(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 2 973 241 B1**

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
21.10.2020 Bulletin 2020/43

(21) Application number: **14715977.6**

(22) Date of filing: **10.03.2014**

(51) Int Cl.:
G06N 3/04 *(2006.01)* ***G06N 3/08*** *(2006.01)*

(86) International application number:
PCT/GB2014/050695

(87) International publication number:
WO 2014/140541 (18.09.2014 Gazette 2014/38)

(54) SIGNAL PROCESSING SYSTEMS

SIGNALVERARBEITUNGSSYSTEME

SYSTÈMES DE TRAITEMENT DE SIGNAUX

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(30) Priority: **15.03.2013 GB 201304795**
24.06.2013 US 201313925637

(43) Date of publication of application:
20.01.2016 Bulletin 2016/03

(73) Proprietor: **DeepMind Technologies Limited**
London
EC4A 3TW (GB)

(72) Inventors:
- **CORNEBISE, Julien Robert Michel**
  London EC3M 5DJ (GB)
- **REZENDE, Danilo Jimenez**
  London EC3M 5DJ (GB)
- **WIERSTRA, Daniël Pieter**
  London EC3M 5DJ (GB)

(74) Representative: **Robinson, David Edward Ashdown**
Marks & Clerk LLP
1 New York Street
Manchester M1 4HD (GB)

(56) References cited:
EP-A2- 0 360 674 **US-A1- 2012 072 215**

- **MARC'AURELIO RANZATO ET AL: "On deep generative models with applications to recognition", COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2011 IEEE CONFERENCE ON, IEEE, 20 June 2011 (2011-06-20), pages 2857-2864, XP032038212, DOI: 10.1109/CVPR.2011.5995710 ISBN: 978-1-4577-0394-2**
- **Danilo Jimenez Rezende ET AL: "Stochastic Backpropagation and Approximate Inference in Deep Generative Models", , 16 January 2014 (2014-01-16), XP055159172, Retrieved from the Internet: URL:http://arxiv.org/abs/1401.4082**

EP 2 973 241 B1

Description

FIELD OF THE INVENTION

[0001] This invention generally relates to electronic hardware, software, and related methods for signal processing, in particular signals processing systems which generate data dependent on, and representative of, previously learnt example data.

BACKGROUND TO THE INVENTION

[0002] We will describe, in the main, signal processors which employ neural networks and other techniques to generate output data examples which match those previously learnt. For example the signal processor may be trained with many different examples of handwritten digits from zero to nine and may then be employed to randomly generate a new example from one of the learnt categories. Thus an output may be generated from a set of learnt distributions (of the training examples) and, in general, the categorisation of the training examples may also be learnt. We will also describe techniques which use an external input to select the category of output example generated, not by precisely specifying the category but instead by providing data which defines a 'context' for the training examples. The signal processor is trained using examples and their context and then afterwards context data can be used to bias the generation of output examples.

[0003] Signal processors of this general type have a range of applications. For example they can be used for prediction, with or without context, and thus have applications in many types of image and audio signal processing, as well as in control applications, for example predicting the position of a robot arm, as well as in other applications, for example evolutionary search techniques for, say, drug discovery. Examples of the signal processor/system may process data including, but not limited to: audio, video, image, game, sensor, actuator, control (including motor control), biological, physical, chemical, spatial, text, search, and other data.

[0004] It is known to use a Boltzmann machine to provide a so-called generative model as described, for example, in Salakhutdinov and Hinton, "Deep Boltzmann Machine", in Proceedings of the International Conference on Artificial Intelligence and Statistics, volume 5, pages 448-455, 2009 (http://www.cs.utoronto.ca/~rsalakhu/papers/dbm.pdf). However Deep Boltzman Machines require a great deal of processing power to implement. Ranzato et al., "On deep generative models with applications to recognition", Proceedings CVPR '11, pages 2857-2864, describes a deep generative model with multiple Restricted Boltzmann Machine layers, but this is also computationally intensive.

[0005] A Helmholtz machine can also be employed to provide a generative model, but whilst such machines have some interesting features in practice they learn very slowly and the output examples they generate are poor.

[0006] We will describe improved signal processors and related architectures which address both these problems.

SUMMARY OF THE INVENTION

[0007] The invention is set out in the appended independent claims.

[0008] In examples the relationship between the category vector and the probability vector, and also the stored category vectors themselves, have been learnt by training the signal processor using a set of examples, as described further below. The training system may comprise part of the signal processor or the variable parameters in the signal processor may be trained when an instance of the signal processor is created, and afterwards the signal processor may operate independently of a training module/system. In some preferred implementations the probability vector generation system operates to translate between a probability vector defining the probability of a set of output data points (output vector) and a compressed representation of the probability vector as a category vector. Thus a category vector is a compressed representation of a probability vector.

[0009] The data compression may be implemented using a neural network, in particular a deterministic (rather than stochastic) neural network. Employing a shared mapping system of this type reduces the number of parameters to be learnt by the signal processor since, in effect, the weights of the neural network are common to all the compressed representations (categories). Furthermore employing a deterministic neural network counterintuitively facilitates learning by facilitating a deterministic, closed-form calculation of the weights during training of the signal processor.

[0010] The skilled person will appreciate that, in this context, the reference to deterministic is to the weight calculation and does not preclude, for example, use of the 'dropout' technique to reduce the risk of complex co-adaptation where there are potentially many degrees of freedom (Hinton et al 'Improving Neural Networks by Preventing Co-adaptation of Feature Detectors', Arxiv: 1207.0580 v1, 3 July 2012). More generally in this specification, where reference is made to a 'deterministic' neural network this should be taken to include, for example, a neural network which employs dropout or similar techniques to reduce overfitting during training.

[0011] As previously mentioned, preferably the probability vector generation system comprises a deterministic neural

network, for example a non-linear multilayer perceptron. Here, by non-linear, it is meant that one or more layers of neurons in the network have a non-linear transfer function so that the network is not constrained to fit just linear data. The skilled person will recognise that, in principle, the mapping need not be performed by a neural network but may be performed by any deterministic function, for example a large polynomial, splines or the like, but in practice such techniques are undesirable because of the exponential growth in the number of parameters needed as the length of the input/output vectors increases.

[0012] The signal processor includes a context vector input to receive a context vector which defines a relative likelihood of each of the plurality of categories (for training examples and/or output examples). This provides an input to the stochastic selector so that the selection of a category of output example is dependent on the context (vector). Then the context vector, or data derived from the context vector, may be provided as a further input to the probability vector generation system, in examples as an additional vector input to the deterministic neural network. Thus an input layer of this network may have a first set of nodes to receive a category vector output from the memory storing these vectors, and a second set of nodes to receive the context vector.

[0013] In some preferred examples a length of the context vector may be different to the number of categories and a mapping unit is included to translate from one to the other. This mapping unit preferably comprises a second neural network, preferably a deterministic neural network, preferably non-linear (including a non-linear function applied to the signals from at least one layer of nodes). In examples this mapping unit comprises a second multilayer perceptron. The stochastic selector may then select a category according to a set of probabilities defined by a modified context vector of length K (the number of categories). In such a system, if there is no external context then the context vector, or the modified context vector of length K output from the mapping unit, may be defined to be constant (that is, setting the categories to be equally likely).

[0014] Where the context vector is not constant the context vector mapping neural network should have at least one hidden layer; similarly in examples the neural network in the probability vector generation system also preferably has at least one hidden layer although, depending upon the complexity of the data, two or more hidden layers may be preferable for this neural network. Providing a context vector input for the signal processor enables output examples from a learnt context to be provided. Although, typically, an output example may comprise a large number of data points (for example it may be an image), and the context vector will often be much shorter (for example 1-100 values), this is not essential. Thus in other implementations the context vector may be large, for example an image, and the output example small, for example defining a classification or category of the image. In this case the probability vector generation system may not be needed to provide data compression between the probability vector and category vector, in which case the probability vector generation system may, in effect, provide an identity operation (straight through connection). Data compression may then effectively be provided by the context vector mapping unit (A).

[0015] One particularly advantageous extension to the above described signal processor is to connect a sequence of the signal processors in a chain such that each successive signal processor receives a context vector from at least a previous signal processor in the chain, in embodiments from all the previous signal processors in the chain. More particularly, the context vector input to a signal processor in the chain may comprise data identifying the selection of the stochastic selector in the previous signal processor of the chain. In some sense this corresponds to a 'belief' the previous signal processor has regarding the output example to generate because what is provided is an example selected based on the likelihoods (distributions) it has learnt from the training examples. The selection of the stochastic selector may be provided to the next signal processor from various stages following the selection. Thus the information may be provided as a probability vector or as a category vector or, potentially, as a stochastic selection (sample) with data values chosen according to probabilities defined by the probability vector. It is preferable, however, to use the 'compressed' category vector level data as this reduces the number of parameters which the subsequent signal processor must learn and, in effect, leverages the compression mapping (MLP - multilayer perceptron - weights) learnt by the previous signal processor.

[0016] Thus it will also be appreciated that the output data from a signal processor for an output example may either comprise a category vector, or a probability vector (defining likelihood values for data points of the output example) which, if desired, may be employed for generating an output example. Additionally or alternatively the output may comprise an output example per se, with data point values selected stochastically according to corresponding probabilities defined by the probability vector.

[0017] Similarly the output from the chain of signal processors may either comprise a probability vector from the end processor of the chain and/or an output stochastic selector may be provided to generate an output example according to probabilities defined by this probability vector.

[0018] The skilled person will recognise that in a chain of signal processors the first signal processor in the chain may or may not have a context vector input, depending upon whether it is desired to make the signal processor chain dependent on an external context vector input.

[0019] The number of categories available in a signal processor is a design choice. In part this choice may be made dependent on a priori knowledge of the data - how many categories, very roughly, might be expected to be present. For

example with learnt hand written digits 10 different categories would be expected, for digits 0-9. In general, however, it is advantageous to provide for a very large number of categories and, in effect, allow the training of the signal processor to determine how many categories are needed. In theory there is a risk of over-fitting with such an approach (in effect the signal processor may simply memorise the training examples). In practice, however, this is not necessarily a problem and if it was could be addressed by, for example, dropout or imposing a sparse representation (on one or both neural networks) or in other ways, for example by detecting over fitting and adjusting (reducing) a number of free parameters. Thus it is generally desirable to make provision for a large number of categories.

[0020] In one approach a large number of categories may be implemented on a single signal processor, but with more than a few thousand categories this becomes computationally expensive. Counterintuitively it is much more computationally efficient to implement a relatively small number of categories on each processor of a chain of processors: with this approach the effective number of categories grows exponentially with the number of processors in the chain (the number of levels) whilst the computational cost of sampling from the structure grows linearly with the number of processors (levels), and the computational cost of training the chain grows sub-linearly with the number of levels. For example with, say, 20 categories and four levels there are effectively 20^4 =160,000 categories. There is not complete equivalence with this same number of categories implemented on a single processor but there is very little decrease in flexibility for a huge saving in computational cost. By way of illustration consider an example with two categories on each processor: The first processor splits the data domain into two (in general divided by some complex surface), the second processor then splits each of these categories within the data domain into two, the third processor splits each of the domains created by the first and second processors into two, and so forth. In effect the context vector received by a processor labels which of the available regions generated by previous processors the current processor is to split the category vector inherited from the previous processor provides this information in compressed form (it represents, for example, a compressed form of the image it has chosen). One processor receives the category vector which, for say an image, defines a compressed image which the previous processor believes should be the output example, and this is combined with a belief of the present processor regarding the output example image, the present processor adding detail. This process continues down the chain with sequential refinement of the output example.

[0021] There is also described a signal processing system for generating output examples from categories of a plurality of categories, wherein a distribution of training examples across said plurality of categories has been learnt by said signal processing system, the signal processing system comprising: a chain of signal processors, wherein each signal processor of the chain has learnt a distribution of said training examples across a limited number of categories less than said plurality of categories; wherein at least each said signal processor after a first said signal processor in the chain has a context input and is configured to generate an output example from said learnt distribution conditional on said context input; wherein each successive signal processor in said chain receives the output example from the preceding processor in the chain as said context input; wherein a first said input processor in said chain is configured to stochastically select a said output example according to its learnt distribution; and wherein a last said signal processor in said chain is configured to provide one or both of an output example and a probability distribution for stochastically selecting a said output example.

[0022] There is also described a method of signal processing to generate data for an output example from a plurality of learnt categories of training examples, the method comprising: storing a plurality of category vectors each defining a learnt category of training example; stochastically selecting a stored said category vector; generating a probability vector, dependent upon said selected category vector; and outputting data for said output example, wherein said output example comprises a set of data points each having a probability defined by a respective component of said probability vector.

[0023] As previously described, in some preferred examples of the method selection of a stored category vector is dependent upon category likelihoods defined by a context vector, in particular one provided by a preceding signal processor in a chain of signal processors.

[0024] In examples the stored category vectors and probability vectors, more particularly the probability vector generation system, comprise, that is are defined by, a learnt representation of real-world data. More generally the (output example) data may comprise one or more of: image data, sound data, signal data, sensor data, actuated data, spatial data, text data, game data and the like examples of the signal processor may be employed to generate/predict/classify or otherwise process such data.

[0025] A signal processor/method as described above may be implemented in hardware, for example as electronic circuitry, or in software, for example as code running on a digital signal processor (DSP) or on a general purpose computer system, or in a combination of the two. As the skilled person will appreciate, the signal processing we describe may be distributed between a plurality of coupled components in communication with one another. Processor control code and/or data (for example learnt parameter data) to implement examples may be provided on a physical (non-transitory) data carrier such as a disc, programmed memory, for example non-volatile memory such as Flash, or in Firmware. Code and/or data to implement examples may comprise source, object or executable code in a conventional programming language (interpreted or compiled) such as C, or code for a hardware description language such as Verilog.

[0026] There is also described a method of training a signal processor, signal processing system, or method, in

particular as previously described, the method comprising: presenting training examples to the signal processor system or method, wherein a said training example comprises a set of data points corresponding to data points of a said output example; computing from a said training example a set of responsibility values, one for each said category, wherein a said responsibility value comprises a probability of the training example belonging to the category, each category having a respective stored category vector; computing a gradient vector for a set of parameters of the signal processor, system or method from said set of responsibility values, wherein said set of parameters includes said stored category vectors and defines a shared mapping between said stored category vectors and a corresponding set of said probability vectors defining probability values for a said set of data points of a said output example; and updating said set of parameters using said computed gradient vector.

[0027] Examples of this training procedure are efficient in part because the category vectors represent a compressed version of the data, say image, space represented by the probability vectors. Because of this and, in examples, because the neural network between the category and probability vectors provides a shared parameterisation for the training examples, learning is relatively quick and computationally efficient. In effect the category vectors provide a reduced dimensionality codebook for the examples, say images.

[0028] Broadly speaking a responsibility value defines the likelihood of a category given an example set of data points; in examples this is computed from the probability of the set of data points given a category (preferably normalised by summing over all the available categories). In preferred examples the responsibility value is also conditional on the context vector so that parameters are learnt based on a combination of training examples and their context vector data. In examples the learnt set of parameters comprises the context vectors stored in memory (one per category) and the weights of the two neural networks, for the context and category vectors respectively (MLPs A and B later). The skilled person will appreciate that the aforementioned probability of an example set of data points given a category and context vector is a probability of the example given a category vector, context vector, and weights of the neural network connecting these to a probability vector, that is ***B(m,c)***. The skilled person will further appreciate that the calculation of this probability will depend upon the implementation of the neural network and also on the type of data. For example for binary data a binomial distribution applies and if b_i is the probability of bit i then:

$$p\left(x_i \middle| B\left(\mathbf{m},\mathbf{c}\right)\right) = b_i^{x_i}\left(1-b_i\right)^{1-x_i} \qquad x_i \in \{0,1\}$$

[0029] Ideally, when computing the gradient vector this would be computed over the entire set of training examples, but in practice one example or a 'minibatch' of a few examples is sufficient to provide a noisy but usable approximation to what the gradient would be if integrated over the full set of training examples. When updating the parameters the gradient vector is multiplied by a step size (η). In theory different step sizes may be employed with different parameters and η may be a diagonal or full matrix, but in practice this does not appear necessary. Since there may be many thousands of parameters (the parameters include the weights of the neural networks) it is convenient to chose the step size as a constant small number, say 0.001 (although, again, in theory the step size could be reduced towards 0 as training progresses, for example as a function of iteration number). In practice it is useful to chose a step size to be as large as practicable without the training procedure failing. Broadly speaking averaging the gradient vector over a minibatch corresponds to a change in step size.

[0030] Merely to give a feel for the numbers which may be involved, the output neural network (B) may have of order 10 input side nodes (category vector and context) and of order 1000 nodes in each of two hidden layers and an output 'visible' layer. The input side neural network (A) may have of order 100-1000 input layer (context vector) nodes, of order 1000 hidden layer nodes, and a number of output nodes equal to the number of categories, depending on the implementation say 10-10000. In some implementations the context vector may have length one, that is it may comprise a single, scalar value. As previously mentioned, a category vector may be relatively compressed, for example having a length of order 1-100.

