FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Master's Degree Thesis in
Master of Science in Physics

Machine Learning for Gravitational Waveform Modelling

Name:
Prasoon Pandey

Marticulation Number:
205171

Supervisor:
Prof. Dr. Sebastiano Bernuzzi

Co-Supervisor:
Dr. Luís Felipe Longo Micchi

Academic Year:
2024-25

Dedication...

Abstract

Analyzing gravitational wave (GW) data from compact binary mergers through Bayesian methods requires a deep dive into the posterior probability distribution of detected signals. Central to this process are accurate waveform models, which simulate the GW signals generated by systems defined by a specific set of parameters. Typically, sampling the posterior distribution for a single GW event demands the creation of around ten million waveforms, making the speed of waveform generation critically important. This urgency is amplified with the advent of next-generation GW detectors, which are poised to capture a far greater number of events than current instruments can. Optimizing the computational efficiency for these signals is vital for maximizing the scientific returns from large-scale experiments in the future.

One of the most promising strategies to achieve the required efficiency gains is template acceleration using machine learning. While there is an extensive body of literature on this approach for binary black holes, studies focusing on binary neutron stars remain comparatively scarce.

In my thesis research, I propose `mlgw_bns_HOM`, a neural network-based surrogate model designed to accelerate the generation of Effective-One-Body (EOB) waveforms while fully incorporating all higher-order modes (HOM) and precession effects for enhanced generalization and accuracy. This model achieves substantial computational gains through an innovative training algorithm that combines data compression techniques with analytical insights.

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Contents

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

1 | Introduction

Primo capitolo [1]

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

2 | Gravitational Waves Theory

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

3 | Machine Learning Techniques

In this chapter, we will discuss the machine learning techniques employed in this work. It does not aim to be a comprehensive guide to the subject, but rather a summary of the most important concepts and techniques so that the reader can understand the main ideas and the choices made in this work. One can refer to [2] for a more detailed and comprehensive guide to the subject.

3.1. Introduction

In the recent years, machine learning, and particularly deep learning, has become a powerful tool in every scientific field. Some common examples include Large Language Models (LLMs) [3, 4], synthetic image generation [5], stellar astronomy [6], geoinformatics [7].

Let's begin by defining machine learning in a general way:

> An algorithm which is used to extract patterns and relationships from data, to make predictions, and to solve problems that are too complex to be solved by traditional methods.

Since the development of advanced engineering tools and numerical programming, we have access to huge amount of data, and machine learning helps us to "learn" an intrinsic structure of it.

This "learning" can be done in different ways, and we broadly classify them into three categories:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Since, our work focuses on the supervised learning, we will only discuss it in this chapter. However, later in this chapter, we will provide a brief overview of the other two types of learning.

3.1.1. Supervised Learning

The most common type of machine learning algorithm is the supervised learning. It is used when we have a dataset with labeled examples, and we want to learn a function f that maps inputs $\mathbf{x}$ to outputs $\mathbf{y}$. Some examples of supervised learning problems include spam email detection, image classification, and speech recognition. The figure 3.1 summarizes a typical supervised learning workflow, where the labeled training data is passed to a machine learning algorithm for fitting a predictive model that can make predictions on new, unlabeled data inputs.

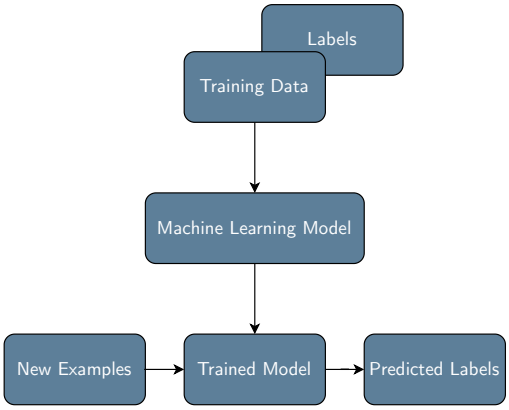

Figure 3.1: Supervised learning pipeline.

Formally, the supervised learning problem can be written as:

$$\hat{\mathbf{y}} = f(\mathbf{x}; \theta). \tag{3.1}$$

Here, both the input $\mathbf{x}$ and output $\mathbf{y}$ are vectors of a predetermined and fixed size. The model is just a mathematical equation with a fixed form. It represents a family of different relations between the input and the output. The model also contains *parameters* θ. The choice of parameters determines the particular relation between input and output.

