

Filling in the Gap: a general method using Neural Networks

Rui Rodrigues¹

¹Dep. Matemática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal

Abstract

When a set of medical signals has redundant information, it is sometimes possible to recover one signal, from its past and the information provided by the other signals. In this work, we present a general method to realize that task.

It has been known for a long time that multilayered networks are universal approximators, but, even with the backprop algorithm, it was not possible to train such a network, to realize complex real life tasks. In the last years, Geoffrey Hinton presented a training strategy that allows to overcome the previous difficulties. We describe a way of adapting Hinton's strategy to our task.

An example of a situation considered here, consists on training a Multilayered perceptron to take ECG leads II and I as input and produce as output missing lead V.

This method got the best scores among participants in the Physionet/ Computing in Cardiology Challenge 2010.

1. Introduction

When continuous observations are required for clinicians and researchers, it is important to be able to reconstruct, in medical records, signals that were temporary corrupted or lost. The aim of the PhysioNet/Computing in Cardiology Challenge 2010 was to develop robust methods for filling in gaps in multiparameter physiologic data [1]. The participants were given ten-minute records containing 6 to 8 signals, acquired from Intensive Care Unit(ICU) patient monitors, and the final 30-second segment from one signal was set to zero. The challenge was to reconstruct this missing signal segment (gap) on each record.

The method we present consists on creating a multilayer perceptron neural network (MLP) that outputs the signal to be reconstructed, the target signal. This MLP receives as input a part of the other signals present in the record; for example, if the target signal is respiration (RESP), under certain conditions we use ECG II and central venous pressure(CVP) as input signals. The neural network we build has four hidden layers. Known difficulties to train such a network, that in some cases has about 800 000 weights, are overcome by a procedure that is adapted from Geoffrey

Hinton's ideas [2–4].

2. Methods

2.1. Data

The Data is the PhysioNet/Computing in Cardiology Challenge 2010 set C¹[1,5]. It consists on 100 10-minute records from an intensive care unit, each one composed by 6 to 8 signals (ECG, continuous blood pressure, fingertip plethysmograms(PLETH) and others). After 9 minutes and 30 seconds one of the signals dropped to zero: this is the signal to be reconstructed.

2.2. Choosing the input signals

For each record it is chosen a subset of the available signals to reconstruct the target signal; it contains from 1 to 3 signals. We'll refer to these subset as the *input signals*. For example, when reconstructing RESP, used input signals were: PLETH and ECG II in record C00, ECG II and CVP in record C03, ICP and PLETH in record C12 and only ICP in record C63.

2.3. Creating patches

The MLP we build receives as input, and outputs, small patches from the signals.

The patches duration is:

- 3 seconds for input and target signal when the target signal is RESP
- 1 second for target signal and all input signals but RESP, when target signal is not RESP; when RESP is used as an input signal, we take RESP patches with the duration of 2 seconds, starting 1 second before the target signal patch.

We produce the training patches from the first 9 minutes and 30 seconds of the input signals and target. Two consecutive patches start with a delay of 5 samples (0.04 seconds). Like this we get about 14 000 training patches for each signal.

¹viewable with a browser in <http://www.physionet.org/cgi-bin/ATM>

2.4. Preprocessing

All signals are normalized to zero average and unit standard deviation, before patches are extracted; to reduce computational burden, all but ECGs and ICP patches are subsampled.

2.5. MLP structure

Accordingly to [6], an MLP with several hidden layers is expected to produce better results than a single hidden layer network; even if our task is reasonably simpler than those referred by the authors. We use a four hidden layer MLP, with linear units in the input and output layers, and logistic units in the hidden layers.

2.6. Training

Following the ideas of [2–4, 7, 8] first we create an autoencoder for the input signals and another for the target signal.

2.6.1. The autoencoder

An autoencoder is an MLP where the output layer should reproduce the input. The middle layer can be seen as a *code* to represent the input. The layers until the middle layer define an *encoder* and the layers after the middle layer, inclusive, form a *decoder*.

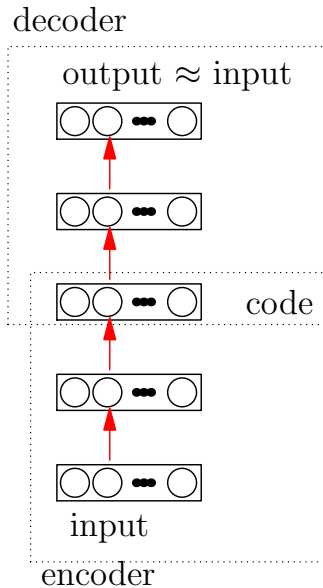


Figure 1. The autoencoder used in this work

For input signals we use a code longer than the input: our code is the set of the input signals features that are available to produce the target signal. More precisely, as

we will see, we use the input signals code to produce the target code. For the target signal we use a short code: computationally it gets more efficient.

Instead of initializing the weights of the autoencoders with random values we initialize them with those of a stack of Restricted Boltzmann Machines.

