (260×300) topographic measurements (altitudes in feet) of a mountain range near Walker Lake. Figure 20 depicts the exhaustive data set which clearly shows the mountain ridges and the flat salt lakes in the valleys. A normal score transform of the data was taken. To obtain a training and a cross-validation set we split the image in two halfs (bottom and top). The aim of this study is to produce unconditional simulations on small 130×150 grid and conditional simulations on a larger 260×300 grid. To be able to reproduce the long range structure along the NS-direction we used the multiple grid approach. For conditional simulation, three successive grids were retained of size 65×75, 130×150

and 260×300. For each of these grids we used the same template, see Figure 18, but with different sizes $d = 20,\ 10$ and 5. For unconditional simulation we retained the first two grids only. The topography of the mountain ranges is much more erratic than that of the valleys. Hence the conditional distribution modelled in valleys should have smaller variance than those in mountain ranges, which means that the conditional distribution should be heteroscedastic. To check that this property is featured through the modeling of the neural network, we selected a location y_{lake} in the lake and y_{mnt} in the nearby mountain ridge. For these two locations, the template associated to the second grid was used to find the neigbouring values $\mathbf{x}_{lake}$ and $\mathbf{x}_{mnt}$ of y_{lake} and y_{mnt}. For the mountain location we have

$$\mathbf{x}_{mnt} = \{1.88,\ 0.82,\ 0.98,\ 1.69,\ 1.21,\ 1.22,\ 1.73,\ 1.41,\ 1.50,\ 0.57,\ 2.05,\ 0.92\} \quad (28)$$

For the mountain location we have

$$\mathbf{x}_{lake} = -\{1.91,\ 2.66,\ 1.75,\ 1.94,\ 1.93,\ 1.48,\ 1.66,\ 1.66,\ 1.65,\ 1.62,\ 1.67,\ 2.15\} \quad (29)$$

Figure 19 shows the distribution $f(y_{mnt}|\mathbf{x}_{mnt})$ and $f(y_{lake}|\mathbf{x}_{lake})$ modelled through a neural network with 20 first-hidden-layer nodes and 10 second-hidden-layer-nodes. The conditional distribution appears indeed heteroscadastic.

390 regularly gridded sample locations (0.5 % of the total amount of pixels) were selected on the reference Walker Lake data set to provide conditioning data. Figure 20 shows two resulting conditional simulations. The large scale feature of mountain versus lake area is well reproduced, whereas smaller details within the mountain are not reproduced. Figure 21 shows the reference semivariogram of the conditional simulation versus the variogram in North-South and East-West directions. Although not explicitely modelled, the variograms are well reproduced.

Two unconditional simulations on a 130×150 grid are also shown in Figure 20. The unconditional simulations reproduce well the transition between mountain and lake area characteristic of the exhaustive data set. The variograms are again well reproduced, see Figure 21.

Channels. Fluvial reservoirs are characterized by the presence of sinuous sand-filled channels within a background of mudstone. A realistic modeling of the curvi-linear aspect of such channels is critical for reliable connectivity assessment and for flow simulation. The modeling of such channels usually occurs within well-defined stratigraphic horizons. An object-based method (fluvsim, Deutsch and Wang, 1996) was used to generate a training and cross-validation image, see Figure 23. The corresponding petrophysical properties were generated as a random texture within both the mudstone and the channel. The training and cross-validation images clearly exhibit long-range structure due to the channeling. To reproduce such long-range structures, we must simulate on multiple grids. Three grids were selected of sizes 25×25, 50×50 and 100×100. For each of these grids we used the same template, see Figure 18, but with different sizes $d = 20,\ 10$ and 5. For each grid, a neural network is learned from the

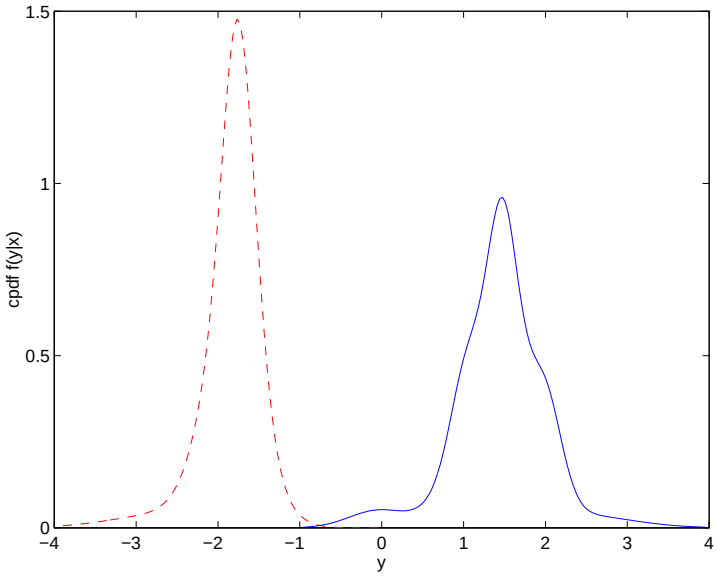

Figure 19: *Conditional density model* $f(y|\mathbf{x})$ *with data* $\mathbf{x}$. *(- -) in the lake area, (–) in the mountain range.*

scanned data using the corresponding template. In each case we opted for a net with 30 nodes in the first and 15 nodes in the second hidden layer. Training was completed for each of the nets at about 25 EM-steps. Figure 24 shows three unconditional simulations and two conditional simulations, each with a different amount of conditioning data, in order to assess the influence of conditioning data on the positioning of the channels. When using conditioning data, the meandering and thickness of the channels is better reproduced. The conditioning data locations were taken randomly from the training image. For the unconditional simulation we found more variation in channel thickness as shown in Figure 24. Also, in the latter case the connectivity is less well reproduced. Figure 25 shows that the reproduction of the uniform histogram is excellent.

