Time series analysis of Gravitational Wave signals using neural networks

M. Carrillo, J. A González, M. Gracia-Linares, F. S. Guzmán
Instituto de Física y Matemáticas, Universidad Michoacana de San Nicolás de Hidalgo.
Edificio C-3, Cd. Universitaria, 58040 Morelia, Michoacán, México.

Abstract. We present an algorithm based on neural networks that predicts the mass ratio in a binary black hole collision out of given Gravitational Wave (GW) signals. In this brief analysis, the network is trained with a small sample of GW signals generated with numerical simulations. The effectiveness of the algorithm is evaluated with GW generated again with simulations with given mass ratios unknown to the network and found to be in the worst scenarios less than 10%.

1. Introduction
The sources of Gravitational Waves (GWs) are of garden variety, from sources such as binary systems involving black holes and neutron stars, supernovae core collapse, down to high energy accretion processes [1]. Compact binary systems are the most promising ones to be detected [2], because of the large expected event rate [3].

The set of parameters characterizing a binary black hole collision involves the parameters of the orbit before the collision, the orientation of the independent spins of the black holes, their mass ratio and an extraordinary amount of parameters involved if matter surrounding the system are taken into account. Speeding up the estimation of these parameters is a matter of the major importance because of the amount of computational time for the signal analysis and filtering. Parameter estimation counts with efficient techniques, including for instance the use of Markov chains [4] and interpolation of waveforms [5], which have shown to be very efficient. However, it is still important to reduce the computational costs of estimations.

The characterization of the GW source involves the analysis of the properties of the signal (assuming this has been filtered) involves inverse problem strategies. As a first step, and in order to investigate whether neural networks can help at the estimation of parameters, we introduce in this paper an algorithm based on neural networks that is capable to predict the parameters of the source. At this stage we still do not consider noise filtering or statistical inference and only assume GW signals are given, and the goal is to give an approximate of the parameters value. At this point we also are wondering about the precision of the parameter prediction and will worry later about the efficiency.

We restrict to the case of binary black hole collisions and only one parameter, namely the mass ratio between the black holes. We generate GW signals using the Einstein Toolkit code [6] to generate GW signals that we use to train our network. Then provided a new signal, also generated through simulations, we challenge the network to guess the black holes mass ratio. This allows us to estimate the effectiveness of the network and the possibilities it has to tackle problems involving more parameters.
The paper is organized as follows. In section 2 we present a general description of artificial neural networks, in section 3 we present the specific properties of the simulations used to train the network, in section 4 we show the efficiency of the algorithm and in section 5 we draw some conclusions and perspectives.

2. Neural network description

Artificial Neural Networks (ANN) arose as an attempt to emulate the nervous system in both organization and stimuli processing from internal and external environment of living organisms [7]. Basically, a neural network is fed with some variables or parameters of a particular problem, and after a series of calculations it spits out a concrete outcome correlated with the input data. This kind of artificial intelligence is commonly used in pattern classification and recognition, which makes it specially appropriate to analyze the problem we intended to engage here.

The first notions on artificial neural network were established in 1943 by Warren McCulloch and Walter Pitts [8], creating a mathematical model for simple computational logical operations. Neural networks were proposed, in analogy to their biological counterparts, as a set of interconnected processing units called neurons, responsible for all calculations and data transfer, as described in Figure 1. A neuron can be defined with four components:

(i) Input nodes or synapses. A neuron has a determined number of nodes, depending on input connections. The value from each node \( x_i \) is weighted by a determined value \( w_i \), that will vary according to the relevance of the connection.

(ii) Transfer function. It handles the income information from synapses, mapping a vector to a real number, it usually is the sum over all the weighted inputs.

(iii) Activation function. It takes the outcome from transfer function to a determined range through a certain mathematical function. The most popular activation functions, see figure 2, are the sigmoid function:

\[
\frac{dy}{dx} = y(1-y) \tag{2}
\]

or logistic function and the hyperbolic tangent:

\[
y(x) = \tanh x \tag{3}
\]

\[
\frac{dy}{dx} = 1 - y \tag{4}
\]

however one may propose other functions that suit the problem, like sinusoidal, heaviside or even a linear function [9].

(iv) Connections. Deliver the neuron’s outcome in a certain layer to neurons in the next layer.

