Architectural optimization and feature learning for high-dimensional time series datasets

Robert E. Colgan,[1,2] Jingkai Yan,[2,3] Zsuzsa Márka,[4] Imre Bartos,[5] Szabolcs Márka,[6] and John N. Wright[2,3]

[1]*Department of Computer Science, Columbia University in the City of New York, 500 West 120th Street, New York, New York 10027, USA*
[2]*Data Science Institute, Columbia University in the City of New York, 550 West 120th Street, New York, New York 10027, USA*
[3]*Department of Electrical Engineering, Columbia University in the City of New York, 500 West 120th Street, New York, New York 10027, USA*
[4]*Columbia Astrophysics Laboratory, Columbia University in the City of New York, 538 West 120th Street, New York, New York 10027, USA*
[5]*Department of Physics, University of Florida, P.O. Box 118440, Gainesville, Florida 32611-8440, USA*
[6]*Department of Physics, Columbia University in the City of New York, 538 West 120th Street, New York, New York 10027, USA*

(Received 5 July 2022; accepted 3 January 2023; published 25 January 2023)

As our ability to sense increases, we are experiencing a transition from data-poor problems, in which the central issue is a lack of relevant data, to data-rich problems, in which the central issue is to identify a few relevant features in a sea of observations. Motivated by applications in gravitational-wave astrophysics, we study a problem in which the goal is to predict the presence of transient noise artifacts in a gravitational-wave detector from a rich collection of measurements from the detector and its environment. We argue that feature learning—in which relevant features are optimized from data—is critical to achieving high accuracy. We introduce models that reduce the error rate by over 60% compared to the previous state of the art, which used fixed, hand-crafted features. Feature learning is useful not only because it can improve performance on prediction tasks; the results provide valuable information about patterns associated with phenomena of interest that would otherwise be impossible to discover. In our motivating application, features found to be associated with transient noise provide diagnostic information about its origin and suggest mitigation strategies. Learning in such a high-dimensional setting is challenging. Through experiments with a variety of architectures, we identify two key factors in high-performing models: sparsity, for selecting relevant variables within the high-dimensional observations, and depth, which confers flexibility for handling complex interactions and robustness with respect to temporal variations. We illustrate their significance through a systematic series of experiments on real gravitational-wave detector data. Our results provide experimental corroboration of common assumptions in the machine-learning community and have direct applicability to improving our ability to sense gravitational waves, as well as to a wide variety of problem settings with similarly high-dimensional, noisy, or partly irrelevant data.

DOI: 10.1103/PhysRevD.107.022009

I. INTRODUCTION

We consider the problem of detecting the presence or absence of some phenomenon of interest from a large collection of time series, a subset of which are predictive but whose precise mathematical relationship to the phenomenon of interest is *a priori* unknown. Variants of this fundamental problem arise in areas such as finance, neuroscience and brain computer interfaces, structural health monitoring, machine diagnostics, and anomaly detection, just to name a few. All of these areas present the analyst with time series, which may be noisy and only a few of which may be relevant to the prediction task at hand.

Our applied motivation comes from gravitational-wave astrophysics, which uses gravitational phenomena to study the properties of the Universe and its occupants. This scientific quest is driven by extraordinarily sensitive detectors, such as KAGRA, Virgo, and Laser Interferometer Gravitational-wave Observatory (LIGO) [1–12], which can detect spatial effects as small as (10^{-19} m/$\sqrt{\text{Hz}}$). A major confounding factor is the frequent presence in the main gravitational-wave data stream of brief, loud noise artifacts commonly known within LIGO as "glitches"—nonastrophysical nuisances caused by factors as varied as seismic and ionospheric activity, nearby road and train traffic,

optical effects within a detector, etc. These glitches, which can appear as frequently as every few seconds, significantly hinder the detectors' sensitivity to the astrophysical phenomena they are intended to measure, because they can drown out real astrophysical events or even mimic them, potentially causing false-positive detections. It is therefore important to distinguish them from true astrophysical signals and ensure data contaminated by glitches are not used for gravitational-wave detection.

Glitches are thought to be attributable to a wide variety of internal and terrestrial sources; identifying, investigating, and mitigating the various types of glitches that appear and their causes has been a major focus of LIGO's engineering efforts for decades [13–39]. The rich body of work that has resulted encompasses a wide variety of distinct but related objectives, problem formulations, and methodologies. One approach is to develop methods that seek to automatically identify glitches based on features of the gravitational-wave data stream so that the surrounding time period can be flagged for possible rejection or further analysis (e.g., Omicron [40]); some of these methods also incorporate other data sources and attempt to identify components or subsystems of the detector that may be related to the physical origin of glitches (e.g., iDQ [22], hVeto [33], and UPV [35]). Other works have focused on characterizing and classifying glitches by common morphology to better understand types of similar, recurring glitches (e.g., GravitySpy [26], PCAT [41,42], WDF-ML [41,42], and LCSS [36]). Still other approaches model the noise (quasistationary or transients) through various means (e.g., using Wiener filters [43] or other approaches such as Nonsens [44], DeepClean [45], Bayeswave [46], and gwsubtract [47]) so that it can be subtracted to recover the underlying signal, which is not possible in all situations but carries the obvious benefit that, if successful, the data can be retained and possibly still used to support a detection. These methods represent only a handful of the multitude of creative approaches developed in this subfield in recent years, an increasing number of which have relied on machine-learning techniques [23].

We focus in this work on the problem of detecting glitches based on nonastrophysical information. Currently, many glitch types are identified via methods that directly analyze the gravitational-wave data stream along with one or more of the hundreds of thousands of auxiliary data channels LIGO's detectors record for purposes such as seismic motion monitoring. Still, many glitches and glitch types are of unknown origin, and, if history is a predictor, many new glitch types will emerge in the future. One avenue to ensure we do not misidentify a true gravitational event as a glitch is to make use of solely terrestrial information about the detector: Glitches which can be predicted based on the nonastrophysically sensitive auxiliary data of the detector alone (without using actual detector output) are unlikely to be astrophysical in origin. Reference [48] introduced a first demonstration of this concept, leveraging the more than 200 000 auxiliary data channels recorded by a LIGO detector to predict glitches with high accuracy. This method is based on classical machine-learning tools: One first extracts hand-crafted features from each of the potentially informative subset (about 40 000) of auxiliary channels around the time in question and then applies sparse logistic regression to perform prediction based on a small subset of these features.

Can we learn better features for glitch detection in gravitational-wave astrophysics? More generally, under what circumstances is it possible to reliably learn from an overwhelmingly large number of noisy and mostly irrelevant data streams? Modern machine-learning architectures, such as deep neural networks, learn adaptive features from raw data as well as how to combine those features hierarchically. Compared to classical, manually defined features, such learned features are better able to capture richer properties of raw data relevant to the task, especially in scenarios with complex and/or high-dimensional data such as natural images and audio. Moreover, the hierarchical, nonlinear nature of deep neural networks makes them far more powerful than classical linear models, enabling them to learn complex ways of aggregating and synthesizing information from the output of their learned feature detectors [49–52].

In this work, we systematically explore the properties of several machine-learning architectures and evaluate the extent to which they are beneficial for learning a successful classifier in the problem setting described above. In Sec. III, we compare the fixed-feature, logistic regression–based model of Ref. [48] (which we refer to as FF) to an equivalent linear model with features learned from raw data (which we refer to as LF) and find significant improvement. In Sec. IV, we confirm previous findings that regularization-induced sparsity is essential to learning effective classifiers in this setting and extend it to other more complex models. In Sec. V, we move from flat, linear classifiers to deeper, nonlinear ones and evaluate the effect of depth on performance. In Sec. VI, we discuss our results and synthesize the observations gleaned from this exploration. We believe the insights gained will be valuable beyond the problem setting of gravitational-wave astrophysics.

II. PROBLEM FORMULATION AND PRIOR WORK

We consider the problem of binary classification of a time of interest t from multiple time series. Given P time series $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P$, where each $\boldsymbol{x}_p$ is a sequence of time-ordered, real-valued scalar samples

$$\boldsymbol{x}_p = (x_{p,1}, \ldots, x_{p,T}) \in \mathbb{R}^T$$

sampled at some frequency f_p, we would like to make a prediction $y_t \in \{-1, 1\}$, where the labels -1 and 1 indicate

the presence or absence, respectively, of some phenomenon of interest at time t. We assume that the P time series (or a subset of them) encode enough information near time t to make such a prediction but make no further assumptions about the structure or content of the time series.

A. Motivating application: Glitch prediction in gravitational-wave astrophysics

The above problem formulation is motivated by concerns in gravitational-wave astrophysics, which uses incredibly sensitive interferometric detectors to measure small distortions in spacetime created by astrophysical events such as merging black holes. In addition to the main gravitational-wave measurement data, the two detectors of the LIGO project continuously record hundreds of thousands of time series describing a wide array of aspects of the detector's internal and external state and environment. These are used for monitoring its many components and subsystems to diagnose errors, identify sources of noise, and so on. In our problem formulation, the time series $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P$ represent these auxiliary measurements.

One particularly troublesome ongoing phenomenon in this type of data, as discussed in Sec. I, is the presence of noise transients, also known as *glitches*, in the detectors' output. They can obscure or even mimic gravitational waves, so it is important to be able to identify them and distinguish them from true astrophysical signals. Previous work [48] has demonstrated the potential of doing so using only information contained in the hundreds of thousands of nonastrophysically sensitive auxiliary data channels. That work achieved a useful level of accuracy in reproducing the output of a commonly used method for glitch detection without access to the actual gravitational-wave data that method analyzes to perform its detections. In this work, we adopt a similar problem formulation and improve on the results outlined in Ref. [48], which were obtained by employing a variant of a classical and well-used machine-learning algorithm known as logistic regression, by testing a series of increasingly flexible and powerful machine-learning architectures based on convolutional neural networks.

