Training strategies for deep learning gravitational-wave searches

Marlin B. Schäfer,[1,2] Ondřej Zelenka,[3,4] Alexander H. Nitz,[1,2] Frank Ohme,[1,2] and Bernd Brügmann[3,4]

[1]*Max-Planck-Institut für Gravitationsphysik, Albert-Einstein-Institut, D-30167 Hannover, Germany*
[2]*Leibniz Universität Hannover, D-30167 Hannover, Germany*
[3]*Friedrich-Schiller-Universität Jena, D-07743 Jena, Germany*
[4]*Michael Stifel Center Jena, D-07743 Jena, Germany*

(Received 16 June 2021; accepted 2 December 2021; published 8 February 2022; corrected 24 February 2022)

Compact binary systems emit gravitational radiation which is potentially detectable by current Earth bound detectors. Extracting these signals from the instruments' background noise is a complex problem and the computational cost of most current searches depends on the complexity of the source model. Deep learning may be capable of finding signals where current algorithms hit computational limits. Here we restrict our analysis to signals from nonspinning binary black holes and systematically test different strategies by which training data is presented to the networks. To assess the impact of the training strategies, we reanalyze the first published networks and directly compare them to an equivalent matched-filter search. We find that the deep learning algorithms can generalize low signal-to-noise ratio (SNR) signals to high SNR ones but not vice versa. As such, it is not beneficial to provide high SNR signals during training, and fastest convergence is achieved when low SNR samples are provided early on. During testing we found that the networks are sometimes unable to recover any signals when a false alarm probability $<10^{-3}$ is required. We resolve this restriction by applying a modification we call unbounded Softmax replacement (USR) after training. With this alteration we find that the machine learning search retains $\geq 91.5\%$ of the sensitivity of the matched-filter search down to a false-alarm rate of 1 per month.

DOI: 10.1103/PhysRevD.105.043002

I. INTRODUCTION

The direct detection of a gravitational wave (GW) on September 14, 2015 [1] started the era of GW astronomy. After the analysis of two and a half observing runs, tens of GWs have been confirmed [2,3]. GW170817 [4] was the first GW event also seen in the electromagnetic spectrum [5–8].

The latency between a GW and its reported detection is a vital aspect of multimessenger missions. Lowering the delay between data aggregation and signal detection allows to maximize electromagnetic observation time and reduces the risk that early emissions are being missed.

To extract GW signals from the instrument data, a well-established technique known as *matched filtering* is used in many search algorithms. It convolves *templates*, i.e., precalculated models of the expected signals, with the measured data [9–13]. When one of these templates matches the data to a given degree and the data quality is high enough, these searches report a candidate detection.

Matched filtering is known to be optimal in stationary Gaussian noise when accurate models of the waveform exist [14]. However, it can be computationally limiting when many templates are required. This is the case when effects such as higher-order modes [15], precession [16], or eccentricity [17] are considered. Furthermore, signals which are not covered by the filter bank may be missed entirely. While there are unmodeled searches that detect coincident excess power in different detectors [18–20], they are less sensitive in regions where accurate models exist.

Recently, new deep learning based searches have started to be explored [21–26]. Summaries of the current state of the field are given in [27,28]. The pioneering works by George *et al.* [21] and Gabbard *et al.* [23] demonstrated that deep neural networks are capable of detecting GWs from two merging black holes (BBH). The networks have also proven to generalize to signals with previously unseen parameters [21,29]. It was shown that these algorithms can distinguish data containing a GW from pure noise as well as matched filtering with a false-alarm probability (FAP) down to 10^{-3}. That means, the networks were tested down to a level at which about 1 in 1000 pure noise samples was falsely classified as containing a signal.

The authors of [30] find that the FAPs determined by the original studies do not directly translate to false-alarm rates (FARs) on continuous data streams. For FARs, the appropriate question to ask is how many false signals does the network identify per time interval of continuous data, as opposed to how many uncorrelated data chunks are falsely identified as containing a signal. The effects of clustering subsequent outputs when the network is applied via a sliding window have to be accounted for. Comparing deep learning searches to traditional matched-filter searches is, therefore, not trivial because matched-filter searches typically operate at FARs that are orders of magnitude smaller than what has been tested for early neural networks. In [26] we suggested a standardized testing procedure which produces statistics which are comparable to traditional search algorithms to resolve these issues.

In this paper we reanalyze and extend the results given in the initial papers [21,23]. Our motivation is twofold. First, we want to verify and test the performance of the networks quoted in those papers. Specifically, we apply the testing procedure outlined in [26]. Second, we discuss how the GW data is prepared for and presented to the network. The form of data preparation is often taken as a given, while comparatively more work is invested in finding a network structure that suits the problem. We carefully examine the influence of different choices of data presentation and training strategies on the ability to detect signals given a fixed network.

Here we focus on signal detection. The problem of deep learning parameter estimation is another vital and active area of research. Multiple groups have made advancements in this field [21,31–35].

We use the network presented by Gabbard *et al.* [23] for most of our studies. It is trained on simulated data containing GWs from BBHs with individual black hole masses ranging from $10\ \mathrm{M}_\odot$ to $50\ \mathrm{M}_\odot$. The search is restricted to a single detector. This restriction reduces the parameter space to the two component masses, the orbital phase, the distance to the source and the time of coalescence.

The network classifies segments of 1 s duration sampled at 2048 Hz into the two categories "noise + signal" and "noise" by returning a value between 0 and 1 we call "p-score." A larger p-score corresponds to a higher confidence of the network that the input contains a signal.

To optimize the training strategy, we focus on the difference between *curriculum learning* [36] and *fixed interval training*. Fixed interval training uses a single training set, i.e., a single, fixed range of signal-to-noise ratios (SNRs). Curriculum learning lowers the SNR of the training signals progressively, thus increasing the complexity with time. We evaluate different variants of both strategies. In total 15 different approaches are tested.

Each strategy is applied to 50 randomly initialized networks. We do this to guard against favorable initializations. All tests are done with two different implementations, to further increase robustness of our results. The different implementations use the two core libraries Tensorflow [37] and PyTorch [38], respectively.

We find that most training strategies are capable of closely reproducing the results given in [23]. We do not see a significant difference in performance between curriculum learning and fixed interval training strategies. However, networks that had access to lower SNR signals during training generally outperformed those that only saw high SNR signals. We find that networks trained on fainter signals can generalize to loud ones, while the opposite is not the case.

Further analysis of the networks showed that the efficiency, which is the fraction of correctly classified input samples containing a signal at a given FAP, drops to zero beyond a FAP of 10^{-3} when the training is carried out for long enough. This drop is caused by numerical instabilities in the final activation and the comparatively low penalty of false positives. We propose a simple modification that does not require retraining of the network to push this problem to significantly lower FAPs. We call this modification unbounded Softmax replacement (USR).

We evaluate 3 different networks of each training strategy on a month of simulated data. The networks are applied using a sliding window with step size of 0.1 s. We follow the procedure outlined in [26] to analyze the results. Our evaluation of the base line network is limited by $\mathcal{O}(10^3)$ false alarms estimated with perfect confidence to contain a signal. By applying the USR modification we are able to eliminate this restriction and can calculate sensitivities down to a FAR of 1 per month. For comparison, we construct a template bank and use it to do a matched-filter search on the same data used to evaluate the networks. We find that the machine learning search retains at least 91.5% of the sensitivity of a matched-filter search for all tested FARs and most strategies.

All code required to reproduce our analysis is public and can be found at [39].

The contents of this paper are structured as follows. In Sec. II we describe the architecture, datasets, training strategies, and evaluation methods. We apply these in Sec. III and describe our findings. In particular we describe the USR modification which allows the networks to be tested at low FAPs. We conclude in Sec. IV.

II. METHODS

A. General setup

We focus our studies on the network presented by Gabbard *et al.* in [23]. They used a convolutional neural network with 6 stacked convolutional layers followed by 3 fully connected layers. All but the last layer use an exponential linear unit (ELU) as activation function.

The architecture is altered in two details compared to the original version of [23]. We added a batch normalization layer before the first convolutional layer to take care of *input normalization*. Input normalization scales all inputs to have a mean-value of 0 and a variance of 1. This is standard practice in contemporary deep learning and has been proven to help the network train efficiently [40]. The second modification is a reduction of the pool sizes.

TABLE I. The modified neural network from [23] as used in this study. The given shapes correspond to the tensor shapes in the TensorFlow version of the code, i.e., data length × number of channels. PyTorch swaps these dimensions. The order of the layers is given by reading the column "layer type" from top to bottom and left to right. Layers are grouped by their influence on the output shape and by trainable weights.

Layer type	Kernel size	Output shape
Input + BatchNorm1d		2048 × 1
Conv1D + ELU	64	1985 × 8
Conv1D	32	1954 × 8
MaxPool1D + ELU	4	488 × 8
Conv1D + ELU	32	457 × 16
Conv1D	16	442 × 16
MaxPool1D + ELU	3	147 × 16
Conv1D + ELU	16	132 × 32
Conv1D	16	117 × 32
MaxPool1D + ELU	2	58 × 32
Flatten		1856
Dense + Dropout + ELU		64
Dense + Dropout + ELU		64
Dense + Softmax		2

This change was required because we lowered the sample rate of the data from 8192 Hz to 2048 Hz. We decided to lower the sample rate for multiple reasons. First of all, the detector sensitivity drops sharply above 1 kHz. Thus little to no SNR is lost by disregarding higher frequencies. For this reason current searches are often limited to the same frequency band as well [12,41,42]. Second, signals within our training set merge at much lower frequencies and do not exceed 1 kHz. Finally, as we will show in Sec. III, our training converged to the same state as previous works. We are thus confident that this reduction in the sample rate has no negative impact on the network's ability to detect signals. A reduction of the size of the input to a neural network usually also helps with training. The resulting network setup is depicted in Table I.