When we talk about *learning*, we mean that the model is able to find the **best parameters** θ that gives meaningful output for a given input. When we say meaningful, we mean that the model is able to generalize to new examples that it has not seen before.

During the learning/training process, we learn the parameters using a set of training data of N pairs of input and output $\{\mathbf{x}_i, \mathbf{y}_i\}$. We aim to modify these parameters such that map of each trainining input to its associated output as closely as possible. Therefore, in order to quantify the deviation between the predicted output and the target output, so that we can guide the training process, we define a *loss function* $\mathcal{L}$. When we train the model, we aim to minimize this loss function:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\mathcal{D}, \theta). \tag{3.2}$$

There is a systematical way called *optimization algorithm* to find the best parameters θ^* that help the model reach global or local minimum of the loss function. This will be discussed later in this chapter.

Let's move on to discuss about the term dataset which comes up often in the context of machine learning.

3.2. Dataset: Training, Validation, and Test Sets

The quality of the data and the amount of useful information that it contains are key factors that determine how well a machine learning algorithm can learn. Similarly, it is extremely necessary to split

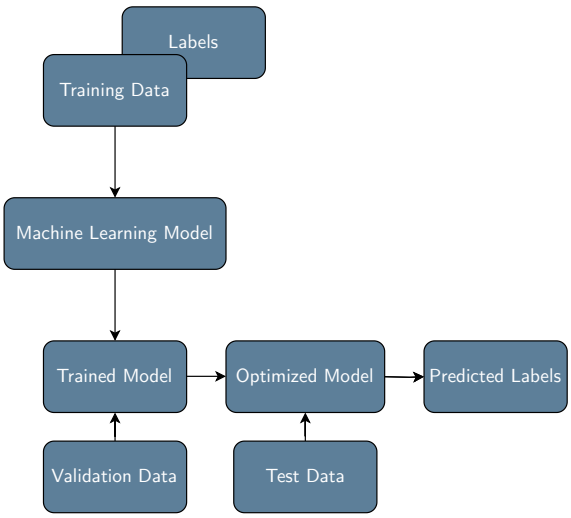

Figure 3.2: Use of training, validation, and test sets.

the collected data into different sets so that we can evaluate the performance of the model on unseen data. Based on the purpose, most of the machine learning pipelines are designed to split the data into three sets: training set, validation set, and test set.

- Training set: This is the subset of data used to fit the model. Generally speaking, we randomly pick 70% or 80% of the data for training.
- Validation set: One of the most important sets which is often overlooked. Before the training process, we fix some of the parameters of the model called *hyperparameters*. The validation set is used to tune these hyperparameters and plays a key role in diagnosing issues such as *overfitting* and *underfitting*. We split 15% or 10% of the data for validation.
- Test set: Finally, this is set of data which my model has not seen before. It serves as the final benchmark to assess the model's performance, robustness, and generalization capability. Like the validation set, it typically comprises 10% to 15% of the total data.

The role of each set can be clearly seen in the figure 3.2.

In the next section, we will discuss about a point mentioned in the paragraph above: *underfitting* and *overfitting*.

3.3. Headache: Overfitting and Underfitting

In the world of machine learning, we rely on building a network of some fundamental building blocks (usually called *neurons*) and achieving the right model complexity is an ongoing struggle. To give a brief idea about these concepts, we can say that *overfitting* is when the model is "too complex" and it fits the training data too well, while *underfitting* is when the model is "too simple" and it does not fit the training data well enough.

Let's take an example were we sample some data points from a 4th order polynomial function with some noise. Our goal is to develop a model that learns the underlying function (not exactly, but close

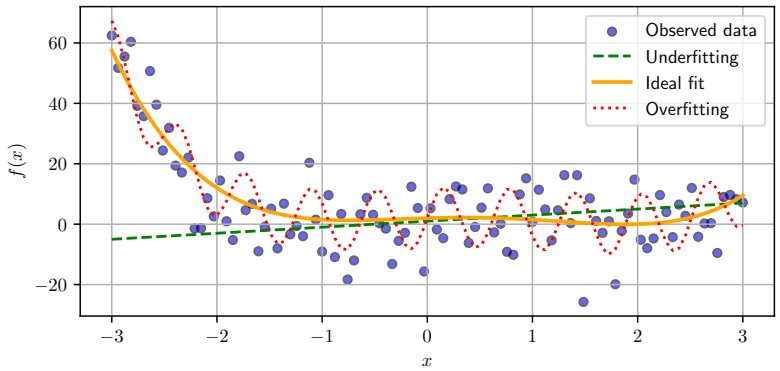

Figure 3.3: Illustration of model underfitting, ideal fitting, and overfitting on noisy data generated from a 4th-order polynomial function. The data points (blue scatter) are sampled from the function $0.5x^4 - x^3 - x^2 + x + 2$ with some noise $\mathcal{N}(0, 10)$. The orange curve represents an underfit model (linear regression), the green curve shows the ideal 4th-order polynomial fit, and the red curve illustrates an overfit model, where a higher-degree polynomial captures the noise.