1. Ground-truth labels

In our general problem formulation, the target labels $y_t \in \{1, -1\}$ represent the presence and absence of glitches in the gravitational-wave data stream, and the goal is to accurately predict these labels. Following Ref. [48], the labels we use for training and evaluation are computed by Omicron [40], an existing excess power–based transient search that directly analyzes the main gravitational-wave data stream to identify glitches. Also following Ref. [48], we choose negative examples ("glitch-free points") by randomly sampling points in time that are sufficiently distant from any time identified by Omicron as containing a glitch.

2. Input data

Following Ref. [48], in this paper, we consider data from LIGO's auxiliary channels during LIGO's engineering run 14 (ER14) in March 2019. We follow the same procedure to reduce the approximately 250 000 auxiliary channels in a detector to approximately 40 000 by excluding channels that are constant or vary only in a predictable fashion (e.g., counting time cycles). Of these, approximately 35 000 have a sample rate of 16 Hz, with the rest having various higher sample rates up to 65 536 Hz; for efficiency, we restrict our analyses in this work to channels with a sample rate of 16 Hz and leave higher-frequency channels to future work. (We note that higher-frequency channels generally have downsampled equivalents already present; also, excluding high-frequency channels does not represent an inherent limitation of any of the models presented. Including the additional information that could be present in higher-frequency channels would likely only improve a given model's performance, so it is notable that the results achieved here without using those channels represent an improvement over previous work that did use those channels.) We further exclude any channels known or suspected to be coupled to the gravitational-wave data stream following the same procedure as Ref. [48].

For efficiency, we draw training data from a shorter subset of the ER14 training period used in Ref. [48] (GPS time 1 235 890 000 to 1 235 900 000, i.e., the final 10 000 s of the 30 000-s period of Ref. [48]), because Ref. [48] demonstrated that 10 000 s is a sufficient amount of time from which to draw training data. As in Ref. [48], we draw validation data from the following 10 000 s (GPS time 1 235 900 000 to 1 235 910 000) and test data from the following 10 000 s (GPS time 1 235 910 000 to 1 235 920 000).

We normalize each channel by computing the mean and standard deviation of the raw channel data over the entire training data period; then we subtract the training mean and divide by the standard deviation for all data in the training, validation, and test periods.

Following Ref. [48], our positive samples are drawn from points in time identified by Omicron [40] as a glitch peak; our negative samples are drawn randomly from periods where no glitch was identified by Omicron within 2 s. We select the same number of negative samples as there are positive samples in each dataset.

B. Prior work: Glitch prediction with fixed features and shallow models

The possibility of making such predictions was recently demonstrated in the initial work of Ref. [48]. The FF method of Ref. [48] is based on classical statistical tools: It extracts certain *hand-crafted* features from the auxiliary channels $\boldsymbol{x}_p$ and then predicts the label $\hat{y}_t$ by linearly combining these features and passing them through a sigmoid function to return a probability estimate:

$$\hat{y}_t = \sigma\left(\sum_{kp} \omega_{kp} [\boldsymbol{f}_k \star \boldsymbol{x}_p]_t + b\right). \tag{1}$$

Here, $\star$ denotes discrete correlation, and $\sigma(\cdot)$ denotes the logistic function: $\sigma(x) = (1 + \exp(-x))^{-1}$. The filters $\boldsymbol{f}_k$ are fixed; in Ref. [48], these correspond to certain intuitively chosen patterns of behavior that might be predictive, such as spikes and level changes. The weights ω_{kp} of this linear combination are learned from training data via gradient descent on an objective function that measures the error between the known ground-truth labels y_t and the current model's predictions $\hat{y}_t$.

This method achieves 80–85% accuracy in glitch detection on unseen validation and test data. These results demonstrated that glitches can indeed be predicted with moderate accuracy using only auxiliary data and, hence, that many glitches can be identified as terrestrial in origin and safely discarded, increasing confidence in remaining detection candidates. While these results were inspiring, the FF method is arguably very far from leveraging all of the structure in these complex datasets and, hence, very far from optimal in its ability to predict glitches based on auxiliary channels. Limitations of this approach include the following.

(i) *Hand-crafted vs learned features.*—The FF method is based on hand-crafted features designed from intuition-based predictions of a few patterns of behavior that might be predictive; it cannot leverage the ability of modern machine-learning techniques to learn more expressive, highly tuned features from raw data [49,52] (Sec. III).

(ii) *Depth.*—Increased depth has consistently been found to improve performance and trainability, even over shallower models with equivalent statistical capacity [51,52] (Sec. V).

(iii) *Linear vs nonlinear models.*—The ability of modern models to deal with nonlinear structure in data is crucial; deeper hierarchical models without nonlinear activation functions can be reduced to an equivalent flat model [52] (Sec. V).

In this paper, we systematically investigate these issues, developing a sequence of models which fundamentally improve over the flat, fixed-feature model discussed above. We also adopt aspects of that method that prove to be essential to both approaches—most notably regularization-induced sparsity (Sec. IV)—and describe how we adapt them to our proposed methods.

III. FROM FIXED TO LEARNED FEATURES

The principal weakness of the FF method [Eq. (1)], as discussed in Ref. [48] and Sec. II B, is that the feature extraction procedure must be defined manually, and optimizing it individually for tens of thousands of time series is not practical. A major factor in the explosive success of modern machine-learning methods in the past decade has been their ability to flexibly learn features from raw data rather than rely on inflexible hand-designed features [49,50,52]. It is natural, then, to consider whether replacing the fixed features of the above model with learned features would improve its performance.

A. Flat model with learned features

To that end, we first consider a nearly equivalent model which differs from FF only in the computation of features from the raw data $\boldsymbol{x}_p$. We replace the fixed, manually defined feature extraction procedure with a convolutional model (defined explicitly in Appendix A 1) that *learns* a filter $\boldsymbol{w}^0_{1p}$ $[p \in (1 \ldots P)]$ for each of the P time series. For comparison, we preserve for now all other aspects of the FF model, including its linearity and (lack of) depth.

In the general notation of Appendix A 1 [Eq. (A1)], our learned feature model takes the form

$$\boldsymbol{\alpha}^1 = \sigma\left(\sum_{p=1}^{P} \boldsymbol{w}^0_{1p} \star \boldsymbol{x}_p + b\right), \tag{2}$$

where, as in Eq. (1), σ is a logistic function. The estimated probability that $\boldsymbol{x}_t$ belongs to the class "glitch" is $\hat{y}_t = \boldsymbol{\alpha}^1_t$. The filters $\boldsymbol{w}^0_{1p}$ are jointly optimized during the training process. Below, we refer to this model as LF. As in the FF model, we also apply a sparsifying regularization term to the filters to encourage $\|\boldsymbol{w}^0_{1p}\|_2 = 0$ for most p (see Sec. IV). The major increase in generality in moving from FF to LF comes from the fact that the $\boldsymbol{w}^0_{1p}$ can be arbitrary vectors—in contrast, FF restricts these filters to be linear combinations $\sum_k \omega_{kp} \boldsymbol{f}_k$ of the fixed filters $\boldsymbol{f}_k$.[1]

B. Performance comparison

We now compare the performance of the FF model with an LF model as described above, setting the input data length to 2.5 s to match the amount of time considered by the FF model for each sample. (In Sec. III C, we show that longer input lengths enable significantly better performance, further underscoring the advantages and flexibility of learned features.) For efficiency, we consider only

[1]The features described by Ref. [48] include several based on standard deviation, a nonlinear function that cannot be implemented as linear convolution. Strictly speaking, therefore, it is not correct to say that the LF model is an exact generalization of an FF model that employs standard deviation or other nonlinear functions. However, the flexibility afforded by learning the feature extractors—even only linear ones—from raw data would almost certainly outweigh any loss of flexibility from restricting ourselves to linear features. This assumption is consistent with our results with the single-layer LF model. Such nonlinear functions could be learned by the deeper models discussed in Sec. V.