All studies presented in Sec. II B and Sec. II C were carried out using the network from George *et al.*[1] [21] as well. However, with our particular training setup, every metric showed performance similar to the network from [23]. We present only results using the network based on the work of Gabbard *et al.*

Each GW signal is defined by the component masses m_1, m_2 and a phase ϕ_0. Two masses are drawn independently from a uniform distribution between 10 $M_\odot$ and 50 $M_\odot$ and the higher and lower values are assigned to m_1 and m_2, respectively, to enforce the condition $m_1 \geq m_2$. Phases are uniformly drawn from the interval $[0, 2\pi]$. We generate signals with 5 different phases for each pair of masses (m_1, m_2). The amplitude, and therefore the distance of the source, is determined by the target SNR we have chosen. We fix the sky position to be overhead the LIGO Hanford detector [43] and the inclination as well as the polarization to 0, because in the case of nonprecessing signals and assuming a single detector, any variation in those parameters can be fully absorbed by modifications of the amplitude and phase of the signal.

The waveforms are generated with a sample-rate of 2048 Hz, a lower-frequency cutoff of 20 Hz, and using the model `SEOBNRv4_opt` [44] (optimized version of `SEOBNRv4` [45]). It is common practice to shift the location of the maximum amplitude by some small time within each training sample. This procedure allows the network to be less sensitive to the exact alignment of the waveform within its input. To achieve this behavior, we shift the position of the merger by a time uniformly drawn from −0.1 s to 0.1 s before projecting onto the Hanford detector. After the projection the signals are *whitened* using the analytic model of LIGO's design sensitivity at its zero detuned high power configuration [46], i.e., we divide the Fourier transformed signal by the square root of the power spectral density (PSD) associated with the power of the background noise at different frequencies, and transform back to the time domain. Whitening the data reduces the power at frequencies where the detector is known to be less sensitive. Next, the waveforms are scaled to an optimal SNR of 1. The optimal SNR ρ_{opt} is defined by

$$\rho_{\mathrm{opt}}{}^2 = 4\mathrm{Re}\left[\int df \frac{\tilde{h}(f)\tilde{h}^*(f)}{S_n(f)}\right], \qquad (1)$$

where $\tilde{h}$ is the Fourier transform of the time domain signal, before it was whitened, $\tilde{h}^*$ is its complex conjugate, S_n is the PSD and Re extracts the real part of the complex number. Finally, we extract a time slice such that the original, not shifted merger time is located 0.7 s from the start of the window.

All noise is simulated from the same PSD used to whiten the signals. After generation, the noise is whitened by the PSD used to create it in the same way the signals are whitened. We choose to explicitly whiten the colored noise to take into account any artifacts the process may introduce. This also eliminates sources of errors and is in principle extendable to real noise.

The whitened signals and noise samples are combined during training. This allows us to rescale the signals at runtime to a desired strength. Since during generation all signals are scaled to SNR 1, rescaling is achieved by a multiplication of the signal with the target SNR.

We have briefly tested training on frequency domain data. This was motivated by studies such as [24,47]. While these studies analyze longer duration signals, there is no conceptual problem to using the frequency representation of short BBH waveforms. To accommodate the complex valued frequency representation we changed the input layer

[1]We adjusted the network from George *et al.* too, by using batch normalization for input normalization and reducing the sample rate of the input to 2048 Hz.

in Table I to a shape of 1025×2 and inserted the real and imaginary parts as different channels. With this being our only modification to the architecture, the network was able to differentiate data containing a signal from pure noise but found up to 65% fewer signals at low SNR. We suspect that with greater effort in finding an optimized architecture, one could regain the performance of the network on time domain data.

We have also explored training on raw data, i.e., data where the computationally expensive whitening is skipped. Even after several hundred epochs the network was not able to distinguish data containing signals from pure noise, irrespective of using the time or frequency domain representation of the data.

Our training set contains 20 000 unique combinations of component masses each of which is used to generate 5 waveforms with random coalescence phases. Therefore, it contains 100 000 individual signals. We generate 200 000 independent noise samples, 100 000 of which are used in combination with the signals. The remaining 100 000 noise samples are used as pure noise. Our training set, therefore, contains 200 000 independent samples.

The validation set is assembled in the same way as the training set. It too contains 100 000 samples of the "signal"-class and 100 000 samples of the "noise"-class for a total of 200 000 samples. The validation set was chosen to be of equal size to the training set due to its influence when curriculum training strategies are used. The conditions for when the complexity of the training set is increased are evaluated on this set.

We use a third dataset to calculate relevant metrics of the network during training. This third set is required, as the validation set directly influences the training for curriculum strategies. Metrics determined on the validation set may, therefore, be biased. We call this third set the efficiency set and describe its usage in Sec. II B. It contains 10 000 unique signals and 400 000 independent noise samples.

Finally, we evaluate the performance of the network on a test set. This test set contains a month of simulated Gaussian noise with injections separated by a time uniformly distributed in the interval [16, 22] s. The injection parameters are drawn from the distributions shown in Table II. Noise is generated using the same PSD used for the training set. A month of data corresponds to ∼26 million correlated samples.

TABLE II. Injection parameters for the dataset used to determine the FAR and sensitive volume of the different networks.

Parameter	Uniform distribution
Component masses	$m_1, m_2 \in (10, 50)\ \mathrm{M}_\odot$
Spins	0
Coalescence phase	$\Phi_0 \in (0, 2\pi)$
Polarization	$\Psi \in (0, 2\pi)$
Inclination	$\cos\iota \in (-1, 1)$
Declination	$\sin\theta \in (-1, 1)$
Right ascension	$\varphi \in (-\pi, \pi)$
Distance	$\mathrm{d}^2 \in (500^2, 7000^2)\ \mathrm{Mpc}^2$

Each network is trained for 200 epochs, i.e., 200 full passes on the training set. We found this to be a sufficient number of training cycles for most of the networks to converge to a stable performance on the validation set. We use the default implementations of the Adam optimizer with a learning rate of 10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ [48]. As loss we use a variant of the binary cross-entropy that is designed to stay finite,

$$L(\boldsymbol{y}_\mathrm{t}, \boldsymbol{y}_\mathrm{p}) = -\frac{1}{N_\mathrm{b}} \sum_{i=1}^{N_\mathrm{b}} \boldsymbol{y}_{\mathrm{t},i} \cdot \log(\epsilon + (1 - 2\epsilon)\boldsymbol{y}_{\mathrm{p},i}). \quad (2)$$

Here $\boldsymbol{y}_\mathrm{t}$ is either $(1,0)^T$ for data containing a signal or $(0,1)^T$ for pure noise, $\boldsymbol{y}_\mathrm{p}$ is the prediction of the network, $N_\mathrm{b} = 32$ is the minibatch size, and $\epsilon = 10^{-6}$.

B. Network performance

A common metric when training neural networks is the *accuracy*, which is the ratio of correctly classified samples over the total number of samples. This approach weighs false-negatives and false-positives equally.

GW searches assign a statistical significance to each event. This is usually given as the FAR of the search at the ranking statistic threshold associated with the candidate event. For the network we use the p-score as ranking statistic. The more false positives a search produces at a given ranking statistic, the less significant each event becomes. Therefore, false-positives severely limit the ability of the search to recover true events. Low latency searches do not distribute any event candidates publicly with a FAR greater than ∼1 per month [12]. For searches which operate on archival data, low FARs are needed to assign a probability for the signal to be of astrophysical origin, based on the expected astrophysical rate of comparable events [2,3].

For these reasons we monitor the *efficiency* of the network rather than the accuracy. The efficiency is the true-positive probability at a fixed false-positive probability, i.e., a fixed FAP. To do so, we sort the p-score outputs of the network on the noise from the efficiency set and use the xth largest as a threshold, where we choose

$$x = \lfloor N_n \cdot \mathrm{FAP} \rfloor \quad (3)$$

Here, N_n is the total number of noise samples used and $\lfloor \cdot \rfloor$ denotes the flooring operation. We then evaluate the signals from the efficiency set scaled to SNRs 3,6,9,12,15,18, 21,24,27 and 30 and count the samples that exceed the threshold. The efficiency is then given by

$$\mathrm{efficiency} = \frac{N_{s>t}}{N_s}, \quad (4)$$

with $N_{s>t}$ being the number of signals assigned a p-score larger than the threshold and N_s the total number of signals. To get a better understanding of the efficiency as a function of the signal strength, we also calculate the efficiencies at each of the SNRs individually. In this work, a FAP of 10^{-4} is used for all efficiency calculations.

For each of the strategies we discuss in Sec. II C the network is trained 50 times from scratch. The parameters of the networks are initially random for each run. The final performance of a single network may depend on these initial values. Training each network-strategy combination multiple times and averaging over their efficiencies reduces the influence of the network initialization, thus yielding greater insight into the impact of the training strategy.

After training has completed for all 50 networks, we choose 3 networks for which we calculate the sensitive volume and the false-alarm rate on a month of simulated data. The networks are chosen by the following scheme. We select the epoch of the maximum efficiency of all networks. At this epoch we pick the best and the worst performing networks, where ranking is based on the efficiency. The last network is chosen to be the one which has the efficiency closest to the average efficiency over all 50 runs at the chosen epoch.

The sensitivity and FAR calculation follows the procedure outlined in [26]. As suggested in [21,23], the network is applied to time series data of duration longer than the input window via a sliding window. We choose a step size of 0.1 s to ensure the correct alignment of the merger time within the input window for at least one step. Each window is whitened individually using the same method and noise model applied to the training set.

To reduce the computational cost, the data are sliced into the input windows and preprocessed only once. We store this sliced data and apply the different networks to it. This allows us to evaluate the entire month of data in about 1 h on a single NVIDIA RTX 2070 SUPER.

The network outputs a value between 0 and 1 for every slice. A value of 1 corresponds to the network being confident that it has seen a signal. We use this output as ranking statistic. Outputs that exceed a threshold, which we call trigger-threshold, are clustered by their time. Within each cluster the first time where the output becomes maximal is picked. The combination of this time and the corresponding network output is called an event.