2.6.2. Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a network with two layers: visible and hidden with no intralayer connections. Connections in a RBM are bidirectional. In the main version all units are binary and stochastic, but here input units will be real valued and we will start with binary hidden units and in a next step real valued. The activation function we use in the visible units is either the logistic function or the identity. Weights and bias are randomly initialized.

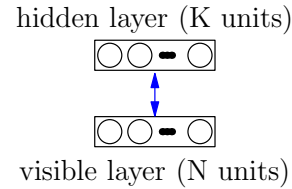


Figure 2. Restricted Boltzmann Machine- bidirectional weights

In the first step of the RBM training phase we will use *Contrastive Divergence* (CD) learning [9, 10], to ensure that the value of each visible unit depends on the value of many hidden units (distributed representation). CD-learning (here we use 1-step) consists on updating the connection weights in the following way:

- Given the values v_i of the RBM input layer, the binary state, h_j , of hidden unit j , is set to 1 with probability

$$\sigma(b_j + \sum_i v_i w_{ij})$$

where σ is the logistic function, b_j is the bias of hidden unit j and w_{ij} is the weight of the connection from visible unit i to hidden unit j .

- From the 0-1 values of the hidden units we generate new values \tilde{s}_i for the visible units. In the case of linear visible units, we set $\tilde{s}_i = b_i + \sum_j h_j w_{ij}$ and in the case of logistic visible units the value of s_i is given by $\tilde{s}_i = \sigma(b_i + \sum_j h_j w_{ij})$.
- The added value Δw_{ij} to w_{ij} will be

$$\Delta w_{ij} = \epsilon \left(\left\langle v_i * \sigma(b_j + \sum_i v_i w_{ij}) \right\rangle - \left\langle \tilde{v}_i * \sigma(b_j + \sum_i \tilde{v}_i w_{ij}) \right\rangle \right)$$

Where ϵ is a small constant and $\langle . \rangle$ represents the average value on the data.

In a second step of the RBM learning procedure, to accelerate reduction of reconstruction error, we use *Mean Field* (deterministic) learning [11]. The difference to CD-learning is: instead of generating the new values, \tilde{s}_i , from the sampled 0-1 values, h_i , of the hidden units, we generate them from the probabilities, $\sigma(b_j + \sum_i v_i w_{ij})$. The updating rule for the weights remains the same.

2.6.3. Stacking two RBMs to build an autoencoder

To initialize the weights and bias of a signal's autoencoder, the first RBM we build has the signal as its visible layer; the values of the hidden layer from the first RBM will be used as the visible layer of the second RBM. Therefore, the first RBM has linear visible units, and, the second has logistic visible units.

Afterwards, the bidirectional weights of the first and second RBM are used for the directional weights of the autoencoder. In a general RBM, we will note W and W^T , respectively, the matrices of the weights from the visible to the hidden layer, and, from the hidden to the visible layer. In this way, we get the matrices W_1, W_1^T, W_2 and W_2^T .

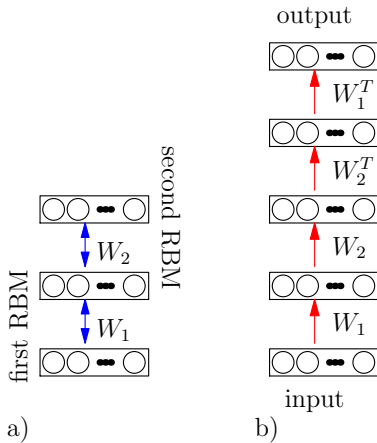


Figure 3. a) the stack of RBMs and b) the autoencoder we build from it

With these weights, we build an MLP whose input layer is the visible layer of the first RBM, the first hidden layer is the hidden layer from the first RBM, the second hidden layer is the hidden layer from the second RBM, the third hidden layer is again the hidden layer from the first RBM and the output layer is the visible layer from the first RBM. The bias from the RBMs units are used in the obvious way in these MLP. This is the way we initialize the autoencoder.

To improve accuracy on reconstructing the input, we update the weights using standard *backprop* algorithm.

2.6.4. Building the final four hidden layer MLP from two autoencoders

To get our final four layer MLP, we connect, the encoder from the input signals, with the decoder from the target signal, using a simple perceptron. This perceptron receives as input, the code from the input signals, and should output the code from the target signal.

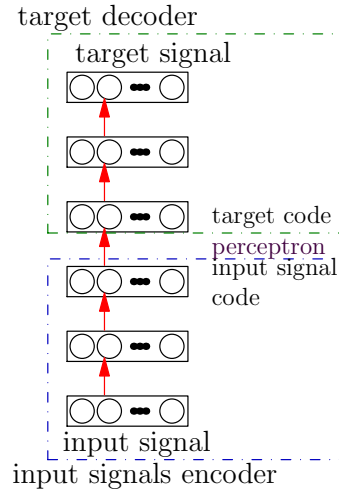


Figure 4. Initializing the final 4-hidden layer MLP from the input signals and target autoencoders

The perceptron is initialize with random weights, and after, we update the weights according to the rule:

$$\Delta w_{ij} = \langle input_i output_j \rangle - \langle input_i \widetilde{output}_j \rangle$$

where $input_i$ and $output_j$ are data values and \widetilde{output}_j is the 0-1 output produced by the perceptron with the present weights, sampling from the Bernoulli distribution with $p = \sigma(b_j + \sum_i s_i w_{ij})$.

