社団法人 人工知能学会
Japanese Society for
Artificial Intelligence

人工知能学会研究会資料
JSAI Technical Report
SIG-Challenge-B201-5 (5/4)

# Likelihood estimated by two types of Neural Networks

C.-H. Lee, L.-W. Lu, N. M. Mayer

Nat'l Chung Cheng University,

Chia-Yi, Taiwan
mikemayer@ccu.edu.tw

*Abstract*—The purpose of this contribution is to review two relatively recent developments in the field of neural networks to the robotics audience. The first are deep belief networks and the second are echo state networks. Since much of sensor processing relates to probability estimates and also sampling of data in this work we focus on how to derive directly a probability and likelihood estimates. As it turns out both show complementary features and strengths which make a combination of both highly eligible.

## 1. Introduction

Most out of the box sensor processing approaches, such as Kalman and particle filters[1] either explicitly or implicitly assume certain constraints on the uncertainty distribution of the sensor input and the internal state of a robot. Usually it is assumed that that uncertainty can be expressed by a normal distribution. For everyday applications that works very well although it is common knowledge that real world noise is often not very similar to a Gaussian distribution. This has consequences, such as the fact that many approaches offer a version in which outliers are discarded in a separated step because outliers are covered badly by the Gaussian assumption and have a strong impact on the mean and variance.

Within this work we present two approaches that base on neural networks echo state networks and deep belief networks [2, 3] which in 2 different ways give estimation of a likelihood without any further assumptions on the underlying probability distribution, where the Deep Belief Network as a generative model generates samples (particles with the correct probability distribution).

Deep Belief Networks (DBNs) are one of the mainstream deep learning approaches, which were initially introduced in [4]. DBNs can learn the networks one layer at a time in a biologicaly plausible way. DBNs have vastly been applied to many types of data in artificial intelligence, such as hand-written digit images of binary pixels in MNIST database[5], windows of melcepstral coefficients in speech recognition[8] and gray-scaled images in 3D object recognition in NORB dataset[7]. In our paper, we will further examine the ability of DBNs in interpreting binary encoded data, and we will validate this by showing how dose DBNs perform in learning various distributions.

In our work, we have tried to construct each layer RBMs into individual dynamic linking module, this layer-like property made it easily duplicated and distributed under our Qi's framework[11]. We have successfully implemented the deep believe network on the hand-written digits task using as presented in [4] using the framework. We have also worked on training the binary-represented random numbers by DBNs and try to reproduce an identical distribution from such random number generator.

Sect. 2.1 introduces the structure of Deep Belief Network and Restricted Boltzmann machines (RBMs), the basic component of DBNs. Sect. 2.2 shows how DBNs can learn the binary encoded data by learning certain distributions. In Sect. 2.3, we will propose some ideas we have came up with on combining DBNs with domestic service robot application.

The second approach presented here is the echo state network [22]. Here, we directly attack the concept of likelihood by the idea that all relevant information for the probability estimate in some sense is represented in the reservoir, thus it should be possible to read it out in order to retrieve the best possible probability estimate. We show one example in a experiment and read use this as a benchmark for our approach. The general field of application would be the analysis of time series.

Time series prediction is important to forecast e.g. economical data, and used to make decisions which, in turn, change economy. The term “causality” is used when past values of a time series provide significant information about future values of another time series [26]. One of the possible methods for causality inference is transfer entropy [25], however, it has the disadvantage of requiring a fair amount of data. Granger causality [20], on the other hand, is based on regression and uses less data, but is a linear method (non-linear extensions exist, see also [6] for a comparison between different methods). In this work, we propose a (non-linear) approach based on regression and a recent recurrent neural network learning method, which we

briefly revisit in Sect. 3.1 Recent advances in this area have shown to be successful in time series prediction [4, 5, 3]. However, instead of using predictions of the neural networks directly, we take a different route and describe an approach using the prediction error to detect causal links in time series (Sect. 3.2). Our approach is demonstrated using simulation data (Sect. 3.3), and finally, we discuss our results in Sect. 3.5.

## 2. Numerical Experiments with DBNs

### 2.1 Deep Belief Networks

DBNs are probabilistic generative models composed of multilayer directed networks of stochastic, latent variables. These stochastic latent variables then form into Restricted Boltzmann Machines (RBMs) a type of neural networks, which are the basic components of DBNs. These networks are "restricted" to a visible layer and a hidden layer, the latter can be viewed as a feature detector capturing correlations that observed at the former.

#### *A. Restrict Boltzmann Machine*

The Restrict Boltzmann Machines are the type of generative models which can be applied on many types of data such as binary images in MNIST database[5][4], gray-scaled images of 3D objects in NORB dataset[7],colored nature image[6], or mel-cepstral coefficients in speech recognition[8]. To train the RBMs, contrastive diversions[12] are often performed in order the obtain generative weights.