[0031] The above described training procedure can straightforwardly be extended to a chain of processors since, in effect, each processor may be trained independent of the others except that it inherits a sample stored category vector from one (or all) previous signal processors in the chain. This sample is made stochastically, with the probability of selecting a category vector dependent on a corresponding responsibility value. Thus in this manner responsibilities are inherited from one processor in the chain to another although, in examples, a computed gradient vector is not inherited or shared between signal processors of the chain. In a modification of the procedure a gradient vector may be computed for a context vector of a processor of the chain and this may then be shared, more particularly accumulated, from one processor in the chain to a subsequent processor in the chain.

[0032] The previously described signal processors/systems may be considered as an architecture in which a stochastic node or nodes (the stochastic selection step of 1 of K categories, is followed by a deterministic neural network (B), which is then followed by a stochastic output stage (stochastic selection of a set of data points according to a probability vector).

This concept may be extended and generalised to provide a neural network architecture in which a (large) deterministic neural network is sandwiched or interleaved between layers of stochastic nodes. Such an approach can address previous difficulties with slow/poor training of deep stochastic neural networks.

[0033] Thus there is also described a neural network architecture, the architecture comprising: a first, input layer of stochastic nodes; a second, output layer of stochastic nodes; and a deterministic neural network connected between said input and output layer nodes.

[0034] Examples of this structure may be employed to propagate signals (features) both up and down through the layers of the deep neural network. Thus the structure is able to implement a (modified) Helmholtz machine which addresses the defects in conventional Helmholtz machines - which has stalled research in this field for a decade or more - providing both extremely fast, and also accurate, sampling.

[0035] Broadly speaking the deterministic neural network (which may optionally be sparse and/or employ dropout) learns an efficient representation of features from amongst training examples from which the stochastic neural network nodes can then select. For example the deterministic neural network may learn to distinguish between a man and a woman and thus, implicitly, the stochastic nodes are forbidden from selecting both a man and woman simultaneously, which is desirable real-world behaviour. By contrast without the deterministic intermediate structure a complicated set of interrelationships between features of say male and female faces would need to be learnt.

[0036] In examples the deterministic neural network includes one, two or more hidden layers, and in preferred implementations is non-linear as previously described.

BRIEF DESCRIPTION OF THE DRAWINGS

[0037] These and other aspects of the invention will now be further described, by way of example only, with reference to the accompanying figures in which:

Figures 1a to 1c show, respectively, a flow diagram of a signal processing method/system according to an example, a neural network architecture according to an example, and an example deterministic neural network for the method/system/architecture of Figures 1a and 1b;

Figure 2 shows a signal processing system comprising a chain of signal processors, according to an example;

Figures 3a and 3b show, respectively, a selection of examples from a set of training data, and a plot of values of K=100 category vectors or embeddings each having a dimension $d_m = 2$ and comprising two real, continuous values illustrating a compressed representation of the dataset from which the examples of Figure 3a are drawn, the 2D coordinates of each point representing an image;

Figure 4 shows output examples generated by successive signal processors in a chain of signal processors according to an example;

Figure 5 shows a block diagram illustrating a signal processor structure/architecture according to an example;

Figure 6 shows an example of a computer system programmed to implement the signal processors/processing chain of Figures 1 and 2; and

Figure 7 shows output examples generated by a chain of signal processors according to an example, illustrating image completion.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

[0038] Broadly speaking we will describe signal processing techniques which we term Compressed Mixtures and, for their powerful extension, Chained Compressed Mixtures. We will describe the structures and procedures implementing these techniques, and the algorithms by which the signal processors can be trained on real observed samples (learning) so that they can generate new, similar, samples (sampling). One advantage of Compressed Mixtures is that they can "imagine" very fast yet very accurately. Further, in examples all the required computations are available in closed form, allowing efficient learning.

[0039] As used herein, a generative model (GM) is computational machinery that learns to imagine. Its main purpose is to generate samples similar to those that it has observed.

[0040] More precisely, a GM is trained by observing a sequence of samples from an unknown real-world probability distribution, and generates new samples from an approximation to this probability distribution. For example, the GM

may be shown images from the NORB (New York University Object Recognition Benchmark) dataset which contains around 200,000 images of 50 different objects, and may then learn to generate new example images which look like the objects. As used herein, a Conditional Generative Model (CGM) additionally learns to generate new samples conditionally on a given context, that is, some data which accompanies the observation. Each observation can have its own context. Different contexts correspond to different distributions of the observations, and a CGM learns this and, given any specific future context, will generate new samples corresponding to the distribution associated with this context. For example, the context may specify the conditions under which an image was captured.

[0041] In general learning and sampling can be intertwined: sampling from the model can be done at any time; and it is possible to refine the model by learning from additional observations without needing to restart from scratch. Conversely, it is also possible to keep sampling without seeing any new observations: New observations are not necessary to generate new samples, thus allowing an extensively trained GM to be used in practical settings where its generative abilities are needed. Thus, for example, a signal processor implementing a generative model may be trained during some initial "calibration" stage, storing the learnt parameters in non-volatile memory, and may then be used as a self-contained module without its training system.

[0042] Because the Compressed Mixtures we describe can operate efficiently on very high-dimensional observation spaces, they are suitable to many domains of applications. The samples, observed or generated, can for example be either large static objects, as large as images, or entire time-series of smaller-dimensional objects where the high dimensionality stems from the number of time steps actually represented in a single observed series. The example training and output data values can be categorical (selected from a discrete number of categories), binary, discrete (eg 0-255), or even continuous. Some example applications are described later.

[0043] In broad terms we will begin by describing our "CMix" architecture and components, then how to sample from, and train this architecture. We then discuss some of its limitations and how these can be addressed by chaining multiple CMix processors, and describe how the chain of processors is trained.

Compressed Mixture processor (CMix)

[0044] A Compressed Mixture (CMix) signal processor defines a distribution, or a conditional distribution where context data is employed, over the sample space.

1. Notation

[0045] We consider an arbitrary d -dimensional sample space Ω^{d_V}. The space Ω is problem-specific, for example $\Omega = \{0,1\}$ in black and white images, $\Omega = \{0,...,255\}$ in greyscale images, or more generally $\Omega = R$ (ie is the set of real numbers).

[0046] We denote by $x \sim p$ the sampling of a realization of a random variable x from a probability distribution $p(\cdot)$. With a slight abuse of notation we do not distinguish between a random variable and its realization, nor between a distribution with its density.

[0047] Composition of functions is noted by $f \circ g(x) := f(g(x))$.

[0048] Vectors and matrices are represented in boldface, e.g. $\mathbf{x}$. The i-th component of vector $\mathbf{x}$ is denoted in subscript x_i, while the i-th row of a matrix $\mathbf{m}$ is noted $\mathbf{m}_i$. The vector of the components 1,2,..., i -1 of $\mathbf{x}$ is noted $\mathbf{x}_{<i}$. Superscript indexing $\mathbf{x}^i$ serves to denote a sequence of vectors.

2. Architecture

[0049] Referring to Figure 1a, this shows the architecture of a Compressed Mixture processor according to an example. Square boxes represent deterministic transformations, while ellipsoids represent sampling a random variable from a given distribution. The dashed boxes are expanded visions of the corresponding square box, illustrating an MLP (multilayer perceptron) in each.

[0050] In some preferred examples at the top of a CMix is vector $\mathbf{c} \in R^{d_C}$ (that is, a vector of dimension (length) d_c), we refer to as the context vector, input vector or conditioning vector. This represents the exogenous information we want to condition the sample on. It can be ignored (for example taken as constant) for a non-conditional generative model.

[0051] The density of the generative distribution for any data-point $\mathbf{x}$ in the visible space Ω^d conditional on a context vector $\mathbf{c}$, is:

$$p(\mathbf{x} \mid \mathbf{c}) = \sum_{k=1}^{K} p(\mathbf{x} \mid k, \mathbf{c}) p(k \mid \mathbf{c}),$$

where $p(\mathbf{x}|k,\mathbf{c})$ is a probability distribution on the visible space Ω^{d_V}, and $p(k|\mathbf{c})$ is a categorical distribution over the index

of classes {1,..., *K*}. Namely, the probability of occurrence of any class index *k* between 1 and *K* is defined as:

$$p(k \mid \mathbf{c}) = \mathrm{Cat}(k \mid \sigma \circ \mathbf{A}(\mathbf{c})),$$
$$= \sigma_k \circ \mathbf{A}(\mathbf{c})$$

where $\sigma_k(\mathbf{x}) = \exp x_k / \sum_j \exp x_j$ is the *k*-th component of the classical softmax function and A is a multilayer perceptron (MLP) with the appropriate input and output dimensions for d_c and *K*. There is no specific constraint on their number of hidden layers and hidden units.

[0052] Similarly, the distribution $p(\mathbf{x}|k,\mathbf{c})$ on the visible space is such that its sufficient statistic is the output of a second MLP **B** :

$$p(\mathbf{x} \mid k, \mathbf{c}) = p(\mathbf{x} \mid \mathbf{B}(\mathbf{c}, \mathbf{m}_k))$$

where $\mathbf{m}_k$ is the *k*-th row of a matrix $\mathbf{m} \in R^{K \times d_m}$. We refer to the rows of the matrix **m** are as embeddings.

[0053] The CMix model thus defines a mixture model of the "visible" distributions $p(\mathbf{x}|k,\mathbf{c})$, where the parameters of the components are shared through the MLPs **A** and **B**.

[0054] In examples the MLP **B** defines a non-linear compression between the dimension (length) d_v of the output example vector and the dimension (length) d_m of the category vector or embedding stored in matrix **m.** Referring to Figure 1a, in block 102 MLP **A** converts the context vector of dimension d_c to a vector of dimension *K*, where *K* is the number of categories of example employed by the signal processor. As previously mentioned, *K* may be chosen based on the application, or simply to be large or, in a chain as described later, to provide a large number of categories for the chain. MLP **A** preferably has at least one hidden layer; preferably this has a larger number of nodes than c, and preferably also a larger number of nodes than *K*.

[0055] In block 104 the "~" indicates that a *k is* chosen to choose a category: there are *K* categories and *k* takes a value indicating one of these. In examples, therefore, *k* may be represented by a vector of length *K* in which every component has a value of 0 except for component *k*, which may have a value of 1. All components may have an equal probability of being selected. Alternatively the context vector **c,** via **A** and a, may define, for each component of the vector, the probability of that component have a value of 1 (the probabilities being normalised so that the sum of these probabilities is equal to 1).

[0056] Block 106 represents a memory storing matrix **m.** This may be implemented as a table comprising *K* rows of data, each row storing d_m values, one for each component of a category vector or "embedding". One of these rows, and hence a category vector or embedding $\boldsymbol{m}_k$, is selected by the value of *k.* The category vector or embedding in effect represents a compressed representation of output example data from the processor.

[0057] In block 108 MLP **B** receives the category vector or embedding as an input together with, where used, the context vector **c** (or data derived from this). MLP **B** translates this input into a probability vector *p*(*p* in equation (4) above; p_v in Figure 1a, the "v" denoting the "visible" nodes of **B**), which has a dimension (length) d_v equal to the number of data points in the desired output example. For example, for an image with 4096 pixels, d_v would have length 4096.

[0058] The output of MLP **B** defines a probability (of a value, say 1 or 0) for each data point x_i of an output example. This probability vector may be used as the output from the signal processor as it effectively provides a representation of the output example (for example, for a processor trained on the NORB dataset, as described later, p_v effectively provides a greyscale image of one of the 50 model objects). However in examples an output example is generated, as indicated in block 110, by stochastically sampling this probability distribution i.e. values for data points are chosen according to the probability defined for each by p_v. By contrast, in the chained processors described later, the compressed representation $\boldsymbol{m}_k$ is used as the output example data from the processor.

[0059] The dimension (length) d_m of the category vector or embedding $\boldsymbol{m}_k$, is chosen according to the desired or sustainable degree of compression of the training/output data. Thus d_m may be chosen with some knowledge of the degree of compression which is potentially applicable to the dataset used with the processor and/or by routine experiment. In examples a high degree of compression is employed between d_v and d_m - for example with images compression by two or three orders of magnitude or more may be employed. However it is not essential for any compression to be employed - for example for a processor in a classification application the context vector may have the dimension of the image and the number of output data points/nodes may be low (one with, say, a continuous value, or a few, to classify the input into a few classes). In general, however, a significantly compressed representation is desirable.

[0060] The MLP B preferably has at least one hidden layer, and works best with two or more hidden layers. Preferably the number of nodes in its hidden layer(s) is at least equal to d_v

Extended archiecture

[0061] The architecture of Figure 1a may be considered as employing a pair of stochastic layers (indicated by ellipses in Figure 1a), the output stage 110 and the stochastic selection stage 104 - although selection stage 104 effectively comprises just a single stochastic node which may take 1 of K values.

[0062] This architecture may be extended to the more general architecture of Figure 1b. Thus Figure 1b shows a neural network 130 comprising a first stochastic layer 132, for example an input layer, a second stochastic layer 136, for example an output layer, and a deterministic neural network, **D,** 134 connected between the stochastic layers.

[0063] In Figure 1b the connections shown between nodes are illustrative only - the connections between nodes of different layers may be global or local. In the stochastic layers the node values may be drawn randomly from/in accordance with a probability distribution which may be defined, for example, by a weight matrix multiplying a non-linear function such as a sigmoid operating on an input vector.

[0064] Broadly speaking, the deterministic neural network D, learns to map features of the training data to patterns with the correct frequencies. Consider, for example, a simple version of D with 4 binary output nodes, which can therefore represent 16 patterns: if, say, a particular pattern should appear ¼ of the time the structure of Figure 1b will learn to map ¼ of the training data features to the same pattern. It will be appreciated that if D is made sufficient large then any mapping is possible. The structure will allocate the correct mapping for the correct frequencies of patterns.

[0065] Advantageously this structure may be employed to implement a Helmholtz machine-type training procedure, but other training procedures may also be employed. The deterministic nature of **D** simplifies training (in effect back-propagation may be employed to train **D**), avoiding the problems that occur with stochastic nodes in a Helmholtz machine, which result in a noisy gradient vector and thus very slow or stalled learning.

[0066] Preferably **D** is large and/or deep, that it is preferably has a large number of nodes in its one or more hidden layers, and/or two, three or more hidden layers. This provides greater representational power for the distribution(s), twisting and expanding these to a larger representational space. It may be constrained to be sparse (only a relatively small percentage of neurons activated by any particular feature, for example less than 20%, 15%, 10% of the neurons having greater than a threshold activation) and/or employ dropout. In effect, D acts as a feature learner for the training data and the stochastic layers operate on these learnt features.

Multilayer Perceptron (MLP)

[0067] An example deterministic neural network which may be used in the architectures of Figures 1a and 1b, for **A, B** and/or **D** is the multilayer perceptron (MLP). The skilled person will be aware of such devices (and further details can be found in, for example, C.M. Bishop, "Neural networks for pattern recognition", Oxford University Press, 1995) but for completeness we will outline an example structure.

[0068] A multilayer perceptron (MLP) is a deterministic function with a specific parametric structure alternating linear and non-linear operations, making it a universal function approximator: it can approximate any real-valued multivariate deterministic function $f : R_{\mathbf{c}}^{d} \rightarrow R_{\mathbf{v}}^{d}$, as long as it has been trained with enough couples **c**, *f*(**c**).

[0069] Figure 1c shows an example architecture of a MLP. This MLP has an input layer containing 2 units plus a bias unit, two hidden layers containing 4 and 6 units plus a bias unit each, and an output layer containing 4 units, with no bias needed at the output. (In principle a bias unit enables the representation of the constant term in $y = mx + c$ but in practice the bias units are optional, particularly in larger neural networks with many nodes in a layer). In Figure 1c arrows represent linear combinations, i.e. multiplications by a given weight and summation of all incoming arrows. Circles represent scalar units. Units labelled tanh operate a non-linear transformation of their input; units labelled 1 are constant bias units. The vector $\mathbf{c} = (c_1, c_2)$ is the input, while the output is collected into the vector $\mathbf{A}(\mathbf{c}) = (A_1(\mathbf{c}),..., A_4(\mathbf{c}))$.

[0070] More formally, a MLP is a composition of linear and non-linear operations on spaces of arbitrary dimensions, each such space being usually named a *layer*, and each component of each space being named a *unit.* A MLP A from $R_{\mathbf{c}}^{d}$ to $R_{\mathbf{v}}^{d}$ will therefore have one *input layer* with d_c unit, 1 *output layer* with d_v units, and an arbitrary number n_H of intermediate *hidden layers* of dimensions $d_{H,1},...,d_{H,n_H}$. Its precise form is the following composition of linear functions $\mathbf{H}^k$ and non-linear functions σ^k:

$$\mathbf{A}(\mathbf{c}) := \mathbf{H}^{n_H+1} \circ \sigma^{n_H} \circ \mathbf{H}^{n_H} \circ \sigma^{n_H-1} \circ \mathbf{H}^{n_H-1} \cdots \sigma^{1} \circ \mathbf{H}^{1}(\mathbf{c}).$$

[0071] The functions $\mathbf{H}^k$, for any k in $\{1,...,n_H + 1\}$, are affine transformations from $R^{d_{H,k-1}}$ to $R^{dH,k}$ where $d_{H,0} := d_c$ and $d_{H,n_H+1} := d_v$. More precisely, with a slight abuse of notation, we identify the function with a $d_{H,k} \times (d_{H,k-1} + 1)$ matrix and

define for any **x** in $R^{d_{H,k-1}}$:

$$\mathbf{H}^k(\mathbf{x}) := \mathbf{H}^k \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}.$$

[0072] The components of the matrices $\mathbf{H}^1 ... \mathbf{H}^{n_H+1}$ are the weights of the MLP, and are the free parameters that are trained by gradient ascent to approximate the function of interest.