The list of events is compared to the known injection times. If the event is separated from the closest injection by more than some maximum time it is called a false positive. Otherwise we consider it a true positive. From these we can calculate the FAR as well as the sensitive volume as detailed in [26]. The FAR is given by

$$\mathrm{FAR} = \frac{N_f}{T_o}, \tag{5}$$

where N_f is the number of false positives and T_o is the duration of the analyzed data. When the injections are distributed uniformly in volume the sensitive volume of the search is given by

$$V(\mathrm{FAR}) = V(d_{\max}) \frac{N_t(\mathrm{FAR})}{N_i}, \tag{6}$$

where $d_{\max}$ is the maximum distance at which sources are injected, $V(d_{\max})$ is the volume of a sphere with radius $d_{\max}$, N_i is the total number of injections and $N_t(\mathrm{FAR})$ is the number of true positives at a given FAR. The FAR can be adjusted by considering only events above a given threshold. To convert the sensitive volume to a distance we calculate the radius of a sphere of the given volume.

We use a p-score of 0.1 as our trigger-threshold. Triggers are said to belong to a cluster if they are within 0.2 s of the cluster bounds. An event is called a true-positive if there was an injection within 0.3 s of the reported event time. Otherwise it is a false-positive. We chose the cluster boundary time as twice the step size to allow for modest smoothing of the network output, while keeping it short compared to the average duration of a signal [$\mathcal{O}(1\ \mathrm{s})$]. The maximum separation between an event and the corresponding injection was chosen to be larger than the cluster boundaries but still small compared to the average signal duration. None of these parameters were optimized.

Figure 1 shows example output from one of the networks. The top panel shows the raw input with the injected waveform overlayed in black. The injection time is marked with a red vertical line and the grey lines highlight ± 0.3 s where events are true positives. The bottom panel shows the network output for the corresponding time. The black vertical lines show the events returned by the search.

C. Training strategies

The two initial publications by George *et al.* [21] and Gabbard *et al.* [23] disagree on the usefulness of curriculum learning. Whereas George *et al.* find a noticeable improvement by using curriculum learning, Gabbard *et al.* find no difference in the final performance of the network.

We aim to determine the impact curriculum learning has on the final performance and speed of convergence of these networks. By doing so we optimize the sensitivity of the networks tested here and hope that our findings generalize also to state-of-the-art machine learning search algorithms [25,26,32,49].

Our study contains 10 curriculum learning and 5 fixed interval training strategies. An overview can be found in Table III. The minimum SNR allowed in any of these strategies is ≥ 5. We choose SNR 5 as a lower bound as this is roughly the lowest single detector SNR at which signals seen in multiple detectors can be confidently distinguished from noise [2,3].

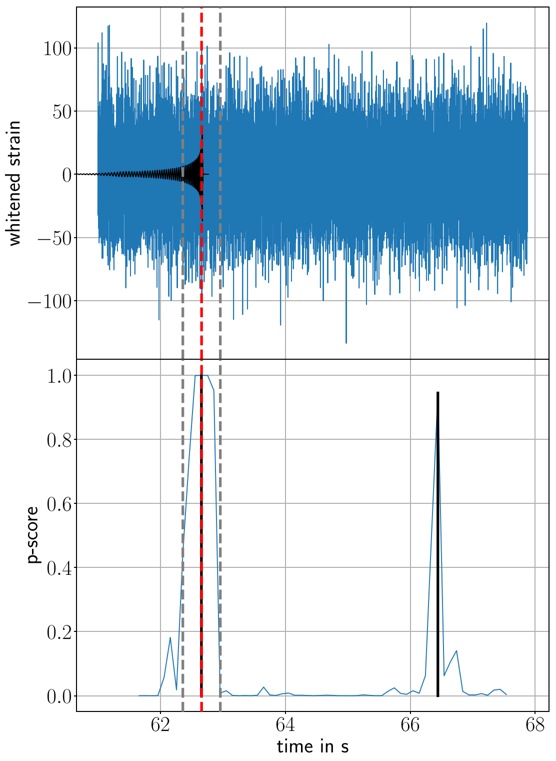

FIG. 1. A sample output from the network on long duration data. The top panel shows the whitened input data. The injected signal is overlayed in black. The red vertical line signifies the time of the injection, i.e., the time that would ideally be returned by the search algorithm. The vertical grey lines mark the interval within which a returned event is classified as a true positive. The bottom panel shows the output of the network corresponding to the input. The vertical red and grey lines, again, show the true injection time and the allowed interval for true positives respectively. The black vertical lines mark the events returned by the search. Their height is the p-score attributed to the event. While the first event is a true positive, the second event is a false-positive originating from noise.

We test 5 different conditions for the optimal SNR contained in the training data for both types of strategies. For curriculum strategies, these conditions prescribe when the SNR of the training data is lowered. For fixed interval strategies the conditions are the interval from which the SNR for each sample is drawn.

Curriculum strategies use either the validation loss, the validation accuracy, or the number of epochs since the last step as conditions. For validation loss and validation accuracy we choose either a threshold or wait until the values stabilize and do not improve anymore. The latter are labeled by a prefix “plateau” throughout this paper. We choose a threshold of 0.95 for the validation accuracy and 0.2 for the validation loss. These values are arbitrary but proved to work well. When using the plateau conditions we lower the training range when the validation loss or validation accuracy, respectively, do not improve by more than 0.01% for 6 consecutive epochs. Finally, we also test

TABLE III. An overview of the different training strategies tested in this work. The “curriculum” type strategies lower the SNR of the training samples whenever the condition in the last column is fulfilled. All of them start with SNR $\in [90, 100]$. Curriculum strategies with the postfix relative in their name lower the boundaries of the interval by 10% at each step, until the lower limit falls below SNR 5. The other curriculum strategies lower the bounds by a fixed value of 5, until the lower limit reaches SNR 5. A metric fulfills the plateau condition when it has not improved by more than 0.01% for 6 consecutive epochs. The “fixed interval” type strategies use a single SNR range for the entire training. Their interval is given in the last column.

Type	Name	Condition
Curriculum	Accuracy Accuracy relative	When validation accuracy ≥ 0.95
	Epochs Epochs relative	Every 5 epochs
	Loss Loss relative	When validation loss ≤ 0.2
	Plateau accuracy Plateau accuracy relative	6 epochs validation accuracy plateau
	Plateau loss Plateau loss relative	6 epochs validation loss plateau
Fixed interval	SNR 30	SNR $= 30$
	SNR 15	SNR $= 15$
	SNR 8	SNR $= 8$
	Low	SNR $\in [5, 15]$
	Full	SNR $\in [5, 100]$

lowering the training SNR irrespective of any of the metrics, by waiting 5 epochs between steps. We choose to wait 5 epochs to allow the network enough time at each signal strength while ensuring we reach the minimum SNR. No extensive studies testing different values were made.

We test two different approaches to lowering the training SNR. All curriculum strategies start with SNRs which are uniformly drawn from the interval [90, 100]. Strategies that are given the postfix “relative” lower the bounds of this interval by 10% at each step. The ranges are not lowered further when the lower bound of the interval reaches SNR 5. Strategies without the postfix “relative” lower the bounds of the interval by a fixed value of 5 at each step. This procedure is also continued down to a minimum bound of SNR 5.

For fixed interval training strategies we test training on a single SNR as well as a fixed size interval of SNRs. We choose to train on fixed single SNRs 8, 15 and 30 to cover the low, mid and high SNRs respectively. Training on a single SNR allows us to test how well the network generalizes to lower and higher SNRs than it has seen during training. By drawing the SNR from an interval we aim to reduce the dependence on a specific signal strength. We choose two strategies that draw SNRs from a fixed range. One covers only the lowest range used by any of the nonrelative curriculum strategies, i.e., it draws the signal SNRs from the interval SNR $\in [5, 15]$. The other draws the SNRs from the entire range of SNRs seen by the curriculum strategies, SNR $\in [5, 100]$.

D. Matched-filter baseline

In order to assess how sensitive the trained networks are in relation to conventional searches, we perform a matched-filter analysis of the test set described in Sec. II A. To do so, we utilize the PyCBC analysis toolkit [50].

The template bank covers component masses from 10 $M_\odot$ to 50 $M_\odot$ and is constructed to lose no more than 3% of the SNR of any signal due to its discreteness. The templates are placed stochastically. In total, the bank contains 598 templates.

The search is implemented by `pycbc_inspiral`. We configured it to output a set of times where any template of the bank convolved with the data exceeds a matched-filter SNR of 5. Unlike the optimal SNR, the matched-filter SNR is the match of a detector data segment with a template, and so it varies based on the noise realization, while the optimal SNR assumes a noise realization that is constant zero. Combining the times where the threshold is exceeded with the corresponding matched-filter SNR and by using this SNR as ranking statistic, we obtain a set of triggers. We then process these triggers as described in Sec. II B to find events and calculate FARs and sensitive distances.

The configuration files are included in the data release [39].

III. RESULTS

A. Sensitivities

We are able to reproduce or in some cases even improve on the results given in [23]. The top panel of Fig. 2 shows the efficiency of one network as a function of the SNR at fixed FAPs calculated on the efficiency set. We compare our findings to theirs and find excellent agreement with the results shown in Fig. 3 of [23], which closely reproduced efficiencies of matched filtering. The efficiencies at FAPs down to 10^{-3} for most other training strategies also closely follow the findings of Gabbard *et al.* We are, therefore, able to robustly reproduce the findings of [23].

In Fig. 3 we show the evolution of the efficiency of the 50 networks trained on the fixed interval SNR $\in [5, 15]$ as the number of training epochs is increased at a FAP of 10^{-4}. Each panel of the plot shows the efficiency for a chosen SNR which allows us to observe how well the networks perform during different stages of the training at different signal strengths. This is especially interesting for curriculum strategies, where the SNR in the training set is adjusted as the network trains. The grey lines show the evolution of the efficiency for the different network initializations. The black, dashed line is the average of the grey lines.