In a second step we use a similar learning rule, but set $\widetilde{output}_j = \sigma(b_j + \sum_i s_i w_{ij})$.

After learning the perceptron weights, we improve the weights between the first, second and third layers, of our final MLP. For that, we use backprop algorithm on a new MLP, with one hidden layer, builded from those layers and the weights between them.

To finish training, we use backprop on the final four layer MLP.

2.7. Using the patches to reconstruct the missing signal

To get the 30-second missing signal, we averaged, for each data point, the contribution of all the reconstructed patches that contain the point.

3. Results

The method is computationally very demanding, but, in most of the signals from the data set, the results are very good.

The autoencoders associated with the signals, input and target, are able to extract the main features, and reproduce the most important characteristics of the signals, even in the presence of much noise. In the great majority of the records, it is possible to get a very small error, in the training data, when producing target signal from the input signals. In a small number of records, that was not enough to get a good reconstruction of the target signal, in the last 30 seconds.

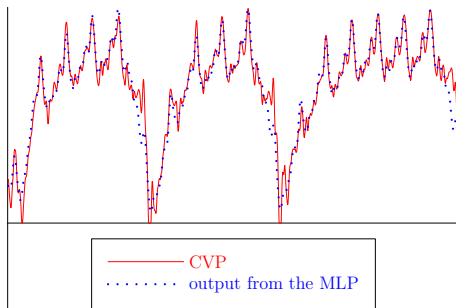


Figure 5. A 10-second patch of target signal CVP and reconstruction, during training time, in record c59.

In the Challenge there were two events, each one with its scoring algorithm:

Event 1- Let K be the quotient between the sum of the squares of the errors and the signal variance. The score of a reconstruction is the maximum between zero and $1 - K$. It measures the overall accuracy of the reconstruction.

Event 2- The score of a reconstructions is the maximum between zero and the correlation coefficient of the signal and its reconstruction.

The final score on each event is the sum of the scores on each of the 100 records.

This method got a score of 83 on event 1 and 90.6 on event 2. These were the best scores among participants in the 2010 PhysioNet/Computing in Cardiology Challenge.

4. Discussion and conclusions

We described here a method to reconstruct the last 30 seconds of a signal, using (some of) the other signals in the record, and the past of the missing signal. The method needs, first of all, to be able to learn to reproduce the target signal from the other signals, during training time (9 minutes and 30 seconds). For that, the input signals must provide enough information, to extract a rule that outputs the missing signal, even if only approximately. Secondly,

to be able to reproduce a certain pattern of the missing signal, this methods needs the same pattern to be significantly present in training time, otherwise that behavior is mostly not learned.

The computational efficiency of this method should be improved, perhaps finding some shortcuts.

References

- [1] Moody GB. The physionet/computing in cardiology challenge 2010: Mind the gap. In Computing in Cardiology 2010, volume 37. Belfast, 2010; [in press].
- [2] Hinton GE. Reducing the dimensionality of data with neural networks. *Science* July 2006;313(5785):504–507.
- [3] Hinton GE. Learning multiple layers of representations. *Trends in cognitive Sciences* 2007;11:428–434.
- [4] Hinton GE. To recognize shapes, first learn to generate images. In P. Cisek TD, (Eds.) JK (eds.), *Computational Neuroscience: Theoretical Insights into Brain Function*. City, State of Publication: Elsevier, 1983; 400–402.
- [5] Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation* 2000 (June 13);101(23):e215–e220. *Circulation Electronic Pages*: <http://circ.ahajournals.org/cgi/content/full/101/23/e215>.
- [6] Bengio Y, Lecun Y. Scaling learning algorithms towards ai. In Bottou L, Chapelle O, Decoste D, Weston J (eds.), *Large-Scale Kernel Machines*. MIT Press, 2007; .
- [7] Hinton GE. Learning to represent visual input. *Philosophical Transactions of Royal Society B* 2010;365(5785):177–184.
- [8] Bengio Y, Lamblin P, Popovici D, Larochelle H. Greedy layer-wise training of deep networks. In Schölkopf B, Platt J, Hoffman T (eds.), *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press, 2007; 153–160.
- [9] Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. *Neural Comput* 2006;18(7):1527–1554. ISSN 0899-7667.
- [10] Carreira-Perpiñan MA, Hinton G. On contrastive divergence learning. In Cowell RG, Ghahramani Z (eds.), *aistats05. Society for Artificial Intelligence and Statistics*, 2005; 33–40.
- [11] Hinton GE. Boltzmann machine. *Scholarpedia* 2007; 2(5):1668.

Address for correspondence:

Rui Rodrigues
Dep. de Matemática, Faculdade de Ciências e Tecnologia
2829-516 Caparica, Portugal
rapr@fct.unl.pt