Neural networks may have a lot of branches, depending on the number of neurons, but essentially are catalogued into recurrent and feedforward networks or non-recurrent. In recurrent networks, loops between neurons are allowed; meanwhile in a non-recurrent configuration, information flows only in one direction. Here we are focused on feedforward networks, since they are more widespread than recurrent networks, and can be used in countless applications despite their relative simplicity. The most common feedforward network is the multilayer perceptron (MLP), derived from the perceptron developed by Rosenblatt in 1958 [10]. The MLP is a fully connected network formed by 3 layers integrated by an arrange of neurons:

- Input Layer. This layer gets the parameters or patterns from outside, and pass them out over the next layer. For historical reasons, neurons at this level do not make calculations at all, and are used just for data transmission.

...
2.1. Mathematical Representation of Neural Network

In order to illustrate a neural network in organization and operation in a mathematical fashion, let us suppose a 3 layer feedforward network with \( n \) inputs, \( m \) hidden neurons and \( l \) outputs. Since in the input layer no calculations are made, the \( j \) hidden neuron has the input vector \( X_j = \{x_{1j}, \ldots, x_{ij}, \ldots, x_{nj}\} \) and its corresponding weight vector \( W_{ij} = \{w_{1j}, \ldots, w_{ij}, \ldots, w_{nj}\} \), where \( i \) is referred to an input neuron. Then the transfer function maps the input vector from \( \mathbb{R}^n \) to \( \mathbb{R} \), taking the linear combination of input vectors with the elements of the weight vector \( W_{ij} \) as coefficients:

\[
\eta = \sum_{i=1}^{n} x_{ij}w_{ij} + w_0 \tag{5}
\]

where \( w_0 \) is the weight of an extra synapse called bias with constant input equal to 1, added to shift the transfer function result or to avoid the particular case of a zero vector, in many cases it can be omitted for simplicity. Immediately after, the activation function \( F \) enters in action.
Figure 3. Scheme of the structure of a neural network and the usual estimation of error

giving the output $\sigma_j$:

$$\sigma_j = F(\eta) = F\left(\sum_{i=1}^{n} x_{ij}w_{ij} + w_0\right)$$

Finally, for the $k$ output neuron we have the result:

$$y_k = G\left(\sum_{j=1}^{m} \sigma_j \theta_j + \theta_0\right) = G\left(\sum_{j=1}^{m} \left(F\left(\sum_{i=1}^{n} x_{ij}w_{ij} + w_0\right) \ast \theta_j\right) + \theta_0\right)$$

where $G$ is the activation function and the $\theta_j$ are the weight coefficients corresponding to the input vector of the $k$ output neuron.

2.2. ANN Training

In order to obtain the expected output for given particular pattern, one must get the appropriate configuration and weight matrix in the ANN. The way to do this is by changing all the synaptic weights through a “training”. Suppose we have a set of $N$ training patterns $\{\tilde{x}^p, \tilde{y}^p\}$, where $p$ is the pattern number, $\tilde{x}^p$ and $\tilde{y}^p$ are the input and target output vectors respectively. In training phase, one seeks to reduce the error between the result $y^p$ thrown by the ANN and the truly desired output $\tilde{y}^p$, figure

The most common way to measure this difference is by taking the sum of squared errors in output neurons:

$$E^p = \frac{1}{2} \sum_k (\tilde{y}_k^p - y_k^p)^2$$

where $k$ is the index of the output neurons. The overall performance or global error is defined by:

$$E^G = \sum_p E^p = \sum_p \sum_k \frac{1}{2} (\tilde{y}_k^p - y_k^p)^2$$

2.2.1. Backpropagation Algorithm One of the most popular learning algorithms due to its acceptable performance in a great variety of problems, and its easily implementation in multilayer perceptron networks is the backpropagation learning algorithm. This method tries to search minimize the error function in terms of synaptic weights:
\[ \nabla E = \left( \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_k} \right) \rightarrow 0 \]  
(10)

The purpose is to change every weight for next time step \( t + 1 \), propagating the error from the outputs to the input neurons at time \( t \) by the rule:

\[ w_k(t+1) = w_k(t) + \Delta w_k(t) = w_k(t) - \gamma \frac{\partial E}{\partial w_k(t)} \]  
(11)

where \( \Delta w_k(t) = -\gamma \frac{\partial E}{\partial w_k(t)} \) and \( 0 < \gamma < 1 \) the learning constant. Continuing with our previous example, we have for the connection from neuron \( i \) to \( j \):

\[ \Delta w_{ij} = -\gamma \sum_{k=1}^{l} \frac{\partial E_k}{\partial y_j} \frac{\partial s_j}{\partial w_{ij}} = \gamma \delta_{ij} y_i \]  
(12)

with \( s = \sum w_{ij} y_i \), \( \delta_{ij} = -\frac{\partial E}{\partial w_{ij}} \) and \( \frac{\partial s}{\partial w_{ij}} = y_i \). Considering \( F \) as the activation function, then \( y = F(s) \) and \( \delta_{ij} \) is calculated as:

\[
\delta_{ij} = \begin{cases} 
(\hat{y}_k - y_k)y_k(1-y_k), & \text{if } j = k, k \in \text{Output neurons.} \\
0, & \text{if } j \neq k, \\
\sum_{k=1}^{l} [\delta_{jk} w_{jk}] \frac{\partial F}{\partial s_j} \frac{\partial s_j}{\partial w_{ij}}, & \text{if } j = h, h \in \text{Hidden neurons.}
\end{cases}
\]  
(13)

In case a sigmoid is used as activation function, see eq. (2), then eq. (13) becomes:

\[
\delta_{ij} = \begin{cases} 
(\hat{y}_k - y_k)y_k(1-y_k), & \text{if } j = k, k \in \text{Output neurons.} \\
0, & \text{if } j \neq k, \\
\sum_{k=1}^{l} [(\hat{y}_k - y_k)y_k(1-y_k)w_{jk}] y_j(1-y_j), & \text{if } j = h, h \in \text{Hidden neurons.}
\end{cases}
\]  
(14)

2.2.2. Scaled Conjugated Gradient Algorithm  The disadvantages of backpropagation come from the choice of the learning rate, momentum, number of trainings and the possibility of getting stuck in a local minimum. That is why new algorithms in neural networks have emerged, like the scaled conjugated gradient [12], which is more efficient at approaching the minimum and without a dependency to a learning rate parameter. However, it is not excluded from the choice of hidden neurons or iterations. This method is based on the hypothesis that the minimum of the functions are locally almost quadratic. The weight change is determined by:

\[ w_{t+1} = w_t + \alpha d_t \]  
(15)

\[ d_t = -g_t + \beta_t d_{t-1} \]  
(16)

with \( g_t \) and \( \beta_t \) defined as:

\[ g_t = \nabla E(w)|_{w=w_t} \]  
(17)

\[ \beta_t = \frac{g_T^T g_t}{g_{t-1} g_{t-1}} \]  
(18)

where \( d_t \) and \( d_{t-1} \) are the conjugate directions in successive iterations and \( T \) defines the transverse vector. The step value is set by the \( \alpha \) coefficient and the minimum search direction by \( b_t \). The \( \alpha_t \) value is calculated as:
\[ \alpha_t = -\frac{d_t^2 g_t}{\delta_t} \]  
\[ \delta_t = d_t^2 H_t d_t + \lambda_t ||d_t||^2 \]  

(19)  
(20)

\( \lambda_t \) is the scale coefficient. In case the error function is not quadratic, \( \lambda \) and \( \delta \) can be negative and the weight increment can lead to an error increase. A sufficiently high value of \( \lambda \) guarantees that the Hessian matrix \( H_t \) is positive definite. In the other case, if \( \delta < 0 \), \( \lambda \) must be increased so that \( \delta \) be greater than 0. A scale coefficient for \( \bar{\delta} \) is suggested:

\[ \bar{\lambda} = 2 \left( \lambda - \frac{\delta_t}{||d_t||^2} \right) \]  

(21)

And the new value for \( \delta \) is:

\[ \bar{\delta}_t = \delta_t + (\bar{\lambda}_t - \lambda_t) ||d_t||^2 \]  

(22)

This scaling also needs an adjustment to validate the quadratic approximation through:

\[ \Delta_t = 2 \frac{E(w_t) - E(w_t + \alpha_t d_t)}{\alpha_t d_t^2 g_t} \]  

(23)

If \( \Delta_t > 0.75 \) then the approximation is sufficiently good, and the value of \( \lambda_t \) can be reduced, proposing \( \lambda_{t+1} = \lambda_t/2 \). On the contrary, if \( \Delta_t < 0.25 \), \( \lambda_t = 4\lambda_t \).

3. Parameter recognition in gravitational waves

The first problem on the implementation of a neural network, is how to enter data, namely defining the meaning the inputs in the neural network. In this case, the data used is the gravitational wave signal generated by binary black hole simulations. The number of inputs in the neural network is a set of sampled points, and the value entered in each input corresponds to the amplitude of the detected signal at time step \( t \). This needs a great number of inputs to include at least the most significant part of the simulated signal, so that the GWs must be within an appropriate range in all cases.

As mentioned before, as a first step, we only consider a single parameter, the mass ratio between the black holes, while all other possible parameters are not considered. Unlike the real gravitational wave signals, with simulations it is possible to provide noise free signals with an arbitrary time resolution. The duration of the simulations is way to longer than the required information, and we only sample 300 time steps from the raw data, covering the region where the signature of the merger of the black holes on the gravitational waves are mostly present. In Figure 4 we show a sample of the GW signals used as input for the network.

These signals were generated with the Einstein Toolkit Cactus based code [6] considering the initial separation between black holes to be that of the first quasi-circular orbit [14] in order to obtain a quick merger for various mass ratios between the two black holes, ranging from 1 to 3.

The signal corresponds to the \( l = 2, m = 0 \) mode of the real part of the \( \Psi_{4} \) Weyl scalar measured at \( r = 30M \). As we can see, the segment where the GWs are appreciated, the signals are similar and the main difference comes up from the amplitude of the signal and a slight shift to the right in the same time segment. Then, the main function of the neural network will be to adjust itself to this differences and decrease the error through training with an appropriate number of GWs samples. The neural networks deployed were trained with 10 GWs samples with different mass ratios: 1.00, 1.10, 1.30, 1.40, 1.50, 1.60, 1.70, 2.10, 2.20, 2.30, and the following configurations:
Figure 4. Some of the gravitational waves used for training the neural networks. See how similar they are in shape due to the similar mass ratio.

(i) 300 input neurons.
(ii) Run over a range from 200 to 800 hidden neurons with hyperbolic tangent as activation function with increments of 100 neurons. To measure the dependency of the number of hidden neurons, four neural networks with initial random weights have been used for each configuration.
(iii) An output neuron with a sigmoid activation function for the mass ratio output prediction.

After the training we provide GW signals generated using simulations for a given mass ratio unknown to the network, that has to be predicted by the network. Considering we know the mass ratio used to generate the test GWs it is possible to measure the error in the network’s prediction.

4. Results
We thus estimate the mass ratio out of a given GW signal. In order to test the network we used 4 GWs corresponding to 1.20, 1.324, 1.9876, 2.2075 mass ratios shown in Figure 5. The algorithm implemented here is the scaled conjugate gradient, because it does not need to adjust a learning rate and momentum, leaving to explore just the number of trainings and hidding neurons. Given that the number of training samples is low, one can have hidden neurons in
Figure 5. Gravitational waves used to test the neural networks, with some variations in mass ratio. The label of each curve indicates the mass ratio between the black holes.

the order of the same inputs as well as the iterations. The mean square error (MSE) is used to measure the performance of neural networks:

$$MSE = \frac{1}{N} \sum_{i} (\hat{m}_i - m_i)^2$$  (24)

where $\hat{m}$ and $m$ are the real and the estimated value with index $i$ and $N$ is the number of mass ratios to be predicted. Since several neural networks are generated with a determined number of hidden neurons, an average of this MSE is calculated as shown in table 1.

Despite the number of samples, the network has reached a considerable low error in the validation test 1. Table 2 displays the averaged percent error predictions for each real mass ratio. A more considerable difference is observed in the region around 2.0 since there is a large gap in the training samples from 1.7 and 2.1 mass ratios.