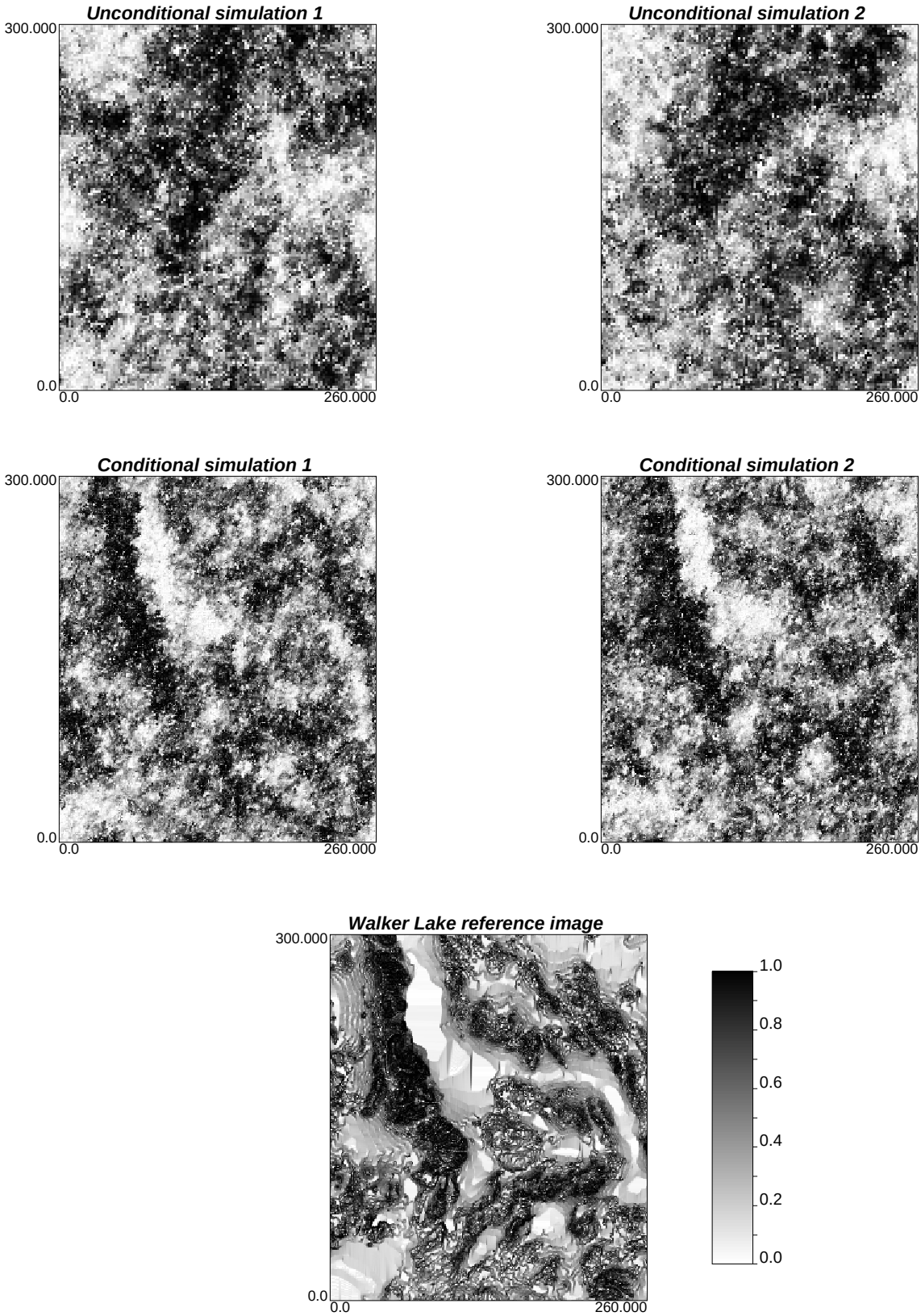

Figure 20: *Walker lake: simulation results (to get a better gray scale, all images have been transformed into uniform scores) .*

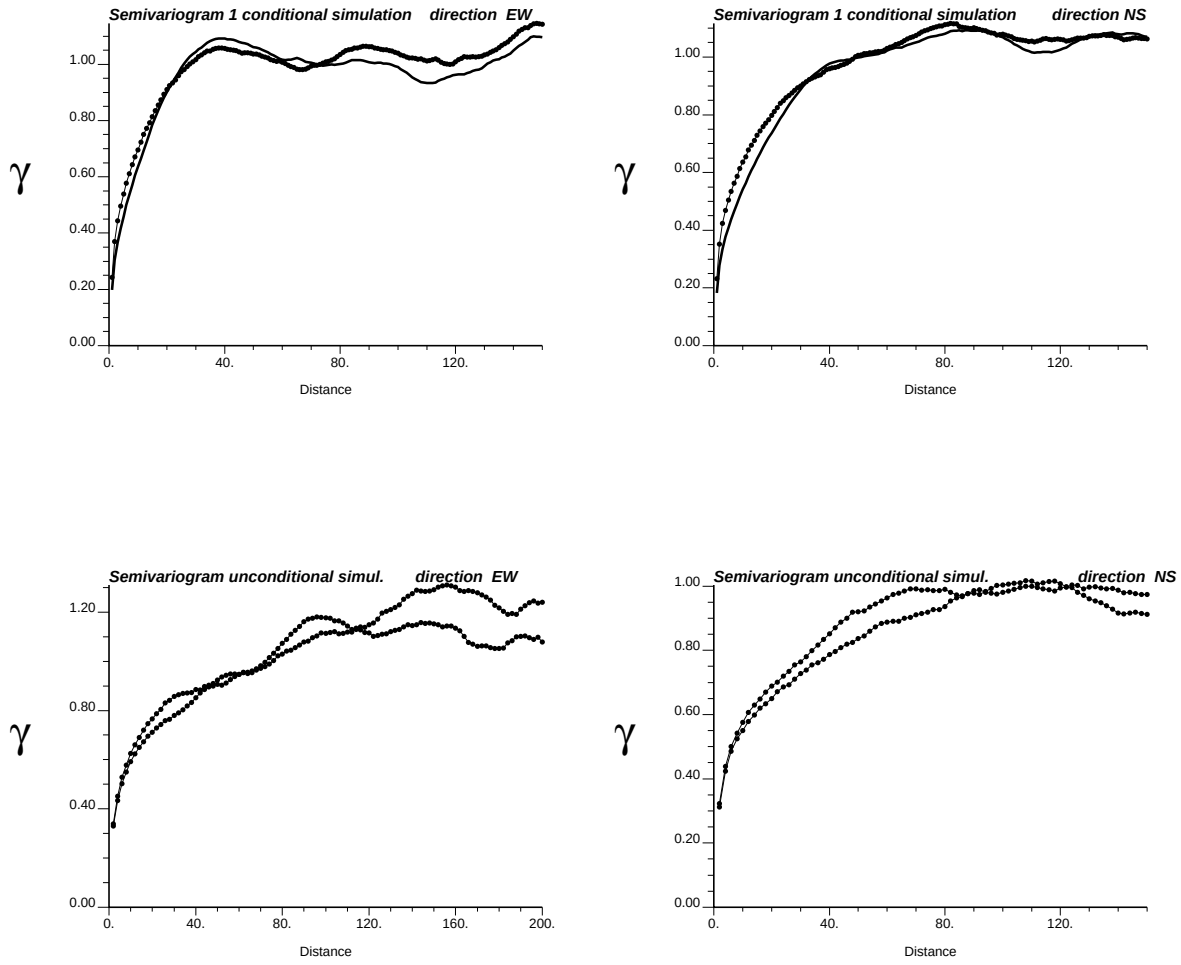

Figure 21: *Walker Lake variograms: (top left) semivariogram of simulation 1 in EW direction versus exhaustive data set (continuous line), (top right) NS-direction, (bottom left) EW-variograms of the two unconditional simulations (bottom right) NS-direction.*