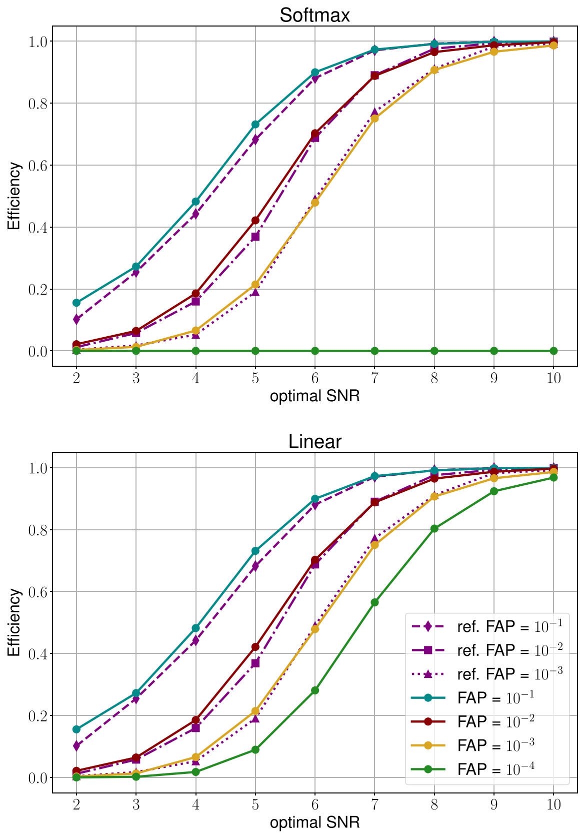

FIG. 2. The efficiency as a function of optimal SNR at different FAPs. The network was trained on SNRs drawn from the fixed interval [5, 15]. We used epoch 186 of the network with the lowest efficiency at that epoch to produce this figure. The top panel shows the efficiency when the last layer uses a Softmax activation, the bottom panel shows the same network with the USR modification. We determine the threshold on the network output using a set of 400 000 pure noise samples. Any of the 10 000 signals at each SNR exceeding this threshold are counted as detected. We compare our findings to Fig. 3 of Ref. [23] which closely reproduces efficiencies of matched filtering.

We highlight the evolution of a single network in dark grey. The red, dashed, vertical line signifies the epoch of maximum efficiency over all 50 networks and 200 epochs.

All networks in Fig. 3 converge to similar efficiencies during the first ~100 epochs. However, as training continues sudden drops to zero efficiency occur which become more frequent at later epochs. As a result the average efficiency drops continuously after some time. All networks show this behavior and thus the influence of an unlucky initialization can be ruled out. Furthermore, the drops are observed at all SNRs simultaneously and, therefore, do not depend on the signal strength.

The same effect can be seen in the top panel of Fig. 2. For FAPs $\geq 10^{-3}$ the curves behave as expected. As one lowers the FAP the efficiency at any given SNR is expected to drop. Visually this manifests in a shift of the efficiency curves toward higher SNRs. Ideally, this behavior would

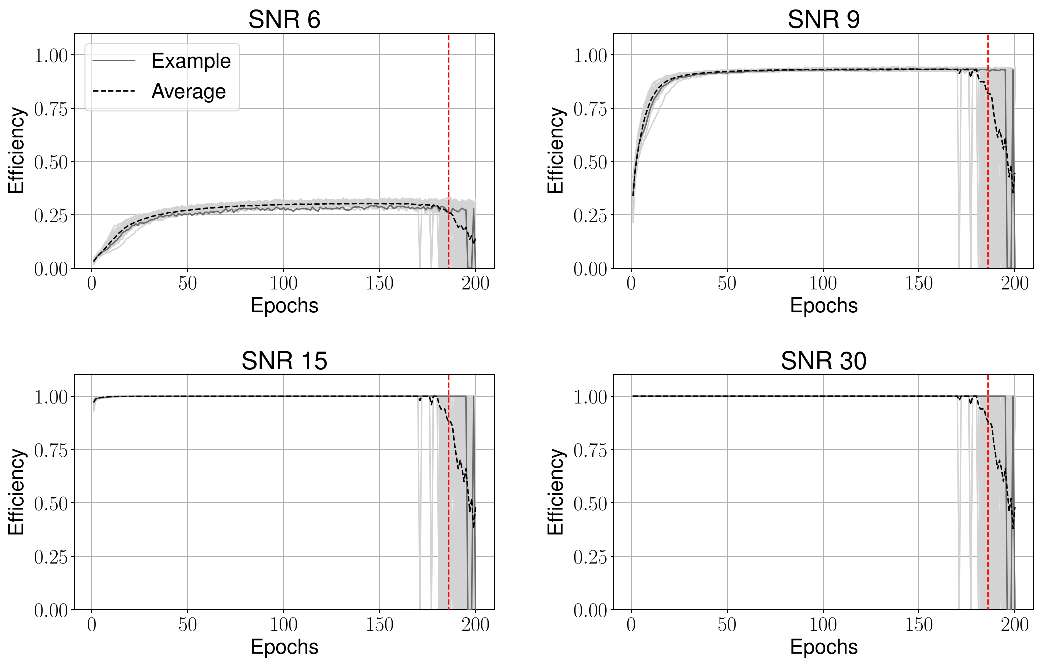

FIG. 3. The evolution of the efficiency as a function of the epochs at different optimal SNRs. Training used the fixed SNR interval [5, 15]. The individual evolutions of all 50 runs are included as grey curves that form overlapping grey bands when plotted together. The dashed black line is the average of those. In dark grey we highlight the evolution of the efficiency for a single network. At the epoch marked by the red, dashed, vertical line we select the network with the highest, lowest and closest to average efficiency for further testing. The curves are computed at a FAP of 10^{-4}.

be true for any FAP. However, at a FAP of 10^{-4} the efficiency collapses and becomes a constant 0.

The drops to zero efficiency are caused by noise samples which are attributed a p-score of 1. Since the Softmax activation on the last layer restricts outputs to the interval [0, 1], no signal samples can achieve a p-score larger than the threshold and thus they cannot be distinguished from noise.

Many of the noise samples attributed a p-score of 1 are caused by numerical rounding errors in the Softmax activation

$$\mathrm{Softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=0}^{N}\exp(x_j)}, \tag{7}$$

where $\boldsymbol{x} = (x_0, x_1, \ldots, x_N)$ is the vector of outputs of the previous layer in the network, and $N + 1$ is the number of neurons in the layer.

The networks operate with single precision (32-bit) floating point numbers. Therefore, small changes in the values of $\boldsymbol{x}$ may cause a roundoff error due to the rapid change in scale of the exponential functions. When this occurs, the fraction may evaluate to 1 even when mathematically (7) may never be 1.

We removed the final activation of the pre-trained network in an attempt to avoid the rounding errors. To do so, we recast (7) for $N = 1$ into

$$\frac{\exp(x_0)}{\exp(x_0) + \exp(x_1)} = \frac{1}{1 + \exp(x_1 - x_0)}, \tag{8}$$

and impose thresholds for the efficiency calculation on the difference $x_0 - x_1$ directly rather than $\mathrm{Softmax}(\boldsymbol{x})_0$. Since (8) is bijective, there exists a direct relation between thresholds in $x_0 - x_1$ and the thresholds on $\mathrm{Softmax}(\boldsymbol{x})_0$. We use $x_0 - x_1$ rather than $x_1 - x_0$ as our ranking statistic since $x_0 - x_1 > \hat{x}_0 - \hat{x}_1 \Leftrightarrow \mathrm{Softmax}(\boldsymbol{x})_0 > \mathrm{Softmax}(\hat{\boldsymbol{x}})_0$. We call this modification unbounded Softmax replacement.

The resulting efficiency is depicted in the bottom panel of Fig. 2. Figure 4 shows the efficiency evolution at different optimal SNRs. We find that the drops to zero efficiency vanish when we apply USR. This is the case for all training strategies we explored and more examples are shown in the Appendix (see Fig. 9–12).

One could also try to resolve the rounding issue by using double precision (64-bit) floating point numbers instead of single precision when applying the Softmax layer. We have tested a numerically safe implementation of the Softmax and found that its first output is rounded up to one even for quadruple (128-bit) precision when the difference $x_0 - x_1 > 45$. This is a relatively low value that indeed occurs for some noise realizations in our experiments. Although using higher precision for the Softmax layer increases the range of values it can operate on, the USR still solves roundoff issues more robustly.

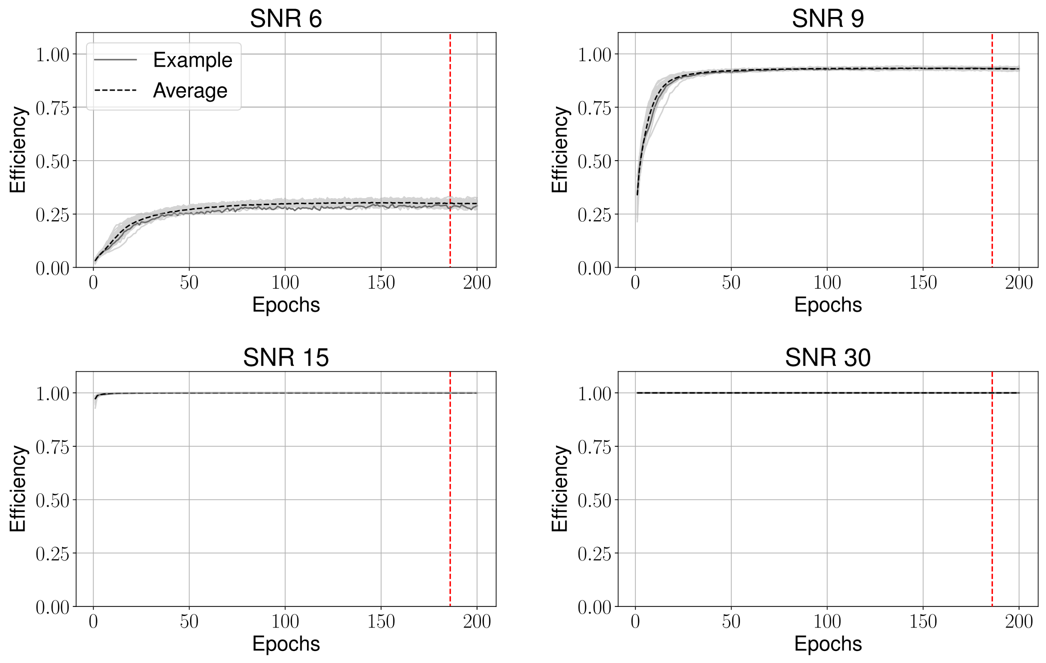

FIG. 4. The evolution of the efficiency as a function of the epochs at different optimal SNRs. Training used the fixed SNR interval [5, 15]. The individual evolutions of all 50 runs are included as grey curves that form an overlapping grey band when plotted together. The dashed black line is the average of those. In dark grey we highlight the evolution of the efficiency for a single network. At the epoch marked by the red, dashed, vertical line we select the network with the highest, lowest and closest to average efficiency for further testing. The curves are computed at a FAP of 10^{-4}. This figure shows the same networks as Fig. 3 after applying USR. This prevents the efficiency to drop to 0.