[0073] The functions σ^k are non-linear functions from $R^{dH,k}$ to $R^{d_{H,k}}$, an activation function or "squashing function" since some common choices map to $[0,1]^{d_{H,k}}$. They are typically chosen as component-wise application of the hyperbolic tangent tanh or of the logistic sigmoid 1/(1-exp(-x))). This activation function is not applied to the output of the last hidden layer, to allow the output of the neural network to take any value in R^{d_V}. In practice, the choice of the number of hidden layers n_H, of their numbers of units $d_{H,k}$, and of the activation functions σ^k may be chosen by trial and error and practical considerations.

[0074] Training a MLP to approximate a function f amounts to choosing the adequate weights, i.e. the components of the matrices $\mathbf{H}^1 ... \mathbf{H}^{n_H+1}$. This is typically achieved by solving the minimization problem

$$\arg\min_{\mathbf{A}} \sum_{(\mathbf{x}, f(\mathbf{x}))} E(\mathbf{A}(\mathbf{x}), f(\mathbf{x}))$$

where the sum is over a training dataset of known pairs (**x**, *f*(**x**)), and $E(\mathbf{A}(\mathbf{x}), f(\mathbf{x}))$ is an error function that measures the divergence between **A**(**x**) and the known outputs f(x). This error function is, for example, a least-square error or a logarithmic loss function. The optimization algorithm employed to solve $\arg\min_{\mathbf{A}} \sum_{(\mathbf{x}, f(\mathbf{x}))} E(\mathbf{A}(\mathbf{x}), f(\mathbf{x}))$ is usually one of many variants of gradient ascent, evaluating the partial derivatives

$$\frac{\partial E(\mathbf{A}(\mathbf{x}), f(\mathbf{x}))}{\partial \mathbf{H}^k_{ij}}$$

by cautious application of the chain-rule of derivation. Such evaluation of the derivatives is referred to as back-propagation of the error.

3. Sampling

[0075] Referring again to the Compressed Mixture (CMix) signal processor of Figure 1a, we now describe a procedure for producing samples from this processor. This is a straightforward application of Equations (1), (3) and (4), and the sampling procedure is detailed in Algorithm 1, below, in terms of samples from a categorical distribution and from the *p* distribution:

Algorithm 1 - Generating a sample from a compressed mixture

[0076]

function GENERATESAMPLE($\mathbf{c}$)

$$\mathbf{p} \leftarrow \sigma \circ \mathbf{A}(\mathbf{c})$$

$$k \sim \mathrm{Cat}(\mathbf{p})$$

$$\mathbf{x} \sim p(\cdot \mid \mathbf{B}(\mathbf{c}, \mathbf{m}_k))$$

return $\mathbf{x}, \mathbf{m}_k$

end function

[0077] Here, in $k \sim \mathrm{Cat}(\mathbf{p})$ "~" denotes choosing a k from a set of K numbers according to probabilities $\boldsymbol{p}$, as previously described. It will be appreciated that in this sampling procedure $\mathbf{c}$ and $\boldsymbol{m}_k$ are known (from previous training).

[0078] For future convenience (i.e. use in a CMix chain), the procedure GENERATESAMPLE returns both a visible sample from the CMix model and the row in the embedding space which served to generate it, but in general the algorithm may return one or more of $\mathbf{x}, \mathbf{m}_k, p(\cdot|\mathbf{B}(\mathbf{c},\mathbf{m}_k))$ (the final sampling step $\mathbf{x} \sim p(\cdot|\mathbf{B}(\mathbf{c},\mathbf{m}_k))$ is optional). Optionally $\mathbf{c}$ may be constant, in which case the output represents the learnt values without context. As previously mentioned, $p(\cdot|\mathbf{B}(\mathbf{c},\mathbf{m}_k))$ may be discrete, continuous, bounded etc, in general any distribution whose sufficient statistics are those of an MLP, i.e. any distribution representable by the output of a MLP.

4. Learning

[0079] The CMix processor may be trained by learning the optimal value of its parameters using an online EM (expectation-maximisation) algorithm, which takes a straightforward form for this processor.

[0080] Here θ is a vector of all parameters in the CMix, i.e. the weights in the MLP A, the matrix $\mathbf{m}$ and the weights in the MLP $\mathbf{B}$. It will be appreciated that there may be many thousands of such parameters.

[0081] For any given data sample x, the first step of the EM procedure is to compute the gradient $\mathbf{G}^{\theta}(\mathbf{x},\mathbf{c})$ of $\log p(\mathbf{x}|\mathbf{c})$ with respect to the parameters θ:

$$\begin{aligned}\mathbf{G}^{\theta}(\mathbf{x},\mathbf{c}) &= \nabla_{\theta} \log p(\mathbf{x} \mid \mathbf{c}) \\ &= \nabla_{\theta} \log \sum_{k} p(\mathbf{x}, k \mid \mathbf{c}) \\ &= E\left[\nabla_{\theta} \log p(\mathbf{x}, k \mid \mathbf{c}) \mid \mathbf{x}\right] \\ &= \sum_{k=1}^{K} p(k \mid \mathbf{x}, \mathbf{c}) \nabla_{\theta} [\log p(\mathbf{x} \mid k, \mathbf{c}) + \log p(k \mid \mathbf{c})].\end{aligned}$$

[0082] Equality (7) is an application of the Fisher identity (see e.g. O. Capp_e, T. Ryden, and E. Moulines, "Inference in hidden Markov models", Springer, 2005, proposition 10.1.6, p. 353). (The notation in (8) with $\mathbf{x}$ on both sides of "|" denotes fixing $\mathbf{x}$ to its value on the right hand side and integrating).

[0083] The posterior mixture weights $p(k|\mathbf{x},\mathbf{c})$, are referred to as the responsibilities of each component of the mixture, and are the posterior distribution of the latent categorical index conditionally on the observation $\mathbf{x}$ and the context $\mathbf{c}$:

$$p(k \mid \mathbf{x}, \mathbf{c}) = \frac{p(\mathbf{x} \mid k, \mathbf{c}) p(k \mid \mathbf{c})}{\sum_{j=1}^{K} p(\mathbf{x} \mid j, \mathbf{c}) p(j \mid \mathbf{c})}.$$

[0084] The second step of the EM algorithm then proceeds to maximizing $\log p(\mathbf{x})$, hence the name *M-step.* We simply increment the parameters θ in the direction given by $\mathbf{G}^{\theta}$. Algorithm 2 describes this procedure, with the optional improvement that it accumulates the gradient over a minibatch of several randomly sampled observations 23 before proceeding to the M-step. A typical minibatch size may be of order 1-10 examples.

Algorithm 2 Training the Compressed Mixture

```
1: function trainCMix()
2:     while isLearning() do
3:         zeroGradParameters()
4:         {x^i, c^i}_{i=1}^N ← getNewDataMiniBatch()
5:         for i ← 1 ... N do
6:             accumulateGradients(x^i, c^i)
7:         end for
8:         updateParameters()
9:     end while
10: end function

11: function zeroGradParameters()
12:     G^θ ← 0
13: end function

14: function updateParameters()
15:     θ ← θ + ηG^θ
16: end function

17: function accumulateGradients(x, c)
18:     r ← computeResponsibilities(x, c)
19:     G^θ ← G^θ + Σ_{k=1}^K r_k ∇_θ [log p(x|k, c) + log p(k|c)]
20:     G^c ← Σ_{k=1}^K r_k ∇_c [log p(x|k, c) + log p(k|c)]
21:     return G^c,
22: end function

23: function computeResponsibilities(x, c)
24:     s ← 0
25:     for k ← 1 ... K do
26:         r_k ← p(x|k, c)p(k|c)
27:         s ← s + r_k
28:     end for
29:     return r/s
30: end function

31: function sampleEmbeddingFromPosterior(x, c)
32:     r ← computeResponsibilities(x, c)
33:     k ∼ Cat(r)
34:     return m_k
35: end function
```

[0085] The algorithmic complexity of a single training step of the CMix model scales as $O(K)$, i.e. linearly with the number of categories K. The procedure ACCUMULATEGRADIENTS at line 17 of Algorithm 2 can (optionally) return the gradient $\mathbf{G}^{\mathbf{c}}$ of $\log p(\mathbf{x})$ with respect to the context $\mathbf{c}$ (to allow propagation of the gradient through a Cmix chain).

[0086] The gradient $\mathbf{G}^{\theta}$ is not explicitly shown as being returned because, in examples, this is a global parameter of the algorithm. The vector of parameters θ may be initialised with random values; the parameter η represents the learning rate of the algorithm, and may be chosen by experiment. The responsibility vector r is preferably returned normalised (that is, divided by sum s as shown). The function SAMPLEEMBEDDINGFROMPOSTERIOR is included in Algorithm

2 although it is not part of the training procedure, because it is used later when chaining CMix processors.

[0087] In line 19, the calculation of $\mathbf{G}^{\theta}$ employs a calculation of $p(\mathbf{x}|k,\mathbf{c})$ and of $p(k|\mathbf{c})$. The term $p(k|\mathbf{c})$ may be determined from equation (3) (knowing the weights of **A**; and **c**). The term $p(\mathbf{x}|k,\mathbf{c})$ may be determined using equation (4), where **x** is known (a training example), **c** is known, and $\mathbf{m}_k$ and the weights of **B** are known (these are the parameters being optimised). The particular form of the calculation of $p(\mathbf{x}|k,\mathbf{c})$ depends on the type of distribution of **x** - for example whether it is Bernoulli (x_i is 0 or 1), Binomial (x_i is in the range 0 to 255, say), or Gaussian. An equation for $p(\mathbf{x}|k,\mathbf{c})$ may be determined analytically, for example by hand (in equation (4) we know the inputs to **B** and the probability is linear on the output of **B**), but in practice this always takes a simple form, linear (or for a Gaussian, polynomial) on **x**, and a function of a logarithm of **B**. Some examples are given below:

Bernoulli Case:

$x_i \in \{0, 1\}$

$$\log p(x|k,c) = \sum_{i=1}^{d} [x_i \log g(B(m_k,c)) + (1-x_i)\log(1-g(B(m_k,c)))]$$

$$g(x) = \frac{1}{1+\exp(-x)}$$

Binomial Case:

$x_i \in \{0, N\}$

$$\log p(x|k,c) = \sum_{i=1}^{d} [x_i \log g(B(m_k,c)) + (N-x_i)\log(1-g(B(m_k,c)))]$$

Compressed Mixture processor chain (CMixChain)

[0088] In practice the above described Compressed Mixtures are limited in the number of distinct samples they can generate by the processing cost. The number K of mixture components in the top layer is arbitrarily chosen and it could theoretically be very large without any impact on the constant $O(1)$ number of operations (algorithmic cost) required for sampling, this being $O(1)$. However, the number of operations of a single learning step grows as $O(K)$, i.e. linearly with the number of categories, making very large numbers of categories impractical.

[0089] We now describe techniques employing chained compressed mixture processors, which alleviate this problem by using the combinatorial explosion of successive compressed mixtures: the first level in the chain provides its sampled category as part of the context of the second level, which in turns passes this and its own sampled category as part of the context of the third level, and so on up to an arbitrary L levels. In practice a small number of levels have proven remarkably powerful. The cost of sampling grows as $O(L)$ with L being very moderate, while the learning cost grows as $O(L^2 \times K^{1/L})$, i.e. sub-linearly with the number of actual categories. Thus by chaining CMix processors a large increase in the number of actual categories that can be sampled from can be obtained, while keeping a scalable training cost using approximation in the EM algorithm by inheriting sampled categories as described.

[0090] Figure 2 shows the architecture of a signal processing system 140 comprising a chain of compressed mixture signal processors 100a,b,c. Apart (optionally) from the last, each compressed mixture signal processor has an output 107a,b to provide its chosen context vector or embedding $\mathbf{m}_k$ to the next processor in the chain; this is concatenated into the context vector provided to the next stage. In Figure 2 the solid arrows indicate the information flow for sampling (red, S) and during learning (blue, L). The calculation of responsibility $p(k|\mathbf{x},\mathbf{c})$ from $p(\mathbf{x}|k,\mathbf{c})$ (arrow L_1) and $p(k|\mathbf{c})$ (arrow L_2) is shown as a multiplication but will in general also involve a summation (for normalisation, as shown in line 27 of Algorithm 2). Although for convenience of representation the same internal dimensions (vector lengths) are suggested for the chained signal processors there is no obligation for these to be the same for each CMix processor - for example the processors could have different category vector/embedding dimensions d_m. and/or the sizes of the neural networks A and/or B could grow with progression down the chain. As indicated by the dashed line accompanying the sampling information flow in the final stage 100c, the output from the chain may be a stochastically selected output example or a probability vector defining probabilities of data points for such an output example.

1. Architecture

[0091] Continuing to refer to Figure 2, a CMix processor chain is constructed from a sequence of *L* CMix processors indexed by $\ell = 1...L$, referred to as the levels in the chain. A key feature of the chain of processors is that each successive CMix in the chain is conditioned on the samples of the preceding CMix processors, as illustrated, yielding a sequential refinement of the generated samples.

[0092] Note that $\mathbf{A}^\ell$, $\mathbf{m}^\ell$, and $\mathbf{B}^\ell$ denote the components of the CMix of level ℓ, and $\mathbf{k} = (k_1 ... k_L)$ is the concatenation of all indices $k_\ell \in \{1,...,K_\ell\}$ in the CMix of all levels in the chain, with a total of L levels. Each level can have a different number K_ℓ of categories. In embodiments the parameters $\mathbf{A}^\ell$, $\mathbf{m}^\ell$, and $\mathbf{B}^\ell$ belong to signal processor f in the chain, and each signal processor has its own parameters (matrix memory (**m**) and MLP weights).

[0093] We can write the joint distributions of any such vector, regardless of the architecture, as the product of sequential conditionals:

$$p(\mathbf{k} \mid \mathbf{c}) := \prod_{\ell=1}^{L} p(k_\ell \mid \mathbf{k}_{<\ell}, \mathbf{c}),$$

where $p(k_\ell|\mathbf{k}_{<\ell},\mathbf{c})$ are conditional categorical distributions defined similarly to (3), and $< \ell$ denotes the previous signal processors in the chain.

[0094] We define a CMixChain conditional distribution over $\Omega^d_{\mathbf{v}}$ as

$$p(\mathbf{x} \mid \mathbf{c}) = \sum_{\mathbf{k}} p(\mathbf{x}, \mathbf{k} \mid \mathbf{c}),$$

where

$$\begin{aligned} p(\mathbf{x}, \mathbf{k} \mid \mathbf{c}) &= p(\mathbf{x} \mid \mathbf{k}, \mathbf{c})\, p(\mathbf{k} \mid \mathbf{c}) \\ &= p(\mathbf{x} \mid \mathbf{k}, \mathbf{c}) \prod_{\ell=1}^{L} p(k_\ell \mid \mathbf{k}_{<\ell}, \mathbf{c}) \end{aligned}$$

and the distributions $p(\mathbf{x}|\mathbf{k},\mathbf{c})$ and $p(k_\ell|\mathbf{k}_{<\ell},\mathbf{c})$ are parametrized as

$$\begin{aligned} p(\mathbf{x} \mid \mathbf{k}, \mathbf{c}) &= p(\mathbf{x} \mid \mathbf{B}^L(\mathbf{c}^L)), \\ p(k_\ell \mid \mathbf{k}_{<\ell}, \mathbf{c}) &= p(k_\ell \mid \sigma \circ \mathbf{A}^\ell(\mathbf{c}^{\ell-1}, \mathbf{m}^\ell_{k_\ell})), \end{aligned}$$

where

$$\begin{aligned} \mathbf{c}^0 &:= \mathbf{c} \\ \mathbf{c}^\ell &:= (\mathbf{c}, \mathbf{m}^1_{k_1}, ..., \mathbf{m}^\ell_{k_\ell}) \text{ for all } \ell \geq 1 \end{aligned}$$

is the concatenation of the original context **c** and the embeddings chosen in the successive categories up to and including level ℓ. That is, at each level ℓ the ℓ-th CMix receives as input the concatenated sampled memories $(\mathbf{c}, \mathbf{m}^1_{\mathbf{k}_1}, ..., \mathbf{m}^{\ell-1}_{k_{\ell-1}})$ of all preceding CMix in the chain together with the global context c. Preferably the size and/or number of hidden layers in the neural nets $\mathbf{A}^\ell$ and $\mathbf{A}^\ell$ is increased as the level depth ℓ increases, to accommodate the growing size of the input.

2. Sampling

[0095] As illustrated in Figure 2, sampling from the CMixChain is performed by sequentially sampling and concatenating the embeddings (category vectors) $\mathbf{m}_k$ from the successive CMix levels until reaching the last level, which is then entirely

sampled though. This procedure is detailed in Algorithm 3, below, which uses the GENERATESAMPLE procedure from Algorithm 1, for a single CMix processor, to return a value **x** dependent on context **c.** In Algorithm 3 the returned embedding is denoted e.