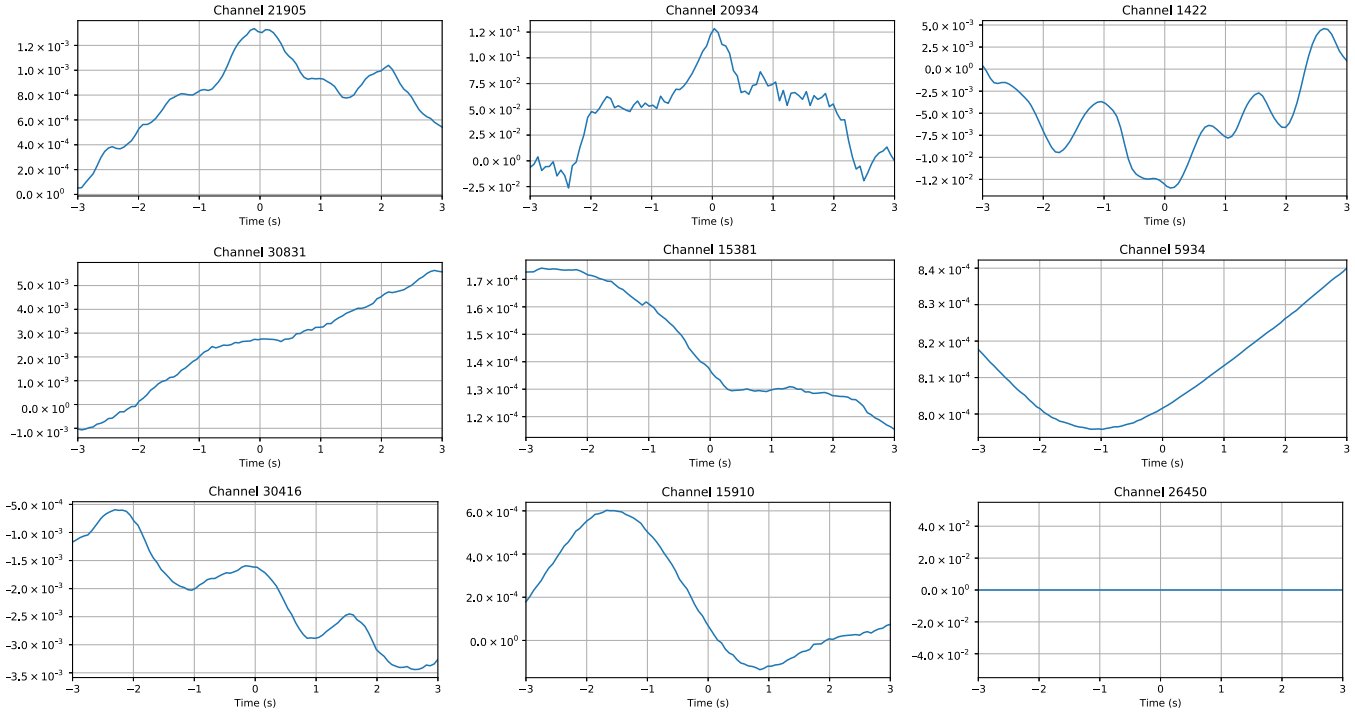

FIG. 1. A few of the features learned by an LF model with 6-s filters. The x axes correspond to filter length in seconds; the y axes show the unitless magnitude of the filter over time, which is relevant for comparison across filters. Intuitively, a higher overall magnitude indicates the associated channel is more important to the model's decisions—when correlated with a higher-magnitude filter, an input data segment will contribute more heavily to the sum and resulting probability estimate than the same segment correlated with a lower-magnitude filter. The lower-right panel shows a filter with magnitude 0, like the vast majority of the learned filters in the model (see Sec. IV).

auxiliary channels that are sampled at 16 Hz for both models.

To train the LF and FF models, we follow the procedures described in Appendix B and in Ref. [48], respectively, with a few minor modifications to facilitate as direct a comparison as reasonably possible (see Appendix B for details). Following Ref. [48], we train and evaluate both models on data from ER14. We draw training data from only the final 10 000 s of the 30 000 s training data period, because that work found that 10 000 s was a sufficient amount of data for good performance.[2] We do not subsample glitches during this period, using instead all 8596 glitches and an equal number of glitch-free points (chosen using the procedure described in Ref. [48]) as training data. As discussed in Appendix B, for the validation results reported, we sample a subset of 500 glitches and an equal number of glitch-free points from the validation period; for the test results, we use all glitches present in the test period and an equal number of glitch-free points.

As in Ref. [48], for both types of model we perform a grid search over the regularization hyperparameters (using the same grid for both), training models with many parameter settings and evaluating their performance on the validation dataset; we choose the setting that gives the best performance on the validation dataset.

We find that the FF model of Ref. [48] achieves an accuracy of 85.9% [with a true-positive rate (TPR), true-negative rate (TNR), and loss of 87.6%, 84.1%, and 0.3392, respectively][3] on the validation dataset, compared to an accuracy of 87.3% (TPR 86.1%, TNR 88.6%, and loss

[2]It is possible that compared to the FF model the models presented here would see greater benefit from a longer training data period because of increased flexibility provided by learned features and other aspects; on the other hand, since the learned features are more closely tuned to the training data, they might be less robust to longer-term changes in the state of the detector, e.g., changes in the shape of glitch-predictive features over time. We leave investigation of the optimal length of time from which to draw training data to future work.

[3]The slightly improved performance of the FF model compared to the same model in Ref. [48] is most likely due to the combination of modifications to the training procedure described above and the shorter training data period, as Ref. [48] reported an overall slight decrease in performance as the length of the training period increased beyond 10 000 s. Also, although we would expect that at least some of the higher-frequency channels contain useful information for the classifier—perhaps even more so, proportionally, than the 16 Hz channels—it is possible that decreasing the data dimensionality improved the model's ability to identify relevant data by enough to outweigh the benefit of the higher-frequency channels.

0.3004) for the LF model, an overall 9.9% reduction in relative error rate. On the test dataset, it achieves an accuracy of 85.8% (TPR 91.2% and TNR 80.5%), compared to an accuracy of 88.6% (TPR 86.7% and TNR 90.4%) for the LF model, an overall 19.7% reduction in relative error rate.

C. Input segment length

So far, for the sake of comparison with the FF model, we have limited the input data segment length for the LF model to the 2.5 s surrounding each sample time, matching the amount of data used to compute the hand-defined features of the FF model. As noted in Ref. [48], those features—including the length of input data they consider—were chosen largely arbitrarily, and we would like to see whether longer (or shorter) input segments might further improve the performance of learned features.

To that end, we consider input segment length as an additional hyperparameter over which to search while maintaining the same (flat) model structure. Although it would be possible to implement FF models that accept other segment lengths—and the FF model may well also have benefited from considering longer data segments—it is more straightforward to do so with the LF model: We simply adjust a single hyperparameter and let the model decide how best to make use of the additional data.

Our results indicate that a segment length of 4–6 s is ideal for this model and data (see Fig. 2), providing significant improvement over the shorter segments considered previously. Too short, and the model may miss relevant behavior that does not coincide precisely with the appearance of the glitch; too long, and the model may become too difficult to optimize because of the presence of too much extraneous data. We emphasize that the input length can be determined through optimization via training on a dataset, which is a useful feature of the method. It is conceivable that for different time frames and datasets this timescale would vary, indicating the data could be dominated by glitch types of much shorter or longer timescales.

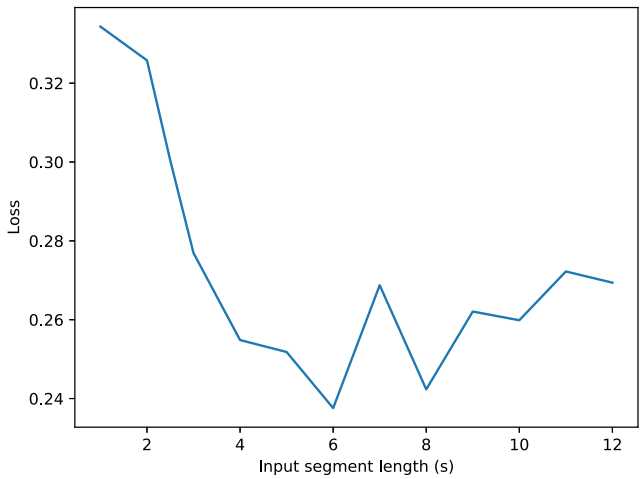

FIG. 2. Loss on validation dataset (lower is better) vs feature or input segment length, showing for a given length the best-performing model over all hyperparameters tested [i.e., initial learning rate as well as elastic net α and λ—see Eq. (3) and Fig. 3].

Figure 1 illustrates some of the features learned in this model. They can reflect behavior such as local maxima (top left and top center); level changes (center left); oscillatory behavior (bottom center); and more complicated effects specific to each channel that are useful for distinguishing between glitchy and glitch-free times. Their shapes represent clues to physical or environmental effects that result in glitches and could help diagnose their origins, highlighting an important added benefit of the LF approach.

By accuracy on our validation dataset, the best-performing LF model over every hyperparameter setting tested achieves an accuracy of 90.9% (TPR 85.8%, TNR 95.6%, and loss 0.2423). This represents a 35.5% reduction in error rate over the FF model discussed above (and a 25.5% reduction over the LF model with input length limited to 2.5 s). The lowest validation loss achieved was 0.2376, but this model had slightly worse accuracy at 90.4% (TPR 91.3% and TNR 89.5%). In Sec. V, we present experimental results with deeper models that further improve performance.

IV. THE ROLE OF SPARSITY

A crucial factor in the success of the FF model was the incorporation of a sparsifying regularization term in the optimization objective function—i.e., a term that encourages many of the weights ω_{kp} to be set to 0 during training, leaving only the most relevant features to be considered. Not only did this improve the model's efficiency and interpretability, it also significantly improved its performance compared to a standard, nonsparsifying regularizer on the overall L2 norm of $\boldsymbol{\omega}$. The effectiveness of sparse regularization has been observed in a variety of problem settings, leading to widespread adoption of regularizers such as the L1 norm or LASSO [53] and the elastic net [54]. It is particularly relevant in this problem, because the vast majority of the P time series contain no useful information for the task.