The efficiency is a metric that is easy to calculate and physically more relevant than the accuracy of the network. However, it does not deal with samples where waveforms are misaligned in the data or take into account longer stretches of time. It is, therefore, only an approximation to the true statistic we want to calculate: the sensitive volume.

To assess if the efficiency is a good approximation to this statistic, we calculate the sensitive volume of three chosen networks for every training strategy as described in Sec. II B. The networks are chosen from the 50 different initializations based on their efficiency at a selected epoch. We pick the networks with the highest, lowest and closest to average efficiency and denote them with "High," "Low," and "Mean," respectively, from here on out. If the efficiency at a fixed FAP is a good indicator of the networks sensitivity we expect the sensitive volume to scale with the efficiency.

Figure 5 shows the sensitive distance as a function of the FAR computed for the three networks trained on the fixed, low SNR interval. It compares the networks with (dashed) and without (continuous) the final Softmax activation and shows an equivalent matched-filter search in purple as reference. We find that the network is sensitive to sources up to a distance of 2150 Mpc with 1 false alarm per month.

The sensitive radii of all converged deep learning searches lie within 3.4% of each other for FARs where all of them are nonzero. However, the sensitivity of the

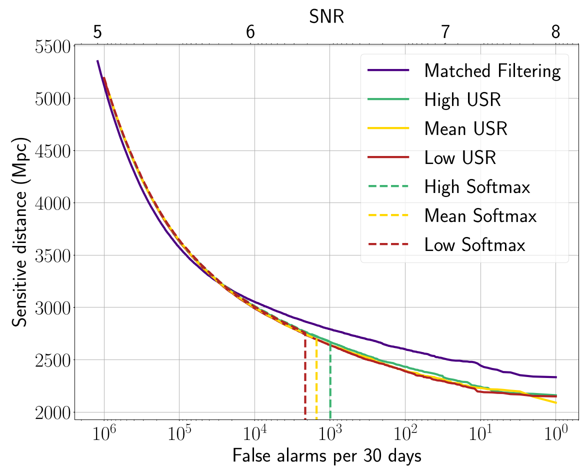

FIG. 5. The sensitive distance as a function of the FAR (bottom horizontal axis) for different search algorithms. We compare differently initialized networks trained on data containing signals with SNR $\in [5, 15]$ to an equivalent matched-filter search. The dashed lines show the original networks, the filled lines show the corresponding network when USR is applied. The labels "High" (green), "Mean" (yellow) and "Low" (red) correspond to the networks with the highest, closest to average and lowest efficiency at epoch 186, respectively. In purple we show the equivalent matched-filter search that operates with a template bank containing 598 templates. The top horizontal axis shows the SNR threshold for the matched-filter search corresponding to the FAR on the bottom axis.

networks with the final Softmax activation drops to zero for FARs $\leq \mathcal{O}(10^3)$ per month. This drop is caused by $\mathcal{O}(10^3)$ false alarms with a p-score of 1. This saturation of the final activation can be alleviated by applying the USR modification and using the new output as a ranking statistic.

All tested networks have also been reevaluated using higher precision floating point data types for the final activation function evaluation (example shown in Fig. 6). This resulted in the networks remaining sensitive at FARs down to 3 per month. However, applying the USR modification allowed us to test the network down to a FAR of 1 per month. Additionally, casting to a higher precision considerably increases computation time in the network due to hardware optimizations of GPUs for single precision floating point operations. In our view, the effectiveness of the USR outweighs the benefits of using higher precision, hence we only report results obtained with the USR modification.

We had expected to find networks with higher efficiencies to be more sensitive, even within different initializations of the same training strategy. As such in Figure 5 we expected to find the sensitivity curve labeled "High" to be above the one labeled "Mean" above the one labeled "Low". While this is true in the example shown in Figure 5 in some regions, for other training strategies the order is arbitrary. All initializations converge to basically the same sensitivity. Sensitivity plots for all training strategies are provided in the data release [39].

The machine learning algorithms are compared to an equivalent matched-filter search, shown in purple in Fig. 5. All searches perform equally well for FARs $\geq 10^5$ per month. For smaller FARs the matched-filter search is sensitive to sources which are up to 200 Mpc farther away. The deep learning search retains at least 91.5% of the sensitivity compared to the matched-filter search at all FARs.

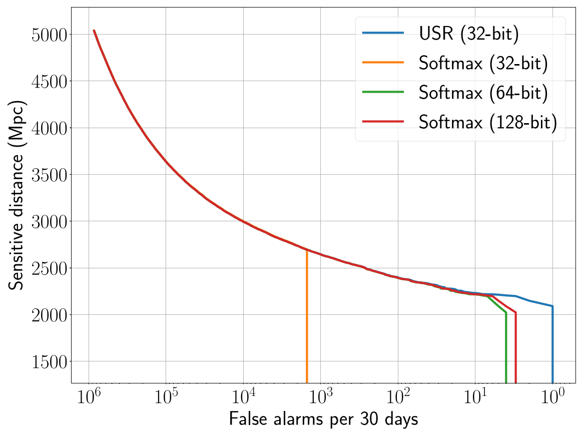

FIG. 6. Sensitivity of the "mean" run of the "fixed low" strategy using the Softmax layer of various floating point precisions as well as the USR. A similar behavior of USR performing at least as well as the Softmax with all precisions was observed in all 45 evaluated runs.

This result shows that with minimal modification to the architecture the original network from [23] achieves a sensitivity comparable to matched filtering for short BBH signals in simulated Gaussian noise even at FARs previously untested for this particular network architecture.

All results above were obtained on data generated and whitened by the exact same PSD used during training. For realistic searches, this assumption does not hold as the PSD in the detectors drifts over time [51]. To assert that the network does not depend strongly on the exact PSD used during training, we also evaluated the sensitivity using a version of the training-PSD scaled by a constant factor of 1.05 in all frequency bins. This reduced the sensitive distance at all FARs by roughly $1/\sqrt{1.05}$, in agreement with the theoretical expectation.

We also tested the effects of using a realistic variation of the PSD. To determine the variation, we used 20 PSDs derived on real data from the O3a observing run [52], chose one as reference, and divided it by all the others. We then determined the PSD ratio that had the largest mean deviation from unity and multiplied it with the training PSD to obtain a realistically varied PSD. Generating and whitening the data by this varied PSD reduces the sensitivity at FARs below 100 per month to around the same level as is observed for the scaled PSD.

Finally, we tested whitening by a different PSD than the one used for generating the data. For this purpose we used a second PSD variation to whiten the data generated by the first PSD variation described above. To obtain the second PSD, we used the PSD ratio that had the smallest, instead of the largest, mean deviation from unity. This simulates a worst-case scenario for realistic PSD variations. We find that the sensitivity drops by as much as 20% compared to using the correct PSD for whitening. This analysis shows that the network is robust against differences between the training PSD and the PSD of the analyzed data, as long as the correct PSD is used for whitening.

B. Training strategies

We trained 50 networks for every training strategy discussed in Sec. II C. Figure 7 shows the evolution of the efficiency at SNR 9 for every training strategy. The networks use a Softmax activation on the final layer. While we also monitor different SNRs, it is this region we are most interested in for three reasons. The first is practical in nature. Above an SNR of 9 networks trained with almost all training strategies recover close to 100% of the signals. It is, therefore, impossible to separate them by efficiency. Second, most GWs are expected to be detected at low SNRs [53]. Hence, efficiency at low SNRs is most important. Lastly, SNR 8 is often used as a threshold above which matched-filter searches can comfortably detect most signals (compare Fig. 5). By probing the efficiency close to