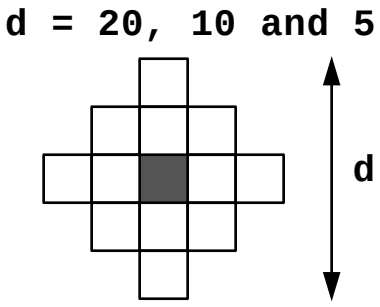

Figure 22: *Template used in multigrid simulation of channels.*

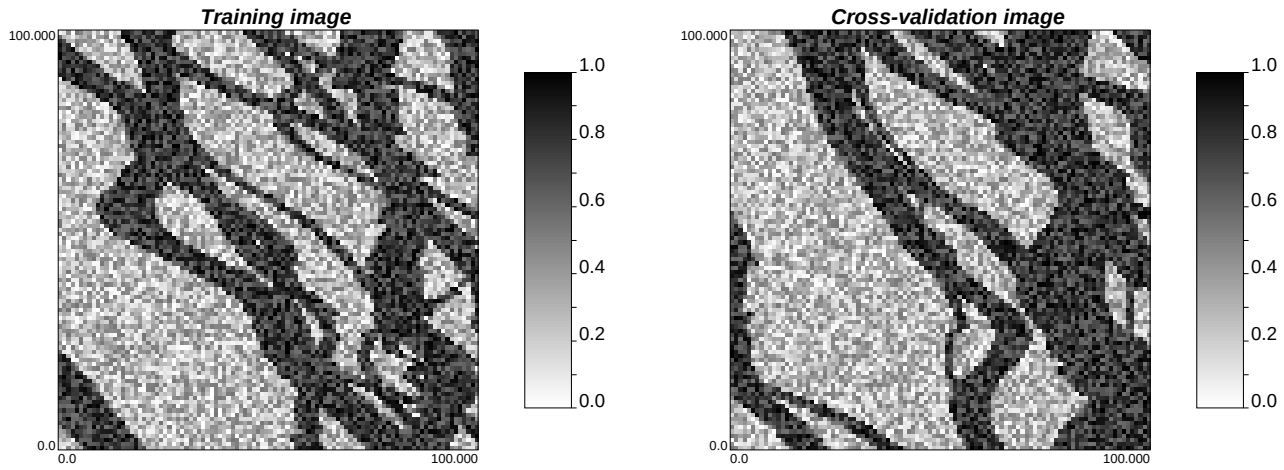

Figure 23: *Channels: Training and cross-validation image.*

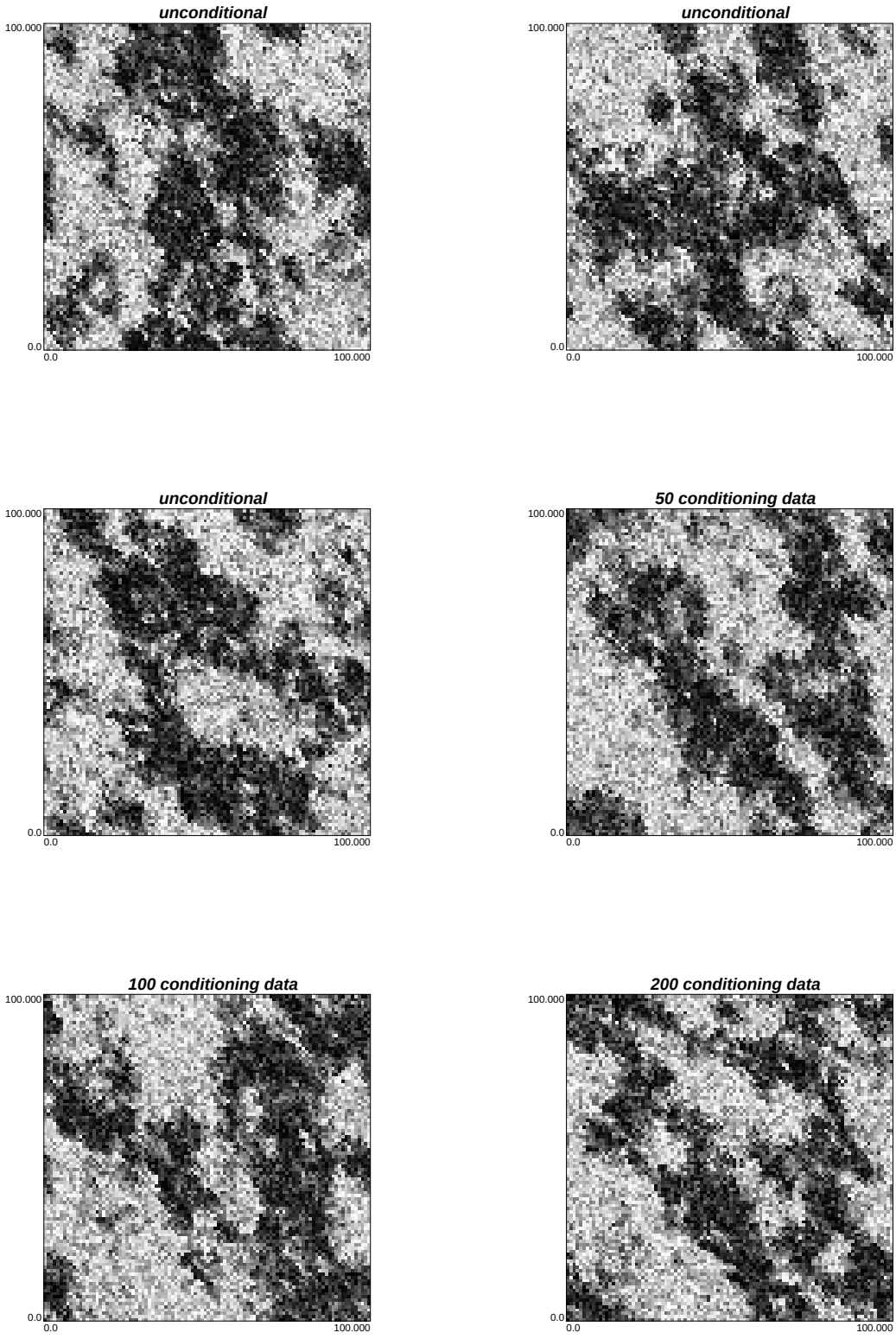

Figure 24: *Channels: results.*

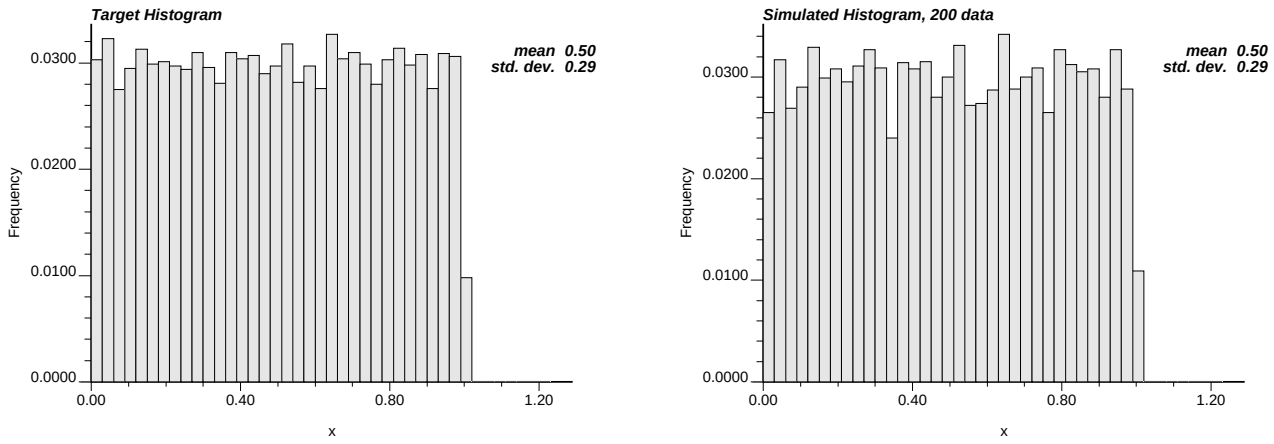

Figure 25: *Channels: Histogram reproduction.*

10 Future research: Integration of data of diverse type on multiple scale and with different precision

The neural network approach provided us with a first attempt at using so-called "artificial intelligence" for incorporating conceptual geological models present in training images. Many important challenges remain, among which:

A better understanding of what the neural network "does". The neural net is trained using unsupervised training, i.e., there is no calibration of the output with actual "true output". One advantage is that the methodology is automatic and thus easy to use in practice. But unsupervised means also uncontrolled, and might lead to incorrect results. The function of each layer in the two layer neural network is clearly different. The first layer performs a non-linear regression which provides mean values m_{k_2} through the weights $\mathbf{w}$ and $\mathbf{u}$ and variance v_{k_2} attached to each node k_2 in the network. The second layer combines these means and variances into a Gaussian mixture model. The first layer provides a type of clustering of the patterns. Similar input patterns $\mathbf{x}_i$ will produce similar mean values $m_{k_2}(\mathbf{x}_i)$, and the conditional distribution of the y_i's will be similar. The second layer mixes all these "clusters" into one model.