enough). Now, while performing the training, we can have two different scenarios, one where model complexity is too low and it returns a quadratic function, and the other where model complexity is too high and it returns the 10th order polynomial function.

Overfitting corresponds to the case where the model output 10th order polynomial function, meaning it performs well on training data but does not generalize well to unseen data (test data). If a model suffers from overfitting, we also say that the model has a high variance, which can be caused by having too many parameters, leading to a model that is too complex given the underlying data.

Underfitting, on the other hand, corresponds to the case where the model output a linear function, meaning it performs poorly on both training and test data. If a model suffers from underfitting, we also say that the model has a high bias, which can be caused by having too few parameters, leading to a model that is too simple given the underlying data.

We can clearly see in the figure 3.3 the trade-off between model complexity and generalization, with overfitting resulting in excessive sensitivity to noise, and underfitting failing to capture the true underlying trend.

I want to make one thing clear at this point. There are certain keywords which have not been clearly defined in our discussion above. However, we will follow up on them in the upcoming sections. For now, let's move on to the next section where we will discuss about the heart of the machine learning model: *neural networks*.

3.4. Neural Networks

The core idea of modern machine learning or deep learning lies in the idea of building effective and problem-specific neural networks. Let's take a look at what we mean by it and what is it's origin.

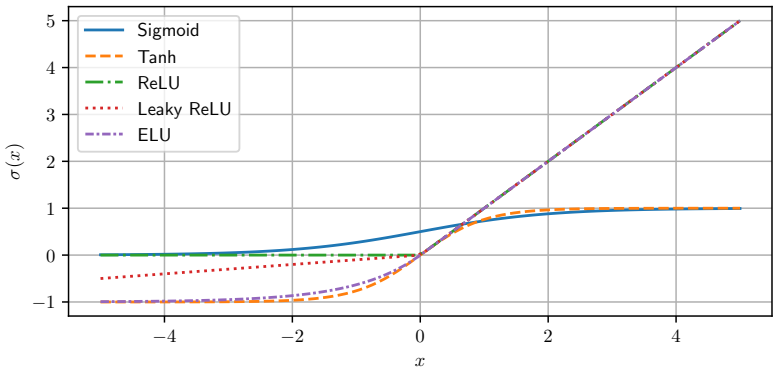

Figure 3.4: Activation functions.

3.4.1. Building Blocks: Perceptrons

The origins of neural networks (or artificial neural networks) can be traced to the early 20th century, inspired by the biological neural networks that underpin cognitive functions in living organisms. This early work lead to the development of the perceptron, which is a simple model of a neuron (artificial neuron). This forms the basis of the modern neural networks where we have multiple layers of perceptrons.

A single perceptron is a simple model that takes multiple inputs, $\mathbf{x} \in \mathbb{R}^n$, and produces a single scalar output, y. The output is a weighted sum of the inputs, plus a bias term is passed through an *activation function*, $\sigma : \mathbb{R} \to \mathbb{R}$.

$$y = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{3.3}$$

where, $\mathbf{w} \in \mathbb{R}^n$ is the weight vector and $b \in \mathbb{R}$ is the bias term.

3.4.2. Multi-layer Perceptrons

When we have multiple perceptrons, we can stack them up to form what we call a *neural network*. The output of the first layer is the input of the second layer, and so on. This way, we can build a deep neural network. The layers between the input and output layers are called *hidden layers*. We start with inputs $\mathbf{x} \in \mathbb{R}^n$ and we have L hidden layers. The output of the l-th hidden layer is given by:

$$\mathbf{h}_l = \sigma\left(\mathbf{W}_l \mathbf{h}_{l-1} + \mathbf{b}_l\right) \tag{3.4}$$

where, $\mathbf{W}_l \in \mathbb{R}^{n \times n_{l-1}}$ is the weight matrix and $\mathbf{b}_l \in \mathbb{R}^{n_l}$ is the bias vector. It is these *weights* and *biases* which are the *parameters* tuned during the training process.