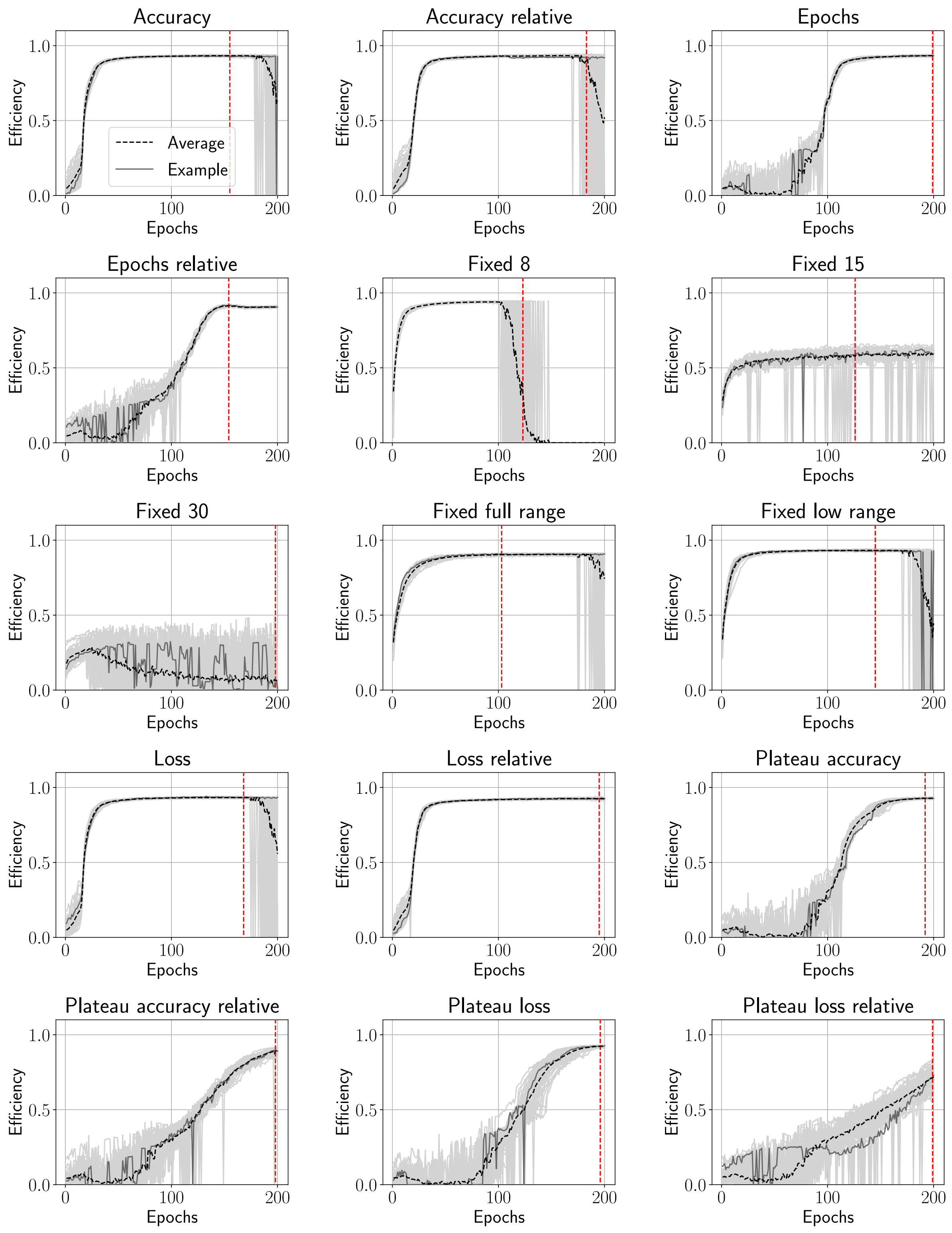

FIG. 7. The efficiency for all 15 tested training strategies as a function of the training epochs at SNR 9 and a FAP of 10^{-4}. The light grey curves show the efficiency for the 50 independent initialized training runs. The black dashed line shows the average over these individual runs. We highlight the evolution of a single run in dark grey. The vertical, red, dashed line signifies the epoch with the largest efficiency. We choose 3 networks at this epoch for which to calculate the sensitive volume.

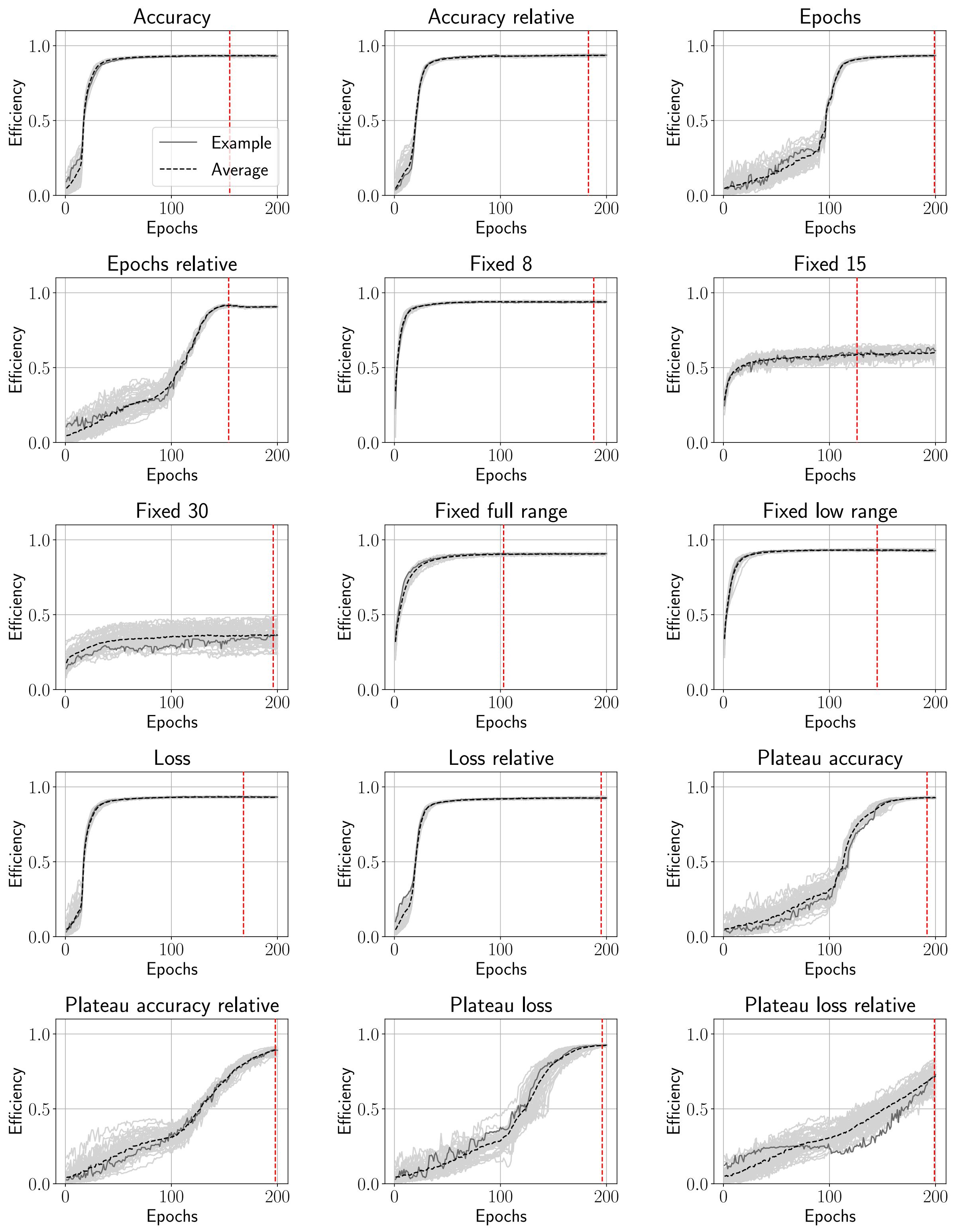

FIG. 8. The efficiency for all 15 tested training strategies as a function of the training epochs at SNR 9 and a FAP of 10^{-4}. The light grey curves show the efficiency for the 50 independent initialized training runs. The black dashed line shows the average over these individual runs. We highlight the evolution of a single run in dark grey. The vertical, red, dashed line signifies the epoch with the largest efficiency. We choose 3 networks at this epoch for which to calculate the sensitive volume. This figure shows the same networks as Fig. 7 with the USR modification applied. With this modification the efficiency stays >0 at all times.

this threshold we can get a sense of how well the search is doing overall.

Most training strategies do not have a major impact on the maximum efficiency. With the exception of training with a fixed SNR 15 and 30 all converged networks reach efficiencies of 90% ("Fixed full range") to 94% ("Fixed 8"). At SNR 6 the efficiency consistently drops to 24% ("Fixed full range") to 33% ("Loss relative") for all converged networks other than the above mentioned exceptions. Above an SNR of 12 the efficiencies reach 100% for all networks except "Fixed 15" and "Fixed 30." Those only achieve 100% efficiency at SNR 15 and 21 respectively.

The relative plateau strategies did not manage to converge within the first 200 epochs. For these, we have extended the training length to 400 epochs, which has allowed these runs to converge. They have reached comparable efficiencies to those mentioned above.

The main difference between all runs is the number of epochs required to reach a converged state. One can see that strategies which supply low SNR signals earlier reach their maximal efficiency earlier. This is especially emphasized with the runs "Fixed 8," "Fixed full range," and "Fixed low range." The curriculum strategies that use the accuracy or loss as their condition also converge quickly. They too supply low SNR samples very early on, as the respective condition is fulfilled at each of the first few epochs. Waiting for a set number of epochs to pass hinders the ability of the network to see low SNR signals early on and, therefore, takes more time to converge. The slowest converging strategies wait for the loss or accuracy to stop improving. They effectively have to wait at least 6 epochs before lowering the training range. Using a relative approach to lowering the SNR range further decreases the speed at which low SNRs are explored. In the most extreme cases the networks do not converge within the given number of epochs.

Finally, some training strategies become unstable toward the end. All of these unstable strategies converge relatively fast. This suggests that the longer one trains a converged network the more likely the efficiency is to collapse. We, therefore, expect that strategies where the efficiency did not collapse during the first 200 epochs would see a similar problem during later epochs.

The breakdown of the efficiency was resolved by the USR modification in Sec. III A. Figure 8 shows the evolution of the efficiency at SNR 9 when this fix is applied. We find that the drops to zero efficiency are removed but the qualitative features of the efficiency curves stay the same.

All efficiency plots were generated from the TensorFlow version of the networks. When training with PyTorch we found the results virtually indistinguishable from the TensorFlow version.

We repeated our tests on networks with different capacities, although this was not the main focus of the present work, to ensure that our findings are robust against a few specific architecture changes. We found no significant differences in the final efficiency, although the speed of training convergence varied. Such studies are left to future work.

IV. CONCLUSIONS

In this paper, we revisited the first deep learning GW search algorithms and compared them directly to a matched-filter search. We showed that for the considered parameter space and for a single detector the networks retain performance closely following matched filtering even on long duration continuous datasets and when considering FAR thresholds down to once per month. While there are now more sophisticated deep learning algorithms available that enhance the capabilities of the first proofs of concept, we think that there is still a lot to be learned from these first steps.

Our initial focus was the optimization of the data presentation to these networks. Two kinds of training strategies were previously explored; curriculum learning, where training samples become more difficult to classify as training continues, and fixed interval training, where the complexity of the training set stays constant.

We found that the particular strategy is of little importance to the eventual performance of the network. It depends a lot more on the presence of sufficiently complex samples in the training set. In particular, we found that the networks are able to generalize low SNR signals to high SNR ones but not vice versa.

On the other hand, the training strategy does have an impact on the time it takes the network to converge. Since high SNR examples are not as important to the performance of the network, strategies that provide low SNR samples earlier converge faster. In conclusion, we recommend training deep learning search algorithms on a fixed range of low SNR signals.

We use efficiency as our metric of performance during training. As this statistic has been used in previous publications, it allows us to verify that we have converged to the expected performance.

The efficiency at FAPs $\leq 10^{-4}$ dropped to zero when networks were trained for extended periods of time. This was unexpected and limited our ability to test the search.