In developing the LF model, we similarly found sparsity to be an essential property. Following Ref. [48], we employ the elastic net as a regularizer on the magnitudes of the learned filters in the LF model to encourage $\|\boldsymbol{w}_{1p}^0\| = 0$ for most p. The elastic net is a linear combination of L2 and L1 regularization, with tunable weight on each component:

$$R(\boldsymbol{\eta}) = \frac{\lambda}{2}\sum_p \eta_p^2 + \alpha\sum_p |\eta_p|, \tag{3}$$

where the hyperparameters λ and α control the strengths of the L2 and L1 components, respectively. In our case, we want the regularization to apply to each channel's learned

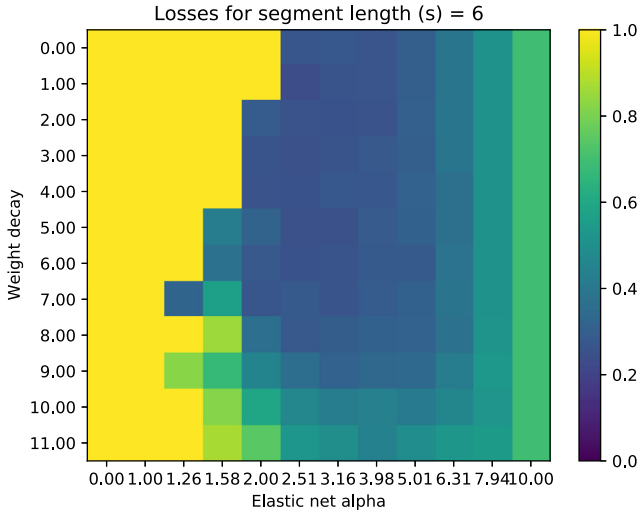

FIG. 3. Loss on validation dataset (lower is better; values greater than 1 are clipped) as a function of α (x axis) and λ (y axis) of Eq. (3) for the best-performing initial learning rate.

filter $\boldsymbol{w}_{1p}^0$ and act on the filter as a whole, rather than on every sample of all filters independently, so we take $\eta_p = \|\boldsymbol{w}_{1p}^0\|_2$ in the above equation. We implement the L1 regularization update following the technique of Ref. [55].

Following the training procedure described in Appendix B with an LF model with no sparsifying regularization and despite trying a much larger grid of hyperparameter settings, no model at any setting tested was able to achieve more than 64.4% validation accuracy. In contrast, as discussed in Sec. III, sparse models were able to achieve an accuracy of more than 90% while learning nonzero features for only a small fraction of the 33 939 channels considered (i.e., all but a few channels are ignored by the model when making predictions). See Fig. 3 for an illustration of validation accuracy as a function of the sparsity hyperparameters α and λ of Eq. (3) and Fig. 4 for an illustration of how validation accuracy correlates with the resulting sparsity of the model.

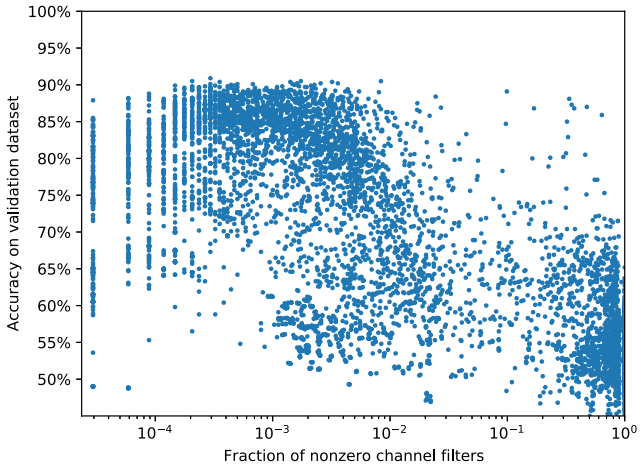

FIG. 4. Accuracy on validation dataset as a function of the fraction of nonzero channels in the model, for all hyperparameters tested (including filter length). The best-performing models achieved an accuracy of around 90% with as few as 0.02–1% of their approximately 35 000 filters nonzero, corresponding to all but around ten to a few hundred channels considered irrelevant.

We also observed that, even when we do not explicitly induce sparsity, under certain circumstances training will spontaneously converge to a model that is sparse in one or more respects. We describe these findings in Appendix C.

V. DEEPER MODELS

In the previous sections, we argued that (i) feature learning and (ii) sparse channel selection are essential ingredients in the design of high-performing glitch predictors. We illustrated these ingredients in the simplest possible setting of shallow (single-layer) architectures. However, experience in application areas such as vision, audio, and natural language processing suggests that feature learning becomes even more powerful in *deeper* architectures, which learn hierarchical features. Deeper models have the following potential advantages in glitch prediction.

(i) *Higher order interactions* between channels are better captured by deep models. A canonical example is the exclusive or relationship, which cannot be represented by a single-layer model. In our setting, this would correspond to the situation in which there are two auxiliary channels which are jointly predictive of a certain type of glitch in the sense that exactly one channel is active (but not both). Deep models are capable of capturing this and other higher-order interactions across channels.[4]

(ii) *Robust feature extraction* from individual channels is facilitated by deeper models, in which low-level features are repeatedly combined to produce a hierarchy of increasingly abstract, higher-level features. This robustness is amplified by including *pooling* operations at various levels, which increases robustness to temporal shifts, variations in signal shape, etc.[5]

[4]Determining precisely *what*, if any, higher-order interactions are present across the LIGO auxiliary channels demands a combination of device modeling and exploratory data analysis. The deeper models proposed in this paper provide one tool for empirically probing the relationship between auxiliary channels and their utility in classifying various glitch types and diagnosing their origins. We will report on this direction in future work.

[5]Of course, one can also perform pooling in shallow models. Later in this section, we introduce deep models with pooling (VGG6, VGG13, and VGG13-BN), which achieve state-of-the-art performance on our datasets of interest. The excellent performance of these models should arguably be attributed not just to depth but to the combination of depth, nonlinearity, and pooling.

TABLE I. Architectural features and performance over the models discussed. Column six shows the best (lowest) loss on the validation dataset over every hyperparameter setting tested for a given model, along with that model's accuracy on the validation dataset. Column seven shows the best accuracy on the validation dataset, along with the loss. To compute accuracy on the test dataset in column eight, we used the model that achieved the best validation loss.

Model	Feature learning?	Depth	Nonlinear?	Pooling?	Best val loss (acc)	Best val acc (loss)	Test acc
FF	×	1	×	×	0.3392 (85.9%)	86.0% (0.3567)	85.8%
LF	✓	1	×	×	0.2376 (90.4%)	90.9% (0.2423)	89.6%
1Hid	✓	2	×	×	0.2385 (90.5%)	91.2% (0.2486)	89.3%
1HidReLU	✓	2	✓	×	0.2330 (91.0%)	91.0% (0.2330)	91.0%
VGG6	✓	6	✓	✓	0.2010 (91.9%)	93.0% (0.2050)	94.0%
VGG13	✓	13	✓	✓	0.1956 (93.4%)	93.4% (0.1956)	93.6%
VGG13-BN	✓	13	✓	✓	0.1732 (93.1%)	93.6% (0.1822)	94.7%

(iii) *Increased statistical capacity.*—Deeper models can accommodate more complicated statistical relationships between the auxiliary channels and the gravitational-wave strain.

In the remainder of this section, we illustrate the power of depth by introducing a sequence of increasingly deeper models, culminating in nonlinear deep models that significantly outperform the previous state of the art for glitch prediction on the datasets considered here. These results corroborate the utility of depth in feature learning, with the caveat that many of the specific architectures considered here vary in other ways (e.g., presence vs absence of nonlinearities and temporal pooling).

A. Models with one hidden layer

1Hid and 1HidReLU both contain a convolutional layer mapping P one-dimensional time-series inputs of length T to a single hidden layer with P^1 scalar-valued feature maps. This is followed by a single fully connected layer that linearly combines the hidden-layer outputs into a single scalar output and adds a scalar bias; the result is then passed through a sigmoid nonlinearity. 1HidReLU contains a rectifying nonlinearity before the fully connected layer, whereas 1Hid does not. We set P^1 to 100 and did not comprehensively study or optimize it, but in limited preliminary experiments we observed little impact from halving or doubling it. (In fact, as discussed in Appendix C 2, in many cases the training spontaneously converges to a model with only one or a few active feature maps, but these models can perform as well as or better than models with more active feature maps.)

B. Models with many hidden layers

We also experimented with three deeper models inspired by the VGG16 network [51], with reduced depth for computational efficiency. In both models, all convolutional kernels have length three and all max-pooling layers have a kernel size and stride of two. We fix the length of the input segments to 80 samples (5 s) for VGG6 and 200 samples (12.5 s) for VGG13. VGG6 consists of a total of five convolutional layers followed by one fully connected layer, with max pooling after the second and fifth convolutional layers. VGG13 consists of a total of 11 convolutional layers, four max-pooling layers, and two fully connected layers. It is identical in structure to VGG6 through the second max-pooling layer, which is followed by two groups of three convolutional layers and a max-pooling layer and then by two fully connected layers. VGG13-BN is identical to VGG13 except for the insertion of a batch normalization layer [56] before every nonlinearity.

C. Experimental results with deeper models

We test these deeper models on the same ER14 dataset used in the previous sections. Table I and Fig. 5 report the validation accuracy and loss achieved by each model and compares these to the shallow models (FF and LF) introduced in previous sections. Validation performance

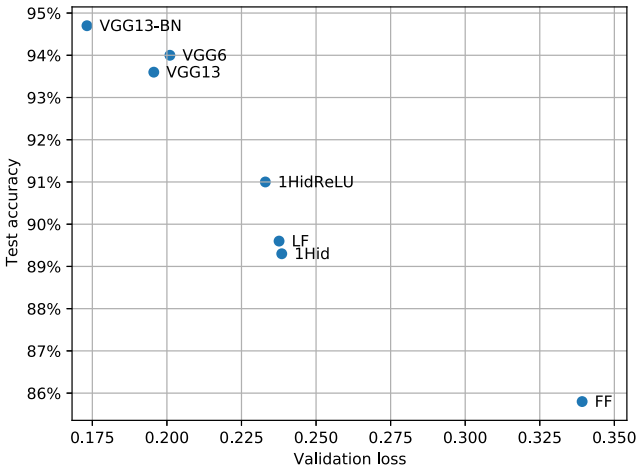

FIG. 5. Scatter plot of Table I showing best validation loss vs test accuracy for each model. For validation loss on the x axis, lower is better, so the models in the upper left have the best overall performance. Model complexity generally increases from the lower right to the upper left.