A better understanding of the fitting process performed by the two layer network would be obtained if we could split the process of training into two steps,

(1), since certain similar patterns $\mathbf{x}_i^{(o)}$ may be recurring many times, we could try to classify similar patterns $\mathbf{x}_i^{(o)}$ into one distinct class. That is, we make a classifi-

cation of the $\{\mathbf{x}_i, i = 1, \ldots, N\}$ by clustering them into C distinct groups, each group labeled as $c = 1, \ldots, C$. Some classes will be well informed and form a distinct cluster, others might correspond to a wide variety of patterns. The classification procedure thus entails a mapping of each $\mathbf{x}_i$ into a class c. Any other new $\mathbf{x}_i$ (in the generalization phase) can then be classified into one of the classes C. Many methods or available for such clustering: the algorithms Autoclass (developed by NASA) or snob (developed by Monash University) seem to be the most appropriate since they also search for the optimal number of classes C and leave the capability for the user to combine or split classes whenever relevant (so some supervision over the algorithm is given). The latter clustering algorithms also perform a "soft" clustering or "partial assignment", i.e. the patterns are not assigned fully (with probability 1) to one single class, instead each pattern may belong to various classes, the weight of assignment is the probability (actually a posterior probability) to belong to a given class.

(2), we proceed by modeling the conditional distribution $f(y_i|\mathbf{x}_i)$ as $f(y_i|c)$ since pattern $\mathbf{x}_i$ belongs to class c. For each of the patterns $\mathbf{x}_i$, we have a value y_i. Then, each distribution $f(y_i|c)$ can be modelled separately. The modeling can be done parametrically as $h_c(y|\mathbf{x}, \underline{\theta}_c)$ or non-parametrically as $h_c(y|\mathbf{x}, \Delta)$ using step functions Δ (the tails might need some modeling). Next, what should be done with those y_i value that correspond to fuzzily classified $\mathbf{x}_i$, i.e. those patterns that are shared by multiple classes or clusters ? We could write the conditional distribution $f(y_i|\mathbf{x}_i)$ for these cases as a mixture of distributions parametrically as

$$f(y|\mathbf{x}, \underline{\theta}) = \sum_{c=1}^{C'} o_c h_c(y|\mathbf{x}, \underline{\theta}_c)$$

or non-parametrically as

$$f(y|\mathbf{x}, \mathbf{o}) = \sum_{c=1}^{C'} o_c h_c(y|\mathbf{x}, \Delta)$$

where C' is the number of poorly-informed classes. Again we can proceed using a rapid EM-algoritm with timing of order $O(C' \times N)$. Now we need not solve the linear systems in (26), since the patterns are already classified (no means and variances need to be determined).

Using this approach, one can integrate training images of various sources, on various scales and with various precisions. The clustering technique would allow mixing both continuous (seismic, petrophysical) data as well as discrete (facies) data. It allows mixing data of various precisions through the partial assignment principle. Imprecise data have a higher probability of being assigned to multiple classes. Indeed, most good clustering algorithms require the precision of the data to be specified.

It is therefore worthwhile to forward simulate seismic through a geological model and subsequently use the seismic image as an additional contraint in the building of the

local ccdf, i.e. $f(y_i|\mathbf{x}_i, \mathbf{s}_i)$, where $\mathbf{s}_i$ is neigbouring or co-located seismic data, possibly with multiple attributes. Such technique would require the geologist and geophysicist to build *one* deterministic model of the reservoir, which need not be constrained to actual local data. Such deterministic model should be deemed *similar* to the actual reservoir. Model building and actual conditioning occur here as two separate steps. The technique will then build from that one deterministic model *multiple* realizations constrained to the actual data.

11 Preliminary conclusions

In this paper we propose a stochastic simulation method based on sequential modeling of local conditional distributions (cpdf) using neural networks. The conditional distribution of a set of unknown variables given a set of known variables is at the root of many statistical prediction algorithms and the methology developed here carries over to other estimation problems, not necessarily spatial ones. The key aspect of the methodology proposed consists of integrating into the cpdf's, right from the start, essential textures read from training images. This approach is markedly different from most geostatistical method where the texture is embedded into the random function model (such as Multi-Gaussian or multiple indicator models) which borrows from reality only 2-point statistics. It is also different from the simulated annealing approach where an objective function is defined to impose reproduction of various multi-point statistics scanned from the training image. Simulated annealing does not provided a method to *decide* which multi-point statistics are essential and thus should be retained in the objective function. The use of a cross-validation data set allowed the neural network approach to avoid overfitting the multi-point patterns scanned from the training image. In case of simulated annealing, if too many multi-point statistics borrowed from the training image are imposed, the training image would be reproduced exactly, which is never the ultimate goal. The neural network approach allows for data expansion, a concept that is of critical importance in any imaging technique within the earth sciences.

The proposed method does not involve any of the classical steps of a geostatistical study such as kriging, modeling of variograms or making univariate transforms, nor does it involve any kind of consistency condition and correction. At the same time this simplification entails its major disadvantage: we do not know what textures are being borrowed from the training images. From the examples it appears that the neural network approach has the capability of reproducing curvilinear features, fractures or repetitive patterns.

Many practical issues still need to be addressed: how large should the template(s) be ? How many nodes should each hidden layer in the net contain ? When should we stop the visit of the whole image in the Metropolis-Hastings sampler? How can we speed up the learning phase ? Many of the answers lie in a better understanding of the

inner working of the neural network and how the two hidden layer interact with each other during learning and generalization phases.

12 Acknowledgements

I would like to thank Prof. André G. Journel for his Ciceroan patience in reviewing this paper. After the comment "Quousque tandem abutere patientia mea", André quoted (after the sixth and yet not final version): "Labor omnia vincit improbus".

13 References

Besag, J., 1974. Spatial interaction and the statistical interaction of lattice systems (with discussion). *J. Roy. Stat. Soc. B*, 36, 192–236.

Besag, J., 1975. Statistical interaction of non-lattice data. *The Statistician*, 24, 179–195.

Besag, J., Green, P., Higdon, D. and Mengersen, K., 1995. Bayesian computation and stochastic systems (with discussion). *Statistical Science*, 10, 3–66.

Bishop C.M, 1995. Neural networks for pattern recognition. Oxford University Press.

Dempster, A.P., Laird, N.M. and Rubin, D.B., 1977. Maximum likelihood from incomplete data via the EM-algorithm (with discussion). *J. Roy. Stat. Soc.*, B, 39, 1–38.

Deutsch, C., 1992. Annealing techniques applied to reservoir modeling and the integration of geological and engineering (well test) data. Ph.D. thesis, Department of Petroleum Engineering, Stanford University, Stanford.

Deutsch, C., and Wang, L., 1996. Hierarchical object-based stochastic modeling of fluvial reservoirs. *Math. Geol.*, 28, 857–880.

Deutsch, C., and Journel, A.G., 1997. GSLIB: Geostatistical Software Library. Oxford University Press.

Guardiano, F. and Srivastava, M., 1992. Borrowing complex geometries from training images: the extended normal equations algorithm. *SCRF Report*, 1992.

Igelnik, B. and Pao, Y.H., 1995. Stochastic choice of basis functions in adaptive functional approximation and the Functional Link net. *IEEE Trans. on Neural Networks*, 6, 1320-1329.

Goodman J. and Sokal A.D., 1989. Multigrid Monte Carlo method. Conceptual foundations. *Physical Revue D, Particles and Fields*, 40, 2035-2071.

Goovaerts, P., 1997. Geostatistics for natural resources evaluation. Oxford University Press.

Husmeier, D. and Taylor, J.G., 1997. Neural networks for predicting conditional probability densities: improved training scheme combining EM and RVFL. King's College, Department of Mathematics, London. Pre-print KCL-MTH-97-47.

Jordan, M.I. and Jacobs, R.A., 1994. Hierachical mixtures of experts and the EM-algorithm. *Neural Computation*, 6, 181–214.

Journel, A.G. and Alabert, F., 1989. Non-Gaussian data expansion in the earth sciences. *Terra Nova*, 1, 123–134.

Hastings, W.K., 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57, 97–109.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P., 1996. Numerical recipes in Fortran 90, volume 2 of Fortran numerical recipes. Cambridge University Press.

Ripley, B.D., 1987. Stochastic simulation. Wiley, New York.

Ripley, B.D., 1996. Pattern recognition and neural networks. Cambridge University Press.

Srivastava, M., 1992. Iterative methods for spatial simulation. *SCRF Report*, 1992.

Tjelmeland, H., 1996. Stochastic models in reservoir characterization and markov random fields for compact objects. Ph.D. Thesis, Department of Mathematical Sciences, Norwegian University of Science and Technology, Trondheim.

Tran, T., 1994. Improving variogram reproduction on dense simulation grids. *Computers and Geosciences*, 20, 1161–1168.

Wang, L., 1995. Modeling complex reservoir geometries with multi-point statistics. *SCRF Report*, 1995.

Xu, W. and Journel, A.G., 1994. DSSIM: A general sequential simulation algorithm. *SCRF report*, 1994.

14 Notation

- y : value at any node
- y_i : value at node i, Image has N nodes.
- $\mathbf{y}$: values at all nodes = one image
- $\mathbf{y}^{(1)}$, $\mathbf{y}^{(2)}$ = several images
- $\mathbf{x}$: set of neighbourhood data in template centred at any node
- x_j : one specific neighbourhood datum in any template (L in total)
- $\mathbf{x}_i$: set of neighbourhood data in template centred at i
- $\underline{\theta}$: set of parameters
- f : density (conditional, marginal or joint)
- F : cumulative distribution (conditional, marginal or joint)
- K_2 : number of neural nodes in second hidden layer
- K_1 : number of neural nodes in first hidden layer
- T, S : non-linear functions
- $\mathbf{u}$: set of parameters feeding in the first hidden layer
- $\mathbf{w}$: set of parameters feeding in the second layer
- v_k : scale parameters of the radial basis function
- o_k : weight into the output layer
- ℓ : likeliness
- Er : error function
- $\mathbf{D}$: set of scanned data $\{y_i|\mathbf{x}_i\}$

15 Appendices

15.1 Appendix A

The moment generating function of a conditional distribution generated by a single layer feed-forward neural network is calculated through the derivative of the ccdf

$$F(y|\mathbf{x}) = \sum_{k=1}^{K} o_k \, T\left(v_k \left(y - \sum_{l=1}^{L} w_{k,l} x_l\right)\right)$$

into the cpdf

$$f(y|\mathbf{x}) = \sum_{k=1}^{K} o_k v_k T'\left(v_k \left(y - \sum_{l=1}^{L} w_{k,l} x_l\right)\right)$$

with T' the derivative with respect to the argument. The moment generating function is defined as

$$\begin{aligned} M(\omega) &= \int f(y|\mathbf{x}) \exp(\omega y) \, \mathrm{d}y \\ &= \sum_{k=1}^{K} o_k \int_0^1 \exp(\omega y) \, \mathrm{d}T \end{aligned}$$

Suppose that

$$T() = \frac{1}{1 + \exp()}$$

then

$$\exp(\omega y) = \left(\frac{1-T}{T}\right)^{-1/v_k} \exp(\omega \sum_{l=1}^{L} w_{k,l} x_l)$$

and

$$M(\omega) = \sum_{k=1}^{K} o_k \exp(\omega \sum_{l=1}^{L} w_{k,l} x_l) \int_0^1 \left(\frac{1-T}{T}\right)^{-1/v_k} \mathrm{d}T$$

where we know that (Abramowitz and Stegun, 1970, p256)

$$\int_0^1 \left(\frac{1-T}{T}\right)^{-1/\sigma} \mathrm{d}T = \frac{\pi T/\sigma}{\sin(\pi T/\sigma)}$$

To obtain the moments, one makes the derivative of $M(\omega)$ and implies that $\omega \to 0$ This can be done by considering the Taylor expansion of $M(\omega)$ as follows

$$\begin{aligned} M(\omega) &\overset{\omega \to 0}{\sim} \sum_{k=1}^{K} o_k (1 + \omega \sum_{l=1}^{L} w_{k,l} x_l + \omega^2/2 (\sum_{l=1}^{L} w_{k,l} x_l)^2 + \ldots)(1 + 1/6 \, (\pi\omega/\sigma)^2 + \ldots) \\ &= \sum_{k=1}^{K} o_k + \omega (\sum_{k=1}^{K} o_k (\sum_{l=1}^{L} w_{k,l} x_l)) + \omega^2/2 (\sum_{k=1}^{K} o_k ((\sum_{l=1}^{L} w_{k,l} x_l)^2 \\ &\quad + 1/3 \, (\pi/\sigma)^2)) + \ldots \end{aligned} \tag{30}$$

Thus as one considers the first moment which is the expected value

$$\mathrm{E}[Y^i] = \lim_{\omega \to 0} \frac{\mathrm{d}^i M(\omega)}{\mathrm{d}\omega^i} \tag{31}$$

with $i = 1$ one obtains the conditional mean

$$m_{y|\mathbf{x}} = \sum_{k=1}^{K} \sum_{l=1}^{L} o_k w_{k,l} x_l$$

and recombining the second moment with the mean one obtains the conditional variance

$$\sigma^2_{y|\mathbf{x}} = \sum_{k=1}^{K} o_k \left(\frac{1}{3} \left(\frac{\pi}{v_k} \right)^2 + \left(\sum_{l=1}^{L} w_{l,k} x_k - \sum_{k'=1}^{K} o_{k'} \sum_{j=1}^{M} w_{k',l} x_l \right)^2 \right)$$

Any other moment or cumulant of the conditional distribution can be obtained using Eq. (30) and Eq. (31)

15.2 Appendix B

In this appendix, we derive an expression for the joint density $f(\mathbf{y})$ for the whole image $\mathbf{y}$ *given* that the value of any one pixel only depends on its neighbourhood and from the local conditional density distribution $f(y|\mathbf{x})$. The problem is:

$$given \ : \ f(y_i|\mathbf{x}), \ \ \forall i = 1, \ldots, N$$

$$find \ : \ f(\mathbf{y}) \ \ \mathbf{y} = \{y_1, y_2, \ldots, y_N\}$$

Assume first that all following the conditional distributions are known

$$f(y_i|y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_N), \ \ \forall i \tag{32}$$

the conditioning can later be reduced to the neigbouring data only.

Next, from Bayes' relationship one can write

$$f(\mathbf{y}) = f(y_N|y_1, \ldots, y_{N-1}) \, f(y_1, \ldots, y_{N-1}) \tag{33}$$

But $f(y_1, \ldots, y_{N-1})$ cannot be factorized further into conditional distributions since $f(y_{N-1}|y_1, \ldots, y_{N-2})$ is not easily obtainable from the given conditional distribution in (32), which has a fixed total of $N-1$ conditioning values . Therefore, we introduce a second image $\mathbf{z}$, which also is a sample of the joint density $f()$, and introduce z_N into relation (33)

$$f(\mathbf{y}) = \frac{f(y_N|y_1, \ldots, y_{N-1})}{f(z_N|y_1, \ldots, y_{N-1})} \, f(y_1, \ldots, y_{N-1}, z_N)$$

This relation can be iterated as

$$f(y_1,\ldots,y_{N-1},z_N) = \frac{f(y_{N-1}|y_1,\ldots,y_{N-2},z_N)}{f(z_{N-1}|y_1,\ldots,y_{N-2},z_N)}\, f(y_1,\ldots,y_{N-2},z_{N-1},z_N)$$

To obtain the result

$$\frac{f(\mathbf{y})}{f(\mathbf{z})} = \prod_{i=1}^{N} \frac{f(y_i|y_1,\ldots,y_{i-1},z_{i+1},\ldots,z_N)}{f(z_i|y_1,\ldots,y_{i-1},z_{i+1},\ldots,z_N)}$$

As the conditional distributions only depend on the neighbouring data, we can write the last expression as

$$\frac{f(\mathbf{y})}{f(\mathbf{z})} = \prod_{i=1}^{N} \frac{f(y_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c})}{f(z_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c})}$$

where S_i represents the set $\{1,2,\ldots,i\}$ and S^c its complement. $\partial\mathbf{y}_S$ is the neigbourhood of y defined over the set S. This relation thus states the relative likelihood of two images. Integrating the density $f(\mathbf{z})$ over all $\mathbf{z}$ gives

$$\int_{all\ \mathbf{z}} f(\mathbf{z})\,\mathrm{d}\mathbf{z} = 1 = f(\mathbf{y}) \int_{all\ \mathbf{z}} \frac{\mathrm{d}\mathbf{z}}{\prod_{i=1}^{N} \frac{f(y_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c})}{f(z_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c})}}$$

thus

$$f(\mathbf{y}) = \left[\int_{all\ \mathbf{z}} \frac{\mathrm{d}\mathbf{z}}{\prod_{i=1}^{N} \frac{f(y_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c})}{f(z_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c})}}\right]^{-1}$$

so $f(\mathbf{y})$ is entirely defined by the local cpdf's

$$f(y_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c}) \ \ and \ \ f(z_i|\partial\mathbf{y}_{S_{i-1}},\partial\mathbf{z}_{S_i^c})$$

yet, due to the difficult normalization the likelihood $f(\mathbf{y})$ cannot be obtained.

15.3 Appendix C

The EM-algorithm consists of two steps: the Expectation and the Maximization steps. In the expectation step we calculate the expectation of the complete data likeliness or error function given the actual data. The function has been introduces as Q

$$\begin{aligned} Q(\underline{\theta},\underline{\alpha}) &= -\sum_{i=1}^{N}\sum_{k_2=1}^{K_2} \mathrm{E}^{(p)}[j_{k_2}(i)]\,(\log o_{k_2} + \log h(y_i|\mathbf{x}_i,\underline{\theta}_{k_2})) \\ &= -\sum_{i=1}^{N}\sum_{k_2=1}^{K_2} \pi_{k_2}^{(p)}(i)\,(\log o_{k_2} + \log h(y_i|\mathbf{x}_i,\underline{\theta}_{k_2})) \end{aligned}$$

which is Eq. (21) with

$$\pi_{k_2}^{(p)}(i) = \frac{o_{k_2}^{(p)} h(y_i|\mathbf{x}_i, \underline{\alpha}_{k_2})}{\sum_{j=1}^{K_2} o_j^{(p)} h(y_i|\mathbf{x}_i, \underline{\alpha}_j)}$$

The expectation are function of the present network parameters $\underline{\alpha}$

$$\underline{\alpha} = \{o_{k_2}^{(p)}, v_{k_2}^{(p)}, \mathbf{w}^{(p)}, \mathbf{u}^{(p)},\ k_2 = 1, \ldots, K_2\} = \underline{\alpha}_{k_2} \cup \{o_{k_2}^{(p)}, k2 = 1, \ldots, K_2\}$$

The expectation of the missing variables $j_{k_2}(i)$ are interpreted as posterior probabilities $\pi_{k_2}^{(p)}(i)$. This is the expectation step where the expectation of the indicators are calculated in terms of the current network parameters $\underline{\alpha}$. Note that $\pi_{k_2}^{(p)}(i)$ are not function of the parameters $\underline{\theta}$.

The function Q is then maximized in the maximization step (in the case of an error function it is a minimization) with respect to $\underline{\theta}$. For this purpose, one needs to take the derivative with respect to the parameters.

$$\nabla_{\underline{\theta}} Q = (\nabla_{\theta_1} Q, \nabla_{\theta_2} Q, \ldots, \nabla_{\theta_q} Q) = 0$$

The function Q is further specified by using Gaussian functions for $h(y_i|\mathbf{x}_i, \underline{\theta}_{k_2})$ as

$$Q(\underline{\theta}, \underline{\alpha}) = -\sum_{i=1}^{N} \sum_{k_2=1}^{K_2} \pi_{k_2}(i)[\log o_{k_2} + \log v_{k_2} - \log\sqrt{2\pi} - \frac{1}{2} v_{k_2}(y_i - m_{k_2}(\mathbf{x}_i|\mathbf{w}, \mathbf{u}))^2] = 0$$

In that case the derivative with respect to v_{k_2} and o_{k_2} can be calculated explicitely as follows

$$\frac{\partial Q}{\partial v_{k_2}} = \sum_{i=1}^{N} \pi_{k_2}(i)[(y_i - m_{k_2}(\mathbf{x}_i|\mathbf{w}, \mathbf{u})) - \frac{1}{v_{k_2}}]$$

such that the new optimal value $v_{k_2}^{(p+1)}$ becomes

$$v_{k_2}^{(p+1)} = \frac{\sum_{i=1}^{N} \pi_{k_2}^{(p)}(i)}{\sum_{i=1}^{N} \pi_{k_2}^{(p)}(i)(y_i - m_{k_2}(\mathbf{x}_i, \mathbf{w}^{(p)}, \mathbf{u}^{(p)}))^2}$$

For the determination of o_{k_2}, we need an Lagrangian parameter γ to constrain to $\sum_{k_2=1}^{K_2} o_{k_2} = 1$ to find that

$$\frac{\partial}{\partial o_{k_2}} \left(Q + \gamma \sum_{k_2=1}^{K_2} o_{k_2} \right) = 0 \Rightarrow -\sum_{i=1}^{N} \left(\frac{\pi_{k_2}(i)}{o_{k_2}} + \gamma = 0 \right)$$

such that

$$\gamma = N, \quad o_{k_2}^{(p+1)} = \frac{1}{N} \sum_{i=1}^{N} \pi_{k_2}^{(p)}(i)$$

To find new values for $\mathbf{w}, \mathbf{u}$, we have to minimize the term

$$\sum_{i=1}^{N} \sum_{k_2=1}^{K_2} \pi_{k_2}^{(p)}(i)(y_i - m_{k_2}(\mathbf{x}_i, \mathbf{w}^{(p)}, \mathbf{u}^{(p)}))^2) v_{k_2}^{(p)} \tag{34}$$

which has to be minimized using steepest descent, since the derivatives of the latter term in $\mathbf{w}$ and $\mathbf{u}$ do not lead to explicit formulation for the next iterated value $\mathbf{w}^{(p+1)}$ and $\mathbf{u}^{(p+1)}$. Therefore a Random Vector Functional-link net was introduced (Husmeier and Taylor, 1997) which fixes the values of $\mathbf{u}$ and makes an additional direct link between input and second hidden layer. In such case the non-linear function $m_{k_2}(\mathbf{x}_i, \mathbf{w}, \mathbf{u})$ remain non-linear in $\mathbf{u}$ but become linear in $\mathbf{w}$:

$$m_{k_2}(\mathbf{x}, \mathbf{w}, \mathbf{u}) = \mathbf{w}_{k_2}^T m_{k_2}(\mathbf{x}, \mathbf{u})$$

where we denote by $\mathbf{w}_{k_2}$ all the weights of the links feeding into the ${k_2}^{th}$ node of the second hidden layer and by $m_{k_2}(\mathbf{x}, \mathbf{u})$ the activities of the nodes in the input and in the first hidden layer. The derivatives of the term (34) now lead to an explicit result for the next iteration of $\mathbf{w}_{k_2}$ because the whole term is quadratic in $\mathbf{w}_{k_2}$. Thus

$$\nabla_{\mathbf{w}_{k_2}} Q = \sum_{i=1}^{N} 2\pi_{k_2}(i)(y_i - \mathbf{w}_{k_2}^T m_{k_2}(\mathbf{x}, \mathbf{u}))v_{k_2}$$

In terms of matrices we find the final solution as the linear system

$$(\mathbf{G^{(p)}} \mathbf{\Pi_{k_2}^{(p)}} \mathbf{G^{(p)}}^T) \mathbf{w}_{k_2}^{(p+1)} = \mathbf{G^{(p)}} \mathbf{\Pi^{(p)}}_{k_2} \mathbf{y} \quad \text{for each } w_{k_2}$$

where the $N \times K_2$ matrix $\mathbf{G^{(p)}}$ is a collection of the non-linear output from the first hidden layer plus the direct input (direct links). The $N \times N$ diagonal matrix $\mathbf{\Pi_{k_2}^{(p)}}$ contains the posterior probabilities $\pi_{k_2}^{(p)}(i)$ on its diagonals. The system is solved using a singular value decomposition as the matrix $\mathbf{G^{(p)}} \mathbf{\Pi_{k_2}^{(p)}} \mathbf{G^{(p)}}^T$ might become close to singular.

```
                  Parameters for NNSCAN
                  *********************

START OF PARAMETERS:
wal.tr              \file with image data
walsc4.tr           \file for output patterns
260 150             \nx, ny : size of 2D image x, y
temp34              \file with definition of template
12                  \number of values in template
2                   \pixel distance between each scanned pattern
```

Figure 26: *Parameter file* nnscan.par

16 Program set nnsim: nnscan nntrain nngen

A set of three FORTRAN90 programs constitutes the code nnsim for stochastic simulation using neural networks.

nnscan is a program for scanning an image to prepare a file for training the neural network.

Once patterns have been scanned the neural net is trained using program nntrain. This program requires the network architecture to be specified, i.e. the number of nodes in the first and second hidden layers. The user has the possibility to specify a generalization set (or test set) such that the training can be tested and compared under various network architectures or for calibration of the parameter σ_u (see section on EM-algorithm). The network outputs all trained parameters to a file.

Once the network has been learned, generalization takes place using program nngen. This program reads the network parameters from file and iteratively visits the image being simulated, updating each grid location with the Metropolis-Hastings-sampler criterion. A histogram must be provided as an array of values, preferably the array of values read exhaustively from the training image. For generating an initial random image, the program bootstraps from that histogram by random drawing with replacement. However, the option is given to start from an initial non-random image provided by the user. Such an image could be a realization from traditional simulation codes such as sgsim or sisim. The possibility is given to simulate on multiple grids, by specifying various grids and the respective network for each grid.

We will discuss in more detail the Walker Lake data set presented in the main text.

For program nnscan we retain the following set of parameters:

- **datafl:** the datafile containing the image in GSLIB format.
- **outfl:** the outputfile containing the scanned patterns from the image in datafl. The file states the number of patterns and the number of neighbouring data specified in the template file. 'd' in **outfl** indicates that it is a y-value.

```
8174          12
  -1.2187   d  <- y-value
  -1.4626
  -1.3971
  -2.0839
  -1.1055
  -1.3879            file: walsc4.tr
  -1.4159
  -2.6889
  -1.5660
  -1.4791      <- 12 x-values
  -1.5770
  -1.9727
  -0.9512
  -1.3133    d
  -1.4849
  -1.3836
       ....
```

- **nx, ny:** Size of the image you want to consider.
- **tempfile:** Template file: has to be specified as follows

```
 Template-file temp34

 4      0              Template :
 0      4
-4      0                      -8-4 0 4 8
 0     -4                           x
 4     -4                        x x x
-4      4                      x x y x x
-4     -4                        x x x
 4      4                           x
 0      8
 8      0
-8      0
 0     -8
```

```
                  Parameters for NNTRAIN
                  **********************

START OF PARAMETERS:
20                  \number of first-hidden-layer nodes
10                  \number of gaussian nodes
walsc4.tr           \training data set
walsc4.cv           \validation data set
walsc4.tr           \test or generalization data set
0.5                 \variance of Gaussian distr. for initial w weights
0.5                 \variance of Gaussian distr. for u weights (fixed)
100                 \maximum number of EM-steps
neuralw4.net        \file to output network parameters
```

Figure 27: *Parameter file* nntrain.par

When using multiple grids, the image must be scanned with a template that has the spacing of the grid, such is the case in template file temp34.

- **ntemp:** number of neighbouring values in the template file
- **ilag:** this specifies the lag-distance between scanned pixels. When the image is very large, one might not want to scan every pixel, but every two or three pixels. **ilag** provides this possibility.

Using the file walsc4.tr and walsc4.cv we can now train a neural network using nntrain.

For program nntrain we retain the following set of parameters:

- K_1**:** number of nodes in the first-hidden-layer.
- K_2**:** number of nodes in the second-hidden-layer.
- **trainfl:** training data set provided by nnscan.
- **crossfl:** validation data set provided by nnscan.
- **genfl:** test data set provided by nnscan.
- **sigmaini:** variance of the Gaussian distribution that initializes the weights **w**
- **sigmau:** variance of the Gaussian distribution that sets the fixed weights **u**.

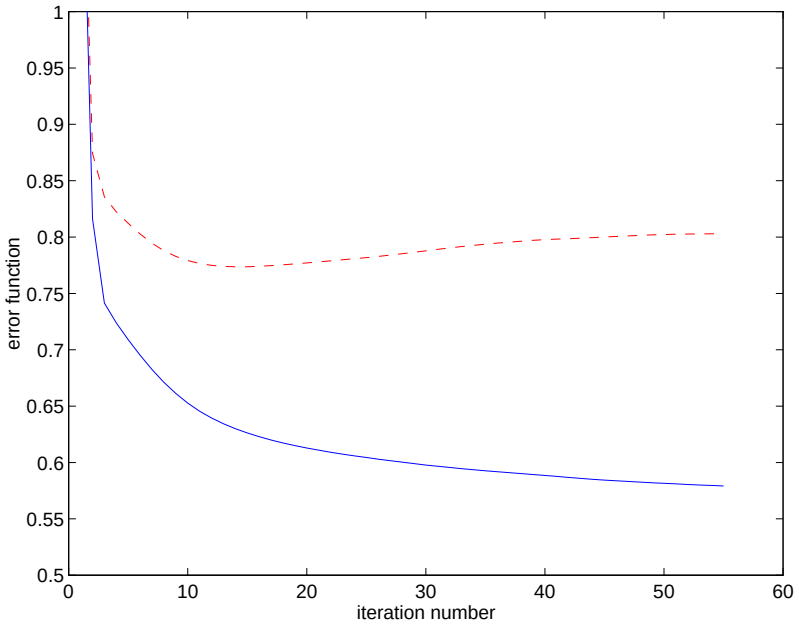

Figure 28: *Walker Lake: (—) training error function versus number of iterations in the EM-algorithm, (- -) cross-validation error function.*

- **maxiter:** maximum number of iterations in the EM-algorithm
- **netfile:** networkfile to store the neural network parameters.

A run of nntrain provides on screen the training and cross-validation log-likeliness. A typical run of the EM-algorithm finishes in 10-100 steps. Figure 28 shows that for the second grid the cross-validation error function is minimal after 15 iterations.

Figure 29 shows the 10 prior probabilities o_{k_2} and variances v_{k_2} for each node in the second hidden layer. There appears to be a negative correlation between prior probabilities and variances. Two histograms of weights $\mathbf{w}$ are shown in Figure 30. The first histogram depicts that part of the weights $\mathbf{w}$ that feed directly into the second hidden layer (the linear part in $\mathbf{x}$, see Figure 11), the second histogram is that part of the weights $\mathbf{w}$ given to the non-linear regression of $\mathbf{x}$. The spread of the weights given to the non-linear part is much larger than the spread of weights given to the linear part. Given all these parameters, the conditional distribution $f(y|\mathbf{x})$ can now be calculated for any $y|\mathbf{x}$.

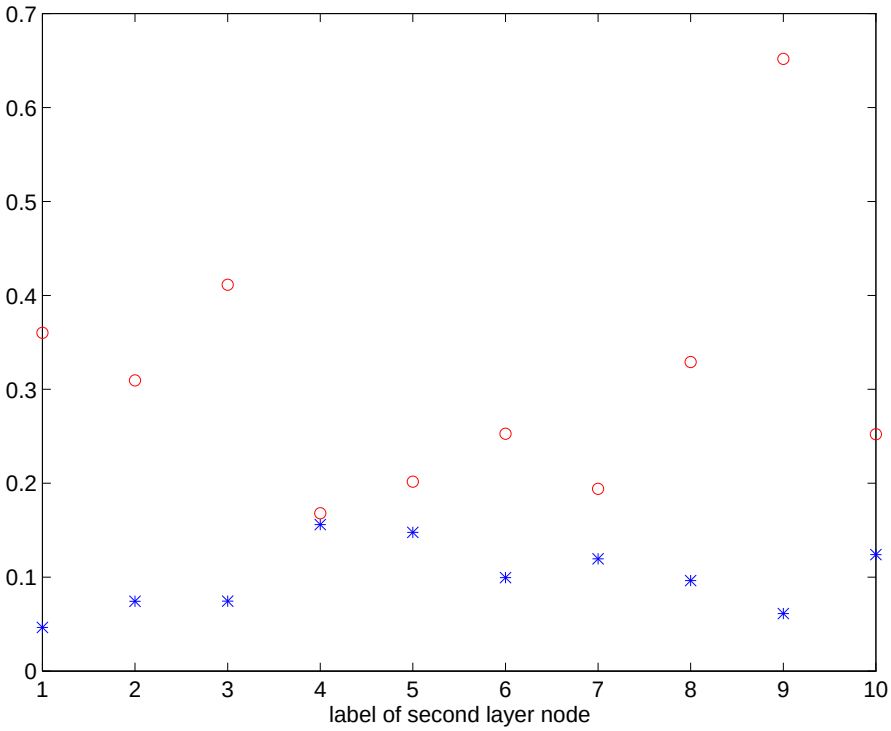

Figure 29: *Walker Lake: (o) prior probabilities o_{k_2}, (*) variances v_{k_2}.*

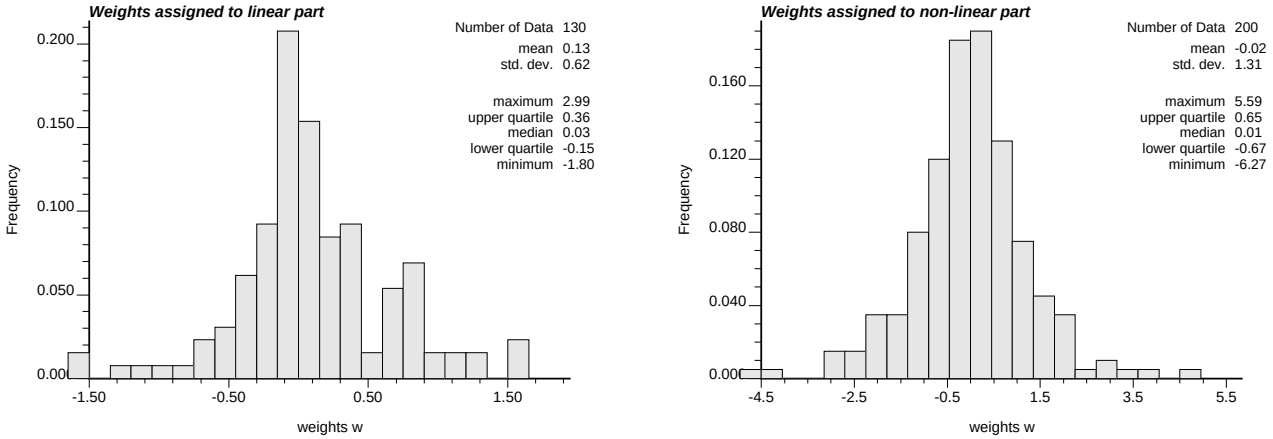

Figure 30: *Walker Lake: (left) histogram of weights* $\mathbf{w}$ *of linear part (right) non-linear part.*

```
                  Parameters for NNGEN
                  ********************

START OF PARAMETERS:
-8                    \seed of random path and drawing
nodatayet             \file with conditioning data
0                     \number of conditioning data to consider
end.im                \file to output image
1                     \number of realizations
0.005                 \scaling of the histogram object-function
false                 \true=prepared image, false=initial random
initial               \already prepared initial image
1                     \edgetype: 1=periodic, 2=ignore
y                     \want to make a movie?
moviefil              \output file for movie
100 100               \final grid definition for movie
nscore.out            \histogram file
78000                 \number of data to consider for histogram
1                     \column with variable in histogram file
200                   \number of percentiles for matching histogram
3                     \number of grids
neuralw4.net          \grid 1 : network parameters file
temp3                 \         template file
12                    \         number of neighbours in template
30 30 4               \         grid definition : nx ny space
200                   \         number of visits of whole image
neuralw2.net          \grid 2 : network parameters file
temp3                 \         template file
12                    \         number of neighbours in template
60 60 2               \         grid definition : nx ny space
200                   \         number of visits of whole image
neuralw1.net          \grid 3 : network parameters file
temp3                 \         template file
12                    \         number of neighbours in template
120 120 1             \         grid definition : nx ny space
200                   \         number of visits of whole image
```

Figure 31: *Parameter file* nngen.par

For program nngen we retain the following set of parameters:

- **seed:** Random number seed.
- **condfile:** GSLIB format file with conditioning data containing 3 columns with x-,y-coordinates and values. x cycles fastest.
- **ncond:** number of conditioning data to consider.
- **outfl:** GSLIB format file for storing multiple realizations.
- **nsim:** number of realizations
- **scaler:** coefficient k_B in equation (2). Usually between 0.01 and 0.001.
- **initfil:** file containing possible non-random images, this GSLIB format file must contain one column of data providing as many initial images as the number of realizations **nsim** specified.
- **edgetype:** **edgetype=1** means that the edges are wrapped. **edgetype=2** ignores pixels at the edges, they are never updated.
- **moviefil:** a file matlab m-file "movies.m" is provided to make a movie of one realization to check convergence of the Metropolis-Hastings-sampler. The m-file reads the file specified by **moviefil**.
- **histfl:** file containing the target histogram.
- **nhist:** number of data you want to consider in file histfl.
- **nperc:** number of percentiles you want to match.
- **ngrid:** number of grids for a multiple grid approach: for each grid we need
  - **netfl :** a file containing neural network parameters produced by nntrain.
  - **tempfl :** a template file, this file is in the coordinates of the current grid, so if you used temp34, you should now use file temp3 specified as

```
Template-file temp3

 1      0              Template :
 0      1
-1      0             -2-1 0 1 2
 0     -1                  x
 1     -1                x x x
-1      1              x x y x x
-1     -1                x x x
 1      1                  x
 0      2
 2      0
-2      0
 0     -2
```

- **ntemp:** number of neighbours in template
- **nx, ny, ispace:** definition of grid length in x, y-direction and the spacing between pixels.
- **maxvisit:** total number of visits of the whole image

The current programs only handle two-dimensional images. A three-dimensional extension is trivial and will be provided in the near future.