Algorithm 3 Sampling from the CMixChain model

```
1: function GENERATESAMPLE(c)
2:     for ℓ = 1 → L do
3:         x, e ←CMix^ℓ:GENERATESAMPLE(c)
4:         c ← (c, e)
5:     end for
6:     return x
7: end function
```

[0096] Figure 3a shows observed samples from the NORB dataset used to train a CMixChain. To aid in understanding the operation of an example single CMix processor, Figure 3b shows a set of compressed representations of images from the dataset of Figure 3a, for a CMix processor with K=100 categories, each with category vector or embedding with a dimension d_m = 2 - that is each image of the training dataset is compressed such that it is represented by just two continuous real values. Figure 3b plots points identified by these two continuous values, using these values as x- and y-coordinates, and plotting 100 points, one for each category.

[0097] Figure 4 shows output examples generated by successive signal processors at levels l = 1,2,3,4 in a CMixChain of signal processors comprising 10 categories per level and 4 levels. These can be compared with the examples of Figure 3a, and illustrate the successive details typically added by each level in the chain. Figure 4 is to aid understanding of the operation of the chain - in an example of the signal processing chain only the output from the final processor of the chain would be provided externally.

3. Learning

[0098] Learning in the CMixChain model employs an approximation of the previously described EM procedure. Algorithm 4, below, details this training procedure; the underlying mathematical basis is described later. The complexity of this approximated algorithm scales as $O(LK(d_{\mathbf{c}} + d_{\mathbf{v}}) + L^2(K + d_{\mathbf{m}}))$ instead of the $O(K^L(d_{\mathbf{c}} + d_{\mathbf{v}} + d_{\mathbf{m}}))$ which would be the cost of the exact algorithm - i.e..the computation cost scales as L^2K rather than as K^L.

Algorithm 4 Learning in the CMixChain model

```
function trainCMix()
    while isLearning() do
        zeroGradParameters()
        {x^i, c^i}_{i=1}^N ← getNewDataMiniBatch()
        for i ← 1 ... N do
            accumulateGradients(x^i, c^i)
        end for
        updateParameters()
    end while
end function
function zeroGradParameters()
    for ℓ ← 1 ... L do
        CMix^ℓ:zeroGradParameters()
    end for
end function
function updateParameters()
    for ℓ ← 1 ... L do
        CMix^ℓ:updateParameters()
    end for
end function
function accumulateGradients(x, c)
    G^c ← 0
    for ℓ ← 1 ... L do
        F ← CMix^ℓ:accumulateGradients(x, c)
        G^c ← G^c + F_{≤d_c}
        e ← CMix^ℓ:sampleEmbeddingFromPosterior(x, c)
        c ← (c, e)
    end for
end function
```

[0099] Algorithm 4 uses the functions defined in Algorithm 2 for a single CMix processor; G_θ does not appear explicitly because it is part of the global parameters of a single CMix, although as previously mentioned, each CMix has its own set of parameters θ and gradient vector G_θ^l which, in examples, is not inherited from one CMix signal processor to the next. Thus in examples each CMix processor is trained independently and just the context vector c is inherited.

[0100] In Algorithm **4** the ACCUMULATEGRADIENTS function for G_c (the gradient with respect to the context **c**) is not required for the chain of signal processors and "memory" from the previous level is provided by inherited context. In principle, however, G_c could be used to inherit more information from one level to the next.

Learning formalism

[0101] To facilitate understanding we will now outline a mathematical justification for the training procedure of Algorithm 4 for a chain of CMix processors.

[0102] For simplicity, we derive here the computations in the case where the number of categories $K_\ell = K$ is constant across all layers. Extension to the general case is straightforward.

[0103] The algorithmic complexity of a single training step of the CMixChain model scales quadratically with ℓ as

$$O\Big(LK(d_{\mathbf{c}}+d_{\mathbf{v}})+L^2(K+d_{\mathbf{m}})\Big),$$

whereas that of a single CMix unchained with an equivalent number K^L of total categories would scale exponentially

with ℓ as

$$O\left(K^{L}(d_{\mathbf{c}}+d_{\mathbf{v}}+d_{\mathbf{m}})\right).$$

[0104] We recall that the gradient of the log-likelihood of a datapoint **x** associated to a generative model $p(\mathbf{x},\mathbf{k})$ with latent variables **k** can always be expressed as an expectation under the posterior over the latent variables

$$\begin{aligned}\nabla \log p(\mathbf{x}) &= \nabla \log \sum_{\mathbf{k}} p(\mathbf{x},\mathbf{k}) \\ &= E_{p(\mathbf{k}|\mathbf{x})}\left[\nabla \log p(\mathbf{x},\mathbf{k})\right],\end{aligned}$$

that is, computing the gradient $\nabla \log p(\mathbf{x})$ requires the knowledge of the posterior distribution $p(\mathbf{k}|\mathbf{x})$.

[0105] In the following we introduce a variational approximation $q(\mathbf{k}|\mathbf{x},\mathbf{c})$ to the posterior $p(\mathbf{k}|\mathbf{x},\mathbf{c})$ and a way of training its internal parameters in order to achieve the desired scaling properties (see, for example, M.J. Beal, "Variational algorithms for approximate Bayesian inference", PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003; and M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models", Machine Learning, 37:183-233, 1999).

[0106] The variational framework replaces the problem of maximizing the data log-likelihood

$$\begin{aligned}\theta^{\star} &= \arg\max_{\theta} \sum_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) \\ &= \arg\max_{\theta} \sum_{\mathbf{x}} \log \sum_{\mathbf{k}} p_{\theta}(\mathbf{x},\mathbf{k}),\end{aligned}$$

with a nested minimization

$$\begin{aligned}(\theta^{\star},Q^{\star}) &= \arg\min_{\theta,Q} E_{Q}\left[\log Q(\mathbf{k}\,|\,\mathbf{x}) - \log p_{\theta}(\mathbf{x},\mathbf{k})\right] \\ &=: \arg\min_{\theta,Q} F\end{aligned}$$

where Q is a distribution over the latent variables **k** referred to as a variational distribution and F is defined as the variational free energy.

[0107] For the model defined in Equation (13), we start by defining a variational distribution $Q(\mathbf{k}|\mathbf{x},\mathbf{c})$ in a factorized form

$$Q(\mathbf{k}\,|\,\mathbf{x},\mathbf{c}) := \prod_{\ell=1}^{L} Q_{\ell}(k_{\ell}\,|\,\mathbf{x},\mathbf{c},\mathbf{k}_{<\ell}).$$

[0108] The free energy associated to this variational posterior is given by

$$F = E_{Q}\left[\log Q(\mathbf{k}\,|\,\mathbf{x},\mathbf{c}) - \log p(\mathbf{x},\mathbf{k}\,|\,\mathbf{c})\right]$$

[0109] Minimizing F with respect to q_{ℓ} under the constraint that it is a probability distribution yields the closed-form solution

$$Q_{\ell}^{\star}(k_{\ell}\,|\,\mathbf{x},\mathbf{k}_{<\ell},\mathbf{c}) \propto p(k_{\ell}\,|\,\mathbf{k}_{<\ell},\mathbf{c}) \exp\left\{E_{Q^{\star}}\left[\log p(\mathbf{x}\,|\,\mathbf{k},\mathbf{c})\,|\,\mathbf{k}_{\leq\ell}\right]\right\},$$

where $\propto$ denotes equality up to a normalization constant. This result can be written as

$$Q_\ell^\star(k_\ell \mid \mathbf{x}, \mathbf{k}_{<\ell}, \mathbf{c}) = \frac{p(k_\ell \mid \mathbf{k}_{<\ell}, \mathbf{c}) f_\ell(\mathbf{x} \mid \mathbf{c}, \mathbf{k}_{\le\ell})}{\sum_{j=1}^{K} p(k_\ell = j \mid \mathbf{k}_{<\ell}, \mathbf{c}) f_\ell(\mathbf{x} \mid \mathbf{c}, \mathbf{k}_{<\ell}, k_\ell = j)}$$

where the quantity

$$f_\ell(\mathbf{x} \mid \mathbf{c}, \mathbf{k}_{\le\ell}) := \exp\left\{E_{Q^\star}\left[\log p(\mathbf{x} \mid \mathbf{k}, \mathbf{c}) \mid \mathbf{k}_{\le\ell}\right]\right\}$$

can be seen as an unnormalized distribution over the visible **x** conditionally on the **c** and on the chosen categories $\mathbf{k}_\ell$ from the variational distribution up the ℓ-th level.

[0110] The variational posterior distribution *Q** obtained in this manner would correspond exactly to the true posterior $p(\mathbf{k}|\mathbf{x},\mathbf{c})$. We can as well identify the correspondence between the solution (35) and more common forward-backward algorithms by noting that the first factor in the numerator of Equation (35) is a backward recursion, while the second factor is a forward simulation of Equation (10) as indicated below:

$$Q_\ell^\star(k_\ell \mid \mathbf{x}, \mathbf{k}_{<\ell}, \mathbf{c}) \propto \overbrace{p(k_\ell \mid \mathbf{k}_{<\ell}, \mathbf{c})}^{\text{forward}} \underbrace{\exp\left\{E_{Q^\star}\left[\log p(\mathbf{x} \mid \mathbf{k}, \mathbf{c}) \mid \mathbf{k}_{\le\ell}\right]\right\}}_{\text{backward}}.$$

[0111] The next step of our derivation is to approximate the expectation

$$E_{Q^\star}\left[\log p(\mathbf{x} \mid \mathbf{k}, \mathbf{c}) \mid \mathbf{k}_{\le\ell}\right].$$

[0112] Since this expectation depends only on $Q_{\ell'}^\star$ for $\ell' > \ell$, we could obtain the exact $Q_\ell^\star$ by solving recursively the equations for $Q_{\ell'}^\star$, starting at the last level $\ell = L$ and going back to the first level $\ell = 1$. This exact solution is not tractable, having an algorithmic complexity $O(K)$. Moreover, the EM algorithm is modifying the parameters of $p(\mathbf{x}|\mathbf{k},\mathbf{c})$ so that $f_\ell(\mathbf{x}|\mathbf{c},\mathbf{k}_{\le\ell})$ defined in Equation (36) approaches the empirical distribution. This learning goal is the same as for the EM iterations for a single CMix model. Therefore, we can adopt the following approximation:

$$E_{Q^\star}\left[\log p(\mathbf{x} \mid \mathbf{c}, \mathbf{k}) \mid \mathbf{k}_{\le\ell}\right] \approx \log p^\ell(\mathbf{x} \mid \mathbf{k}_{\le\ell}, \mathbf{c}),$$

where $p^\ell(\mathbf{x}|\mathbf{k}_{\le\ell},\mathbf{c})$ is the observation model of the ℓ -th CMix model in the chain.

[0113] Replacing this approximation into Equation (35) yields

$$Q_\ell^\star(k_\ell \mid \mathbf{x}, \mathbf{k}_{<\ell}, \mathbf{c}) = \frac{p(k_\ell \mid \mathbf{k}_{<\ell}, \mathbf{c}) p^\ell(\mathbf{x} \mid \mathbf{k}_{\le\ell}, \mathbf{c})}{\sum_{j=1}^{K} p(k_\ell = j \mid \mathbf{k}_{<\ell}, \mathbf{c}) p^\ell(\mathbf{x} \mid \mathbf{k}_{<\ell}, k_\ell = j, \mathbf{c})}.$$

[0114] The approximated solution (40) for *Q** has the same form as the posterior distribution of a single CMix model given in Equation (9). It therefore allows us to re-use the distribution p^ℓ as well as the machinery to learn it inside each CMix in the CMixChain in a modular manner.

[0115] The full variational distribution *Q** thus becomes

$$Q^\star(\mathbf{k} \mid \mathbf{x}, \mathbf{c}) \approx \prod_{\ell=1}^{L} p(k_\ell \mid \mathbf{x}, \mathbf{k}_{<\ell}, \mathbf{c}),$$

where $p(k_\ell|\mathbf{x},\mathbf{k}_{<\ell},\mathbf{c})$ is computed internally by the ℓ-th CMix model given the currently observed data sample **x** and the input from all the precedent CMixes in the chain concatenated with the global context **c**. The maximization in Equation

(31) with respect to the remaining parameters not belonging to Q is performed by gradient ascent of Equation (27), where each parameter update may be computed using a single sample from Q. The resulting procedure is detailed in Algorithm 4.

Example implementation

[0116] Figure 6 shows a schematic block diagram of the structure of electronic hardware/software modules to implement a CMix signal processor 100 as previously described.

[0117] Thus the context data c is provided to a context vector mapping unit 112, implemented by MLP **A,** as well as to MLP **B** of a probability vector generation system 118; these correspond to block 102 and 108 of Figure 1a and implement corresponding functions. The context data is also provided to a training module 122.

[0118] The mapping unit A provides a K-wide output to a stochastic category selector 114, which has a function corresponding to block 104 of Figure 1a, and this in turn provides category selection data, for example in the form of an index or address, to category (embedding) vector memory 116 storing matrix **m.**

[0119] Memory 116 provides a d_m-wide output to probability vector generation system 118, having a function corresponding to block 108 of Figure 1a and implemented by MLP **B.** System 118 also receives context vector **c,** and provides a d_v-wide probability vector output to an optional stochastic output selector 120, which samples from the distribution defined by the probability vector p_v to provide a sampled output example **x** (corresponding to block 101 of Figure 1a).

[0120] Training module 122 receives training data on input 124, and optionally context data **c,** and implements the training procedure of Algorithm 2 to update the weights of MLP **A** (parameters θ_A), the weights of MLP **B** (parameters θ_B), and the category vectors or embeddings stored in memory 116 (parameters θ_m). Training module 122 need not be part of the signal processor 100 - for example the parameters θ could be trained by an external system which is afterwards removed, or the signal processor may be merely programmed with predetermined values, for example storing these into permanent memory such as read-only memory, non-volatile RAM such as Flash™, or on a disk.

[0121] The skilled person will appreciate that the structure of Figure 5 may be implemented in electronic hardware/circuitry. For example it may be defined in a hardware definition language and complied into hardware, or it may be implemented in an ASIC or FPGA. Alternatively some or all of the illustrated blocks may be implemented using a program-controlled signal processor, which may form part of the hardware, for example by including a processor block on an ASIC/FPGA. Alternatively the structure of Figure 5 may be implemented by software modules running on a digital signal processor (DSP) or on, say, a graphics processing unit (GPU). Still further alternatively the structure of Figure 5 may be implemented on a general purpose computer system, or across a plurality of coupled computing systems, for example implementing a high performance computing system.

[0122] Figure 6 shows a general purpose computer system 150 programmed to implement a chain of CMix signal processors as illustrated in Figure 2. Thus the computer system comprises a CMix server 152 including a processor and working memory. Server 152 also includes non-volatile memory 154 storing processor control code to implement a plurality of CMix signal processors 100 of the type shown in Figures 1a and 5, as well as code to sample from the CMix chain to provide an output, and code to implement the training procedure of Algorithms 2 and 4. Server 152 is also coupled to non-volatile storage 156 which stores weights for neural networks **A** and **B** and the embeddings of matrix **m**. The code/data in memory 154 and storage 156 may be provided on a removeable storage medium, illustratively shown as disk 158.

[0123] The CMix server 152 is provided with input data, optionally with associated context data. The input data may be of any type include but is not limited to one or more of: game/search/multimedia data, real-world/sensor data, and external signal data. Applications also include time-series data, training on a temporal series of examples, albeit the examples may be treated as effectively independent rather than as a time-series succession per se. Such time series data may be of any type including the aforementioned types, as well as time-series image (video) data, audio data, weather and other physical/chemical and/or biological data, financial data, and so forth. The neural network server 152 similarly provides corresponding output data, based on the training examples it has learnt and optionally context data provided to the server.

[0124] A user, and/or robot/machine, and/or other computer system(s)/CMix processor(s)/chain(s) may interact with the neural network server 152 to provide input data and/or receive output data via network 160, which may include the Internet. By way of illustration, a user terminal 162, robot/machine 164 and link to other network(s)/computer system(s) 166 are shown in Figure 6.

Example applications

[0125] The CMix signal processors we describe can be employed in a wide range of domains and provide good representative power combined with rapidity of sampling. We describe below some example applications in which these features are advantageous; these merely illustrative and are non-exhaustive. A CMix processor may be trained in a

supervised or unsupervised manner.

1. "Imagining" elements from a category

[0126] Straightforward sampling from the learned conditional distribution **p**(**x**|**c**) can be used to simulate imagination: When trained on labelled data, learning with the label as context **c** and the object as sample **x**, sampling from $p(\mathbf{x}|\mathbf{c})$ outputs example data for, or "imagines", an object from a given category.

2. Classifying objects amongst categories

[0127] Conversely, training a CMix processor (or chain) with the label as observation **x** and the object as the context **c**, then sampling from $p(\mathbf{x}|\mathbf{c})$ turns the CMix processor into a classifier, predicting the category of an unlabelled object.

[0128] For example in a supervised training process for recognising the digits 0-9 it may be known that a particular image corresponds to, say, a "2" and the CMix processor may be trained with the image as the context and **x** denoting the recognised digit. In such a case **x** may be a scalar variable with a range of values denoting different digits, or it may be, say, a vector of length 10 with binary-valued components.