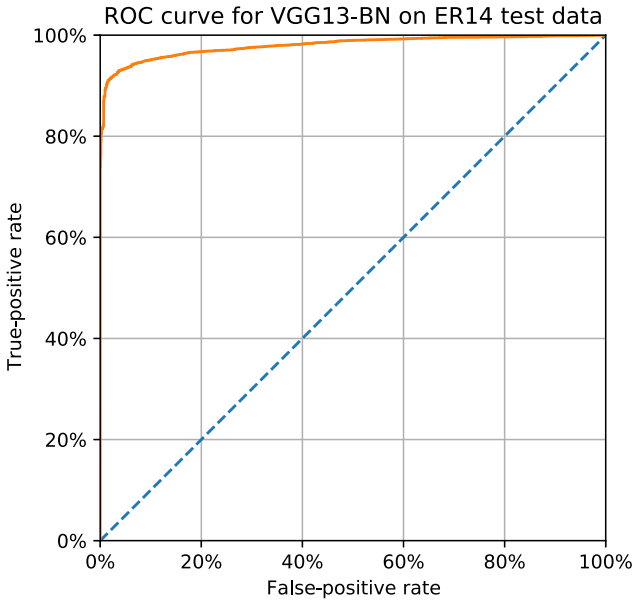

FIG. 6. ROC curve of VGG13-BN on the ER14 test dataset.

increases nearly monotonically with depth; the best-performing model is VGG13-BN, which achieves a validation accuracy of 93.1% and loss of 0.1732 and a test accuracy of 94.7% and loss of 0.1578. Figure 6 shows a receiver operating characteristic (ROC) curve of VGG13-BN's performance on the test dataset. It is worth noting that the improved performance in VGG6 and VGG13 may be attributable not only to their depth but to architectural details such as the use of short convolution filters and pooling. Nevertheless, Table I is consistent with the widely reported finding that deeper networks produce better statistical performance in signal classification tasks.

VI. CONCLUSION

In this paper, we have demonstrated the potential of feature learning for glitch prediction in gravitational-wave astrophysics and, more generally, for learning from high-dimensional time series. We have argued that feature learning and architectural choices including sparsity, depth, and nonlinearity are essential to achieving the best possible performance in this setting. Our architectural explorations culminate in state-of-the-art performance on the ER14 dataset, with a best validation accuracy of 93.6% and test accuracy of 94.7%—an overall approximately 63% reduction in the test error rate compared to the shallow, fixed-feature model.

In general, deeper models require more resources: more training data and more computational resources at both training and test time. Obtaining these best possible resource-performance trade-offs is an important direction for future work; in Ref. [57], we study complexity-performance trade-offs in information extraction from a single time series. One especially important trade-off in learning from high-dimensional time series is the trade-off between sample complexity (how much training data) and test-time performance. In practice, system characteristics can change over time, and it is important to be able to rapidly adapt to these changes, using limited training data. Online learning of deep models, using a combination of large offline datasets and limited streaming data, is an important direction for future work.

Feature learning and deep models introduce new opportunities for using machine learning not just as a tool for prediction but as a tool for generating insights into the data-generating process. The models we have described all employ automatic feature learning, which not only improves performance on the classification task compared to fixed features, but also can provide valuable diagnostic information—for example, by identifying environmental factors or specific subsystems of a gravitational-wave detector associated with transient noise glitches. Insight gained during these investigations enables automated adaptability to slowly changing, time-dependent data as emerging features can be discovered and learned.

Sparse channel selection, as discussed in Sec. IV, leads to models that identify a few especially relevant channels for prediction, which—like feature learning—is also beneficial to both performance and interpretability. Compared to flat, linear, sparse models such as FF, the deep, nonlinear models proposed here squeeze more relevant information out of the small number of selected channels, as witnessed by their substantially improved prediction performance. Mining these more accurate models for insights into the data-generation process is another important direction for future work.

ACKNOWLEDGMENTS

We acknowledge computing resources from Columbia University's Shared Research Computing Facility project, which is supported by NIH Research Facility Improvement Grant No. 1G20RR030893-01, and associated funds from the New York State Empire State Development, Division of Science Technology and Innovation (NYSTAR) Contract No. C090171. This material is based upon work and data supported by NSF's LIGO Laboratory which is a major facility fully funded by the National Science Foundation. This research has also made use of data obtained from the Gravitational Wave Open Science Center, a service of LIGO Laboratory, the LIGO Scientific Collaboration, and the Virgo Collaboration. LIGO is funded by the U.S. National Science Foundation (NSF). Virgo is funded by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale della Fisica Nucleare (INFN), and the Dutch Nikhef, with contributions by Polish and Hungarian institutes. The authors are grateful for the LIGO Scientific Collaboration review of the paper, and this paper is assigned a LIGO DCC number (LIGO-P2200008), with special thanks to Gayathri V. The authors

acknowledge the LIGO Collaboration for the production of data used in this study and the LIGO Laboratory for enabling Omicron trigger generation on its computing resources (National Science Foundation Grants No. PHY-0757058 and No. PHY-0823459). The authors are grateful to the authors and maintainers of the Omicron and Omega pipelines, the LIGO Commissioning and Detector Characterization Teams, and LSC domain expert Colleagues whose fundamental work on the LIGO detectors enabled the data used in this paper. The authors thank colleagues of the LIGO Scientific Collaboration and the Virgo Collaboration for their help and useful comments. The authors thank the University of Florida and Columbia University in the City of New York for their generous support. The authors are grateful for the generous support of the National Science Foundation under Grant No. CCF-1740391. I. B. acknowledges the support of the Alfred P. Sloan Foundation and NSF Grants No. PHY-1911796 and No. PHY-2110060.

APPENDIX A: MODELS

1. Convolutional model and notation

Our models are structured as convolutional neural networks. A convolutional neural network is comprised of a sequence of L layers, which generate features $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^L$. Each $\boldsymbol{\alpha}^\ell$ consists of P^ℓ feature maps (time series) $\boldsymbol{\alpha}^\ell_1, \ldots, \boldsymbol{\alpha}^\ell_{P^\ell} \in \mathbb{R}^{T^\ell}$. For notational consistency, we let $\boldsymbol{\alpha}^0$ denote the input features $\boldsymbol{\alpha}^0_p = \boldsymbol{x}_p$, with $P^0 = P$ and $T^0 = T$.

For a given input $\boldsymbol{x}$, features are generated sequentially by applying an affine map followed by (possible) nonlinearity and pooling operations:

$$\boldsymbol{\alpha}^{\ell+1}_i = \mathcal{P}^\ell \sigma^\ell \left(\sum_{j=1}^{P^\ell} \boldsymbol{w}^\ell_{ij} \star \boldsymbol{\alpha}^\ell_j + b^\ell_i \right). \tag{A1}$$

Here, the $\star$ operation denotes discrete correlation[6]; the $\boldsymbol{w}^\ell_{ij} \in \mathbb{R}^{d^\ell}$ $(i = 1 \ldots P^{\ell+1},\ j = 1 \ldots P^\ell)$ are a collection of filters; and the $b^\ell_i \in \mathbb{R}$ are scalar biases.

Because the affine map in Eq. (A1) is built out of correlation operations, it is *shift equivariant*: If $\mathcal{S}_k$ denotes a temporal shift by k samples,

$$\left(\sum_{j=1}^{P^\ell} \boldsymbol{w}^\ell_{ij} \star \mathcal{S}_k[\boldsymbol{\alpha}^\ell_j] + b^\ell_i \right) = \mathcal{S}_k \left(\sum_{j=1}^{P^\ell} \boldsymbol{w}^\ell_{ij} \star \boldsymbol{\alpha}^\ell_j + b^\ell_i \right).$$

This is a highly desirable property for analyzing time series. It ensures that our feature extraction respects the temporal structure of the $\boldsymbol{\alpha}^\ell_j$, and the resulting mapping requires far fewer parameters and far less computation compared to a generic affine map of the same dimension.

In Eq. (A1), σ^ℓ denotes a scalar activation function, which is extended to vector inputs by applying it elementwise. In our nonlinear models, we often use the ReLU activation $\sigma^\ell(u) = \mathrm{ReLU}(u) = \max\{u, 0\}$. The notation of Eq. (A1) also accommodates "linear layers" with no nonlinear activation, simply by setting $\sigma^\ell(u) = u$. If $\sigma^\ell(u) = u$ for all ℓ, the network output is a linear function of the input $\boldsymbol{x}$. However, nonlinear models are often preferable due to their greater expressive power: In our experiments, nonlinear models typically outperform their linear counterparts, especially when the number of layers L is large.

Finally, the operation $\mathcal{P}^\ell$ performs temporal pooling by taking maxima over contiguous subsets of entries. This operation is observed to improve robustness to temporal shifts and distortions by aggregating feature responses over time, and it is included in several of the deeper networks that we introduce in Sec. V. The general notation of Eq. (A1) is flexible enough to accommodate architectures that do not pool simply by taking maxima over subsets consisting of single indices.

The fixed-feature model FF [Eq. (1)] can be seen as an instance of the general model [Eq. (A1)], with $L = 1$ layers, with inputs $\boldsymbol{\alpha}^0_p = \boldsymbol{x}_p$, a single output $\hat{\boldsymbol{y}} = \boldsymbol{\alpha}^L$, and filters $\boldsymbol{w}^0_{1p} = \sum_k \omega_{kp} \boldsymbol{f}_k$. That is to say, the logistic predictor is a one-layer neural network, which uses a linear combination of the *fixed*, hand-designed features $\boldsymbol{f}_k$.