We found the drops in efficiency to be caused by numerical instabilities in the final activation function of the networks. By removing the Softmax activation on the final layer and imposing thresholds directly on the linear output of the network, we were able to lift the limitations on the testable FAPs. This USR modification has proven to be simple and effective, as no retraining of the networks is required and virtually unlimited low FAPs can be tested.

To compare the deep learning based searches to an equivalent matched-filter search we calculated the sensitive volumes as functions of the FARs on a month of simulated data. We found that the machine learning algorithm is able to closely follow the performance of the traditional algorithm even down to FARs of 1 per month, when USR is used.

The results given here are limited to a single detector, Gaussian noise and signals from BBHs, which are relatively short in duration and comparatively simple to detect with existing methods. Parts of the parameter space, like the inclusion of higher-order modes [15], eccentricity [17], or precession [16], where current searches are computationally limited, are not yet included. However, it is expected that neural networks may generalize efficiently to these more difficult signals. Deep learning detection algorithms for spinning black holes with precession were recently explored for the first time by [49]. There is also ongoing work to construct neural network searches targeting long duration signals [25,26,47,54]. Considering real noise may enable deep learning algorithms to outperform matched filtering, which is only known to be optimal for stationary Gaussian noise. Multiple studies have shown that neural networks adapt well to nonstationary noise contaminated with glitches [22,47,49,55].

ACKNOWLEDGMENTS

We acknowledge the Max Planck Gesellschaft and the Atlas cluster computing team at Albert-Einstein Institut (AEI) Hannover for support, as well as the ARA cluster team at the URZ Jena. F. O. was supported by the Max Planck Society's Independent Research Group Programme. O. Z. thanks the Carl Zeiss Foundation for the financial support within the scope of the program line "Breakthroughs".

APPENDIX: EFFICIENCY CURVE EXAMPLES

This appendix provides examples of the usefulness of USR for various training strategies explored in this paper. The plots show the efficiency as a function of training epochs at 4 distinct SNRs with and without the application of the USR. For both shown examples the USR manages to remove the efficiency breakdown entirely.

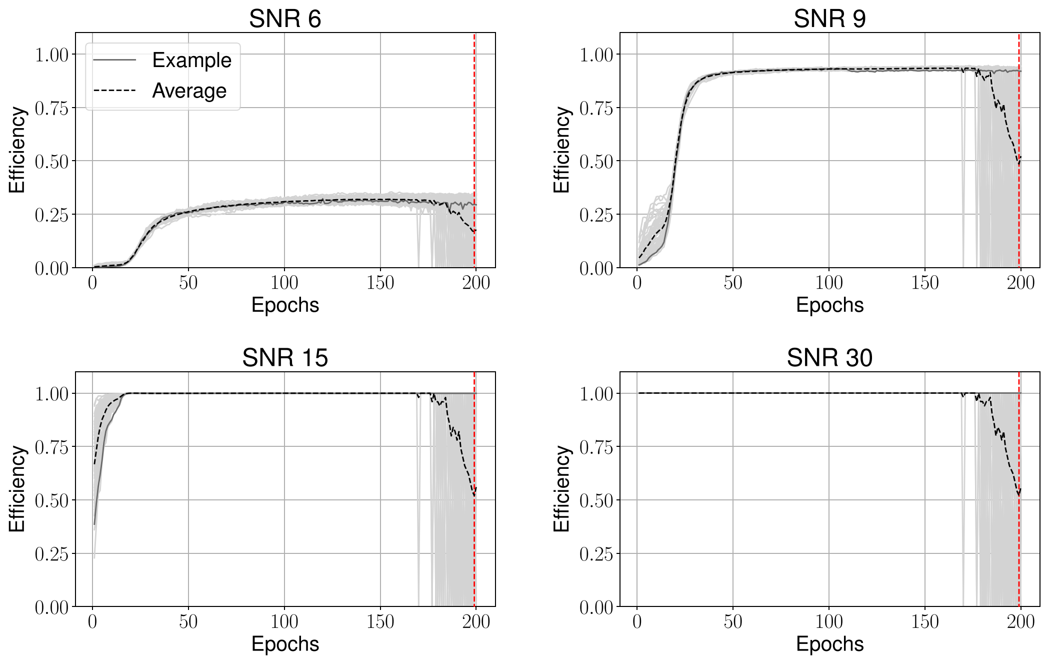

FIG. 9. Efficiency evolution of the "Accuracy relative" strategy using the Softmax output as a ranking statistic.

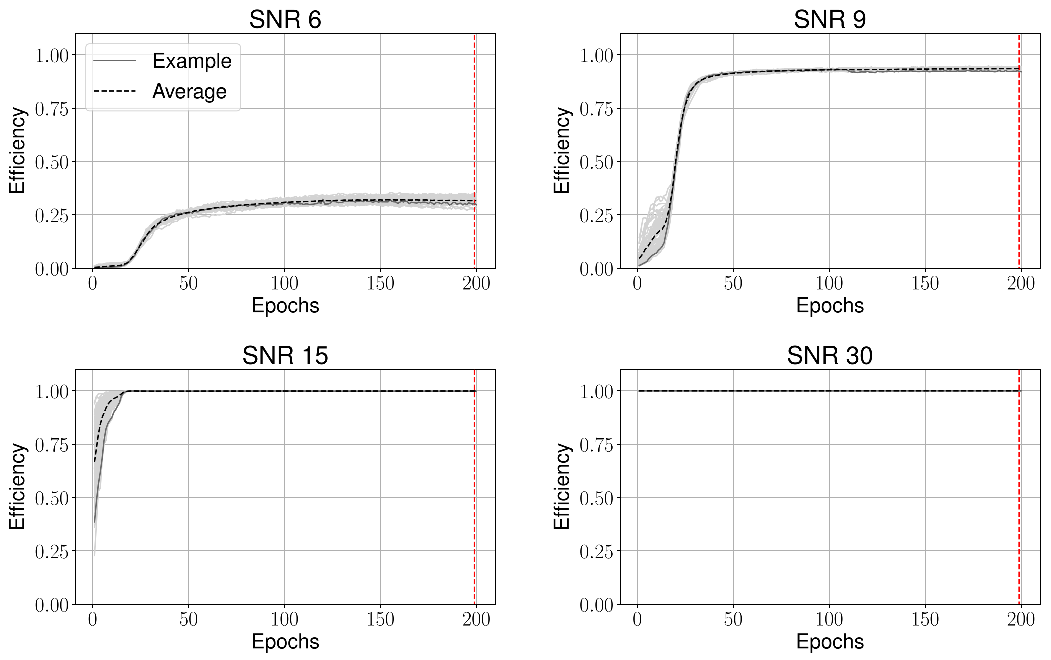

FIG. 10. Efficiency evolution of the "Accuracy relative" strategy using the USR modification.

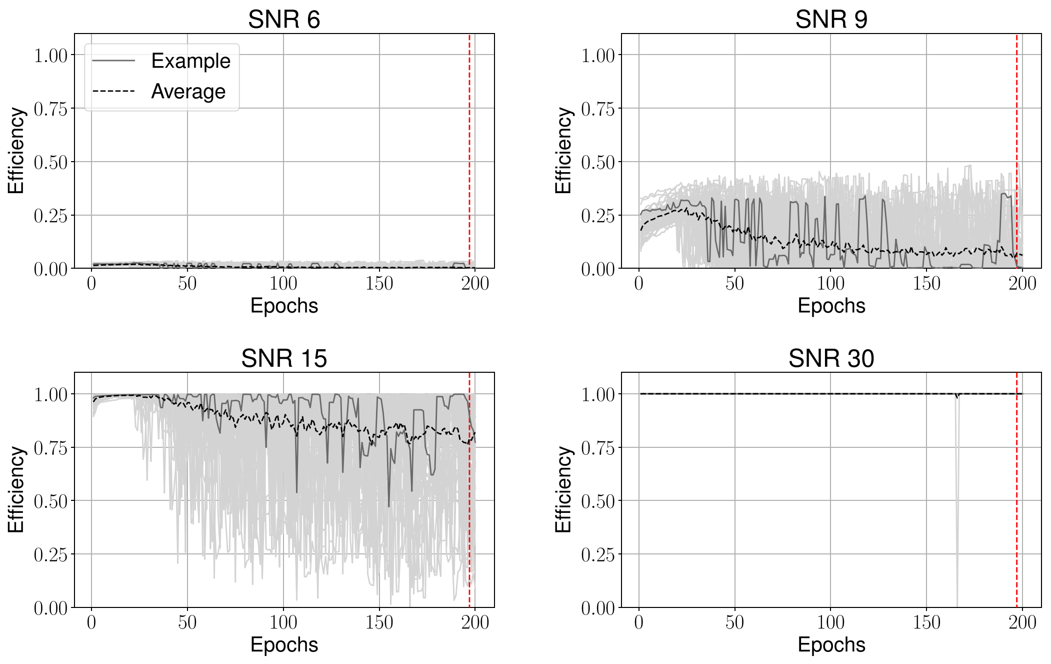

FIG. 11. Efficiency evolution of the "Fixed 30" strategy using the Softmax output as a ranking statistic.

