Improving parametric neural networks for high-energy physics (and beyond)

MACHINE LEARNING Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
27 April 2022

REVISED
29 August 2022

ACCEPTED FOR PUBLICATION
12 September 2022

PUBLISHED
5 October 2022

PAPER

Improving parametric neural networks for high-energy physics (and beyond)

Luca Anzalone[1,3,*], Tommaso Diotalevi[1,2,3] and Daniele Bonacorsi[1,3]

1 Department of Physics and Astronomy (DIFA), University of Bologna, Bologna, Italy
2 European Organization for Nuclear Research (CERN), Geneva, Switzerland
3 Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Bologna, Italy
* Author to whom any correspondence should be addressed.

E-mail: luca.anzalone2@unibo.it

Keywords: parametric neural networks, conditioning mechanism, signal-background classification, high-energy physics

Abstract

Signal-background classification is a central problem in high-energy physics, that plays a major role for the discovery of new fundamental particles. A recent method—the parametric neural network (pNN)—leverages multiple signal mass hypotheses as an additional input feature to effectively replace a whole set of individual classifiers, each providing (in principle) the best response for the corresponding mass hypothesis. In this work we aim at deepening the understanding of pNNs in light of real-world usage. We discovered several peculiarities of parametric networks, providing intuition, metrics, and guidelines to them. We further propose an alternative parametrization scheme, resulting in a new parametrized neural network architecture: the AffinePNN; along with many other generally applicable improvements, like the *balanced training* procedure. Finally, we extensively and empirically evaluate our models on the HEPMASS dataset, along its *imbalanced* version (called HEPMASS-IMB) we provide here for the first time, to further validate our approach. Provided results are in terms of the impact of the proposed design decisions, classification performance, and interpolation capability, as well.

1. Introduction

Selecting events that contain interesting processes is a fundamental requirement of high-energy physics (HEP) experiments and one of the most established areas in advanced computing techniques i.e. machine and deep learning [1]. Physicists are interested in rare events (yield by the collision of known particles), following their theoretical assumptions, measuring the fraction of events that contain a specific decay channel. These rare events—the so-called *signal*—must be separated out from the *background*, i.e. anything else originating from already known processes. The usual way of doing that relies on building an event selection algorithm, estimating its efficiency on selecting signal and rejecting background, and measuring the count of events passing it. Being able to effectively separate background events from the signal is a central problem in HEP, that can help the discovery of new fundamental particles with further analysis. Compared to traditional, expert-designed algorithms based on single cuts driven by physical considerations, Machine Learning algorithms, e.g. (boosted) decision trees [2, 3] and neural networks (NNs) [1, 4], have two main advantages: first of all, they are usually able to deliver an higher selection efficiency; secondly, they save effort, by replacing an HEP-specific manual algorithm solution with an application of a general method, generally stolen from an AI research and adopted also in other fields of study. Since, prior to the analysis, a signal event is not known, these algorithms are trained on synthetic data obtained through expensive Monte-Carlo simulations, trying to mimic data as coming from real-world collisions of particles in accelerators, like the CERN Large Hadron Collider [5].

In addition to the above mentioned challenges, some searches do not even have a clear theoretical prediction on the exact mass values for such potential new particles: for instance, several new physics analyses prefer to keep such mass values floating, generating M different Monte-Carlo samples at different mass hypotheses and splitting the analysis in M different searches, each one with a fixed mass value [6, 7].

Before the introduction of parametrized neural networks (pNNs) [8] (described in section 2.1), therefore, the canonical approach for signal-background classification consisted of training a set of *individual and independent classifiers* [4], each of them trained on a specific signal mass hypothesis $m_i \in \mathcal{M}$ (thus, on a subset of the available data, somehow). The benefit of individual classifiers is that each of them should maximize the separation of the two classes for a single mass hypothesis. The major drawback is that researchers have to design, train, tune, evaluate, and maintain $|\mathcal{M}|$ of such classifiers, one for each mass[4] ($m_i \in \mathcal{M}$), often with different hyperparameters, whose amount can easily be in the order of tens: this can become quickly unfeasible. The other issue is about classification performance. Specifically talking about neural networks, they greatly benefit from the *sharing of weights* and *distributed representation* [1] for improved accuracy, reduced training time, and thus increased data efficiency. pNNs aim at mitigating those issues by means of an additional input (the signal's generating mass), while also bringing additional benefits for the sake of more effective research.

The overall contributions of our work are summarized as follows:

- We propose a novel and more challenging benchmark dataset called `HEPMASS-IMB` [9] (section 3.2), to overcome the simplicity of `HEPMASS` [10], trying to emulate real-world scenarios more closely.
- We developed a novel parametrization scheme: the *affine parametric neural network* (section 4.1).
- We describe and empirically study several design decisions for building effective pNNs (section 4). In this regard, we address the issue of how to assign (or distribute) the mass feature for the background events (section 4.2), as well as providing a novel *balanced training* procedure (section 4.3).
- We attempt a first characterization of the *properties* that parametric networks have (section 5).
- Regarding interpolation (section 5.1), we study it in depth: showing a failure case, and providing guidelines for its assessment.