As we see in the equation 3.3 and 3.4, we have an entity called the activation function σ. It is this function which introduces non-linearity to the model. One can make multiple choices for the activation functions which mainly depends on the problem at hand. We will highlight some of them in the next section.

Table 3.1: Common Activation Functions in Neural Networks

Activation Function	Expression	Key Properties
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	Bounded, smooth, vanishing gradients
Tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	Zero-centered, smooth, vanishing gradients
ReLU	$\text{ReLU}(x) = \max(0, x)$	Sparse activations, fast computation, non-saturating
Leaky ReLU	$\text{LeakyReLU}(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$	Avoids dead neurons, small negative slope
ELU	$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha(e^{x}-1), & x \leq 0 \end{cases}$	Smooth, avoids vanishing gradient for $x < 0$

3.4.3. Activation Functions

A stack of linear layers is equivalent to a single linear layer where we multliply together all the weight matrices. To get more expressive power we can transform each layer by passing it elementwise (pointwise) through a nonlinear function called an activation function. In figure 3.4 and table 3.1, we have highlighted some of the most common activation functions.

Mathematically, a suitable activation function is typically required to be non-linear and differentiable to allow effective learning through gradient-based optimization methods like *backpropagation*. Additional desirable properties include boundedness (to prevent uncontrolled output growth), monotonicity (to aid convergence), and computational efficiency (to ensure scalability).

Just like previous sections, there are a few keywords which have not been defined in proper manner. I just want to make sure that reader is aware that they will be discussed in the upcoming sections.

3.4.4. Loss Functions

So far its quite clear, in supervised learning, a model is trained to approximate a mapping from input data to target outputs. The loss function, also referred to as the **objective function,** plays a critical role in this process by quantifying the discrepancy between the model's predictions and the actual labels. Minimizing this loss function is central to the optimization procedure that drives the learning process.

Mathematically, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ represents the input features and $y_i \in \mathbb{R}$ (or $\mathbb{R}^k$ in the case of multi-class problems) denotes the corresponding label, the model output is denoted as $\hat{y}_i = f_\theta(x_i)$, with θ being the parameters of the model. The loss function $\mathcal{L}(\hat{y}_i, y_i)$ is defined to measure the performance for each example. The total loss over the dataset is:

$$\mathcal{L}_{\text{total}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\hat{y}_i, y_i)$$

The choice of loss function depends on the nature of the problem:

- **Mean Squared Error (MSE):** Widely used in regression problems and defined as

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3.5}$$

  This loss penalizes large deviations more heavily due to the square term, making it sensitive to outliers.

- **Binary Cross-Entropy (BCE):** Commonly used for binary classification tasks:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3.6}$$

  This loss function assumes $\hat{y}_i \in (0, 1)$, making it suitable for models with a sigmoid output layer.

- **Categorical Cross-Entropy:** An extension of BCE used in multi-class classification problems with softmax output layers.

Loss functions not only guide the training process but also influence convergence properties and robustness. Therefore, in many cases, researchers build a custom loss function to handle the specific problem at hand which has even led to a separate field called the *physics-informed machine learning*.

3.4.5. Backpropagation

Upto this point, we have discussed many theoretical ideas involved in machine learning. However, one might question how does training process work, and how do we find the best parameters for the model. This is where the backpropagation and optimization algorithms come into play.

Let's first dig into the core idea of the backpropagation algorithm.

> Backpropagation is a method for computing the gradient of the loss function with respect to the model parameters.

As highlighted in equation 3.2, we want to obtain a set of parameters θ^* that minimizes the loss function $\mathcal{L}$ by updating the parameters like weights matrix $\mathbf{W}$ and biases $\mathbf{b}$. During this process, we iteratively update the parameters by computing the gradient of the loss function with respect to the parameters. The backpropagation algorithm offers a very computationally efficient approach to compute the partial derivatives of a complex, non-convex loss function in multilayer neural networks.

This idea can be better understood when combined with optimization algorithms.