The general model of Eq. (A1) allows for significantly more flexible architectures, in which (i) features can be combined hierarchically and (ii) features can be learned from data. In these more flexible architectures, learning is performed in a similar manner to as described above for the FF model—i.e., gradient descent on a measure of the error between the ground-truth labels and the predictions made by the current state of the model predictions; the only major difference is in the greater number of parameters (see Appendix B for a detailed description of our training procedures).

2. Model details

In all of the following models, we impose sparsity in the form of elastic net regularization (as discussed in Sec. IV) only at the lowest layer, on the connections between the P input channels and the P^1 first-layer feature maps. That is, for a given channel–feature map pair (p, i), the norm of the corresponding learned filter $\|\boldsymbol{w}^0_{ip}\|_2$ corresponds to one element of the vector $\boldsymbol{\eta}$ on which the elastic net regularization $R(\boldsymbol{\eta})$ is computed [Eq. (3)]. We implement the L1 component of the elastic net by explicitly computing $\alpha\|\boldsymbol{\eta}\|_1$ and adding to the loss and the L2 component via standard

[6]Correlation is equivalent to a true convolution up to a flipping of the filters $\boldsymbol{w}^\ell_{ij}$. In implementation, the correlation operation may be subsampled (strided) for efficient computation and storage.

weight decay, which is applied to every parameter of the model at the same magnitude.

a. Models with one hidden layer

As described in Sec. V, 1Hid and 1HidReLU both contain a convolutional layer mapping P one-dimensional time-series inputs of length T to a single hidden layer with P^1 scalar-valued feature maps, followed by a single fully connected layer that linearly combines the hidden layer outputs into a single scalar-valued output, which is passed through a sigmoid nonlinearity to produce a probability estimate. The convolutional layer's filters are each the same length T as the input and, therefore, produce a single scalar value for each pair of an input and a feature map. As is standard in one-dimensional convolutional neural network architectures, for a given feature map $i \in (1 \ldots P^1)$, the values obtained from convolving each input $p \in (1 \ldots P)$ with the learned filter $\boldsymbol{w}_{ip}^0 \in \mathbb{R}^T$ corresponding to that input–feature map pair are summed and a bias term b_i corresponding to that feature map is added. The learned weights for the single convolutional layer, therefore, consist of a three-way tensor $\boldsymbol{W} \in \mathbb{R}^{P^1 \times P \times T}$, and the biases consist of a vector $\boldsymbol{b} \in \mathbb{R}^{P^1}$.

b. Models with many hidden layers

VGG13, VGG6, and VGG13-BN are inspired by the VGG16 models of Ref. [51] (specifically, configuration D of Table 1), with reduced depth for computational efficiency. In both models, all convolutional kernels have length three and all max-pooling layers have a kernel size and stride of two. Each convolutional layer includes a bias for each of its output feature maps, and we employ a rectifying nonlinearity after each convolutional layer. We do not employ padding, so each convolutional layer outputs a segment two samples shorter than the input, and each max-pooling layer halves its input's length.

VGG6 consists of a total of five convolutional layers followed by one fully connected layer, with max pooling after the second and fifth convolutional layers. The first two convolutional layers output 128 feature maps, while the following three output 256. After the final max-pooling layer, the segment length has been reduced to 16. With the 256 output feature maps of length 16 as input, the fully connected layer linearly combines 4096 inputs into a single scalar-valued output, which is passed through a sigmoid nonlinearity to produce a probability estimate.

VGG13 consists of a total of 11 convolutional layers, four max-pooling layers, and two fully connected layers. It is identical in structure to VGG6 through the second max-pooling layer. This is followed by two sets of three convolutional layers with 512 feature maps followed by a max-pooling layer. After the final max-pooling layer, the segment length has been reduced to seven. With the 512 output feature maps of length seven as input, the first fully connected layer has 3584 inputs and 4096 outputs; the second fully connected layer linearly combines its 4096 inputs into a single scalar-valued output, which is passed through a sigmoid nonlinearity to produce a probability estimate.

APPENDIX B: TRAINING AND EVALUATION PROTOCOLS

We follow the following training procedure for all models discussed here unless otherwise specified. As discussed in Sec. II A 2, we draw training, validation, and test data from three separate but nearby time periods. Each model is initialized with the standard Kaiming uniform [58] method with the same random seed. Thereafter, we randomly (again using the same random seed for all models) sample 64 data points from the training period for each training batch and perform stochastic gradient descent with the Adam optimizer [59]. The models and training process are implemented in PYTHON with PyTorch [60].

To evaluate the model during training for learning rate decay and early stopping, we also choose a subset of points from the validation period (the number varies across model types depending on memory constraints, but we use the same points for a given model type). We evaluate the model on this validation batch every 50 training iterations. At each validation, if the loss is lower than previously seen, we retain the model state. When the validation loss fails to decrease for four consecutive validations, we reduce the learning rate by a factor of 4; when the validation loss fails to decrease for ten consecutive validations, training ends and we return the model state that performed best on the validation batch.

We also choose a second, larger (1000-sample) validation batch to evaluate and compare models with different parameter settings. Once training is complete, we evaluate each model on this larger batch and choose the best-performing one. The validation accuracies and losses we report are computed for this model on this batch.

Finally, we calculate test accuracy by running only the best-performing model of a given type (chosen based on the second validation batch, as described above) on a dataset consisting of every glitch from the test period and an equal number of appropriately chosen glitch-free points from the test period.

When directly comparing the FF and LF models, we test several initial learning rates and employ learning rate decay and early stopping based on the loss on a held-out validation subset of the training data rather than running with a fixed learning rate schedule and number of epochs; learning rate decreases by a factor of 5 after each epoch of no improvement on a validation set until reaching a minimum threshold, at which point training terminates.

We also normalize all data based on the mean and standard deviation of the raw time series over the entire training period rather than normalizing after computing features for the subset of training samples chosen (as was done in Ref. [48]). For LF, we also test the same initial learning rates and validate during training with a validation batch as described above every epoch (defined as the model seeing approximately as many samples as present in the training dataset, although not necessarily all of them due to the random sampling procedure used to create the training batches) rather than every 50 training batches. Learning rate decreases by a factor of 5 after each epoch of no improvement until reaching the same minimum threshold as for FF, at which time training terminates.

To determine the best input segment length for LF, as discussed in Sec. III C, we consider it as an additional hyperparameter over which to search while maintaining the same (flat) model structure. Our results indicate that a segment length of 4–6 s is ideal for this model and data (see Fig. 2).

APPENDIX C: IMPLICIT SPARSIFYING REGULARIZATION

In Sec. IV, we argued that sparsifying regularization plays a critical role in successful approaches to prediction from large sets of time series: By regularizing the bottom layer weights, we can force the model to select only the most relevant channels, improving both statistical efficiency and interpretability. We suggested elastic net regularization as a practical and effective means of obtaining sparsity. Interestingly, even if we do not explicitly apply sparsifying regularization to the network weights, it is still possible to induce sparsity indirectly, through various architectural choices. In this appendix, we briefly describe two different forms of implicit sparsifying regularization that emerge in certain experiments described in the main body of the paper.

1. Implicit regularization for channel selection

Our first form of implicit regularization can be motivated through the following experiment, which seems to contradict the claims of Sec. IV. We build a shallow model, in which each input channel is convolved with a channel-specific filter, and then the outputs are linearly combined to produce a final prediction. We apply weight decay (L2 regularization) to all of the weights of the model and train in the same manner described in the body of the paper, with an input segment length of 6 s. This approach achieves a validation accuracy of 89.4%—essentially the same as LF. (In contrast, as discussed in Sec. IV, with neither the extra linear layer nor explicit sparse regularization, the best accuracy achieved is 64.4%.) The resulting model is also quite sparse, with all but a few hundred channels having a magnitude of zero or negligibly close to zero.[7]

There are two surprises here: First, this setup does not involve any explicit sparsifying regularization—just weight decay. Second, the extra linear layer has no effect on the expressiveness of this model class—because the extra linear layer simply applies a (scalar) linear transform to the output of the first layer, the class of input-output relationships that can be implemented by this two-layer model is exactly the same as that which can be represented by LF.

These surprises are actually linked: One can prove that, under L2 regularization, the effect of the extra linear layer is to induce a sparsifying regularization on the first layer filters. This is essentially a consequence of the basic relationship

$$\min_{xy=w} \frac{1}{2}x^2 + \frac{1}{2}y^2 = |w|. \tag{C1}$$

In words, this says that "overparametrizing the scalar w by writing it as a product of two quantities x and y that are L2 regularized is equivalent to L1 regularization on w."

This phenomenon extends quite broadly. Let $f\colon \mathbb{R}^d \to \mathbb{R}$. Consider the problem of minimizing $f(\boldsymbol{w})$ with respect to $\boldsymbol{w}$:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w}). \tag{C2}$$

We are interested in what happens if we introduce an additional scalar variable of optimization β, replace $\boldsymbol{w}$ with $\beta\boldsymbol{w}$, and introduce L2 regularization on both $\boldsymbol{w}$ and β:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d,\beta} f(\beta\boldsymbol{w}) + \frac{\gamma}{2}\beta^2 + \frac{\gamma}{2}\|\boldsymbol{w}\|_2^2. \tag{C3}$$

We argue that this extended problem is equivalent to a regularized problem in $\boldsymbol{w}$ only:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w}) + r(\boldsymbol{w}), \tag{C4}$$

where r is a regularizer. To that end, consider the following problem:

$$\min_{\beta\boldsymbol{w}=\boldsymbol{v}} \frac{1}{2}\beta^2 + \frac{1}{2}\|\boldsymbol{w}\|_2^2. \tag{C5}$$

[7]We use "magnitude" here to refer to the product of the L2 norm of its learned filter and the corresponding linear weight scalar. Because of the lack of an explicit approach to handle the nondifferentiability of the implicit sparsifying regularization, most of these magnitudes do not become exactly 0, but the vast majority are smaller than 10^{-9}.