2. Related work

2.1. Parametric neural networks

A pNN [6–8] is a neural network architecture that leverages an additional input (in our case the *mass* of the hypothetical particle) to replace many individual classifiers, and potentially even improve their classification performance. Let be x the input features, m the generated mass of the signal (or the signal mass hypotheses), and θ a set of learnable weights (or parameters). A pNN can be denoted as $f_\theta(x, m)$, i.e. as a learnable function of both the input features x and the *mass feature* m. A canonical neural network, instead, would be denoted as $f_\theta(x)$, depending only on the input features. The Baldi's pNN [8] first concatenates x with m, then applies five dense layers each with 500 units and activated by ReLU, after that a final dense layer with sigmoid non-linearity outputs the predicted class label. Such architecture results in about 1M learnable parameters.

Indeed the idea of 'parametric' is not new, as in other fields of machine learning, like *imitation learning* [11], *multi-task and meta-learning* [12], *unsupervised reinforcement learning* [13], and *deep generative models* [14], is commonly called 'conditioning'. Here the general idea is to condition the learning (i.e. output) of a neural network on some additional representation z, in order to let the network's output change as z varies. The vector z—called the *task representation*—can take various forms: ranging from a one-hot encoding to a dense embedding, or be a single discrete or continuous variable as well. In our case, the mass feature (i.e. the task representation) is a single scalar m belonging to a finite set $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}$ of mass hypotheses about the signal process we are interested in.

This idea, whether called conditioning or parametrization, is promising in HEP since it may enable to replace (potentially many) individual classifiers with a unique classifier trained on all mass hypotheses. Thus, leveraging the *sharing of weights* for more efficient learning, and *distributed representations* shared among mass hypotheses for improved predictions, which we also found to be beneficial in *low-data regimes*: i.e. when some of the data corresponding to certain masses, is *imbalanced* compared to the most

[4] In this work, the term *mass* refers to the additional input of the parametric network, m: also called *mass feature* by Baldi *et al* [8]. By m_i we denote a generic value of the signal generating mass (or signal mass hypotheses), whose set of values is represented by $\mathcal{M}$. In general, we do not use the term mass to imply the reconstructed (or invariant) mass of the considered particle decay, but at most its theoretical parameter, m_X.

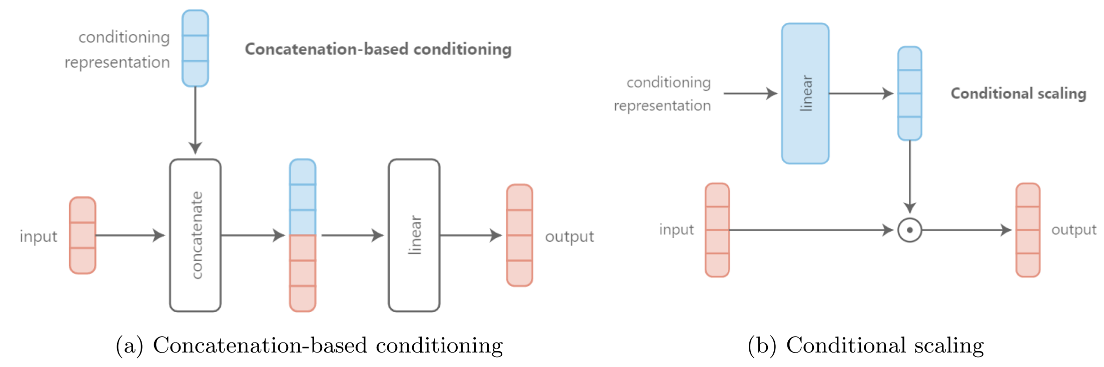

conditioning representation
Concatenation-based conditioning
input
concatenate
linear
output
(a) Concatenation-based conditioning
conditioning representation
linear
Conditional scaling
input
output
(b) Conditional scaling

Figure 1. Two popular conditioning mechanism, that tuned out to be complementary. Reproduced from [15]. CC BY 4.0.

representative masses[5]. The authors also claim that, since the additional input would define a *smoothly varying learning task*, a pNN would be able to *smoothly interpolate* between such learning tasks. Ideally, this means that a pNN would be able to generalize (correctly classify events) beyond the mass hypotheses it was trained on, thanks to the additional mass feature.

