# Evolving Causal Neural Networks

**Marco A. Wiering**
Institute of Information and Computing Sciences
Intelligent Systems Group
Utrecht University
PO Box 80.089, Utrecht
*marco@cs.uu.nl*

## Abstract

We introduce causal neural networks, a generalization of the usual feedforward neural networks which allows input features and target outputs to be represented as input or output units. For inferring the values of target outputs which are represented as input units, we developed a forward-backward propagation algorithm which uses gradient descent to minimize the error of the predicted output features. To deal with the large number of possible structures and feature selection, we use a genetic algorithm. Experiments on a regression problem and 5 classification problems show that the causal neural networks can outperform the usual feedforward architectures for particular problems.

## 1 Introduction

Bayesian belief networks (Pearl, 1988) are of large interest due to their graphical structure modelling independencies between variables, and the sound Bayesian inference algorithm. Given a set of instantiated values of particular observed features, Bayesian belief networks can be used for inferring the probability distribution of unobserved features. Different neural network algorithms such as Boltzmann machines (Aarts and Korst, 1989) and Sigmoid Belief networks (Neal, 1992) are quite similar to Bayesian belief networks, but exact inference is infeasible in all of these algorithms in case the models become more complex.

In this paper we introduce a much simpler class of neural network architectures which we call causal neural networks, since they also allow for representing causal relations between variables. The structure of the causal neural network allows us to represent target output values in the input layer and we can also represent input feature values in the output layer of a feedforward neural network. The latter has already been studied in (Caruana and de Sa, 1997) and was shown to be effective for particular problems. The causal neural network is a further generalization of the well-known structure of feedforward neural networks. For inferring the value of an output value which is represented as an input unit, we can use backward propagation of error signals of the predicted input feature values represented in the output layer. This backward propagation algorithm is a simple variant of the well known backpropagation algorithm (Rumelhart et al., 1986). The new resulting forward-backward propagation algorithm is used recursively for inference to minimize the error of predicted feature values.

Similarly to Sigmoid Belief networks, causal neural networks try to find those values of target outputs which are useful for generating the desired input feature values. Although there is no direct causal relationship from input feature values represented by output units to target outputs represented as input units, minimizing the reconstruction error causes the system to infer those target outputs which cause the generation of the given input feature values. The causal neural networks can therefore be used to put causes in the input units and the effects of these causes in the output units. This can simplify the learning problem drastically and lead to better generalization.

The learning algorithm for training the causal neural networks is a simple extension of the normal backpropagation algorithm and therefore learning is quite fast. To deal with the large number of possible structures, we employ a genetic algorithm for finding the best structures. In the following, we will use the causal neural networks as a supervised learning algorithm which can deal with missing data and causal relations in a principled way.

**Outline of this paper.** In section 2 we describe our causal neural networks with the forward-backward algorithm for inference. In section 3 we describe the genetic algorithm for evolving causal neural network structures. In section 4 we describe experimental results. Finally, section 5 concludes this paper.

## 2 Causal neural networks

**The learning task.** We study supervised learning on a dataset $D$ consisting of $L$ input vectors $X^i$ and their target output vectors $Y^i$. The dataset may consist of unknown input features which may make

the learning task more difficult. We assume that target outputs in the training data are always known, but could be subject to noise.

In this section, we first describe the architecture, and then the usual forward propagation and backpropagation algorithms for training a causal neural network. To deal with unknown values, we add an additional mean-input for all uninstantiated variables which is used to initialize the values of uninstantiated variables. Finally, we describe the recurrent forward-backward propagation algorithm for inferring the desired output values and unknown input features given the values of instantiated variables.

### 2.1 The architecture

The causal neural networks could have an arbitrary (modular) structure, but in this paper we only consider fully-connected feedforward neural networks with a single hidden layer. The architecture consist of one input layer with input units[1]: $I_1, \ldots, I_{|I|}$, where $|I|$ is the number of input units, one hidden layer $H$ with hidden units: $H_1, \ldots, H_{|H|}$, and one output layer with output units: $O_1, \ldots, O_{|O|}$. The network has weights: $w_{ih}$ for all input units $I_i$ to hidden units $H_h$, and weights: $w_{ho}$ for all hidden $H_h$ to output units $O_o$. Each hidden unit and output unit has a bias $b_h$ or $b_o$ with a constant activation of 1. The hidden units use Sigmoid activation functions, whereas the output units use linear activation functions. Finally, the input units and output units have an additional mean-bias $m_i$ and $m_o$ (different for every unit), which is used to deal with unknown initial values for the uninstantiated variables.