It is possible to solve this problem in closed form. $\boldsymbol{w}$ is feasible if and only if $\boldsymbol{w} = s\boldsymbol{v}$ for some s. In this situation, the only feasible β is $\beta = \|\boldsymbol{v}\|_2/\|\boldsymbol{w}\|_2$. Plugging in, we find an equivalent problem:

$$\min_s \frac{1}{2s^2} + \frac{\|\boldsymbol{v}\|_2^2 s^2}{2}. \tag{C6}$$

Setting the derivative equal to zero, we obtain $s_\star = \frac{1}{\|\boldsymbol{v}\|_2^{1/2}}$. Plugging back in, we obtain

$$\min_{\beta\boldsymbol{w}=\boldsymbol{v}} \frac{1}{2}\beta^2 + \frac{1}{2}\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{v}\|_2. \tag{C7}$$

Applying this observation, our extended problem [Eq. (C3)] is equivalent to

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w}) + \gamma\|\boldsymbol{w}\|_2. \tag{C8}$$

Note that here the L2 norm of $\boldsymbol{w}$ is not squared. This is a form of vector sparse regularization which encourages $\boldsymbol{w} = \boldsymbol{0}$. These observations can be extended to a multifilter setting in which there are K vector-valued variables of optimization $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K$. In this setting, adding one extra variable β_i for each $\boldsymbol{w}_i$ induces a sum-of-norms regularization:

$$\begin{aligned}&\min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K,\beta_1,\ldots,\beta_K} f(\beta_1\boldsymbol{w}_1,\beta_2\boldsymbol{w}_2,\ldots,\beta_K\boldsymbol{w}_K) + \sum_{i=1}^K \frac{\gamma}{2}\beta_i^2 + \frac{\gamma}{2}\|\boldsymbol{w}_i\|_2^2 \\ &\equiv \min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K} f(\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K) + \gamma\sum_{i=1}^K \|\boldsymbol{w}_i\|_2.\end{aligned} \tag{C9}$$

Again, this is a vector sparsity regularizer, which encourages just a few of the $\boldsymbol{w}_i$ to be nonzero. It is also possible to work out equivalent problems when other regularizers are placed on the auxiliary variables β_i. This kind of implicit regularization, in which adding redundant optimization variables dramatically changes the effect of the regularizer, has been demonstrated in a number of previous works (see, e.g., [61,62]).

2. Sparsifying feature maps with long steps

A different type of sparsification is observed in deeper models with a ReLU nonlinearity: Training with a larger learning rate produces more models in which many of the second-layer feature maps are identically zero on the entire training dataset, without negatively impacting performance. In Table II, we report both the number of nonzero feature maps and the validation accuracy for 1HidReLU for various initial step sizes s. When s is large, the number of nonzero feature maps can be as small as one. When s is smaller, the fraction of nonzero feature maps approaches 50%. Interestingly, performance varies only moderately across this range of s, even though the nature of the learned model varies significantly.

This phenomenon can be attributed to the ReLU nonlinearity $\sigma(u) = \max\{u, 0\}$; its output is identically zero when u is negative. The composition of the ReLU with an affine function produces a feature $\boldsymbol{\alpha}(\boldsymbol{x}) = \max\{\boldsymbol{w}^*\boldsymbol{x} + b, 0\}$ which is identically zero on the half-space $H_{\text{off}} = \{\boldsymbol{x}|\boldsymbol{w}^*\boldsymbol{x} + b \le 0\}$. If, across the training dataset, all inputs to this

TABLE II. In ReLU models, large steps sparsify by producing dead neurons. We trained 1HidReLU at several initial learning rates, using the same set of other hyperparameter settings, and evaluated what percentage of these settings performed "acceptably well," i.e., achieved a loss better than an appropriately chosen threshold (second column). The third column lists the average percentage of nonzero hidden feature map at each learning rate among those well-performing models. The fourth column lists the lowest loss achieved at that learning rate over all other hyperparameters. Interestingly, as initial step size increases, the models become increasingly sparse at the feature map level without sacrificing performance.

Initial step size s	Successful parameter settings (%)	Average nonzero feature maps (%)	Best loss
0.001	8.9	26.0	0.2881
0.002	14.4	12.2	0.2605
0.02	23.6	9.3	0.2512
0.2	21.0	2.1	0.2330

TABLE III. The same table as above for VGG13. We observe the same behavior as for 1HidReLU in the percentage of nonzero feature maps decreasing as the initial learning rate increases. However, although the best loss remains relatively consistent, the percentage of successful parameter settings decreases rather than increases with increasing step size, suggesting that it is more difficult to train deeper models with higher step sizes.

Initial step size s	Successful parameter settings (%)	Average nonzero feature maps (%)	Best loss
0.00025	30.2	36.7	0.1956
0.0005	23.3	27.9	0.2032
0.001	9.7	23.8	0.2091
0.002	8.3	17.8	0.2166

TABLE IV. The same table as above for VGG13-BN. Batch normalization appears to have a stabilizing effect, reducing the impact of step size on the behaviors observed above.

Initial step size s	Successful parameter settings (%)	Average nonzero feature maps (%)	Best loss
0.00025	22.6	15.2	0.1732
0.0005	22.9	20.3	0.1806
0.001	21.9	15.9	0.1822
0.002	17.4	24.6	0.1977

function map to this half-space, this feature will be identically zero. Moreover, it is likely to stay zero: Since

$$\forall \boldsymbol{x} \in \text{interior}(H_{\text{off}}), \qquad \frac{\partial \alpha(\boldsymbol{x})}{\partial \boldsymbol{w}} = \boldsymbol{0} \quad \text{and} \quad \frac{\partial \alpha(\boldsymbol{x})}{\partial b} = 0,$$

gradient or subgradient updates to $(\boldsymbol{w}, b)$ stay zero. In the literature, this is sometimes referred to as a *dead neuron*. It has been observed both experimentally and theoretically that taking very large steps in $\boldsymbol{w}$ and b tends to push data points $\boldsymbol{x}$ into H_{off}, producing large numbers of dead neurons, leading to very sparse representations, as reported in Table II.

In Tables III and IV, we report the results of the same experiment with the VGG13 and VGG13-BN models respectively. For VGG13, we observe a similar effect as in 1HidReLU in the number of nonzero feature maps decreasing with increasing initial step size, but the percentage of successful parameter settings decreases rather than increases. For VGG13-BN, we observe weaker effects from varying the initial step size, suggesting that batch normalization has a stabilizing effect.

This type of sparsification may have less overt statistical benefits, although it could convey benefits in terms of interpretability of the learned model and test-time efficiency.

[1] T. Akutsu *et al.*, Overview of KAGRA: Detector design and construction history, Prog. Theor. Exp. Phys. **2021**, 05A101 (2021).

[2] A. Buikema *et al.*, Sensitivity and performance of the Advanced LIGO detectors in the third observing run, Phys. Rev. D **102**, 062003 (2020).

[3] B. P.Abbott *et al.*, Prospects for observing and localizing gravitational-wave transients with Advanced LIGO, Advanced Virgo and KAGRA, Living Rev. Relativity **23**, 3 (2020).

[4] M. Tse *et al.*, Quantum-Enhanced Advanced LIGO Detectors in the Era of Gravitational-Wave Astronomy, Phys. Rev. Lett. **123**, 231107 (2019).

[5] D. V. Martynov *et al.*, Sensitivity of the Advanced LIGO detectors at the beginning of gravitational wave astronomy, Phys. Rev. D **93**, 112004 (2016).

[6] K. L. Dooley *et al.*, GEO 600 and the GEO-HF upgrade program: Successes and challenges, Classical Quantum Gravity **33**, 075009 (2016).

[7] R. Abbott *et al.* (The LIGO Scientific Collaboration and The Virgo Collaboration), GW150914: The Advanced LIGO Detectors in the Era of First Discoveries, Phys. Rev. Lett. **116**, 131103 (2016).

[8] J. Aasi, B. Abbott, R. Abbott, T. Abbott, M. Abernathy, K. Ackley, C. Adams, T. Adams, P. Addesso, R. Adhikari *et al.*, Advanced LIGO, Classical Quantum Gravity **32**, 074001 (2015).

[9] F. Acernese, M. Agathos, K. Agatsuma, D. Aisa, N. Allemandou, A. Allocca, J. Amarni, P. Astone, G. Balestri, G. Ballardin *et al.*, Advanced Virgo: A second-generation interferometric gravitational wave detector, Classical Quantum Gravity **32**, 024001 (2015).

[10] C. Affeldt *et al.*, Advanced techniques in GEO 600, Classical Quantum Gravity **31**, 224002 (2014).

[11] Y. Aso, Y. Michimura, K. Somiya, M. Ando, O. Miyakawa, T. Sekiguchi, D. Tatsumi, and H. Yamamoto, Interferometer design of the KAGRA gravitational wave detector, Phys. Rev. D **88**, 043007 (2013).