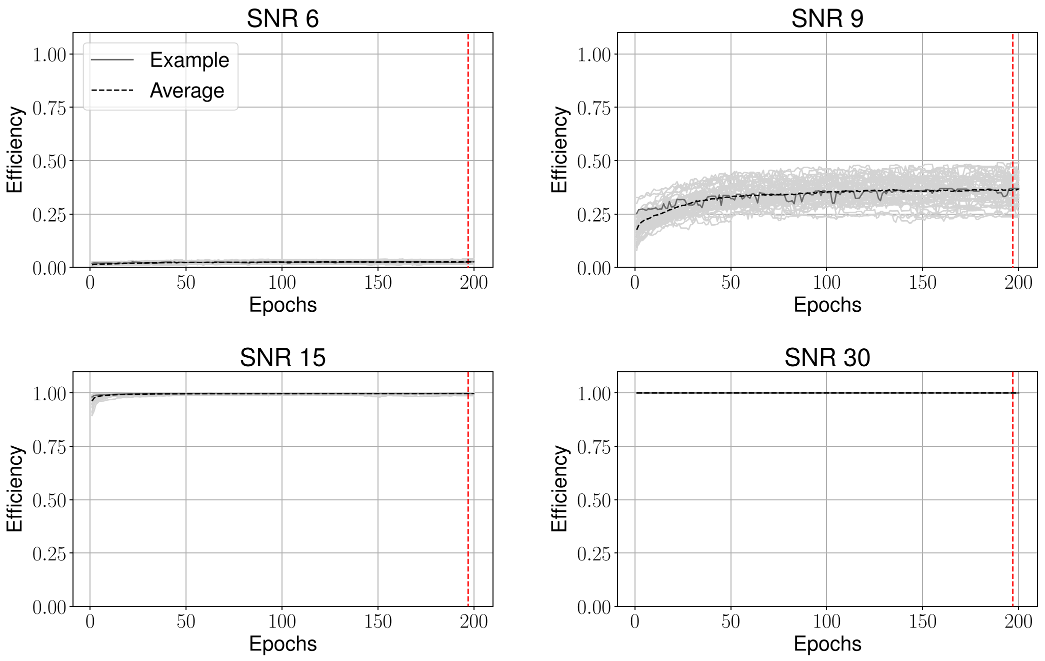

FIG. 12. Efficiency evolution of the "Fixed 30" strategy using the USR modification.

[1] B. P. Abbott *et al.* (LIGO Scientific Collaboration and Virgo Collaboration), Phys. Rev. Lett. **116**, 061102 (2016).
[2] R. Abbott *et al.* (LIGO Scientific, Virgo Collaborations), Phys. Rev. X **11**, 021053 (2021).
[3] A. H. Nitz, C. D. Capano, S. Kumar, Y.-F. Wang, S. Kastha, M. Schäfer, R. Dhurkunde, and M. Cabero, Astrophys. J. **922**, 76 (2021).
[4] B. P. e. a. Abbott (LIGO Scientific Collaboration and Virgo Collaboration), Phys. Rev. Lett. **119**, 161101 (2017).
[5] A. Goldstein *et al.*, Astrophys. J. **848**, L14 (2017).
[6] V. Savchenko *et al.*, Astrophys. J. **848**, L15 (2017).
[7] M. Soares-Santos *et al.* (DES, Dark Energy Camera GW-EM Collaborations), Astrophys. J. Lett. **848**, L16 (2017).
[8] B. P. Abbott *et al.* (LIGO Scientific, Virgo, Fermi GBM, INTEGRAL, IceCube, AstroSat Cadmium Zinc Telluride Imager Team, IPN, Insight-Hxmt, ANTARES, Swift, AGILE Team, 1M2H Team, Dark Energy Camera GW-EM, DES, DLT40, GRAWITA, Fermi-LAT, ATCA, ASKAP, Las Cumbres Observatory Group, OzGrav, DWF (Deeper Wider Faster Program), AST3, CAASTRO, VINROUGE, MASTER, J-GEM, GROWTH, JAGWAR, CaltechNRAO, TTU-NRAO, NuSTAR, Pan-STARRS, MAXI Team, TZAC Consortium, KU, Nordic Optical Telescope, ePESSTO, GROND, Texas Tech University, SALT Group, TOROS, BOOTES, MWA, CALET, IKI-GW Follow-up, H.E.S.S., LOFAR, LWA, HAWC, Pierre Auger, ALMA, Euro VLBI Team, Pi of Sky, Chandra Team at McGill University, DFN, ATLAS Telescopes, High Time Resolution Universe Survey, RIMAS, RATIR, SKA South Africa/MeerKAT Collaborations), Astrophys. J. Lett. **848**, L12 (2017).
[9] L. S. Collaboration and V. Collaboration, Online pipelines (2018), https://emfollow.docs.ligo.org/userguide/analysis/searches.html.
[10] S. Sachdev *et al.*, The gstlal search analysis methods for compact binary mergers in Advanced LIGO's second and Advanced Virgo's first observing runs, arXiv:1901.08580.
[11] T. Adams, D. Buskulic, V. Germain, G. M. Guidi, F. Marion, M. Montani, B. Mours, F. Piergiovanni, and G. Wang, Classical Quantum Gravity **33**, 175012 (2016).
[12] A. H. Nitz, T. Dal Canton, D. Davis, and S. Reyes, Phys. Rev. D **98**, 024050 (2018).
[13] S. Hooper, S. K. Chung, J. Luan, D. Blair, Y. Chen, and L. Wen, Phys. Rev. D **86**, 024012 (2012).
[14] B. Allen, W. G. Anderson, P. R. Brady, D. A. Brown, and J. D. E. Creighton, Phys. Rev. D **85**, 122006 (2012).
[15] I. Harry, J. C. Bustillo, and A. Nitz, Phys. Rev. D **97**, 023004 (2018).
[16] I. Harry, S. Privitera, A. Bohé, and A. Buonanno, Phys. Rev. D **94**, 024012 (2016).
[17] A. H. Nitz, A. Lenon, and D. A. Brown, Astrophys. J. **890**, 1 (2020).
[18] S. Klimenko, S. Mohanty, M. Rakhmanov, and G. Mitselmakher, Phys. Rev. D **72**, 122002 (2005).
[19] S. Klimenko, G. Vedovato, M. Drago, F. Salemi, V. Tiwari, G. A. Prodi, C. Lazzaro, K. Ackley, S. Tiwari, C. F. Da

Silva, and G. Mitselmakher, Phys. Rev. D **93**, 042004 (2016).
[20] R. Lynch, S. Vitale, R. Essick, E. Katsavounidis, and F. Robinet, Phys. Rev. D **95**, 104046 (2017).
[21] D. George and E. A. Huerta, Phys. Rev. D **97**, 044039 (2018).
[22] D. George and E. Huerta, Phys. Lett. B **778**, 64 (2018).
[23] H. Gabbard, M. Williams, F. Hayes, and C. Messenger, Phys. Rev. Lett. **120**, 141103 (2018).
[24] C. Dreissigacker, R. Sharma, C. Messenger, R. Zhao, and R. Prix, Phys. Rev. D **100**, 044009 (2019).
[25] P. G. Krastev, Phys. Lett. B **803**, 135330 (2020).
[26] M. B. Schäfer, F. Ohme, and A. H. Nitz, Phys. Rev. D **102**, 063015 (2020).
[27] E. Cuoco *et al.*, Mach. Learn. Sci. Technol. **2**, 011002 (2020).
[28] E. A. Huerta and Z. Zhao, arXiv:2105.06479.
[29] H. Xia, L. Shao, J. Zhao, and Z. Cao, Phys. Rev. D **103**, 024040 (2021).
[30] T. D. Gebhard, N. Kilbertus, I. Harry, and B. Schölkopf, Phys. Rev. D **100**, 063015 (2019).
[31] A. J. K. Chua and M. Vallisneri, Phys. Rev. Lett. **124**, 041102 (2020).
[32] H. Gabbard, C. Messenger, I. S. Heng, F. Tonolini, and R. Murray-Smith, arXiv:1909.06296.
[33] S. R. Green, C. Simpson, and J. Gair, Phys. Rev. D **102**, 104057 (2020).
[34] M. J. Williams, J. Veitch, and C. Messenger, Phys. Rev. D **103**, 103006 (2021).
[35] S. Schmidt, M. Breschi, R. Gamba, G. Pagano, P. Rettegno, G. Riemenschneider, S. Bernuzzi, A. Nagar, and W. Del Pozzo, Phys. Rev. D **103**, 043020 (2021).
[36] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09 (Association for Computing Machinery, New York, NY, USA, 2009), p. 41–48, 10.1145/1553374.1553380.
[37] M. Abadi *et al.*, TensorFlow: Large-scale machine learning on heterogeneous systems, (2015), software available from https://www.tensorflow.org/.
[38] A. Paszke *et al.*, *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., New York City, 2019), pp. 8024–8035.
[39] M. B. Schäfer, O. Zelenka, A. H. Nitz, F. Ohme, and B. Brügmann, Training Strategies for Deep Learning Gravitational-Wave Searches, https://github.com/gwastro/ml-training-strategies (2021).
[40] S. Ioffe and C. Szegedy, arXiv:1502.03167.
[41] R. Abbott *et al.* (LIGO Scientific, Virgo Collaborations), Astrophys. J. **915**, 86 (2021).
[42] B. P. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), Phys. Rev. D **93**, 042005 (2016).
[43] J. Aasi *et al.*, Classical Quantum Gravity **32**, 074001 (2015).
[44] C. Devine, Z. B. Etienne, and S. T. McWilliams, Classical Quantum Gravity **33**, 125025 (2016).
[45] A. Bohé *et al.*, Phys. Rev. D **95**, 044028 (2017).
[46] L. S. Collaboration, LIGO Algorithm Library—LALSuite, free software (GPL), 10.7935/GT1W-FZ16 (2018).
[47] W. Wei and E. A. Huerta, Phys. Lett. B **816**, 136185 (2021).
[48] D. P. Kingma and J. Ba, arXiv:1412.6980.
[49] W. Wei, A. Khan, E. Huerta, X. Huang, and M. Tian, Phys. Lett. B **812**, 136029 (2021).
[50] A. H. Nitz, I. W. Harry, J. L. Willis, C. M. Biwer, D. A. Brown, L. P. Pekowsky, T. Dal Canton, A. R. Williamson, T. Dent, C. D. Capano, T. J. Massinger, A. K. Lenon, A. B. Nielsen, and M. Cabero, PyCBC Software, https://github.com/gwastro/pycbc (2018).
[51] R. Abbott *et al.* (LIGO Scientific, VIRGO, KAGRA Collaborations), arXiv:2111.03606.
[52] R. Abbott *et al.* (LIGO Scientific, Virgo Collaborations), SoftwareX **13**, 100658 (2021).
[53] B. F. Schutz, Classical Quantum Gravity **28**, 125023 (2011).
[54] C. Dreissigacker and R. Prix, Phys. Rev. D **102**, 022005 (2020).
[55] J. Yan, M. Avagyan, R. E. Colgan, D. Veske, I. Bartos, J. Wright, Z. Márka, and S. Márka, arXiv:2104.03961.

Correction: The copyright license statement was presented incorrectly and has been fixed.