We can represent input features and target outputs inside the input layer or in the output layer. Input features could also not be used at all which allows for feature elimination. The input features which are represented in input (output) units form the set $FI$ ($FO$), and the target outputs which are represented in the input (output) units form the set $TI$ ($TO$). Figure 1 shows an example of a causal neural network architecture. The figure also shows the forward and backward propagation algorithms used for inference.

The set $FO$ is usually disjunct from the set $FI$, since otherwise reconstruction of input feature values in $FO$ would be trivial and not useful for inferring unknown values of target outputs. The input feature values in $FO$ are useful, since the goal is to reconstruct their values from the values of the inputs units. In order to reconstruct these, causal explanations of these values must be found in the (known or unknown) input feature values and the unknown target outputs of the set $TI$. If the set $FI$ is empty, the algorithm tries to infer the values of target outputs in $TI$ which minimize the reconstruction error. If $FI$ is not empty, these feature input values serve as a context for this reconstruction.

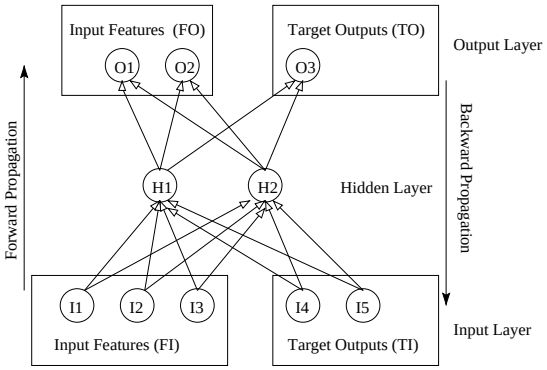


Figure 1: *A causal neural network consisting of one hidden layer. Input features and target outputs can be represented as input or output units.*

**Transformation of examples.** Given an example $(X^i, Y^i)$ we can represent the input features and target outputs as either input or output units. Depending on the neural network structure which maps input features and target outputs to input and output units, we get a new training example $(\hat{X}^i, \hat{Y}^i)$. To deal with unknown values for variables (e.g. missing data or the target outputs) which are represented as input units, we use the values of their mean-bias $m_i$ learned by the delta rule.

### 2.2 Forward propagation

Given the values of all input units, we can compute the values for all output units with forward propagation. The forward propagation algorithm for an instance $X$ looks as follows:

1) Clamp the known input feature values $\in FI$ in the input layer: $I_i = X_i$, and clamp the mean-bias for unknown feature values $\in FI$ and for target output values $\in TI$ to the input layer: $I_i = m_i$.

2) Compute the values for all hidden units $H_h \in H$ as follows:

$$H_h = \sigma\Big(\sum_{i \in FI \cup TI} w_{ih} I_i + b_h\Big)$$

Where $\sigma(x)$ is the Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

3) Compute the values for all output units $O_o \in FO \cup TO$:

$$O_o = \sum_h w_{ho} H_h + b_o$$

### 2.3 Backpropagation

For training the system we extend the backpropagation algorithm (Rumelhart et al., 1986) in which we also learn the mean-bias values using the delta rule. If feature values in $FO$ are unknown, we use

[1] When we refer to a unit, we also mean its activation.

their mean-bias as a target output value. Although leaving them completely out in the learning algorithm would be another possibility, preliminary experimental results indicate that our approach can be more efficient. The learning goal is to learn a mapping from the transformed instance $\hat{X}$ to $\hat{Y}$. For this we first use forward propagation to compute the outputs values $O_o$ and then we use backpropagation to minimize the squared error measure:

$$E = \frac{1}{2} \sum_{o \in FO \cup TO} (\hat{Y}_o - O_o)^2$$

To minimize this error function, we update the weights and biases in the network using gradient descent steps with learning rate $\alpha$:

$$\Delta w_{ho} = -\alpha \frac{\partial E}{\partial w_{ho}} = \alpha (\hat{Y}_o - O_o) H_h$$

and

$$\Delta w_{ih} = \alpha H_h (1 - H_h) I_i \sum_{o \in FO \cup TO} (\hat{Y}_o - O_o) w_{ho}$$

Update the mean-bias values for input and output units using the delta rule:

$$m_i = m_i + \alpha(\hat{X}_i - m_i); \quad \text{and} \quad m_o = m_o + \alpha(\hat{Y}_o - m_o)$$

### 2.4 Forward-backward propagation

For inferring the values of all unknown variables given the instantiated values of known input features we introduce the forward-backward propagation algorithm:

1) Clamp the input units to their input feature values. Unknown values for input features and target outputs $\in TI$ will initially have their mean-bias values as input value, after which their activations will be continually updated.

2) Clamp the target output values in $\hat{Y}$ for all input features $\in FO$ to their value in the instance $X$. For unknown values of input features we use their mean-bias values as target output values in $\hat{Y}$.

3) Use forward propagation to infer the output values $O_i \in FO \cup TO$.

4) Use backward propagation to update the values of uninstantiated input units by gradient descent on the error over the predicted input features $\in FO$:

$$\Delta I_i = \alpha_b \sum_h H_h (1 - H_h) w_{ih} \sum_{o \in FO} (\hat{Y}_o - O_o) w_{ho}$$

5) Repeat Step (3) until some termination condition holds.

Here, $\alpha_b$ is the backward learning rate. We recursively apply the forward-backward algorithm to infer the values of all uninstantiated variables. The algorithm may not always converge, however, since values of uninstantiated variables in the input layer may become infinite to best predict the output features. In the experiments we iterate the algorithm for 400 times which therefore makes inference much slower than with the usual feedforward neural networks. The learning time is exactly the same, since for learning we just make use of the backpropagation algorithm.

## 3 Evolving causal neural networks

The additional freedom we gain with our causal neural networks can also give us problems, since how can we decide whether we should represent an input feature by an input or output unit, and should the target output(s) be represented as input or output unit(s)? For optimizing the structure of the causal neural networks, we use genetic algorithms (Holland, 1975) and cross-validation. Thus, we divide the total dataset $D$ in three data-sets: the learning dataset $D_l$, the cross-validation (halting) dataset $D_c$, and the test dataset $D_t$. In our experiments, we use the causal neural networks for regression and classification purposes. For the regression task it is possible that some of the target output values are represented as input units and others as output units. Therefore we can evolve mixed architectures. For the classification task there is only one target output value (the classification of an input pattern), and we use the genetic algorithm only with populations of homogeneous structures in which the target output value is always represented by an output unit or by an input unit (we call the corresponding networks forward networks (FN) or backward networks (BN)). Input features can be used as input units, output units, or not at all. When the target outputs are represented as input units, the structure should contain at least one input feature in the output layer. Furthermore, we evolve the number of hidden units and the learning rate.

**The genetic algorithm.** We use crossover and mutation operators for evolving the population of structures. We use tournament selection with size 3 and the elitist strategy. Given a structure, we train it $n_t$ times on the learning dataset $D_l$ and use cross-validation after each $t_c$ iterations to look whether we should stop the learning process. The best test-error over these $n_t$ train-test trials is kept as fitness value for the structure. After evolving $G$ generations, we compute the error of the best trained individual during the last generation on the test dataset $D_t$, and return this as a single experimental result.

## 4 Experiments

We performed a number of experiments to validate the usefulness of our approach compared to the usual feedforward neural networks structures trained with

backpropagation. We use two different experiments. The first is a regression task in which there are 3 input features and 3 target outputs. In the second experiment, we use 5 real-world datasets from the UCI repository (Merz et al., 1997).

### 4.1 The regression problem

The input consists of three inputs $X_1, X_2, X_3$ which are drawn from a random uniform distribution between 0 and 0.33. The target outputs are computed as:

$$Y_1 = -log(\frac{10.0}{\sigma(X_1) + \sigma(X_2) + X_3} - 1)$$

$$Y_2 = (X_1^2 + X_2^2) \times -log(\frac{10.0}{\sigma(X_1) + \sigma(X_2) + X_3} - 1)$$

$$Y_3 = (\sqrt{X_1} + \sqrt{X_2}) \times -log(\frac{10.0}{\sigma(X_1) + \sigma(X_2) + X_3} - 1)$$

Here $\sigma(x)$ denotes the Sigmoid function. This is an artificial problem which we constructed to show the advantage of using mixed architectures. We can see that $Y_1$ is a building block for $Y_2$ and $Y_3$, which means that we could put $Y_1$ in the input layer, together with $X_1$ and $X_2$ to infer $Y_2$, $Y_3$, and also $X_3$. We will refer to this architecture as a mixed architecture. Another advantage of this mixed architecture is that inferring $X_3$ from $Y_1$, $X_1$ and $X_2$ may be easier than inferring $Y_1$ from $X_1$, $X_2$, and $X_3$, since the function is the same as: $X_3 = 10\sigma(Y_1) - \sigma(X_1) - \sigma(X_2)$. Thus, we do not have to approximate the logarithmic function in this way.