3. Completion

[0129] In another example application a CMix processor or chain can observe part of an image, and be asked to "imagine" or complete the full-image which best matches the partial image provided. Thus Figure 7 shows examples of image completion by a CMixChain. This illustrates, on the left, 4 test samples (never seen by the model before) to be completed. The yellow region (to the right of the dashed line) indicates the pixels that have been occluded to the model. On the right is shown the CMixChain's "best guess" for the occluded pixels - it can be seen that the result is representative of the occluded portion of the input to be completed. This technique can also be applied to completion of time series for missing data, smoothing, and the like.

[0130] In more detail, completion involves sampling from another conditional than $p(\mathbf{x}|\mathbf{c})$. Instead, for a given context **c,** we observe only a part $\mathbf{x}_\mathbf{v}$ of the object (e.g. half of the pixels in the image) while the rest of the image, $\mathbf{x}_\mathbf{h}$ is hidden from view. Here **v** and **h** are two disjoint sets of indices such that $\mathbf{x} = (\mathbf{x}_\mathbf{v}, \mathbf{x}_\mathbf{h})$. The only requirement is that the hidden part $\mathbf{x}_\mathbf{h}$ and the visible part $\mathbf{x}_\mathbf{v}$ are independent of each other conditionally on the context and the category; that is, the distribution $p(\boldsymbol{x}|k,\mathbf{c})$ can be factorized as

$$p(\mathbf{x} \mid k, \mathbf{c}) = p(\mathbf{x}_\mathbf{v} \mid k, \mathbf{c}) p(\mathbf{x}_\mathbf{h} \mid k, \mathbf{c}).$$

[0131] Such a factorization is typically the case in image generation without lateral connection in the bottom layer of the neural net **B**. For example the distribution of Equation (4) factorizes as a product over the pixels where each pixel follows for example a Bernoulli or a Binomial distribution whose parameter is the output of the MLP **B**:

$$p(\mathbf{x} \mid \mathbf{B}(\mathbf{c}, \mathbf{m}_k)) = \prod_{i=1}^{d_\mathbf{v}} p(x_i \mid B_i(\mathbf{c}, \mathbf{m}_k)).$$

[0132] By way of illustration two examples are given:

Black and white image: if we are modelling a vector **x** of binary data $x_i \in \{0,1\}$, for example pixels in a black and white image, we map each output unit B_i of the MLP **B** to the interval [0,1] by applying the sigmoid function

$$g(b) := \frac{1}{1+\exp(-b)},$$

and use the result as the parameter *q* of a Bernoulli distribution with density

$$\mathrm{Ber}(x \mid q) := q^x (1-q)^{1-x}$$

used to model the corresponding pixel x_i in the image. This leads to the full equation

$$p(\mathbf{x} \mid \mathbf{B}(\mathbf{c}, \mathbf{m}_k)) = \prod_{i=1}^{d_\mathbf{v}} g(B_i(\mathbf{c}, \mathbf{m}_k))^{x_i} (1 - g(B_i(\mathbf{c}, \mathbf{m}_k)))^{1-x_i} .$$

or equivalently, in the log-form used by the learning algorithm as described in connection with line 19 of Algorithm 2:

$$\log p(\mathbf{x}|\mathbf{c}, \mathbf{m}_k) = \sum_{i=1}^{d_\mathbf{v}} x_i \log g(B_i(\mathbf{c}, \mathbf{m}_k)) + (1 - x_i) \log (1 - g(B_i(\mathbf{c}, \mathbf{m}_k)))$$

Grayscale image: in another example we model a vector **x** of value between 1 and some value *N*, for example grayscale images for which *N* = 255. We then use the same sigmoid transformation of the output units, and use this image as the parameter of a Binomial distribution with second parameter *N*:

$$\mathrm{Bin}(x \mid q, N) := q^x (1-q)^{N-x} ,$$

leading to the full equation

$$p(\mathbf{x} \mid \mathbf{B}(\mathbf{c}, \mathbf{m}_k)) = \prod_{i=1}^{d_\mathbf{v}} g(B_i(\mathbf{c}, \mathbf{m}_k))^{x_i} (1 - g(B_i(\mathbf{c}, \mathbf{m}_k)))^{N-x_i} .$$

or equivalently, in the log-form used by the learning algorithm as described in connection with line 19 of Algorithm 2:

$$\log p(\mathbf{x}|\mathbf{c}, \mathbf{m}_k) = \sum_{i=1}^{d_\mathbf{v}} x_i \log g(B_i(\mathbf{c}, \mathbf{m}_k)) + (N - x_i) \log (1 - g(B_i(\mathbf{c}, \mathbf{m}_k)))$$

[0133] This image completion problem can be written as sampling from the distribution $p(\mathbf{x_h}|\mathbf{x_v},\mathbf{c})$ using the learned GM $p(\mathbf{x}|\mathbf{c})$. For a single CMix model computing $p(\mathbf{x_h}|\mathbf{x_v},\mathbf{c})$ is straightforward. From Equation (1) we get

$$\begin{aligned} p(\mathbf{x}_\mathrm{h} \mid \mathbf{x}_\mathrm{v}, \mathbf{c}) &= \frac{\sum_k p(\mathbf{x}_\mathrm{v}, \mathbf{x}_\mathrm{h} \mid k, \mathbf{c}) p(k \mid \mathbf{c})}{\sum_{\mathbf{x}_\mathrm{h}} \sum_k p(\mathbf{x}_\mathrm{v}, \mathbf{x}_\mathrm{h} \mid k, \mathbf{c}) p(k \mid \mathbf{c})} \\ &= \sum_k p(\mathbf{x}_\mathrm{h} \mid k, \mathbf{c}) p(k \mid \mathbf{x}_\mathrm{v}, \mathbf{c}), \end{aligned}$$

where we have used the fact that the different pixels of the image are independent of each other given *k* and $p(k|\mathbf{x_v},\mathbf{c})$ is given by

$$p(k \mid \mathbf{x}_\mathrm{v}, \mathbf{c}) = \frac{p(\mathbf{x}_\mathrm{v} \mid k, \mathbf{c}) p(k \mid \mathbf{c})}{\sum_j p(\mathbf{x}_\mathrm{v} \mid j, \mathbf{c}) p(j \mid \mathbf{c})}.$$

[0134] The marginal observation likelihood $p(\mathbf{x_v}|k,\mathbf{c})$ may be computed by simply ignoring the factors corresponding to the unobserved pixels in Equation (4). A procedure for sampling from the distribution $p(\mathbf{x_h}|\mathbf{x_v},\mathbf{c})$ for a single CMix signal processor is detailed in Algorithm 5, below:

Algorithm 5 Completing missing data with CMix

```
function GENERATECOMPLETION(x_v, c)
    for k ← 1...K do
        p(x_v|k, c) ← ∏_{i∈v} p(x_i|B(c, m_k))
    end for
    for k ← 1...K do
        r_k ← p(k|x_v, c)
    end for
    k ~ Cat(r)
    x_h ~ p(x_h|k, c) = ∏_{i∈h} p(x_i|B(c, m_k))
    x ← (x_v, x_h)
    return x, m_k
end function
```

[0135] For a CMixChain signal processing system, the procedure is similar (although approximated). In this case, Algorithm 5 may be applied successively from the first level $\ell = 1$ to the last level $\ell = L$, as detailed below in Algorithm 6:

Algorithm 6 Completing missing data with CMixChain

```
function GENERATECOMPLETION(x_v, c)
    for ℓ ← 1...L do
        x, e ←CMix^ℓ:GENERATECOMPLETION(x_v, c)
        c ← {c, e}
    end for
    return x
end function
```

4. Classification via learning joint distribution

[0136] An alternative form of classifier/imagining system may be implemented by training a CMix processor or chain on examples which include, together in the same example, an example and its label. Thus, for example, an image of an object may include text with the name of the object; or a label may be concatenated with a training example vector. The CMix processor/chain may then be trained on the joint examples and labels, thus learning to recreate the missing part of the example, imagining an object or providing a label classifying an object, in both cases thereby completing an input.

[0137] More generally, therefore, when learning labelled data, the Cmix processor/chain may be used to process a concatenation of the label and the object as the observation, and learn their joint distribution. Using the completion Algorithms described above then allows both imagination and classification in a unified manner.

[0138] Usefully, learning this joint distribution also allows for semi-supervised learning, i.e. learning from datasets where only certain objects are labelled and many others are not. This facilitates access to very rich sources of training data.

[0139] No doubt many other effective alternatives will occur to the skilled person. It will be understood that the invention is not limited to the described examples and encompasses modifications apparent to those skilled in the art lying within the scope of the claims appended hereto.

Claims

1. A signal processing system for image generation, wherein the signal processing system is configured to generate an image conditional on a context defined by a context vector (c), wherein said context vector defines a relative likelihood of each of a plurality of categories of images, and wherein the signal processing system is configured to generate an image from a selected category, the signal processing system comprising:

a probability vector generation system (**B**; 108), wherein said probability vector generation system has an input to receive a category vector ($\mathbf{m}_k$) for a category (k) of image output example, and an output to provide a probability vector (p_v) for said category of image output example, wherein said category vector comprises a compressed representation of said probability vector, wherein said image output example comprises a set of data points, and wherein said probability vector defines a probability of each of said set of data points for said category of image output example, learned from training example images;
a memory storing a plurality of said category vectors (106; m), one for each of a plurality (K) of said categories of image output example, learned from the training example images; and
a stochastic selector to select a said stored category of image output example (x) for presentation of the corresponding category vector to said probability vector generation system;
wherein said signal processing system is configured to output data comprising an image for an image output example corresponding to said selected stored category, wherein said probability vector generation system comprises a deterministic neural network (B), and wherein said signal processing system further comprises an output stochastic selector (110; $x \sim p_v$) having an input coupled to receive said probability vector from said probability vector generation system and configured to generate and output said data for an image output example;
a context vector input to receive a context vector (**c**), wherein said context vector defines a relative likelihood of each of said plurality of said categories; wherein said context vector input is coupled to said stochastic selector such that said selection of said category of image output example is dependent on said context vector ($p(k|\mathbf{c})$).

2. A signal processing system as claimed in claim 1, wherein said probability vector generation system is coupled to said context vector input, and wherein said probability vector is dependent upon said context vector.

3. A signal processing system as claimed in claim 1 or 2, further comprising a mapping unit coupled between said context vector input and said stochastic selector, wherein said context vector has length d_c, wherein K is a number of said categories, and wherein said mapping unit is configured to map said context vector of length d_c to a category probability vector of length K, and wherein said stochastic selector is configured to select said stored category of output example dependent on said category probability vector, wherein said mapping unit optionally comprises a deterministic neural network.

4. A signal processing chain comprising a chain of signal processing systems (100a-c) each as claimed in any preceding claim, each signal processing system after a first signal processing system further comprising a context vector input to receive a context vector, wherein said context vector defines a relative likelihood of each of said plurality of said categories;
wherein said context vector input is coupled to said stochastic selector such that said selection of said category of output example is dependent on said context vector; and wherein said output data from one said signal processing system provides at least part of said context vector for a next said signal processing system in the chain, wherein:

for each successive said signal processing system after said first signal processing system, said output data from the preceding signal processing systems in the chain combine to provide said context vector input for the successive signal processing system;
wherein said output data from a said signal processing system comprises said category vector selected by the stochastic selector of the signal processing system; and
wherein said output data from a last said signal processing system of the chain comprises said probability vector, optionally further comprising an output stochastic selector having an input coupled to receive said probability vector from said last signal processing system of said chain and configured to generate and output a said output example comprising a said set of data points having values selected with probabilities defined by said probability vector.

5. A signal processing system or signal processing chain as recited in any preceding claim, further comprising a training module coupled to said memory, to said probability vector generation system, to said context vector input, and to said context vector mapping unit, and having a training data input to receive training examples, wherein said training module is configured to compute a responsibility value for each said category, dependent on said a training example presented at said training data input and a context vector presented at said context vector input, and to adjust said stored category vectors, weights of said neural network of said probability vector generation system and weights of said neural network of said mapping unit responsive said computed responsibility values.

6. A signal processing system or signal processing chain as recited in any preceding claim wherein said stored category

vectors, and said probability vectors from said probability vector generation system which depend on said category vectors, comprise a learnt representation of real-world data.

7. A signal processing chain as in claim 4 wherein each signal processing system of the chain has learnt a distribution of said training examples across a limited number of categories less than said plurality of categories.

8. A method of training a signal processing system or signal processing chain as recited in any preceding claim, the method comprising:

presenting training examples to the signal processing system or signal processing chain, wherein a said training example comprises a set of data points corresponding to data points of a said output example;
computing from a said training example a set of responsibility values, one for each said category, wherein a said responsibility value comprises a probability of the training example belonging to the category, each category having a respective stored category vector;
computing a gradient vector for a set of parameters of the signal processing system or signal processing chain from said set of responsibility values, wherein said set of parameters includes said stored category vectors and defines a shared mapping between said stored category vectors and a corresponding set of said probability vectors defining probability values for a said set of data points of a said output example;
updating said set of parameters using said computed gradient vector; and
further comprising presenting a said context vector with said training example, and wherein a said responsibility value is further conditional on said context vector.

9. The signal processing system or signal processing chain, of any preceding claim embodied in: software on a non-transitory data courier, a programmed computer system, programmed memory, or electronic hardware/circuitry.

Patentansprüche

1. Signalverarbeitungssystem zur Bilderzeugung, wobei das Signalverarbeitungssystem dafür konfiguriert ist, ein Bild zu erzeugen, das durch einen Kontext bedingt ist, der durch einen Kontextvektor (c) definiert ist, wobei der Kontextvektor eine relative Wahrscheinlichkeit jeder aus einer Vielzahl von Kategorien von Bildern definiert, und wobei das Signalverarbeitungssystem dafür konfiguriert ist, ein Bild aus einer ausgewählten Kategorie zu erzeugen, wobei das Signalverarbeitungssystem umfasst:

ein Wahrscheinlichkeitsvektor-Erzeugungssystem (B; 108), wobei das Wahrscheinlichkeitsvektor-Erzeugungssystem einen Eingang hat, um einen Kategorievektor (m_k) für eine Kategorie (k) von Bildausgabebeispielen zu empfangen, und einen Ausgang, um einen Wahrscheinlichkeitsvektor (p_v) für die Kategorie von Bildausgabebeispielen bereitzustellen, wobei der Kategorievektor eine komprimierte Darstellung des Wahrscheinlichkeitsvektors umfasst, wobei das Bildausgabebeispiel eine Menge von Datenpunkten umfasst und wobei der Wahrscheinlichkeitsvektor eine Wahrscheinlichkeit für jeden aus der Menge von Datenpunkten für die Kategorie von Bildausgabebeispielen definiert, die aus Trainingsbeispielbildern gelernt wurde;
einen Speicher, der eine Vielzahl der Kategorievektoren (106; m) speichert, einen für jede aus einer Vielzahl (*K*) der Kategorien von Bildausgabebeispielen, die aus den Trainingsbeispielbildern gelernt wurden; und
einen stochastischen Selektor, um eine gespeicherte Kategorie von Bildausgabebeispielen (x) zur Vorlage des entsprechenden Kategorievektors dem Wahrscheinlichkeitsvektor-Erzeugungssystem auszuwählen;
wobei das Signalverarbeitungssystem dafür konfiguriert ist, Daten auszugeben, die ein Bild für ein Bildausgabebeispiel umfassen, das der ausgewählten gespeicherten Kategorie entspricht, wobei das Wahrscheinlichkeitsvektor-Erzeugungssystem ein deterministisches neuronales Netz (B) umfasst, und wobei das Signalverarbeitungssystem ferner einen stochastischen Ausgabeselektor (110; $x \sim p_v$) umfasst, der einen Eingang aufweist, welcher dafür gekoppelt ist, den Wahrscheinlichkeitsvektor von dem Wahrscheinlichkeitsvektor-Erzeugungssystem zu empfangen, und dafür konfiguriert ist, die Daten für ein Bildausgabebeispiel zu erzeugen und auszugeben;
einen Kontextvektoreingang, um einen Kontextvektor (c) zu empfangen, wobei der Kontextvektor eine relative Wahrscheinlichkeit für jede aus der Vielzahl der Kategorien definiert; wobei der Kontextvektoreingang mit dem stochastischen Selektor gekoppelt ist, sodass die Auswahl der Kategorie von Bildausgabebeispielen von dem Kontextvektor ($p(k|c)$) abhängig ist.

2. Signalverarbeitungssystem nach Anspruch 1, wobei das Wahrscheinlichkeitsvektor-Erzeugungssystem mit dem

Kontextvektoreingang gekoppelt ist und wobei der Wahrscheinlichkeitsvektor von dem Kontextvektor abhängig ist.

3. Signalverarbeitungssystem nach Anspruch 1 oder 2, ferner eine Abbildungseinheit umfassend, die zwischen dem Kontextvektoreingang und dem stochastischen Selektor gekoppelt ist, wobei der Kontextvektor eine Länge d_c hat, wobei K eine Anzahl der Kategorien ist, und wobei die Abbildungseinheit dafür konfiguriert ist, den Kontextvektor der Länge d_c auf einen Kategoriewahrscheinlichkeitsvektor der Länge K abzubilden, und wobei der stochastische Selektor dafür konfiguriert ist, die gespeicherte Kategorie von Ausgabebeispielen in Abhängigkeit von dem Kategoriewahrscheinlichkeitsvektor auszuwählen, wobei die Abbildungseinheit optional ein deterministisches neuronales Netz umfasst.

4. Signalverarbeitungskette, umfassend eine Kette von Signalverarbeitungssystemen (100a-c), jedes nach einem der vorhergehenden Ansprüche, wobei jedes Signalverarbeitungssystem nach einem ersten Signalverarbeitungssystem ferner einen Kontextvektoreingang umfasst, um einen Kontextvektor zu empfangen, wobei der Kontextvektor eine relative Wahrscheinlichkeit für jede aus der Vielzahl der Kategorien definiert;
wobei der Kontextvektoreingang mit dem stochastischen Selektor gekoppelt ist, sodass die Auswahl der Kategorie von Ausgabebeispielen von dem Kontextvektor abhängig ist; und wobei die Ausgabedaten aus einem Signalverarbeitungssystem mindestens einen Teil des Kontextvektors für ein nächstes Signalverarbeitungssystem in der Kette bereitstellen, wobei:

für jedes nachfolgende Signalverarbeitungssystem nach dem ersten Signalverarbeitungssystem die Ausgabedaten aus den vorhergehenden Signalverarbeitungssystemen in der Kette sich kombinieren, um den Kontextvektoreingang für das nachfolgende Signalverarbeitungssystem bereitzustellen;
wobei die Ausgabedaten aus einem Signalverarbeitungssystem den Kategorievektor umfassen, der durch den stochastischen Selektor des Signalverarbeitungssystems ausgewählt wurde; und
wobei die Ausgabedaten aus einem letzten Signalverarbeitungssystem der Kette den Wahrscheinlichkeitsvektor umfassen, optional ferner einen stochastischen Ausgabeselektor umfassend, der einen Eingang hat, welcher dafür gekoppelt ist, den Wahrscheinlichkeitsvektor von dem letzten Signalverarbeitungssystem der Kette zu empfangen, und dafür konfiguriert ist, ein Ausgabebeispiel zu erzeugen und auszugeben, das eine Menge von Datenpunkten umfasst, welche Werte aufweist, die mit durch den Wahrscheinlichkeitsvektor definierten Wahrscheinlichkeiten ausgewählt wurden.

5. Signalverarbeitungssystem oder Signalverarbeitungskette nach einem der vorhergehenden Ansprüche, ferner ein Trainingsmodul umfassend, das mit dem Speicher, mit dem Wahrscheinlichkeitsvektor-Erzeugungssystem, mit dem Kontextvektoreingang und mit der Kontextvektor-Abbildungseinheit gekoppelt ist und einen Trainingsdateneingang aufweist, um Trainingsbeispiele zu empfangen, wobei das Trainingsmodul dafür konfiguriert ist, für jede Kategorie einen Verantwortlichkeitswert zu berechnen, der von dem einen an dem Trainingsdateneingang vorgelegten Trainingsbeispiel und einem an dem Kontextvektoreingang vorgelegten Kontextvektor abhängig ist, und um die gespeicherten Kategorievektoren, Gewichte des neuronalen Netzes des Wahrscheinlichkeitsvektor-Erzeugungssystems und Gewichte des neuronalen Netzes der Abbildungseinheit als Antwort auf die berechneten Verantwortlichkeitswerte anzupassen.

6. Signalverarbeitungssystem oder Signalverarbeitungskette nach einem der vorhergehenden Ansprüche, wobei die gespeicherten Kategorievektoren und die Wahrscheinlichkeitsvektoren aus dem Wahrscheinlichkeitsvektor-Erzeugungssystem, die von den Kategorievektoren abhängen, eine gelernte Darstellung von Realweltdaten umfassen.

7. Signalverarbeitungskette nach Anspruch 4, wobei jedes Signalverarbeitungssystem der Kette eine Verteilung der Trainingsbeispiele über eine begrenzte Anzahl von Kategorien, die kleiner als die Vielzahl von Kategorien ist, gelernt hat.

8. Verfahren zum Trainieren eines Signalverarbeitungssystems oder einer Signalverarbeitungskette nach einem der vorhergehenden Ansprüche, wobei das Verfahren umfasst:

Vorlegen von Trainingsbeispielen dem Signalverarbeitungssystem oder der Signalverarbeitungskette, wobei ein Trainingsbeispiel eine Menge von Datenpunkten umfasst, die Datenpunkten eines Ausgabebeispiels entsprechen;
Berechnen einer Menge von Verantwortlichkeitswerten aus einem Trainingsbeispiel, einen für jede Kategorie, wobei ein Verantwortlichkeitswert eine Wahrscheinlichkeit dafür umfasst, dass das Trainingsbeispiel zu der Kategorie gehört, wobei jede Kategorie einen jeweiligen gespeicherten Kategorievektor aufweist;

Berechnen eines Gradientenvektors für eine Menge von Parametern des Signalverarbeitungssystems oder der Signalverarbeitungskette aus der Menge von Verantwortungswerten, wobei die Menge von Parametern die gespeicherten Kategorievektoren einschließt und eine gemeinsame Abbildung zwischen den gespeicherten Kategorievektoren und einer entsprechenden Menge der Wahrscheinlichkeitsvektoren definiert, die Wahrscheinlichkeitswerte für eine Menge von Datenpunkten eines Ausgabebeispiels definiert;
Aktualisieren der Menge von Parametern unter Verwendung des berechneten Gradientenvektors; und
ferner umfassend: Vorlegen eines Kontextvektors mit dem Trainingsbeispiel, und wobei ein Verantwortlichkeitswert ferner durch den Kontextvektor bedingt ist.

9. Signalverarbeitungssystem oder Signalverarbeitungskette nach einem der vorhergehenden Ansprüche, verkörpert durch: Software auf einem nichtflüchtigen Datenträger, ein programmiertes Computersystem, einen programmierten Speicher oder eine elektronische Hardware/Schaltungsanordnung.

Revendications

1. Système de traitement de signal pour la génération d'images, dans lequel le système de traitement de signal est configuré pour générer une image qui est conditionnée par un contexte qui est défini par un vecteur de contexte (c), dans lequel ledit vecteur de contexte définit une vraisemblance relative de chacune d'une pluralité de catégories d'images, et dans lequel le système de traitement de signal est configuré pour générer une image à partir d'une catégorie sélectionnée, le système de traitement de signal comprenant :

un système de génération de vecteur de probabilité (B ; 108), dans lequel ledit système de génération de vecteur de probabilité comporte une entrée pour recevoir un vecteur de catégorie (m_k) pour une catégorie (k) d'exemple de sortie d'image, et une sortie pour fournir un vecteur de probabilité (p_v) pour ladite catégorie d'exemple de sortie d'image, dans lequel ledit vecteur de catégorie comprend une représentation comprimée dudit vecteur de probabilité, dans lequel ledit exemple de sortie d'image comprend un jeu de points de données, et dans lequel ledit vecteur de probabilité définit une probabilité de chacun dudit jeu de points de données pour ladite catégorie d'exemple de sortie d'image, appris à partir d'images d'exemple d'apprentissage ;
une mémoire qui stocke une pluralité desdits vecteurs de catégorie (106 ; m), à raison d'un pour chacune d'une pluralité (*K*) desdites catégories d'exemple de sortie d'image, appris à partir d'images d'exemple d'apprentissage ; et
un sélecteur stochastique pour sélectionner une dite catégorie stockée d'exemple de sortie d'image (x) dans le but de la présentation du vecteur de catégorie correspondant audit système de génération de vecteur de probabilité ;
dans lequel ledit système de traitement de signal est configuré pour émettre en sortie des données qui comprennent une image pour un exemple de sortie d'image qui correspond à ladite catégorie stockée sélectionnée, dans lequel ledit système de génération de vecteur de probabilité comprend un réseau neuronal déterministe (B), et dans lequel ledit système de traitement de signal comprend en outre un sélecteur stochastique de sortie (110 ; $x \sim p_v$) qui comporte une entrée qui est couplée pour recevoir ledit vecteur de probabilité en provenance dudit système de génération de vecteur de probabilité et qui est configurée pour générer et émettre en sortie lesdites données pour un exemple de sortie d'image ;
une entrée de vecteur de contexte pour recevoir un vecteur de contexte (c), dans lequel ledit vecteur de contexte définit une vraisemblance relative de chacune de ladite pluralité desdites catégories ; dans lequel ladite entrée de vecteur de contexte est couplée audit sélecteur stochastique de telle sorte que ladite sélection de ladite catégorie d'exemple de sortie d'image soit fonction dudit vecteur de contexte ($p(k|c)$).

2. Système de traitement de signal tel que revendiqué selon la revendication 1, dans lequel ledit système de génération de vecteur de probabilité est couplé à ladite entrée de vecteur de contexte et dans lequel ledit vecteur de probabilité est fonction dudit vecteur de contexte.

3. Système de traitement de signal tel que revendiqué selon la revendication 1 ou 2, comprenant en outre une unité de cartographie qui est couplée entre ladite entrée de vecteur de contexte et ledit sélecteur stochastique, dans lequel ledit vecteur de contexte présente une longueur d_c, dans lequel K est un nombre desdites catégories et dans lequel ladite unité de cartographie est configurée pour cartographier ledit vecteur de contexte de longueur d_c par rapport à un vecteur de probabilité de catégorie de longueur K et dans lequel ledit sélecteur stochastique est configuré pour sélectionner ladite catégorie stockée d'exemple de sortie en fonction dudit vecteur de probabilité de catégorie, dans lequel ladite unité de cartographie comprend facultativement un réseau neuronal déterministe.

4. Chaîne de traitement de signal comprenant une chaîne de systèmes de traitement de signal (100a-c) dont chacun est tel que revendiqué selon l'une quelconque des revendications qui précèdent, chaque système de traitement de signal après un premier système de traitement de signal comprenant en outre une entrée de vecteur de contexte pour recevoir un vecteur de contexte, dans lequel ledit vecteur de contexte définit une vraisemblance relative de chacune de ladite pluralité desdites catégories ;
dans lequel ladite entrée de vecteur de contexte est couplée audit sélecteur stochastique de telle sorte que ladite sélection de ladite catégorie d'exemple de sortie soit fonction dudit vecteur de contexte ; et dans lequel lesdites données de sortie en provenance d'un dit système de traitement de signal fournissent au moins une partie dudit vecteur de contexte pour un dit système de traitement de signal suivant dans la chaîne, dans lequel :

pour chaque dit système de traitement de signal successif après ledit premier système de traitement de signal, lesdites données de sortie en provenance des systèmes de traitement de signal précédents dans la chaîne se combinent pour fournir ladite entrée de vecteur de contexte pour le système de traitement de signal qui suit ;
dans lequel lesdites données de sortie en provenance d'un dit système de traitement de signal comprennent ledit vecteur de catégorie qui est sélectionné par le sélecteur stochastique du système de traitement de signal ; et
dans lequel lesdites données de sortie en provenance d'un dernier dit système de traitement de signal de la chaîne comprennent ledit vecteur de probabilité, facultativement comprenant en outre un sélecteur stochastique de sortie qui comporte une entrée qui est couplée pour recevoir ledit vecteur de probabilité en provenance dudit dernier système de traitement de signal de ladite chaîne et qui est configurée pour générer et émettre en sortie un dit exemple de sortie qui comprend un dit jeu de points de données qui comportent des valeurs qui sont sélectionnées avec des probabilités qui sont définies par ledit vecteur de probabilité.

5. Système de traitement de signal ou chaîne de traitement de signal tel(le) qu'énoncé(e) dans l'une quelconque des revendications qui précèdent, comprenant en outre un module d'apprentissage qui est couplé à ladite mémoire, audit système de génération de vecteur de probabilité, à ladite entrée de vecteur de contexte et à ladite unité de cartographie de vecteur de contexte, et comportant une entrée de données d'apprentissage pour recevoir des exemples d'apprentissage, dans lequel ledit module d'apprentissage est configuré pour calculer une valeur de responsabilité pour chaque dite catégorie, en fonction dudit un exemple d'apprentissage qui est présenté au niveau de ladite entrée de données d'apprentissage et d'un vecteur de contexte qui est présenté au niveau de ladite entrée de vecteur de contexte, et pour régler lesdits vecteurs de catégorie stockés, des poids dudit réseau neuronal dudit système de génération de vecteur de probabilité et des poids dudit réseau neuronal de ladite unité de cartographie en réponse auxdites valeurs de responsabilité calculées.

6. Système de traitement de signal ou chaîne de traitement de signal tel(le) qu'énoncé(e) dans l'une quelconque des revendications qui précèdent, dans lequel/laquelle lesdits vecteurs de catégorie stockés et lesdits vecteurs de probabilité en provenance dudit système de génération de vecteur de probabilité qui sont fonction desdits vecteurs de catégorie comprennent une représentation apprise de données du monde réel.

7. Chaîne de traitement de signal selon la revendication 4, dans laquelle chaque système de traitement de signal de la chaîne a appris une distribution desdits exemples d'apprentissage sur un nombre limité de catégories qui est inférieur à ladite pluralité de catégories.

8. Procédé d'apprentissage d'un système de traitement de signal ou d'une chaîne de traitement de signal tel(le) qu'énoncé(e) dans l'une quelconque des revendications qui précèdent, le procédé comprenant :

la présentation d'exemples d'apprentissage au système de traitement de signal ou à la chaîne de traitement de signal, dans lequel un dit exemple d'apprentissage comprend un jeu de points de données qui correspondent à des points de données d'un dit exemple de sortie ;
le calcul, à partir d'un dit exemple d'apprentissage, d'un jeu de valeurs de responsabilité, à raison d'une pour chaque dite catégorie, dans lequel une dite valeur de responsabilité comprend une probabilité que l'exemple d'apprentissage appartienne à la catégorie, chaque catégorie comportant un vecteur de catégorie stocké respectif ;
le calcul d'un vecteur de gradient pour un jeu de paramètres du système de traitement de signal ou de la chaîne de traitement de signal à partir dudit jeu de valeurs de responsabilité, dans lequel ledit jeu de paramètres inclut lesdits vecteurs de catégorie stockés et définit une cartographie partagée entre lesdits vecteurs de catégorie stockés et un jeu correspondant desdits vecteurs de probabilité qui définissent des valeurs de probabilité pour un dit jeu de points de données d'un dit exemple de sortie ;
la mise à jour dudit jeu de paramètres en utilisant ledit vecteur de gradient calculé ; et

comprenant en outre la présentation d'un dit vecteur de contexte avec ledit exemple d'apprentissage, et dans lequel une dite valeur de responsabilité est en outre conditionnée par ledit vecteur de contexte.

9. Système de traitement de signal ou chaîne de traitement de signal selon l'une quelconque des revendications qui précèdent mis(e) en œuvre dans : un logiciel sur un support de données non transitoire, un système informatique programmé, une mémoire programmée ou un composant matériel/circuit électronique.

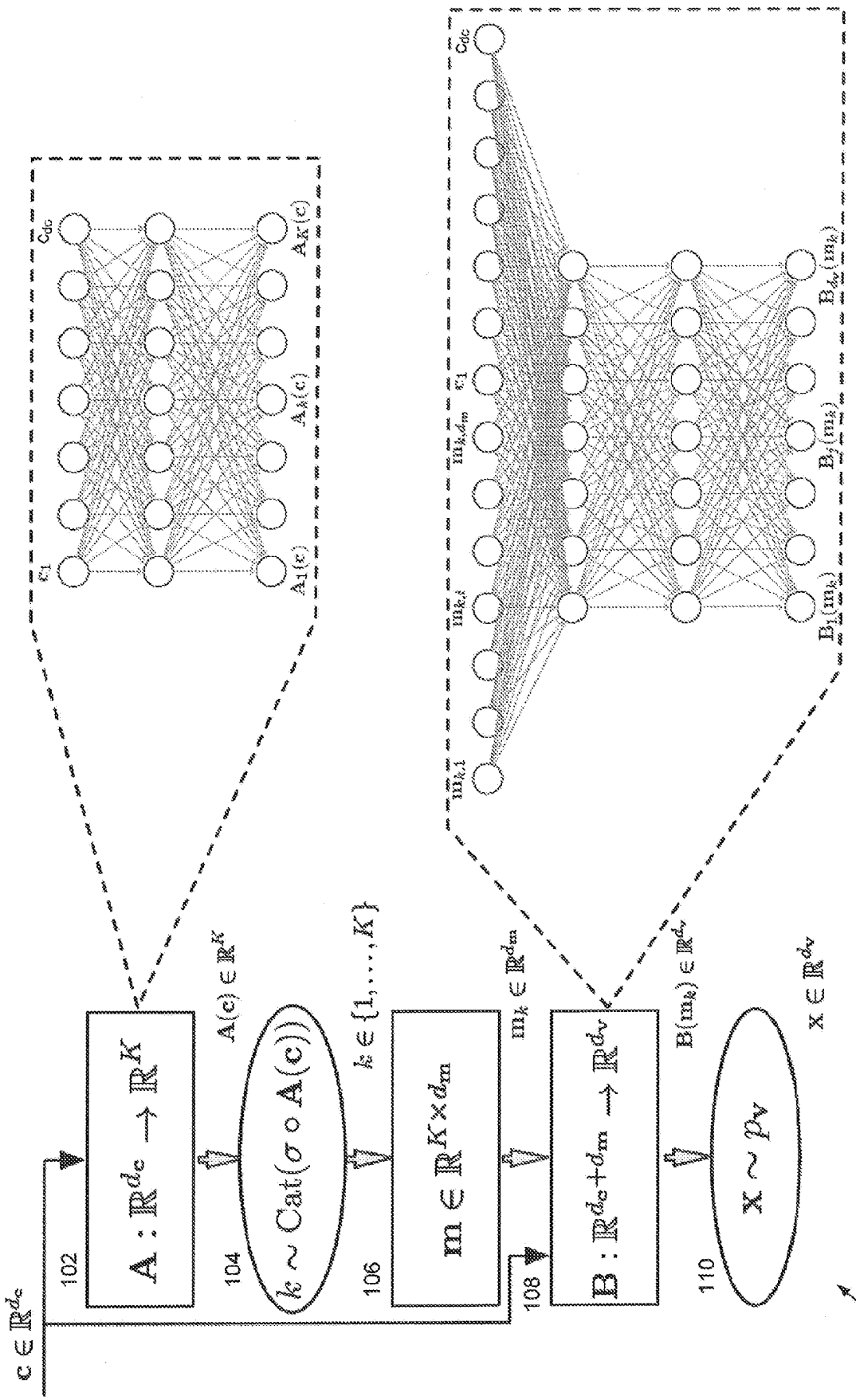
$\mathbf{c} \in \mathbb{R}^{d_c}$
102
$\mathbf{A} : \mathbb{R}^{d_c} \to \mathbb{R}^K$
$\mathbf{A}(\mathbf{c}) \in \mathbb{R}^K$
104
$k \sim \mathrm{Cat}(\sigma \circ \mathbf{A}(\mathbf{c}))$
$k \in \{1, \ldots, K\}$
106
$\mathbf{m} \in \mathbb{R}^{K \times d_m}$
$\mathbf{m}_k \in \mathbb{R}^{d_m}$
108
$\mathbf{B} : \mathbb{R}^{d_c + d_m} \to \mathbb{R}^{d_v}$
$\mathbf{B}(\mathbf{m}_k) \in \mathbb{R}^{d_v}$
110
$\mathbf{x} \sim p_{\mathbf{v}}$
$\mathbf{x} \in \mathbb{R}^{d_v}$
100
c_1
c_{d_c}
$A_1(c)$
$A_i(c)$
$A_K(c)$
$m_{k,1}$
$m_{k,j}$
m_{k,d_m}
$B_1(m_k)$
$B_j(m_k)$
$B_{d_v}(m_k)$

Figure 1a

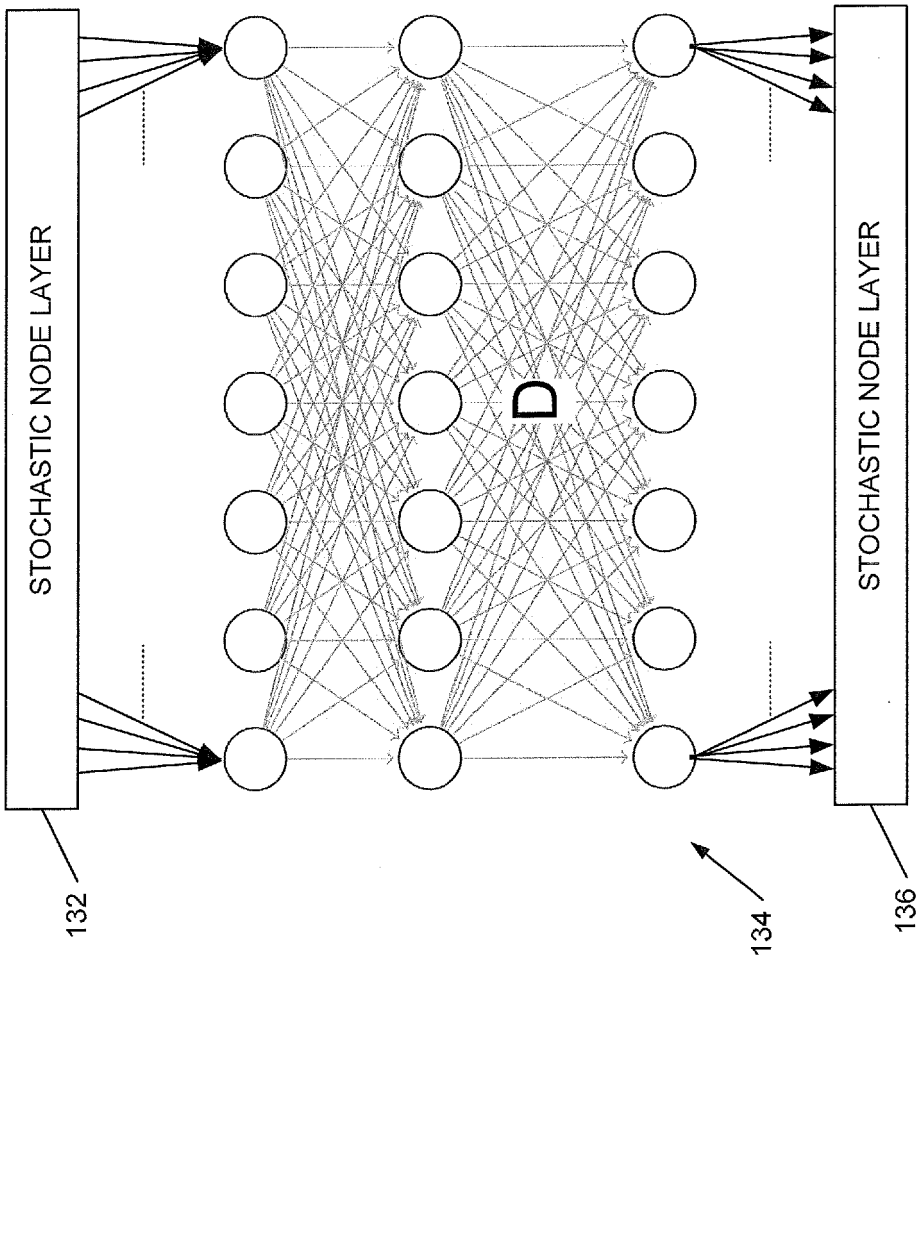
STOCHASTIC NODE LAYER
D
STOCHASTIC NODE LAYER
132
134
136
130

Figure 1b

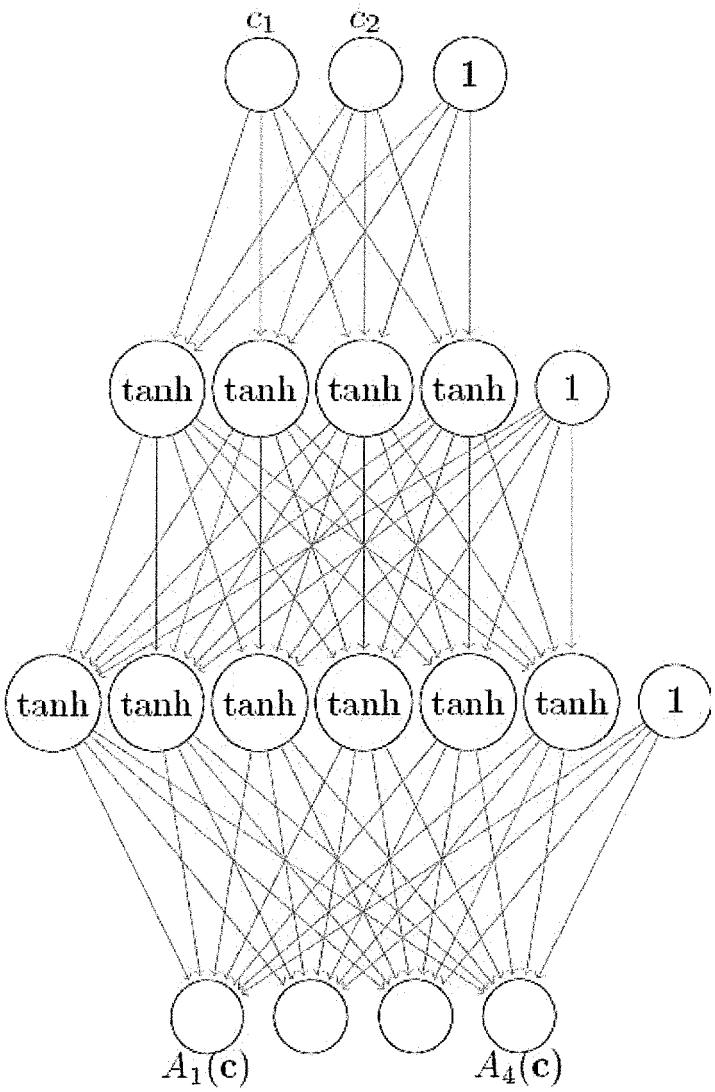
c_1
c_2
1
tanh
tanh
tanh
tanh
1
tanh
tanh
tanh
tanh
tanh
tanh
1
$A_1(\mathbf{c})$
$A_4(\mathbf{c})$

Figure 1c

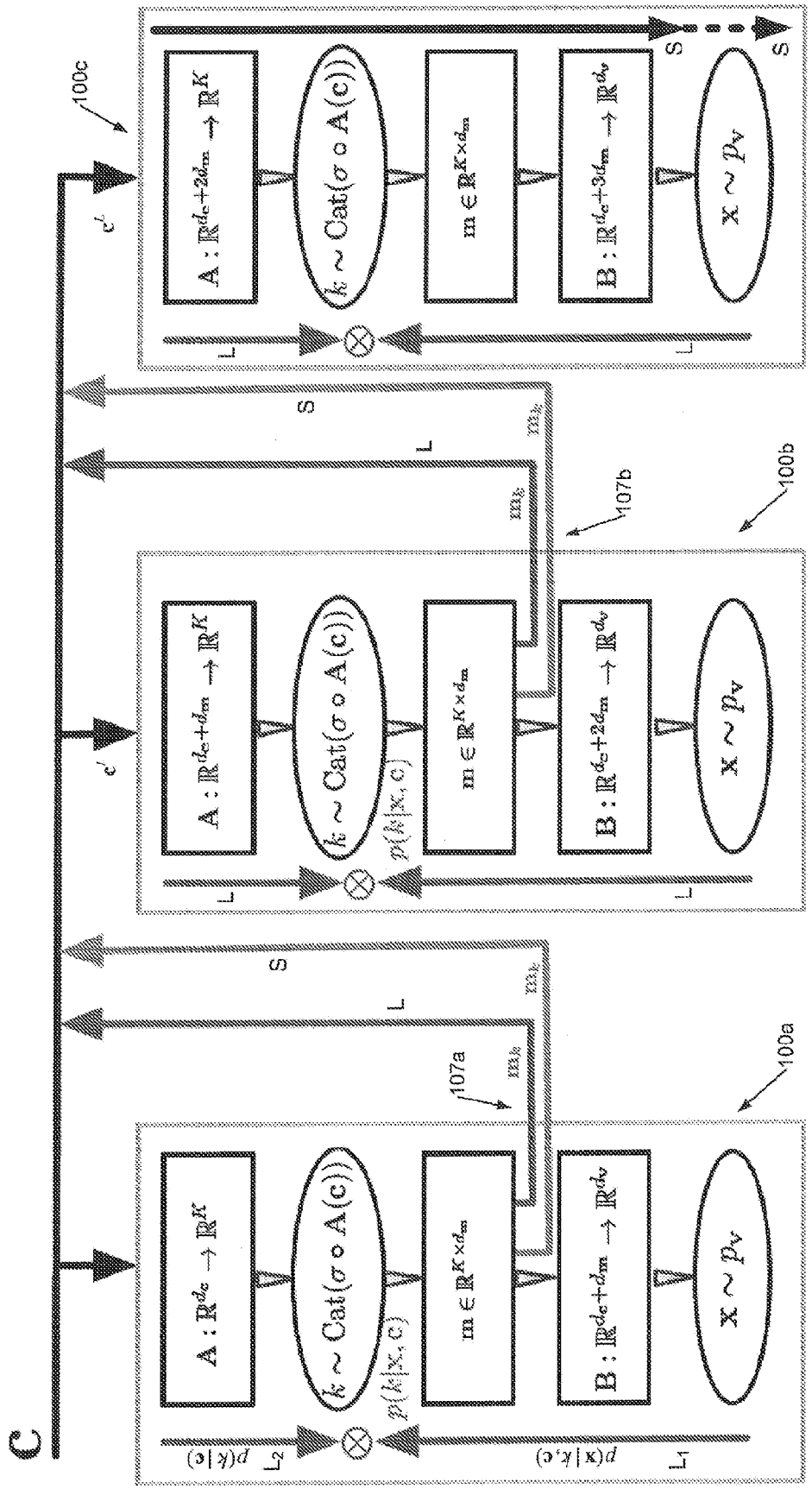
c
100a
100b
100c
107a
107b
140
L
S
$\mathbf{A} : \mathbb{R}^{d_c} \to \mathbb{R}^K$
$\mathbf{A} : \mathbb{R}^{d_c+d_m} \to \mathbb{R}^K$
$\mathbf{A} : \mathbb{R}^{d_c+2d_m} \to \mathbb{R}^K$
$k \sim \mathrm{Cat}(\sigma \circ \mathbf{A}(\mathbf{c}))$
$p(k|\mathbf{x}, \mathbf{c})$
$\mathbf{m} \in \mathbb{R}^{K \times d_m}$
$\mathbf{B} : \mathbb{R}^{d_c+d_m} \to \mathbb{R}^{d_v}$
$\mathbf{B} : \mathbb{R}^{d_c+2d_m} \to \mathbb{R}^{d_v}$
$\mathbf{B} : \mathbb{R}^{d_c+3d_m} \to \mathbb{R}^{d_v}$
$\mathbf{x} \sim p_v$
$p(k|\mathbf{c})$
$p(\mathbf{x}|k, \mathbf{c})$
$\mathbf{m}_k$
$\mathbf{m}_{k_1}$
$\mathbf{m}_{k_2}$