5. Final comments
Using a neural network we have predicted the mass ratio of binary black hole out of the gravitational wave form produced during the collision, with a success of less then 1% in the best case and around 10% in the worst example, this depends on both the number of training patterns and whether the signal to be interpreted lies far from a sampled region.
Neural networks performance

| Neural network configuration | Training averaged MSE | Test averaged MSE |
|-----------------------------|-----------------------|------------------|
| NN-200                      | 1.12436E-03           | 2.0302E-02       |
| NN-300                      | 1.48354E-04           | 8.5020E-03       |
| NN-400                      | 1.1737E-03            | 8.4419E-03       |
| NN-500                      | 1.2051E-03            | 1.1810E-02       |
| NN-600                      | 1.1591E-03            | 2.0541E-02       |
| NN-700                      | 1.5617E-03            | 1.9152E-02       |
| NN-800                      | 1.4744E-03            | 1.4716E-02       |

Table 1. An average of the mean square error eq. 24, is calculated over all training and validation samples, for each initialized random neural network configuration with 500 trainings. The NN-XXX indicates the number of hidden neurons.

| Neural network | Real mass ratio | 1.2 | 1.324 | 1.9876 | 2.2075 |
|----------------|-----------------|-----|-------|--------|--------|
| NN-200         | 10.992          | 1.700| 12.551| 0.75   |
| NN-300         | 3.169           | 0.800| 8.846 | 0.158  |
| NN-400         | 5.109           | 1.792| 8.540 | 0.226  |
| NN-500         | 6.425           | 2.123| 9.300 | 0.271  |
| NN-600         | 11.904          | 6.647| 12.046| 0.2548 |
| NN-700         | 11.079          | 2.826| 11.641| 7.970  |
| NN-800         | 5.459           | 1.364| 11.386| 0.310  |

Table 2. Averaged percent mean squared error of the outcomes versus the real mass ratios, thrown by the 4 neural networks with different architectures. The NN-XXX indicates the number of hidden neurons.

We plan to extend this model to the analysis of more parameters concerning binary black holes, and eventually incorporate the algorithm to signal filtering libraries that will allow to measure the effectiveness in real situations of our methods. In regard to neural network improvements, it is needed to increase the number of training waves, explore a larger space of neural network architectures, compare with other training methods or even make use of genetic algorithms to minimize error. As well as define new ways to input information into the neural network like spectral analysis that potentially will speed up any parameter estimate.

Acknowledgments
This research is partly supported by CIC-UMSNH projects 4.9 and 4.23.

References
[1] B.S. Sathyaprakash and Bernard F. Schutz, Living Rev. Relativity 12 (2009), 2
[2] Gregory M. Harry and LIGO Scientific Collaboration, Advanced LIGO: The next generation of gravitational wave detectors, Classical Quantum Gravity 27, 084006 (2010).
[3] J. Abadie et al. (LIGO Scientific Collaboration and Virgo Collaboration), Classical Quantum Gravity 27, 173001 (2010).
[4] V. Raymond, M. V. van der Sluys, I. Mandel, V. Kalogera, C Rover and N. Christensen, Classical Quantum Gravity 26, 114007 (2009).
[5] R. J. E. Smith, K. Cannon, C. Hanna, D. Keppel, and I. Mandel, Phys. Rev. D 87, 122002 (2013).
[6] http://einstein Toolkit.org/
[7] Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, (Prentice-Hall, NJ, 1995), pp. 563-566
[8] W. McCulloch and W. Pitts, Bull. of Math. Biophys. 5, (1943), 115.
[9] B. Karlik and V. Olgac. Int. J. of Artificial Intelligence and Expert Syst., 1(4), (2010), 111.
[10] Frank Rosenblatt, Perceptrons and the Theory of Brain Mechanisms, (Spartan, NY, 1992).
[11] G. Cybenko, Mathematics of Control, Signals and Systems, 2(4), (1989), 599.
[12] Martin Møller, Neural Networks Lecture Notes in Computer Science, 6(4), (2012), pp. 525-533.
[13] IOP Publishing is to grateful Mark A Caprio, Center for Theoretical Physics, Yale University, for permission to include the iopart-num Bi\TeX\ package (version 2.0, December 21, 2006) with this documentation. Updates and new releases of iopart-num can be found on www.ctan.org (CTAN).
[14] G. B Cook, Living Rev. Relativity 3 (2000), 5.