[12] G. M. Harry (The LIGO Scientific Collaboration), Advanced LIGO: The next generation of gravitational wave detectors, Classical Quantum Gravity **27**, 084006 (2010).

[13] S. Soni *et al.*, Discovering features in gravitational-wave data through detector characterization, citizen science and machine learning, Classical Quantum Gravity **38**, 195016 (2021).

[14] H. Yu, R. X. Adhikari, R. Magee, S. Sachdev, and Y. Chen, Early warning of coalescing neutron-star and neutron-star-black-hole binaries from the nonstationary noise background using neural networks, Phys. Rev. D **104**, 062004 (2021).

[15] J. Merritt, B. Farr, R. Hur, B. Edelman, and Z. Doctor, Transient glitch mitigation in Advanced LIGO data, Phys. Rev. D **104**, 102004 (2021).

[16] D. Davis, LIGO detector characterization in the second and third observing runs, Classical Quantum Gravity **38**, 135014 (2021).

[17] S. Bianchi, A. Longo, G. Valdes, G. González, and W. Plastino, An automated pipeline for scattered light noise characterization, Classical Quantum Gravity **39**, 195005 (2022).

[18] P. Nguyen, Environmental noise in advanced LIGO detectors, Classical Quantum Gravity **38**, 145001 (2021).

[19] K. Cannon, GstLAL: A software framework for gravitational wave discovery, SoftwareX **14**, 100680 (2021).

[20] C. Stachie, T. D. Canton, E. Burns, N. Christensen, R. Hamburg, M. Briggs, J. Broida, A. Goldstein, F. Hayes, T. Littenberg, P. Shawhan, J. Veitch, P. Veres, and C. A. Wilson-Hodge, Search for advanced LIGO single interferometer compact binary coalescence signals in coincidence with Gamma-ray events in Fermi-GBM, Classical Quantum Gravity **37**, 175001 (2020).

[21] D. Davis, L. V. White, and P. R. Saulson, Utilizing aLIGO glitch classifications to validate gravitational-wave candidates, Classical Quantum Gravity **37**, 145001 (2020).

[22] R. Essick, P. Godwin, C. Hanna, L. Blackburn, and E. Katsavounidis, iDQ: Statistical inference of non-gaussian

noise with auxiliary degrees of freedom in gravitational-wave detectors, Mach. Learn. **2**, 015004 (2020).
[23] E. Cuoco, Enhancing gravitational-wave science with machine learning, Mach. Learn. **2**, 011002 (2021).
[24] M. Razzano and E. Cuoco, Image-based deep learning for classification of noise transients in gravitational wave detectors, Classical Quantum Gravity **35**, 095016 (2018).
[25] N. Mukund, S. Abraham, S. Kandhasamy, S. Mitra, and N. S. Philip, Transient classification in LIGO data using difference boosting neural network, Phys. Rev. D **95**, 104059 (2017).
[26] M. Zevin *et al.*, Gravity Spy: Integrating advanced LIGO detector characterization, machine learning, and citizen science, Classical Quantum Gravity **34**, 064003 (2017).
[27] G. A. Valdes Sanchez, Data analysis techniques for Ligo detector characterization, Ph.D. thesis, The University of Texas at San Antonio, 2017.
[28] T. J. Massinger, Detector characterization for advanced LIGO, Ph.D. thesis, Syracuse University, 2016.
[29] L. K. Nuttall *et al.*, Improving the data quality of Advanced LIGO based on early engineering run results, Classical Quantum Gravity **32**, 245005 (2015).
[30] R. Biswas, L. Blackburn, J. Cao, R. Essick, K. A. Hodge, E. Katsavounidis, K. Kim, Y.-M. Kim, E.-O. Le Bigot, C.-H. Lee, J. J. Oh, S. H. Oh, E. J. Son, Y. Tao, R. Vaulin, and X. Wang, Application of machine learning algorithms to the study of noise artifacts in gravitational-wave data, Phys. Rev. D **88**, 062003 (2013).
[31] D. MacLeod, Improving the sensitivity of searches for gravitational waves from compact binary coalescences, Ph.D. thesis, Cardiff University (United Kingdom), 2013.
[32] J. Aasi, The characterization of Virgo data and its impact on gravitational-wave searches, Classical Quantum Gravity **29**, 155002 (2012).
[33] J. R. Smith, T. Abbott, E. Hirose, N. Leroy, D. MacLeod, J. McIver, P. Saulson, and P. Shawhan, A hierarchical method for vetoing noise transients in gravitational-wave detectors, Classical Quantum Gravity **28**, 235005 (2011).
[34] N. Christensen (LIGO Scientific Collaboration and Virgo Collaboration), LIGO S6 detector characterization studies, Classical Quantum Gravity **27**, 194010 (2010).
[35] T. Isogai (LIGO Scientific Collaboration and Virgo Collaboration), Used percentage veto for LIGO and virgo binary inspiral searches, J. Phys. Conf. Ser. **243**, 012005 (2010).
[36] S. Mukherjee, R. Obaid, and B. Matkarimov, Classification of glitch waveforms in gravitational wave detector characterization, J. Phys. Conf. Ser. **243**, 012006 (2010).
[37] L. Blackburn, The LSC glitch group: Monitoring noise transients during the fifth LIGO science run, Classical Quantum Gravity **25**, 184004 (2008).
[38] D. Sigg, R. Bork, and J. Zweizig, Detector characterization and global diagnostics system of the laser interferometer gravitational-wave observatory (LIGO), in *The Ninth Marcel Grossmann Meeting*, edited by V. G. Gurzadyan, R. T. Jantzen, and R. Ruffini (2002), pp. 1841–1842, 10.1142/9789812777386_0401.
[39] R. Gurav, B. Barish, G. Vajente, and E. E. Papalexakis, Unsupervised matrix and tensor factorization for LIGO glitch identification using auxiliary channels, in *AAI 2020 Fall Symposium on Physics-Guided AI to Accelerate Scientific Discovery* (Association for the Advancement of Artificial Intelligence, 2020).
[40] F. Robinet, Omicron: An algorithm to detect and characterize transient noise in gravitational-wave detectors, https://tds.ego-gw.it/ql/?c=10651 (2015).
[41] J. Powell, D. Trifiro, E. Cuoco, I. S. Heng, and M. Cavaglià, Classification methods for noise transients in advanced gravitational-wave detectors, Classical Quantum Gravity **32**, 215012 (2015).
[42] J. Powell, A. Torres-Forné, R. Lynch, D. Trifirò, E. Cuoco, M. Cavaglià, I. S. Heng, and J. A. Font, Classification methods for noise transients in advanced gravitational-wave detectors II: Performance tests on Advanced LIGO data, Classical Quantum Gravity **34**, 034002 (2017).
[43] J. C. Driggers *et al.* (The LIGO Scientific Collaboration Instrument Science Authors), Improving astrophysical parameter estimation via offline noise subtraction for Advanced LIGO, Phys. Rev. D **99**, 042001 (2019).
[44] G. Vajente, Y. Huang, M. Isi, J. C. Driggers, J. S. Kissel, M. J. Szczepańczyk, and S. Vitale, Machine-learning nonstationary noise out of gravitational-wave detectors, Phys. Rev. D **101**, 042003 (2020).
[45] R. Ormiston, T. Nguyen, M. Coughlin, R. X. Adhikari, and E. Katsavounidis, Noise reduction in gravitational-wave data via deep learning, Phys. Rev. Res. **2**, 033066 (2020).
[46] N. J. Cornish and T. B. Littenberg, Bayeswave: Bayesian inference for gravitational wave bursts and instrument glitches, Classical Quantum Gravity **32**, 135012 (2015).
[47] D. Davis, T. B. Littenberg, I. M. Romero-Shaw, M. Millhouse, J. McIver, F. D. Renzo, and G. Ashton, Subtracting glitches from gravitational-wave detector data during the third LIGO-Virgo observing run, Classical Quantum Gravity **39**, 245013 (2022).
[48] R. E. Colgan, K. R. Corley, Y. Lau, I. Bartos, J. N. Wright, Z. Márka, and S. Márka, Efficient gravitational-wave glitch identification from environmental data through machine learning, Phys. Rev. D **101**, 102003 (2020).
[49] Y. Bengio, A. Courville, and P. Vincent, Representation learning: A review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. **35**, 1798 (2013).
[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Commun. ACM **60**, 84 (2017).
[51] K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556.
[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, MA, 2016), http://www.deeplearningbook.org.
[53] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc. Ser. B **58**, 267 (1996).
[54] H. Zou and T. Hastie, Regularization and variable selection via the elastic net, J. R. Stat. Soc. Ser. B **67**, 301 (2005).
[55] Y. Tsuruoka, J. Tsujii, and S. Ananiadou, Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty, in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP* (Association for Computational Linguistics, Singapore, 2009), pp. 477–485.

[56] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *International Conference on Machine Learning* (PMLR, Lille, France, 2015), pp. 448–456.

[57] J. Yan, M. Avagyan, R. E. Colgan, D. Veske, I. Bartos, J. Wright, Z. Márka, and S. Márka, Generalized approach to matched filtering using neural networks, Phys. Rev. D **105**, 043006 (2022).

[58] K. He, X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in *Proceedings of the IEEE International Conference on Computer Vision* (IEEE, Santiago, Chile, 2015), pp. 1026–1034.

[59] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), pp. 8024–8035.

[61] P. D. Hoff, Lasso, fractional norm and structured sparse estimation using a Hadamard product parametrization, arXiv:1611.00040.

[62] P. Zhao, Y. Yang, and Q.-C. He, High-dimensional linear regression via implicit regularization, Biometrika **109**, 1033 (2022).