A RBM is a two-layer network in which visible layer and hidden are symmetrically connected and the siblings within one layer are not connected (see Fig. 1).

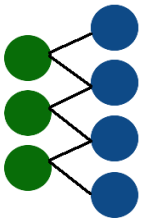

Fig. 1. Structure of RBMs, one RBMs contains a hidden layer and a visible layer.

The network takes the idea of "*energy*" as described by *Hopfield*, 1982, in which the joint configuration of a pair (v,h) has an energy *E(v,h)* :

$$E(v,h) = -\sum_{i \in visible} a_i v_j - \sum_{j \in hidden} b_i h_j - \sum_{i,j} v_i h_j w_{ij} \quad (1)$$

where *i* indicates visible unit and *j* indicate hidden unit, $a_i$, $b_j$ are their biases respectively and $w_{ij}$ is the weight between them.

For each possible pair *(v,h)*, the energy function is:

$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \quad (2)$$

The probability of visible vector or training image, v, is given as (3).

$$p(v) = \frac{\sum_h e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \quad (3)$$

Hence, we can compute the derivatives of the log probability of data, and the maximum likelihood learning rule for data vector v can be defined as (4).

$$\frac{\partial log p(v)}{\partial \omega_{ij}} = < v_i h_j >_{data} - < v_i h_j >_{model} \quad (4)$$

Then we can repeatedly update the weight, $w_{ij}$, of visible unit i and hidden unit j. Which lead to (5).

$$\triangle w_{ij} = \varepsilon(< v_i h_j >_{data} - < v_i h_j >_{model}) \quad (5)$$

where ε is a learning rate.

A fast learning algorithm is proposed in [4] which has termed *contrastive divergence*. Refer to [4] and [14] for further details.

#### *B. Deep Belief Networks*

As stated before, DBNs are probabilistic generative models composed of multilayer directed networks of stochastic, latent variables. These stochastic latent variables then form a Restricted Boltzmann Machine, which is the basic components of DBNs. According to [16], DBNs have two most significant properties:

*1. "There is an efficient, layer-by-layer procedure for learning the top-down, generative weights that determine how the variables in one layer depend on the variables in the layer above."*

*2. "After learning, the values of the latent variables in every layer can be inferred by a single, bottom-up pass that starts with an observed data vector in the bottom layer and uses the generative weights in the reverse direction."*

### 2.2 DBNs on Learning Binary Data

Deep Belief Networks can deals with many different types of sensory input data, including binary colored image, colored nature image, speech phones or even video sequence. In this paper, we want to examine the ability of DBNs on binary encoded sensory input. In this section, we will first depict our experimental setup and we will show that DBNs works well on extracting features from binary encoded input data.

### *A. Experimental Setup*

In this experiment, we first generate a double precision number of base 10 from a certain distribution, then we encoded this number into binary base according to IEEE-754[17]. (see the shaded area of Fig. 2).

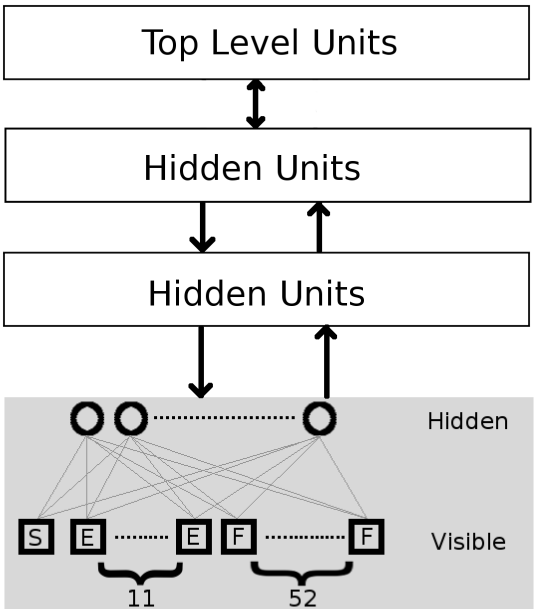


Fig. 2. Structure of our Deep Belief Networks. The shaded area explain how we encoded into binary number. Shape squares stands for each node in visible layer, and circle stands for each node in hidden layer. Character 'S' stands for *sign bit* in IEEE-754, 'E' stands for *exponent bits* and 'F' stands for *fraction bits.* This two layer form a Restricted Boltzmann Machine in Deep Belief Networks.

Later, we construct the DBNs as in Fig. 2, there are four hidden layer and one visible layer(input layer), the size of each hidden layer is set as following respectively, 256, 512, 256 and 64. For each RBMs, we train each layer for 100 epochs, each epoch training through the whole data set once. After training, in order to examine the result, we let the network runs as generative mode for the exact the same amount of data as training set.

### B. Experiment Result

#### *1. Normal Distribution*

In this experiment, we have tested two normal distribution, one with mean at 0 and another with mean at 10. We generate the number from the function of these two distributions as input data for DBNs respectively. Fig. 3 shows that DBNs can learn the exact mean and deviation of normal distributions and reconstruct the identical distribution.

#### *2. Poisson Distribution*

In experiment 1, we have tested the **continuous** distribution. This experiment test test the networks with Poisson distribution, which has the **discrete** property. Fig. 4 shows DBNs can also

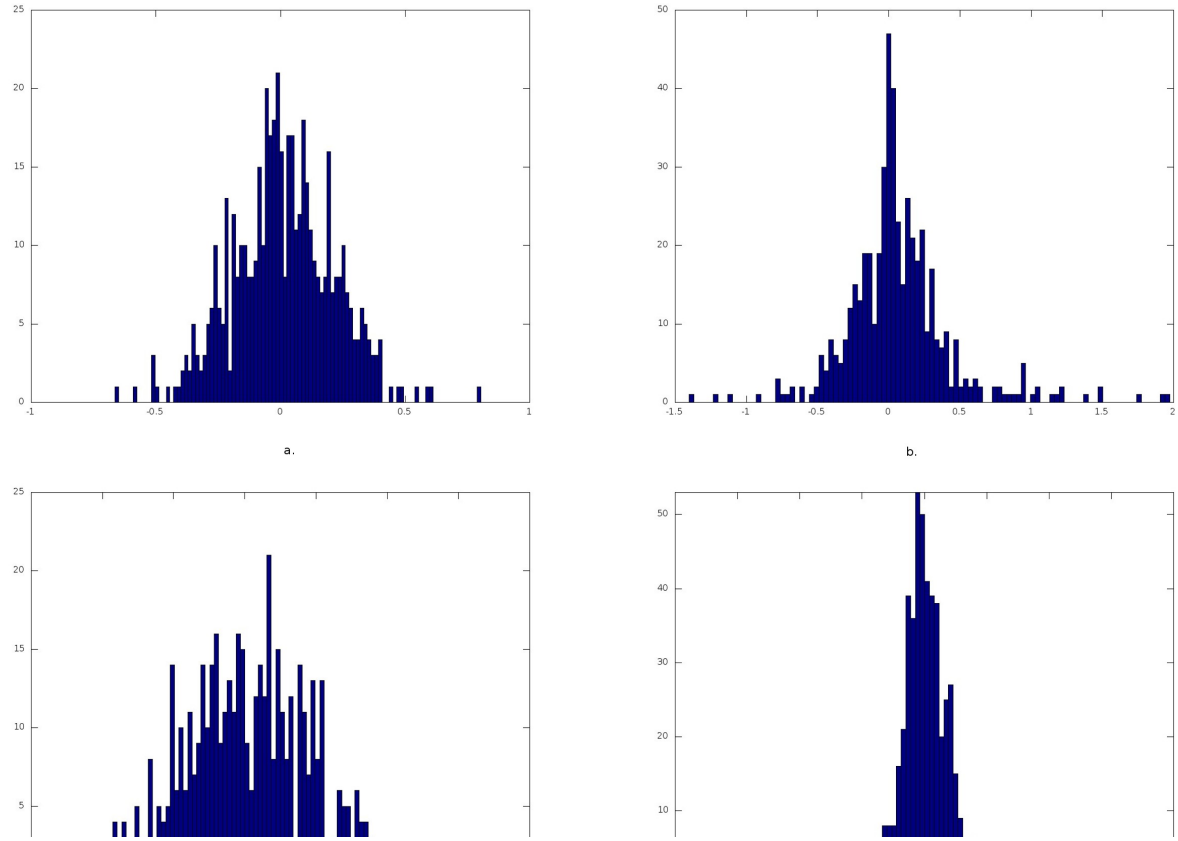


Fig. 3. The result of learning normal distribution with mean 0 and 10. Picture **a** is the normal distribution with mean 0, **b** is the learning result of reconstructed distribution. Picture **c** is the normal distribution with mean 10, **d** is the result of reconstructed distribution learning from **c**.

learn the discrete distribution.

*3. Random Number Generator*

In the last experiment, we examnine with a uniform distribution. We tested the network with 5000 random numbers. The result is presented in Fig. 6, one can see that most of the reconstructed numbers lay within the range between 0 and 1. This result needs more fine tuning, for example, there's a small gap neer 0 with extremely low probabiliy compared to the reigon nearby and not every reconstructed number is between 0 and 1. In conclusion, the network can get a approximation of our test set.

## 2.3 Combing DBNs with Domestic Service Robots

Among the ideas, we took the advantages of great performance of deep autoencoder[8]. In the service robot competitions, the robots must operate in everyday human environments, where they have to perform a given set of service tasks, such as following instructions like gesture, voice or even remote control signals. In voice recognition task, we first train the users voice using CMUSphinx[9], the output sentence of CMUSphinx need further treatment since we can not assure zero error in speech recognition. In natural language, one or more mistakes in one sentence may lead to the meaningless task. Hence, we feed those results into our DBNs modules and train the network to categorize semantics into certain task. Another application could be categorizing environment. We take kitchen as example, one could easily imagine what types of objects may appeare inside the kitchen environment such as a pot, oven or stove. We will leave the topic as an open issue here, additional works need to be done in the future. Thus, we see good reason to investigate for the service robot tasks. In the following passage we give a example in phone recognition as present in [4]. By doing above experiments, we have shown that Deep Believe Networks can successfully learn the probability distribution with binary encoded dataset. With this property, we can further

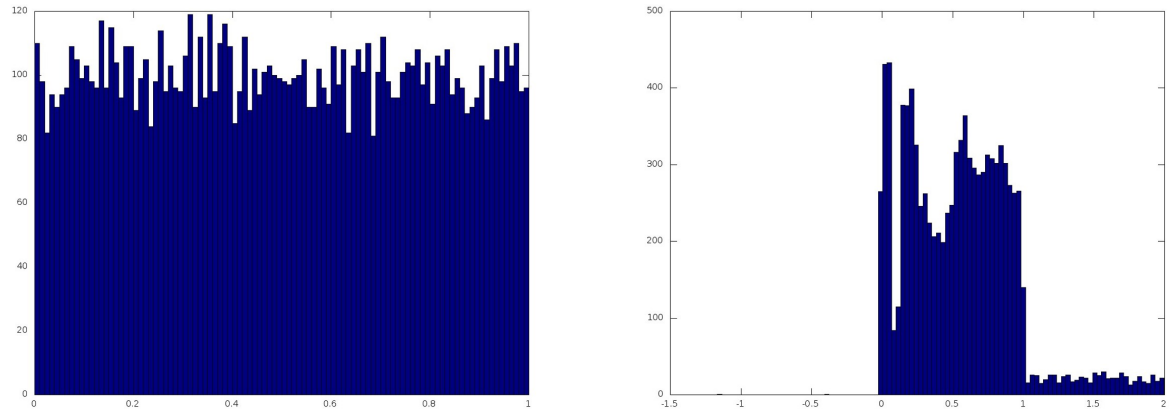

Fig. 4. The result of learning Uniform distribution from random number generator. Left picture shows the original distribution, on the right hand side is the reconstruction of it.

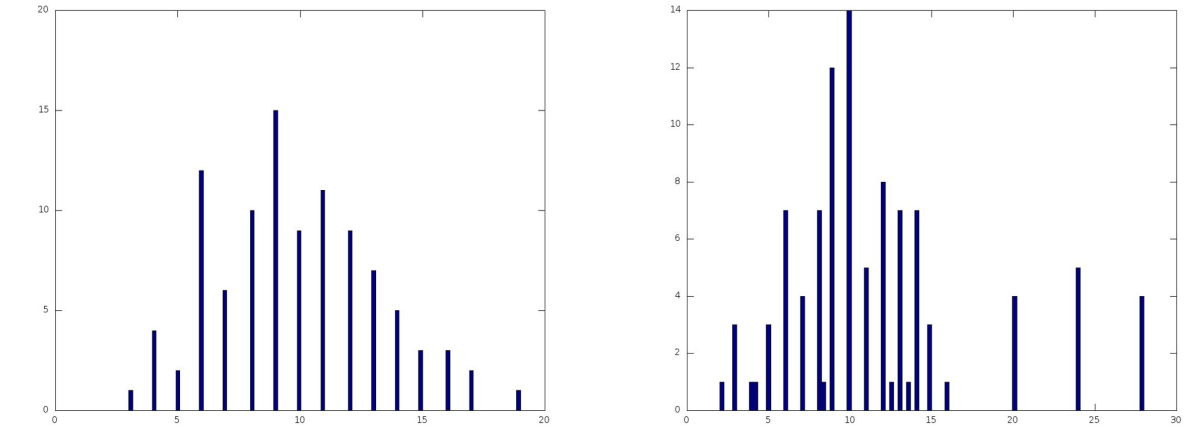

Fig. 5. The result of learning Poisson distribution. Left picture shows the original distribution, on the right hand side is the reconstruction of it.

extend DBNs into state prediction application. In robotics, we often are confronted with with the problem of state prediction, for instance, ball prediction. In domestic service robot, particularly in mapping and tracking, state prediction of self-localization and state of the tracked object, one may use DBNs to calculate the posterior of Bayesian filters.

## 3. ESN as a likelihood estimator

### 3.1 Echo State Networks

Echo State Networks (ESN) are an approach to address the problem of slow convergence in recurrent neural network learning. ESN consist of three layers (see Fig. 6): a) an input layer, where the stimulus is presented to the network; b) a randomly connected recurrent hidden layer; and c) the output layer. Connections in the output layer are trained to reproduce the training signal. The network dynamics is defined for discrete time-steps t, with the following equations:

$$x_{lin,t+1} = Wx_t + w^{in}u_t \quad (6)$$

$$x_{t+1} = \tanh(x_{lin,t+1}) \quad (7)$$

$$O_t = w^{out}x_t \quad (8)$$

where the vectors $\mathbf{u_t}$ , $\mathbf{x_t}$ , $\mathbf{o}_t$ are the input and the neurons of the hidden layer and output layer respectively, and $\mathbf{w^{in}}$ , $\mathbf{W}$, $\mathbf{w^{out}}$ are the matrices of the respective synaptic weight factors.

Connections in the hidden layer are random but the system needs to fulfil the so-called echo state condition. Jaeger [22] gives a definition; in the following a slightly more compact form of the echo state condition:

Consider a time-discrete recursive function $x_{t+1} = F(x_t , u_t)$ that is defined at least on a compact sub-area of the vector-space $x \in R^n$ . and where $x_t$ are to be interpreted as internal states and $u_t$ is some external input sequence, i.e. the stimulus.

The definition of the echo-state condition is the following: Assume an infinite stimulus sequence: $\bar{u}^{\infty} = u_0 , u_1 , \ldots$ and two random initial internal states of the system $x_0$ and $y_0$ . To both initial states $x_0$ and $y_0$ the sequences $\bar{x}^{\infty} = x_0 , x_1 , \ldots$ and $\bar{y}^{\infty} = y_0 , y_1 , \ldots$ can be assigned.

$$x_{t+1} = F(x_t , u_t) \quad (9)$$

$$y_{t+1} = F(y_t , u_t) \quad (10)$$

Then the system F (.) fulfils the echo-state condition if independent from the set $u_t$ and for any $(x_0 , y_0)$ and all real values $\in > 0$ there exists a $\delta(\in)$ for which

$$d(x_t , y_t) \leq \in \quad (11)$$

for all $t \geq \delta$. The ESN is designed to fulfil the echo state condition.

#### 3.1.1 Online learning using recursive least squares

ESN can be trained using either an offline or an online learning procedure. For our approach, we are online learning the output layer using the recursive least square method (RLS). The combination of ESN and RLS has first been published by Jaeger [23]. The following update rule was used:

$$\alpha_t = s^t_{teach} - w^{out}_{t-1} \cdot o_t , \quad (12)$$

$$g_t = p_t \cdot o_t / (\lambda + o_t^T \cdot p \cdot o_t) , \quad (13)$$

$$pt = 1./\lambda \cdot p_t - g_t \cdot o_t^T \cdot p_t / \lambda , \quad (14)$$

$$w_t{}^{out} = w_t{}^{out} + (\alpha_t \cdot g_t^T) , \quad (15)$$

where $\alpha_t$ represents the linear error vector and $p_t$ the inverse of the autocorrelation, $\lambda$ is close to 1 and is used as forgetting factor.

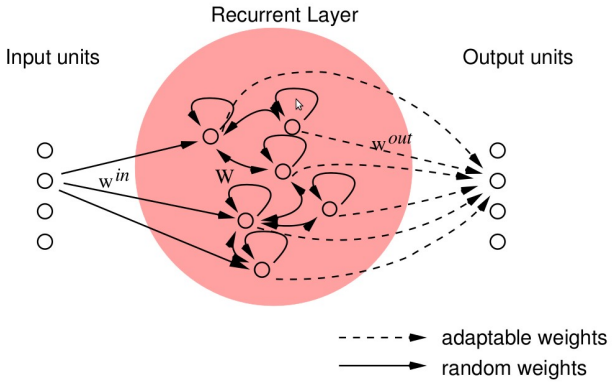


Fig. 6. ESN networks: Principle setup

### 3.2Modelling probability distributions by using the mean square error

In the following we reproduce one idea that was outlined in the tutorial of Jaeger [22] that we also used as the basis in our previous publication[3]. Instead of training the output $\omega_t \in \Omega$ directly, we model a probability that a specific event has occurred with regard to the output. In other words: the aim is to train the network in that way that each of the output units represents the probability of an event. As one simplest way to do this we teach the output or of the network to reproduce the probability that the as a range of the – statistical – output variable $\Omega_r \subset \Omega$ that is of interest for the given task. The task of network is to find $p(\Omega_r | x_t (\bar{u}^{\infty}))$ in the following written short $p(\Omega_r)$. We define u the teaching signal dr as:

if ($\omega_t \in \Omega_r$) $d_r = 1$

else $d_r = 0$

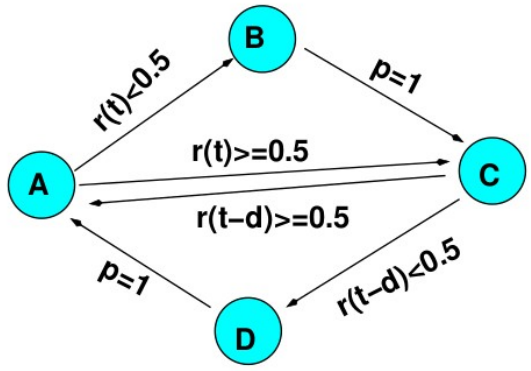


Fig. 7. Test model set-up.

The mean square error (MSE) is the

$$E_{mse} =< (d_{r,t} - o_r)^2 >= p(\Omega_r)(1 - o_r)^2 + (1 - p(\Omega_r))o^2{}_r \qquad (16)$$

The derivative $\partial E_{mse} / \partial o_r$ set equal to zero yields the point at which $E_{mse}$ is minimal:

$$o_r - p(\Omega_r) = 0 \qquad (17)$$

Thus, the MSE is reached when or = $p(\Omega_r)$; we can assume

$$o_r \rightarrow p(\Omega_r), \qquad (18)$$

for sufficiently long learning sequences. Since – with the common restrictions of reservoir computing– the full information of the input history is encoded in the activity state of the reservoir. Thus, –without additional efforts in the hidden layer– more information about statistical variables can be retrieved from additional output units: Because the optimal solution (absolute minimum of the MSE) can be derived, the network is going to find the true probability as far as it is detectable by linear regression from the current state of the reservoir. Usually, the quality of the network performance and the learning progress can be checked by measuring $E_{mse}$ , where values close to zero represent a good network performance. It should be noted that for the learning rule outlined above the theoretical limit is above zero. Under the assumption that the p(Ωr ) is the true probability we get:

$$E_{min}(\Omega_r) = \min_{or}(E_{mse}) = p(\Omega_r)(1 - p(\Omega_r)). \qquad (19)$$

However, since in fact the true value p(Ωr ) is unknown, it is not a good idea to use Emse − Emin as a measure. However, it can be used to find out if the output node is deterministic (i.e. the output node takes either 0 or 1). In this case the minimal error is in fact 0 again.

Instead one could go the following way in that we can get a set of outputs that covers a complete range of the random variable in the way that for a range r ∈ R:

$$\cup_{r \in R} \Omega_r = \Omega, \qquad (20)$$

$$\Omega_i \cap \Omega_j = \varnothing, \qquad (21)$$

for all i = j. Obviously, we have $r \in R\ p(\Omega_r) = 1$. We can test the constraint in the network. We test the quality of the network output by testing measuring r or which should be close to one if the network has adapted sufficiently.

Basing on the plausibility constraint it is very easy to define an error function for the network. In the simplest case, it can be assumed to train 2 outputs, of which the first represents the occurrence of an event $e \in \Omega_x$ , whereas the second output is trained to record the non-occurrence $e \in \Omega_x$ . Thus, I train (d1 = 1, d2 = 0) in the case $e \notin \Omega_x$ and else ($d_1 = 0$, $d_2 = 1$). In this case it can be assumed that the cost function

$$E_{total} = (o_1 + o_2 - 1)^2 \qquad (22)$$

approaches zero after a sufficient long learning process since from Eq. 18 we get

$$E_{total} \rightarrow (p(\Omega_x) + p(\Omega_x) - 1)^2 = (p(\Omega_x) + (1 - p(\Omega_x)) - 1)^2 = 0. \qquad (23)$$

Thus, this energy function may serve better as an estimator how well the network has adapted to the particular current input history.

## 4 .Simulation details

We demonstrate the approach on a prediction task. Our test model cycles between four states (A, B, C, D). Figure 7 outlines the transition probabilities. Every time in which the state A is reached a random number $0 \leq r(t) < 1$ is drawn. In every time step the model transfers from one state to the next state. Is the current state the state A and the random value r(t) smaller then 0.5 the model goes to state B else the model transfers to state C. From state B the model transfers to C. From C a the next step is either D if r(t − d) < 0.5 else the model returns to state A, where d is a positive or zero delay constant (0 delay indicates that the transition C-A happens in the same cycle as A-C, i.e. the complete cycle becomes A-C-A. Is the model in state D it always transits back to A.

The task of the network is to predict the next state from the previous. Had the states A, B, C, D been interpreted directly as Markov states the transitions from C to the next state would appear random with equal probability either to state A or D, i.e. in

that interpretation the model is non-Markov. However, a perfect hidden Markov model (HMM) with 2d hidden states would – presumably– be able to detect the fact that the choices of the transitions from A and those from C are linked.

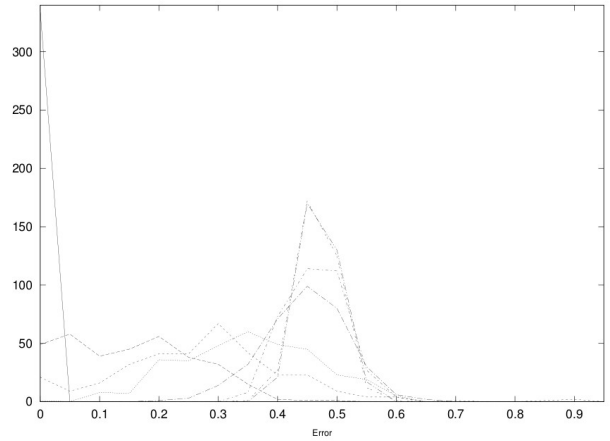


Fig. 8. Histograms of errors for different delays. Depicted is the MSE of probability of the transition to state D in the event of initial state C. If the network cannot detect the causality of transitions from C the error is at best 0.5. Full lines depicts the error at delay 0, dashes delay 1, small dashes delay 2, dotted delay 3, dash-dotted delay 4, double dashed delay 5, small dash-dotted delay 6.

The model is presented as an 8 dimensional vector in the following way to the network:

$$A = [1, 0, 0, 0, 0, 1, 1, 1]$$
$$B = [0, 1, 0, 0, 1, 0, 1, 1]$$
$$C = [0, 0, 1, 0, 1, 1, 0, 1]$$
$$D = [0, 0, 0, 1, 1, 1, 1, 0]$$

The second half of each vector represents the inverse of the first. Thus, it can serve to find the cost function according to Eq. 23. Obviously, the task becomes more complex as the delay constant increases. The inverse correlation matrix was set to $p = 0.0001 \cdot I$ where I is the identity matrix. The forgetting factor $\lambda$ was set to 1.0, which sets the RLS into the non-forgetting mode. The recurrent matrix was set to random orthonormal matrix which was multiplied by 0.98 (Fig. 8) and 1.14 (Fig. 9), which gives a slightly over-critical network. Since the input practically is never close to 0, the network stays non critical. The online learning was performed from the 1000th step onwards up to 18000th step. The different MSEs were recorded in the last 100 steps of each simulation.

## 5 .Results

We tested the network for several delays and network sizes. To estimate the network performance we used the MSE error at the transition from node C, in following $EM\ SE,C$ . A network that is able to detect the causality between the history and the transition from node C can reach zero MSE, whereas for network that cannot detect the causality the transition appears to be stochastic with equal probability to state D and A. The MSE in this case can be determined by Eq. 19.

A histogram of errors for different delays is depicted in Fig. 8. For sake of simplicity we used the MSE of transitions from A (stochastic, $EM\ SE,A$ ) and B (deterministic, $EM\ SE,B$ ) in Fig. 9. The plot depicts the values of

$$E_{norm} = (E_{MSE,C} - E_{MSE,B})/(E_{MSE,A} - E_{MSE,B}). \quad (24)$$

Thus, values of Enorm around 0 can be interpreted in that way that the network is able to detect the causality relation between transitions from A and C. Fig. 8 results for different network sizes and delays. Each line represents different the performance of one network and different delays. The network sizes are 10,20,30,50,100,150, and 200 neurons in the hidden layer.

The value of $E_{total}$ (cf. Eq. 23)) shows a very fast convergence to the final range almost immediately after the learning starts.

## 6 .Discussion

In the case of DBNs we investigated several ways to insert continuous input to the network and in this way to retrieve samples with correct probability distributions. The results are promising though we were not able to exactly achieve the correct distribution ranges. In the case of ESN we demonstrated an approach able to detect causalities in time series by using the mean square error of an ESN online learning procedure. Our current results indicate that the ability of this approach is limited to a few cycles– equivalent some dozens of steps. It can be expected that our results can be further improved by adapting the reservoir to the stimulus statistics. It would be very good if both approaches could be connected in order to combine virtues of both ESNs and DBNs.

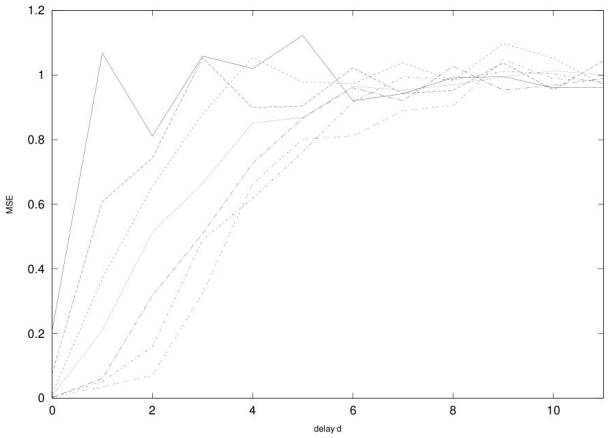


Fig. 9. Normalised errors for different network sizes at the hidden layer as a function of the delay. Full line 10 neurons, dashed 20 neurons, small dashes 30 neurons, dotted 50 neurons, dash-dotted 100 neurons, double dashed 150 neurons, and small dash-dotted 200 neurons.

[1] S. Thun, W. Burgard, D. Fox, Probabilistic robotics, MIT Press, 2006

[2] L-W. Lu, C-H. Lee, N. M. Mayer, Combining Deep Believe Networks with Domestic Service Robot**,** The 43rd Intl. Symp. on Robotics (ISR2012),Taipei, Taiwan, Aug. 29-31, 2012(in preparation)

[3] Norbert Michael Mayer, Oliver Obst, Chang Yu-Chen , Time Series Causality Inference Using Echo State Networks. Latent Variable Analysis and Signal Separation - 9th International Conference, LVA/ICA 2010, St. Malo, France, September 27-30, 2010. Proceedings; 01/2010

[4] Hinton, G. E., Osindero, S. and Teh, Y., "A fast learning algorithm for deep belief nets," Neural Computation 18, pp 1527-1554., 2009.

[5] THE MNIST DATABASE of handwritten digits, Yann LeCun, Corinna Cortes, http://yann.lecun.com/exdb/mnist/

[6] Ranzato, M., Krizhevsky, A. and Hinton, G. E., "Factored 3-way restricted Boltzmann machines for modeling natural images," Proc. Thirteenth International Conference on Artificial Intelligence and Statistics, 2010.

[7] THE NORB DATASET of 3D object recognition from shape, Fu Jie Huang, Yann LeCun, Courant Institute, New York University, 2004, http://www.cs.nyu.edu/ ylclab/data/norb-v1.0/

[8] Mohamed, A. R., Dahl, G. E. and Hinton, G. E., "Deep belief networks for phone recognition," NIPS 22 workshop on deep learning for speech recognition, 2009.

[9] Deng, L., Seltzer, M., Yu, D., Acero, A., Mohamed A. and Hinton, G., "Binary Coding of Speech Spectrograms Using a Deep Auto-encoder," Interspeech, Makuhari, Chiba, Japan, 2010.

[10] Carnegie Mellon University, The cmu sphinx group open source speech recognition engines, http://cmusphinx.sourceforge.net/html/cmusphinx.php

[11] Qi is a tool for rapid prototyping of integrated behavior systems on a team of distributed embedded systems we developed.

[12] G. E. Hinton, "Training products of ecperts by minimizing contrastive divergence," Neural Computation , vol. 14, pp. 1771-1800, 2002

[13] K.-F. Lee and H.-W. Hon, "Speaker-independent phone recognition using hidden markov models," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, no. 11, pp. 1641-1648, 1989.

[14] G. E. Hinton, "A practical guide to training restricted Boltzmann machines", 2010

[15]Arel, I.; Rose, D.C.; Karnowski, T.P.; , "Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]," Computational Intelligence Magazine, IEEE , vol.5, no.4, pp.13-18, Nov. 2010, doi: 10.1109/MCI.2010.938364

[16] Geoffrey E. Hinton (2009), Scholarpedia, 4(5):5947.http://www.scholarpedia.org/article/Deep_belief_networks

[17] IEEE Computer Society (August 29, 2008), IEEE Standard for Floating-Point Arithmetic, IEEE, doi:10.1109/IEEESTD.2008.4610935, IEEE Std 754-2008

[18] T. Lee and D. Mumford, "Hierarchical Bayesia inference in the visual cortex," J. Opt. Soc. Amer., vol.20, pt.7, pp.1434-1448, 2008

[19] Joschka Boedecker, Oliver Obst, N. Michael Mayer, and Minoru Asada. Initialization and self-organized optimization of recurrent neural network connectivity. HFSP J., 3(5):340–349, 2009. doi: 10.2976/1.3240502.

[20] C. W. J. Granger. Investigating causal relations by econometric models and cross- spectral methods. Econometrica, 37(3):424–438, 1969.

[21] Barbara Hammer, Benjamin Schrauwen, and Jochen J. Steil. Recent advances in efficient learning of recurrent networks. In ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning, pages 213–226, 2009.

[22] H. Jaeger. The 'echo state' approach to analysing and training recurrent neural networks. In GMD Report 148, GMD German National Research Insitute for Computer Science, 2001.

[23] Herbert Jaeger. Adaptive nonlineaer systems identification with echo state networks.

In Advances in Neural Information Processing Systems; Proceedings of the NIPS 15, pages 609–615, 2003. AA14.

[24] E. Pereda, R. Quian Quiroga, and J. Bhattacharya. Nonlinear multivariate analysis of neurophysiological signals. Progress in Neurobiology, (77):1–37, 2005.

[25] Thomas Schreiber. Measuring information transfer. Physical Review Letters, 85(2):461–464, July 2000.

[26] Takashi Shibuya, Tatsuya Harada, and Yasuo Kuniyoshi. Causality quantification and its applications: structuring and modeling of multivariate time series. In KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 787–796, New York, NY, USA, 2009. ACM.