3.4.6. Optimization Algorithms

The whole basis of machine learning in build on the idea of paramter estimation for my neural network so that it can mimic the underlying function. This requires solving an optimization problem, where we try to find the values for a set of variables $\theta \in \Theta$, that minimize a scalar-valued loss function or cost function $\mathcal{L} : \Theta \to \mathbb{R}$:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \mathcal{L}(\theta) \tag{3.7}$$

The algoithms involved in this optimization problem are called **solver**. There are multiple solvers available but the most common ones are:

- **Gradient Descent:** A first-order optimization algorithm that iteratively updates the parameters in the direction of the steepest descent of the loss function. Upon initialization, the parameters are set to some random values θ_0. Then at each iteration t, they perform an update of the following form:

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t) \tag{3.8}$$

  where, η is the **learning rate**. The learning rate is another hyperparameter that controls the step size of the gradient descent. It is a crucial parameter that determines the speed and accuracy of the optimization process.

  Formally, we require that there exists an $\eta_{\max} > 0$ such that:

$$\mathcal{L}(\theta_{t+1}) \leq \mathcal{L}(\theta_t) \tag{3.9}$$

  for all $\eta \in [0, \eta_{\max}]$.

- **Stochastic Gradient Descent (SGD):** A variant of gradient descent that uses a small subset of the training data (mini-batch) to compute the gradient. We can write the update rule for SGD as:

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}_i(\theta_t) \tag{3.10}$$

  where, $\mathcal{L}_i(\theta_t)$ is the loss function evaluated on the i-th training example.

  One should also note that stochastic gradient is an unbiased estimator of the true gradient:

$$\mathbb{E}[\nabla_\theta \mathcal{L}_i(\theta_t)] = \nabla_\theta \mathcal{L}(\theta_t) \tag{3.11}$$

  Finally, we can use it on a mini-batch of size $\mathcal{B}$ to compute the gradient:

$$\mathbf{g}_t \approx \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \nabla_\theta \mathcal{L}_i(\theta_t) \tag{3.12}$$

  and the update rule becomes:

$$\theta_{t+1} = \theta_t - \eta \mathbf{g}_t \tag{3.13}$$

  where, $\mathcal{B}_t$ is the mini-batch of size $\mathcal{B}$ at iteration t.

- **ADAM:** A second-order optimization algorithm that uses the gradient and the Hessian matrix to update the parameters.

  ADAM is a combination of lot of ideas from the field of optimization. To trace it back, we move:

  $$\textbf{AdaGrad} \rightarrow \textbf{RMSProp} \rightarrow \textbf{ADAM}$$

  I would recommend reading [2] for more details on this approach. However, to break it down formally we introduce two new terms namely momentum and RMSProp:

  $$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t \tag{3.14}$$
  $$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2. \tag{3.15}$$

  The update rule translates to:

  $$\theta_{t+1,d} = \theta_{t,d} - \eta_t \frac{1}{\sqrt{s_{t,d}} + \epsilon} \mathbf{m}_{t,d} \tag{3.16}$$

  where, β_1 and β_2 are the momentum and RMSProp parameters respectively.

Now, we can clearly find the application of the backpropagation algorithm in the context of neural networks. For dense neural networks, the gradient calculation could have been a nightmare. However, with the help of the backpropagation algorithm, we can compute the gradient of the loss function with respect to the parameters of the any neuron in the network.

3.5. Gaussian Process Regression

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

4 | Conclusion

Conclusioni

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Bibliography

[1] D. Bossini, M. Pancaldi, L. Soumah, M. Basini, F. Mertens, M. Cinchetti, T. Satoh, O. Gomonay, and S. Bonetti. Ultrafast Amplification and Nonlinear Magnetoelastic Coupling of Coherent Magnon Modes in an Antiferromagnet. *Physical Review Letters*, 127(7):077202, August 2021. doi: 10.1103/PhysRevLett.127.077202.

[2] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL http://probml.github.io/book1.

[3] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[4] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. URL https://arxiv.org/abs/2006.11239.

[6] Ostdiek, B., Necib, L., Cohen, T., Freytsis, M., Lisanti, M., Garrison-Kimmmel, S., Wetzel, A., Sanderson, R. E., and Hopkins, P. F. Cataloging accreted stars within Gaia DR2 using deep learning. *A&A*, 636:A75, 2020. doi: 10.1051/0004-6361/201936866. URL https://doi.org/10.1051/0004-6361/201936866.

[7] Claire Robin, Christian Requena-Mesa, Vitus Benson, Lazaro Alonso, Jeran Poehls, Nuno Carvalhais, and Markus Reichstein. Learning to forecast vegetation greenness at fine resolution over africa with convlstms, 2022. URL https://arxiv.org/abs/2210.13648.

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

A | Appendice A

Appendice A

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

List of Figures

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

List of Tables

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Ringraziamenti

Ringraziamenti

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA