



LEIBNIZ UNIVERSITÄT HANNOVER  
AND  
MAX PLANCK INSTITUTE FOR GRAVITATIONAL  
PHYSICS (ALBERT EINSTEIN INSTITUTE)

MASTER THESIS

# Analysis of Gravitational-Wave Signals from Binary Neutron Star Mergers Using Machine Learning

*Marlin Benedikt Schäfer*

*Supervisors: Dr. Frank Ohme and Dr. Alexander Harvey Nitz*

May 27, 2019

This page is intentionally left blank. LÖSCHEN!!! Damit Eigenständigkeitserklärung nicht auf Rückseite gedruckt ist.

I hereby assure that the thesis at hand has been constituted independently and without the use of any other than the cited sources. I furthermore assure, that all passages taken textually or analogously from other sources are marked as such.

This thesis, in its current or a similar form, has not been submitted to any other examination office.

---

Hiermit versichere ich, dass die vorliegende Arbeit selbständig und ohne Verwendung anderer Quellen, als den angegebenen, verfasst wurde. Zudem versichere ich, dass alle Stellen, die wörtlich oder sinngemäß aus anderen Quellen entnommen wurden, als solche gekennzeichnet sind.

Diese Arbeit hat so oder in einer ähnlichen Form noch keiner anderen Prüfungsbehörde vorgelegen.

---

Ort, Datum

---

Marlin Benedikt Schäfer

This page is intentionally left blank. LÖSCHEN!!! Damit Inhaltsverzeichnis nicht auf Rückseite gedruckt ist.

## Abstract

Put the abstract here

This page is intentionally left blank. LÖSCHEN!!! Damit Inhaltsverzeichnis nicht auf Rückseite gedruckt ist.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Gravitational-Wave signals from binary neutron star mergers</b>	<b>2</b>
2.1	The waveform . . . . .	2
2.2	Matched filtering . . . . .	2
<b>3</b>	<b>Neural networks</b>	<b>3</b>
3.1	Neurons, layers and networks . . . . .	3
3.2	Specific layers . . . . .	5
3.2.1	Dense layer . . . . .	5
3.2.2	Convolution layer . . . . .	5
3.2.3	Inception layer . . . . .	5
3.2.4	Batch Normalization layer . . . . .	5
3.2.5	Dropout layer . . . . .	5
3.2.6	Max Pooling layer . . . . .	5
<b>4</b>	<b>Our network</b>	<b>4</b>
4.1	Training data . . . . .	4
4.2	Training and final performance . . . . .	4
4.3	Comparison to matched filtering . . . . .	4
<b>5</b>	<b>Conclusion</b>	<b>5</b>
<b>6</b>	<b>Acknowledgments</b>	<b>6</b>
<b>A</b>	<b>Full adder as network</b>	<b>7</b>
	<b>Glossary</b>	<b>10</b>

This page is intentionally left blank. LÖSCHEN!!! Damit erste Seite nicht auf Rückseite gedruckt ist.



### 3 Neural networks

Explain the use for this section.

Neural networks are machine learning algorithms inspired by research on the structure and inner workings of brains. [Insert quote (Rosenblatt?)] Though in the beginning NN were not used in computer sciences due to computational limitations [Citation] they are now a major source of innovation across multiple disciplines. Their capability of pattern recognition and classification has already been successfully applied to a wide range of problems not only in commercial applications but also many scientific fields. [Quote a few scientific usecases here. Of course using the one for gw but also other disciplines.] Major use cases in the realm of gravitational wave analysis have been classification of glitches in the strain data of GW-detectors [Citation] and classification of strain data containing a GW versus pure noise [Citation]. A few more notable examples include [list of citations].

In this section the basic principles of NN will be introduced and notation will be set. The concept of backpropagation will be introduced and extended to a special and for this work important kind of NN. (maybe use the term "convolution" here already?) It will be shown that learning in NN is simply a mathematical minimization of errors that can largely be understood analytically.

#### 3.1 Neurons, layers and networks

What is the general concept of a neural network? How does it work? How does back-propagation work? How can one replicate logic gates? (cite online book)

The basic building block of a NN is - as the name suggests - a *neuron*. This neuron is a function mapping inputs to a single output. In the early days this output was always either 1 or 0 [Citation], whereas nowadays it is usually some real number.

In general there are two different kinds of inputs. Those that are specific to the neuron itself and those that the neuron receives as an outside stimulus. We write the neuron as

$$f : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}; \quad (\vec{x}, \vec{w}, b) \mapsto f(\vec{x}, \vec{w}, b) := a(\vec{w} \cdot \vec{x} + b), \quad (3.1)$$

where  $\vec{x}$  is the outside stimulus,  $\vec{w}$  are called weights,  $b$  is a bias value and  $a$  is a function known as the *activation function* (change this to not be emphasized if it is not used for the first time here). A usual depiction of a neuron and its structure is shown in Figure 3.1. The activation function is a scalar function

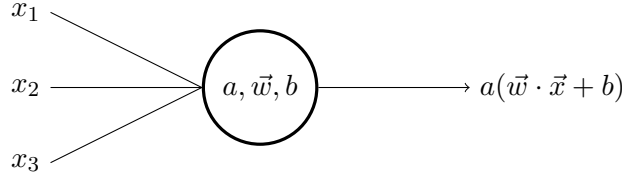
$$a : \mathbb{R} \rightarrow \mathbb{R} \quad (3.2)$$

determining the scale of the output of the neuron. To understand the role of each part of the neuron, consider the following activation function:

$$a(y) = \begin{cases} 1, & y > 0 \\ 0, & y \leq 0 \end{cases}. \quad (3.3)$$

$x_1$	$x_2$	$a(\vec{w} \cdot \vec{x} + b)$
0	0	0
0	1	0
1	0	0
1	1	1

**Table 3.1:** Neuron activation with activation function (3.3), weights  $\vec{w} = (w_1, w_2)^T = (1, 1)$ , bias  $b = -1.5$  and inputs  $(x_1, x_2) \in \{0, 1\}^2$ . Choosing the weights and biases in this way replicates an "and"-gate.



**Figure 3.1:** Depiction of a neuron with inputs  $\vec{x} = (x_1, x_2, x_3)^T$ , weights  $\vec{w}$ , bias  $b$  and activation function  $a$ .

With this activation function, the neuron will only send out a signal (or "fire") if the input  $y$  is greater than 0. Therefore, in order for the neuron to fire, the weighted sum of the inputs  $\vec{w} \cdot \vec{x}$  has to be larger than the negative bias  $b$ . This means, that the weights and biases control the behavior of the neuron and can be optimized to get a specific output.

The effects of changing the weights makes individual inputs more or less important. The closer a weight  $w_i$  is to zero, the less impact the corresponding input value  $x_i$  will have. Choosing a negative weight  $w_i$  results in the corresponding input  $x_i$  being inverted, i.e. the smaller the value of  $x_i$  the more likely the neuron is to activate and vice versa.

Changing the bias to a more negative value will result in the neuron having fewer inputs it will fire upon, i.e. the neuron is more difficult to activate. The opposite is true for larger bias values. So increasing it will result in the neuron firing for a larger set of inputs.

As an example consider a neuron with activation function (3.3), weights  $\vec{w} = (w_1, w_2)^T = (1, 1)$ , bias  $b = -1.5$  and inputs  $(x_1, x_2) \in \{0, 1\}^2$ . Choosing the weights and biases in this way results in the outputs shown in Table 3.1. This goes to show, that neurons can replicate the behavior of an "and"-gate. Other logical gates can be replicated by choosing the weights and biases in a similar fashion (See first section of Appendix A).

Use the introduction of the and-neuron from above to introduce the concept of networks in a familiar way. Having logic gates enables us to build more complex structures, such as full adders and hence we can, in principle, calculate any function a computer can calculate. Only afterwards introduce layers as a way of structuring and formalizing networks.

Since all basic logic gates can be replicated by a neuron, it is a straight forward idea

to connect them into more complicated structures, like a full-adder (see Appendix A). When doing this we connect multiple neurons, which therefore form a network - a neural network. The example of the full-adder demonstrates the principle of a NN perfectly. It's premise is to connect multiple simple functions to form a network, that can solve difficult tasks.

The example of a full-adder-network however shows another thing: Sufficiently complicated networks can - in principle - calculate any function an ordinary computer can calculate, as a computer is mostly a machine consisting of multiple adders and logic gates.

## **3.2 Specific layers**

Explain that there are not just dense layers.

### **3.2.1 Dense layer**

What is a dense layer, how does it work (can maybe be omitted)

### **3.2.2 Convolution layer**

What are the advantages of convolution layers and why do we use them? Disadvantages?

### **3.2.3 Inception layer**

Explain what it is, how it works. (cite google paper) ONLY IF IT IS REALLY USED IN THE FINAL ARCHITECTURE!

### **3.2.4 Batch Normalization layer**

Explain how batch normalization works and why it is useful. (cite according paper)

### **3.2.5 Dropout layer**

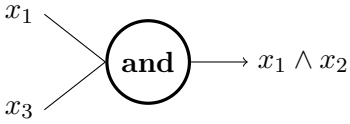
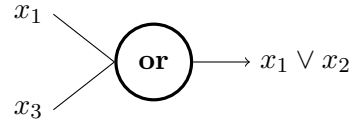
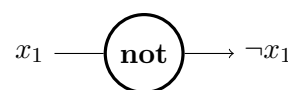
Explain what a dropout layer is, what it does, why it is useful.

### **3.2.6 Max Pooling layer**

Explain what max pooling does and why it is useful, even when it is counter intuitive.

## A Full adder as network

To create a full adder from basic neurons, the corresponding logic gates need to be defined. The equivalent neuron for an "and"-gate was defined in subsection 3.1. There are two more basic neurons which need to be defined. The neuron corresponding to the "or"-gate, which is given by the same activation function (3.3), weights  $\vec{w} = (w_1, w_2)^T = (1, 1)$  and bias  $b = -0.5$ , and the equivalent neuron for the "not"-gate, which is given by the activation function (3.3), weight  $w = -1$  and bias  $b = 0.5$ . These definitions are summarized in Table A.1.

"and"-neuron	"or"-neuron	"not"-neuron																																				
																																						
$\vec{w} = (1, 1) \quad b = -1.5$	$\vec{w} = (1, 1) \quad b = -0.5$	$w = -1 \quad b = 0.5$																																				
<table> <tr> <th><math>x_1</math></th> <th><math>x_2</math></th> <th><math>a(x_1 + x_2 - 1.5)</math></th> </tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	$x_1$	$x_2$	$a(x_1 + x_2 - 1.5)$	0	0	0	0	1	0	1	0	0	1	1	1	<table> <tr> <th><math>x_1</math></th> <th><math>x_2</math></th> <th><math>a(x_1 + x_2 - 0.5)</math></th> </tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	$x_1$	$x_2$	$a(x_1 + x_2 - 0.5)$	0	0	0	0	1	1	1	0	1	1	1	1	<table> <tr> <th><math>x_1</math></th> <th><math>a(-x_1 + 0.5)</math></th> </tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </table>	$x_1$	$a(-x_1 + 0.5)$	0	1	1	0
$x_1$	$x_2$	$a(x_1 + x_2 - 1.5)$																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				
$x_1$	$x_2$	$a(x_1 + x_2 - 0.5)$																																				
0	0	0																																				
0	1	1																																				
1	0	1																																				
1	1	1																																				
$x_1$	$a(-x_1 + 0.5)$																																					
0	1																																					
1	0																																					

**Table A.1:** A summary and depiction of the main logic gates written as neurons. All of them share the same activation function (3.3).

Using the basic logic gates a more complex structure - the "XOR"-gate - can be built. A "XOR"-gate is defined by its truth table (see Table A.2).

$x_1$	$x_2$	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	0

**Table A.2:** Truth table for the "XOR"-gate.

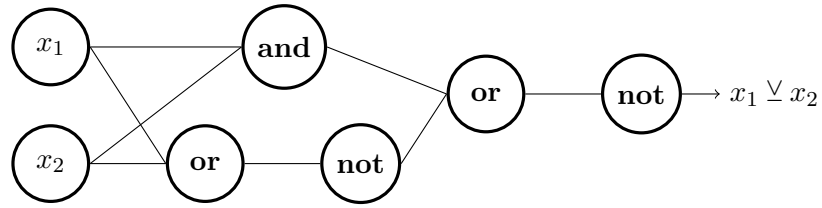
It can be constructed from the three basic logic operations "and", "or" and "not"

$$x_1 \vee x_2 = \neg((x_1 \wedge x_2) \vee \neg(x_1 \vee x_2)). \quad (\text{A.1})$$

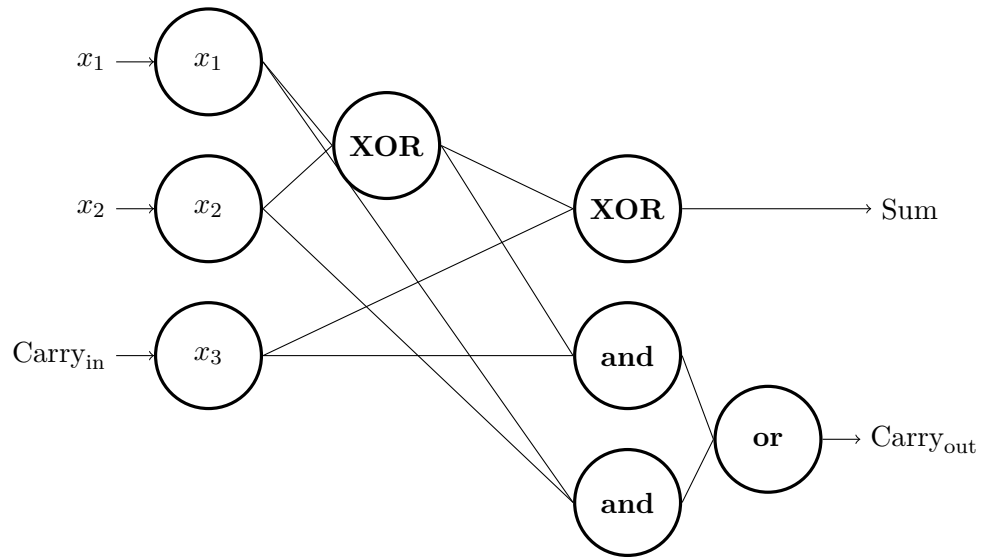
Therefore the basic neurons from Table A.1 can be combined to create a "XOR"-network (see Figure A.1).

To simplify readability from here on out a neuron called "XOR" will be used. It is defined by the network of Figure A.1 and has to be replaced by it, whenever it is used.

With this "XOR"-neuron a network, that behaves like a full-adder, can be defined. A full-adder is a binary adder with carry in and carry out, as seen in Figure A.2.



**Figure A.1:** The definition of a network that is equivalent to an "XOR"-gate.



**Figure A.2:** A network replicating the behavior of a binary full adder.



## Glossary

**GW** gravitational wave.

**NN** Neural network.