Figure 2

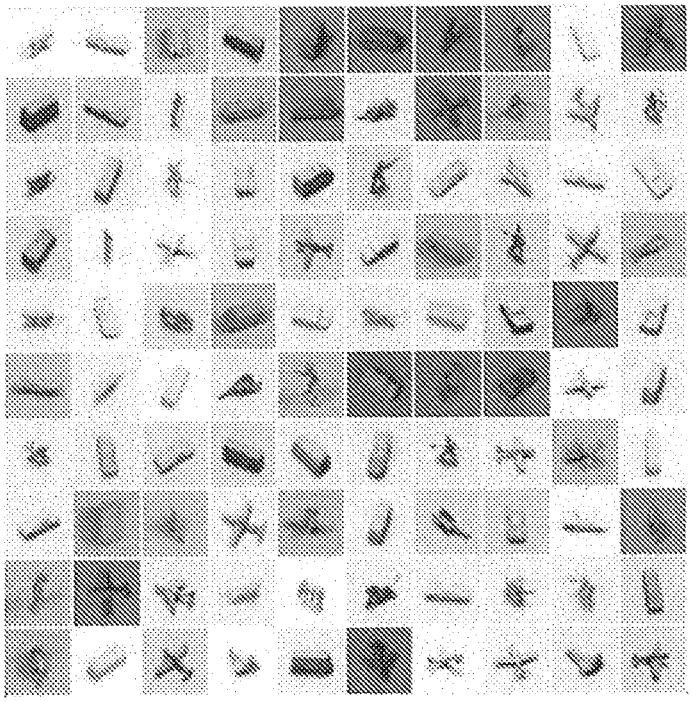

Figure 3a

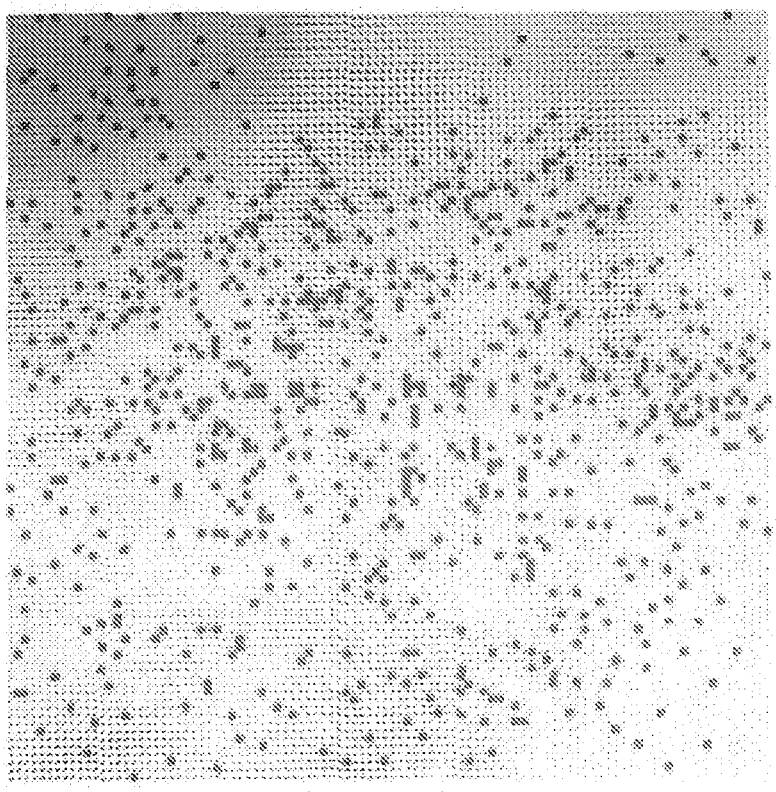

Figure 3b

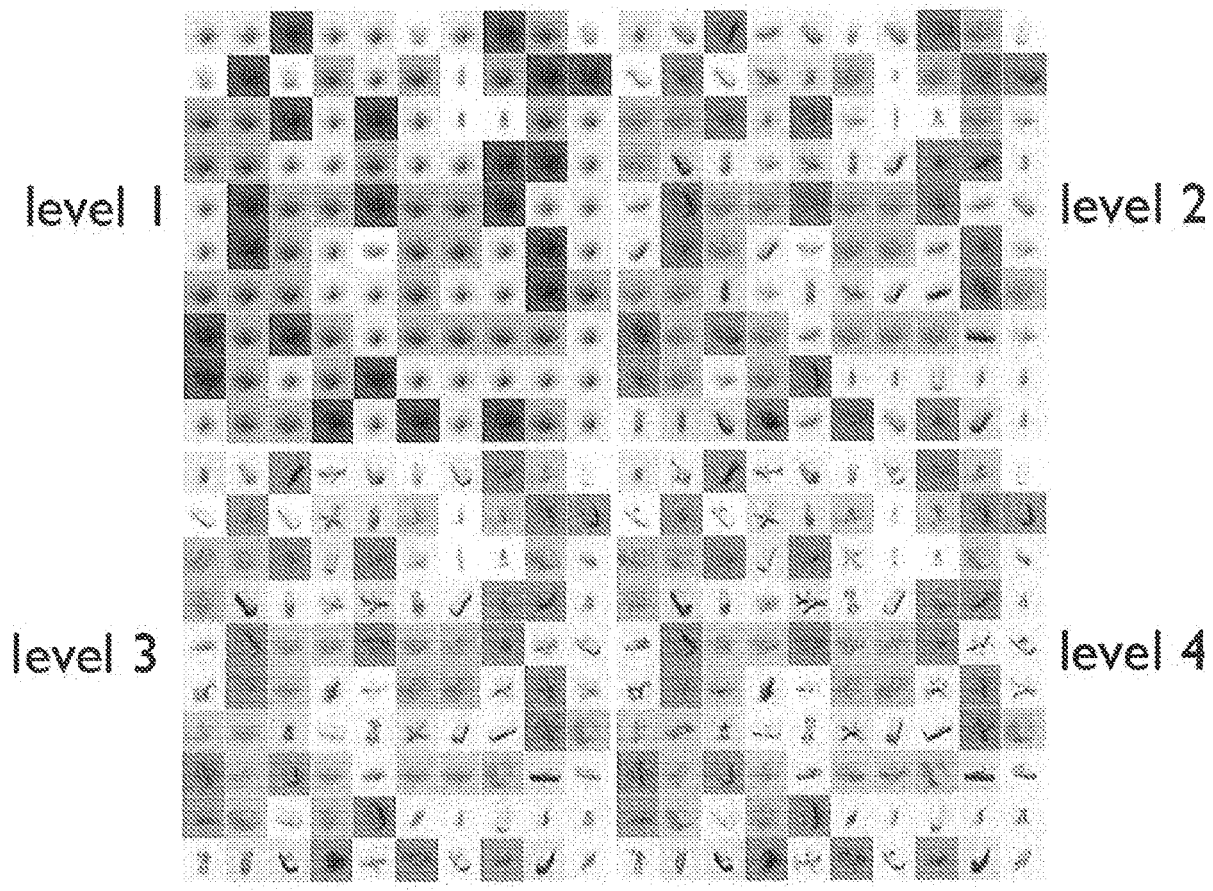

Figure 4

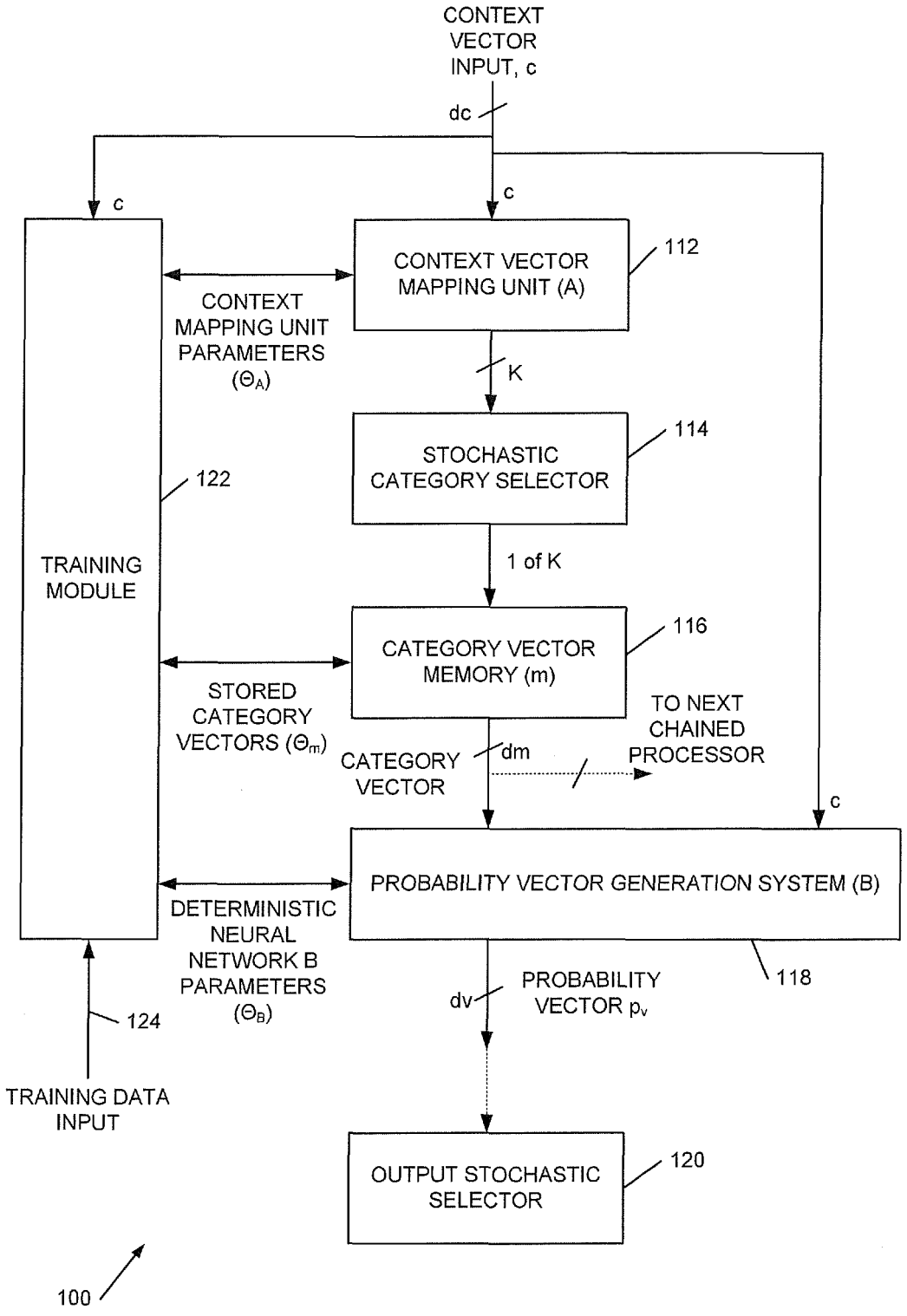
CONTEXT VECTOR INPUT, c
dc
c
c
CONTEXT VECTOR MAPPING UNIT (A)
112
CONTEXT MAPPING UNIT PARAMETERS (Θ_A)
K
STOCHASTIC CATEGORY SELECTOR
114
122
TRAINING MODULE
1 of K
CATEGORY VECTOR MEMORY (m)
116
STORED CATEGORY VECTORS (Θ_m)
TO NEXT CHAINED PROCESSOR
CATEGORY VECTOR
dm
c
PROBABILITY VECTOR GENERATION SYSTEM (B)
DETERMINISTIC NEURAL NETWORK B PARAMETERS (Θ_B)
118
dv
PROBABILITY VECTOR p_v
124
TRAINING DATA INPUT
OUTPUT STOCHASTIC SELECTOR
120
100

Figure 5

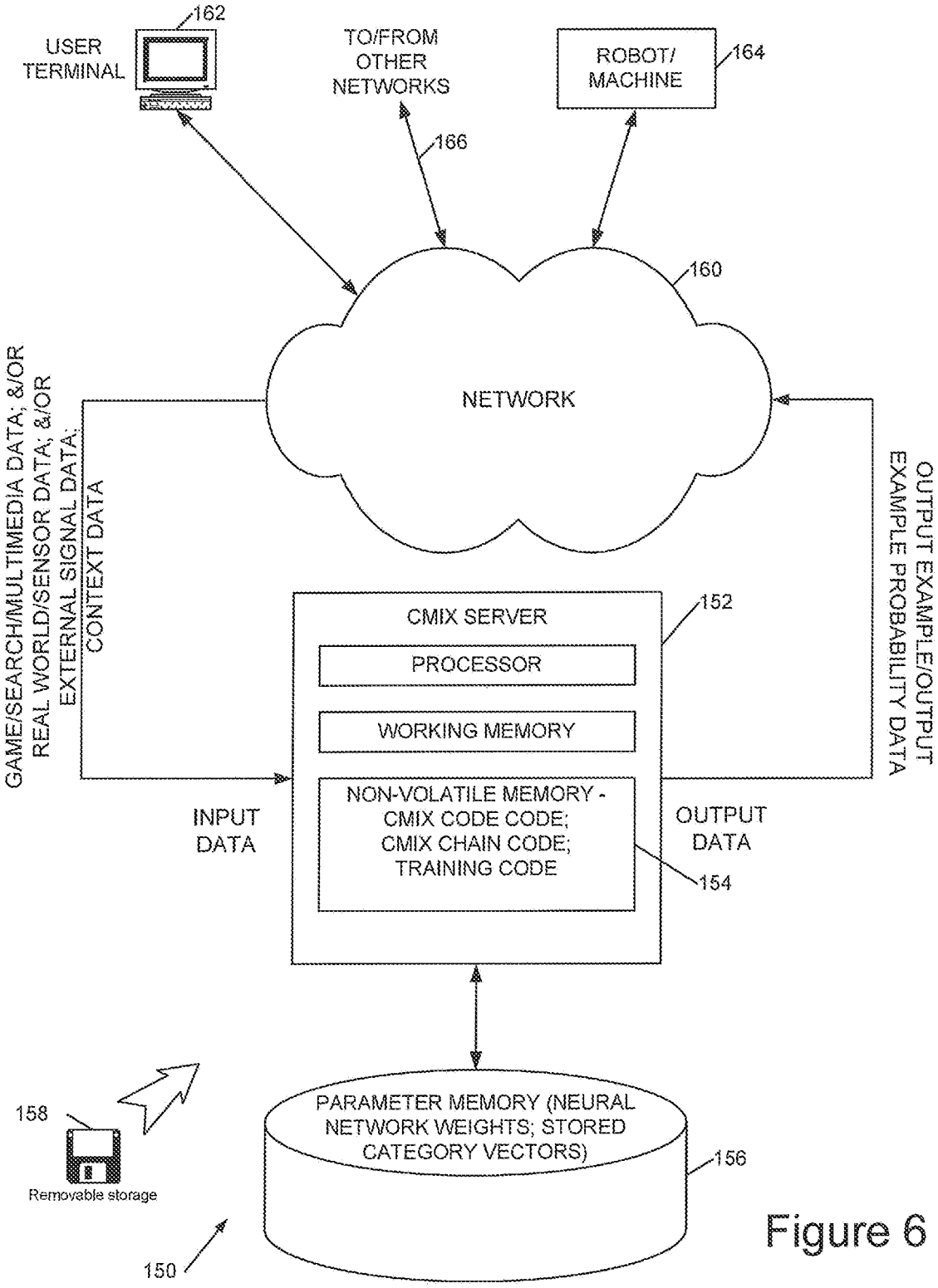
USER TERMINAL
162
TO/FROM OTHER NETWORKS
166
ROBOT/ MACHINE
164
160
NETWORK
GAME/SEARCH/MULTIMEDIA DATA; &/OR REAL WORLD/SENSOR DATA; &/OR EXTERNAL SIGNAL DATA; CONTEXT DATA
OUTPUT EXAMPLE/OUTPUT EXAMPLE PROBABILITY DATA
CMIX SERVER
152
PROCESSOR
WORKING MEMORY
NON-VOLATILE MEMORY - CMIX CODE CODE; CMIX CHAIN CODE; TRAINING CODE
154
INPUT DATA
OUTPUT DATA
PARAMETER MEMORY (NEURAL NETWORK WEIGHTS; STORED CATEGORY VECTORS)
156
158
Removable storage
150

Figure 6

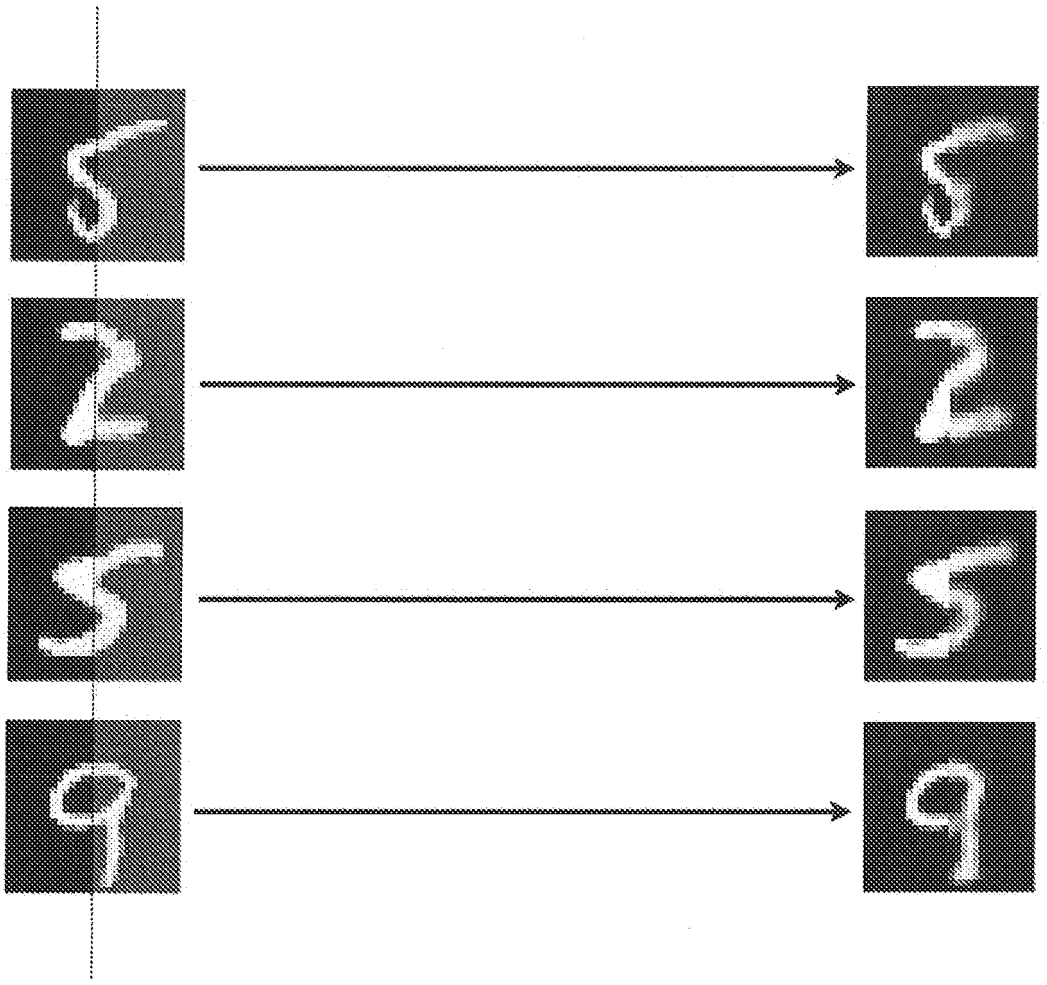

Figure 7

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Non-patent literature cited in the description

- **SALAKHUTDINOV ; HINTON.** Deep Boltzmann Machine. *Proceedings of the International Conference on Artificial Intelligence and Statistics,* 2009, vol. 5, 448-455, http://www.cs.utoronto.ca/~rsalakhu/papers/dbm.pdf **[0004]**
- **RANZATO et al.** On deep generative models with applications to recognition. *Proceedings CVPR '11,* 2857-2864 **[0004]**
- **HINTON et al.** *Improving Neural Networks by Preventing Co-adaptation of Feature Detectors,* 03 July 2012 **[0010]**
- **C.M. BISHOP.** Neural networks for pattern recognition. Oxford University Press, 1995 **[0067]**
- **O. CAPP_E ; T. RYDEN ; E. MOULINES.** Inference in hidden Markov models. Springer, 2005, 353 **[0082]**
- Variational algorithms for approximate Bayesian inference. **M.J. BEAL.** PhD thesis, Gatsby Computational Neuroscience Unit. University College, 2003 **[0105]**
- **M. I. JORDAN ; Z. GHAHRAMANI ; T. S. JAAKKOLA ; L. K. SAUL.** An introduction to variational methods for graphical models. *Machine Learning,* 1999, vol. 37, 183-233 **[0105]**