**Simulation set-up.** We compare the normal feedforward network (FN) with no input features in the output layer to the backward network (BN) with all outputs in the input layer and no other inputs (which is of course not suited for this task), and to the mixed architecture with $Y_1$ and $X_1$, $X_2$ in the input layer, and $Y_2, Y_3$ and $X_3$ in the output layer. We also evolve mixed architectures using the genetic algorithm with a population size of 30, 50 generations, and $n_t = 1$. For the non-evolving architectures, the number of hidden units is 5. The learning rate $\alpha = 0.2$ and $\alpha_b = 0.1$. We perform 100 simulations with different distributions for the three datasets consisting of 100 examples, for which 10 networks of each type are trained and tested using cross-validation. The trained network with the smallest cross-validation error is used for testing on the test dataset $D_t$. For this the root of the mean squared error is computed. The results are shown in table 1.

**Experimental results.** The figure shows that the mixed architecture performs better than the usual feedforward network. Thus, putting target output values inside the input layer of a feedforward neural network can be useful. As expected, the complete backward network is not suited for this task. The genetic algorithm is shown to be able to almost always find the best possible architecture (the Mix-NN architecture).

Table 1: *The Training results (RMSE) on the 3 input 3 output function. Averages are computed over 100 simulations.*

| FN | BN | MIX-NN | GA MIX-NN |
|---|---|---|---|
| 0.0095±0.0075 | 0.105±0.012 | 0.0068±0.0026 | 0.0071±0.0034 |

### 4.2 Experiments on real datasets

We also performed experiments on real datasets from the UCI repository (Merz et al., 1997). We use the following 5 datasets consisting of three medical diagnosis datasets: Hepatitis (155 examples), Liver disease (345), Pima Indians (768), and two other datasets: Chess Kr-vs-kp (3196), and Vote (435). We also make 10% of the input features in the four datasets unknown to examine how unknown (missing) values affect the performance of the different neural network architectures. For these binary classification problems, we threshold at 0 to compute the output class. The results are given as percentage of wrong classifications.

**Simulation set-up.** We run 10 simulations with different random partitionings of the total dataset into $D_l$, $D_c$, and $D_t$; all datasets have $\frac{1}{3}$ of the number of examples of the total dataset. We use the genetic algorithm on forward (GA-FN) and backward networks (GA-BN) and let it run for 50 generations using a population size of 30 and $n_t = 3$. For the normal forward and backward architectures which are not evolved, we use 4 hidden units for which 500 networks are trained and tested each on the cross-validation dataset. The best was used for testing on the test dataset $D_t$.

Table 2: *The Training results on the 5 datasets with and without adding missing (unknown) input feature values.*

| Data Set | Mis. | FN | BN | GA-FN | GA-BN |
|---|---|---|---|---|---|
| Hepatitis | 0% | 0.46±0.11 | 0.37±0.04 | 0.38±0.07 | 0.37±0.05 |
| Hepatitis | 10% | 0.51±0.09 | 0.36±0.06 | 0.42±0.08 | 0.36±0.04 |
| Liver Dis. | 0% | 0.43±0.04 | 0.39±0.05 | 0.44±0.05 | 0.40±0.04 |
| Liver Dis. | 10% | 0.43±0.05 | 0.39±0.06 | 0.43±0.04 | 0.41±0.07 |
| Pima Ind. | 0% | 0.25±0.04 | 0.24±0.03 | 0.26±0.05 | 0.24±0.03 |
| Pima Ind. | 10% | 0.35±0.07 | 0.24±0.03 | 0.30±0.05 | 0.24±0.03 |
| Chess | 0% | 0.05±0.01 | 0.25±0.02 | 0.07±0.02 | 0.07±0.01 |
| Chess | 10% | 0.14±0.01 | 0.25±0.02 | 0.17±0.02 | 0.14±0.01 |
| Vote | 0% | 0.06±0.02 | 0.11±0.02 | 0.07±0.02 | 0.07±0.01 |
| Vote | 10% | 0.08±0.02 | 0.11±0.02 | 0.07±0.01 | 0.08±0.02 |
| Average | | 0.276 | 0.271 | 0.261 | 0.238 |

**Experimental results.** The results are given in table 2. For the Hepatitis problem the neural networks do not perform very well; this can be explained by the small number of learning examples (there were only 52 examples used for training). Different learning rates or numbers of hidden units did not improve the results. The forward networks perform better than the backward networks on Vote and Chess, but it is interesting to see that the backward networks perform better on the medical diagnosis datasets in which the disease causes the observed symptoms. The backward models also suffer much less from missing data. Using the genetic algorithm can help to find better structures, which is especially clear for the backward networks which can combine input features with target outputs in the input layer to learn correct mappings for Vote and Chess.

## 5 Conclusion

We described a new neural network architecture which is a generalization of normal feedforward neural networks. These causal neural networks can represent feature inputs and target outputs in the input or output layer of a feedforward network. Although we only used them for supervised learning tasks in this paper, the causal neural networks can also be used for pattern completion, and extended for unsupervised learning purposes. The training algorithm of the causal neural networks is very fast, since it is a simple extension of the normal backpropagation algorithm, although the question may arise whether this is a valid thing to do if there are missing values. For inference in our causal networks we developed the forward-backward propagation algorithm which should be executed recursively to infer all the values of all uninstantiated variables. We want to look closer at stopping conditions for this propagation and the effects of early stopping on the obtained approximation.

Experiments on a regression task and 5 datasets from the UCI repository show that the causal neural networks with target outputs in the input layer can outperform the usual feedforward neural networks for medical datasets in which target outputs (diseases) cause the input features (symptoms). Furthermore, these backward networks are much less sensitive to missing data. For other problems, architectures found by a genetic algorithm which mix input features and target outputs in the input layer performs quite well compared to the usual feedforward neural networks.

The causal neural networks are similar to Sigmoid Belief networks in some way, since they both try to infer causal output variables given instantiated feature values. The difference is that Sigmoid Belief network use a probabilistic setting and try to maximize the likelihood of generating the instantiated feature values, whereas in causal neural networks the goal is to minimize the squared error of the instantiated feature values represented as ouput units.

It is interesting to analyse the meaning of hidden unit representations in the case of having target outputs represented by input units. In the case of non-linear dimensionality reduction (DeMers and Cottrell, 1993) with a four layer feedforward neural network with a bottleneck, the hidden units in the second layer represent a non-linear manifold for reconstructing the input. This may also be applied for supervised learning in which the input and output consist of feature inputs and target outputs. Reconstructing the output could then be done by propagating the values of known feature inputs. A similar technique was applied in (Hancock and Thorpe, 1994) for training a car to follow roads. Here the authors computed the eigenvectors of the generated camera images and a given corresponding steering command to train the system. Afterwards, the system was applied by linearly combining the eigenvectors for the newly generated camera images which also gives the control commands. It remains a question whether such reconstructions can always be used to generate the correct output.

In causal neural networks, we are less interested in dimensionality reduction, but try to evolve an architecture which can use the input units to infer the values of separate output units. In this case the hidden units repesentations should be mixed to infer the values of output units. The mixture is dependent on the values of input units. Therefore if an target output is represented as input unit, forward-backward propagation derives its value which causes the hidden units to be mixed in the best possible way for reconstructing the given values of output units. It should be clear that this cannot be done with all possible architectures, therefore a suitable architecture needs to be evolved.

In future work, we want to study more general graphical structures to model the dependencies between variables. For learning and inference we can employ variations of the backpropagation and forward-backward propagation algorithms.

## References


A.H.L. Aarts and J.H.M. Korst. 1989. *Simulated Annealing and Boltzmann Machines.* Wiley, Chichester.

R. Caruana and V.R. de Sa. 1997. Promoting poor features to supervisors: Some inputs work better as outputs. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9.* Morgan Kaufmann, San Mateo, CA.

D. DeMers and G.W. Cottrell. 1993. Non-linear dimensionality reduction. In S.J. Hanson C.L. Giles

and J.D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 580–587. Morgan Kaufmann, San Mateo, CA.

J.A. Hancock and C.E. Thorpe. 1994. ELVIS: Eigenvectors for land vehicle image system. In *CMU-RI-TR*.

J.H. Holland. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

C.J. Merz, P.M. Murphy, and D.W. Aha. 1997. UCI repository of machine learning databases.

R.M. Neal. 1992. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.