In this case, our supervised dataset $\mathcal{D}$ have the form $\{(x,m,y)_i\}_{i=1}^N$, in which we have input features x (i.e. the variables associated to each event), their mass hypothesis m, and the target class labels y we aim to predict. For each signal mass hypothesis, $m_i \in \mathcal{M}$, we can *slice* our dataset such that only the features and targets corresponding to mass m_i are retained, i.e. $\mathcal{D}_{m_i} = \{(x,y)_j \ : m_j = m_i, \forall j = 1,\ldots,N\}$, where N is the total number of events[6]. In this way, we obtain $|\mathcal{M}|$ datasets for which each of them can be used to train an individual classifier (but also to evaluate our pNN at single mass points). The pNN can replace all of them, and if trained on a subset of the mass hypotheses, $\bar{\mathcal{M}} \subset \mathcal{M}$, it is, in principle, able to automatically account for the missing masses ($\tilde{m}_j \in \mathcal{M} \setminus \bar{\mathcal{M}}$) thanks to the interpolation capability (discussed in section 5.1), which should work on novel intermediate mass points as well.

2.2. Conditioning mechanisms

The authors of the original pNN [8] utilize a simple conditioning mechanism: they just *concatenate* the features with the mass (or task representation, in general) obtaining a new set of features, $\bar{x} = [x, m]$, going back to a standard feed-forward neural network formulation, i.e. $f_\theta(\bar{x})$, that learns from an *extended* set of features $\bar{x}$. Indeed, many arbitrarily-complex conditioning mechanisms exists, two of them (figure 1) are yet simple but powerful [15]:

- **Concatenation-based conditioning.** The task or conditioning representation m (our mass) is first concatenated along the last dimension (axis) of the input features x, and then the result is passed through a linear layer. Notice that in the original pNN, the linear layer after concatenation is missing.
- **Conditional scaling.** A linear layer first maps the conditioning representation m to a scaling vector s, to which follows an element-wise multiplication (Hadamard product) with the input features x, i.e. $x \odot s$.

These two conditioning mechanisms are widely applicable, although it is not yet clear in which case one mechanism is preferable to the other(s). Anyway, these two mechanisms can be both combined into a third one[7]: a *conditional affine transformation* [15], which motivates our new parametric architecture (refer to section 4.1).

3. Datasets

In this section, we provide details about the two datasets we used to conduct our study.

[5] This is often the case in practice, since when doing MC simulations to reproduce the data, some kind of events (for certain masses) are more frequently generated (thus being less rare) than others, naturally resulting in an *imbalanced dataset*, that is imbalanced not necessarily with respect the class labels (since the background class is independently produced) but with respect the *mass feature*.

[6] From a ML jargon perspective we can equivalently refer to events as (ex)*samples* or *datapoints*, of some dataset.

[7] An affine transformation, i.e. $y = Ax + b$, does not involve concatenation directly: turns out that concatenation-based conditioning is equivalent to *conditional biasing*, in which the task representation is first mapped to a bias vector that is then added to the input, element-wise, effectively replacing a concatenation operation. Further details in [15].

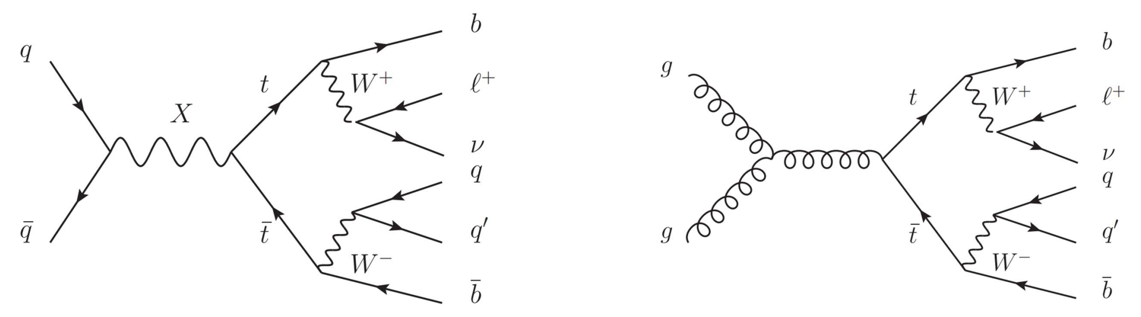


Figure 2. Feynman diagrams depicting the hypothetical particle decay: $X \to t\bar{t} \to W^+bW^-\bar{b}$. Reproduced from [8]. CC BY 4.0.

3.1. HEPMASS

The HEPMASS dataset [8, 10] was utilized by the pNN's authors to demonstrate their novel idea. The dataset contains 7M training samples, and 3.5M test samples. The physical case under consideration is the search of a new particle X with an unknown mass. This particle decays into a $t\bar{t}$ pair, and the final state consists in the most probable decay product: $t\bar{t} \to W^+bW^-\bar{b} \to qq'bl\nu\bar{b}$. The dominant background considered for this specific signal is the standard model $t\bar{t}$ production, identical in the final state but different in kinematics due to the absence of the X resonance. The Feynman diagrams showing the signal and background processes are shown in figure 2. There are a total of 27 features (without considering the 28th *mass feature*) already normalized to have approximately zero-mean and unitary variance. Each datapoint $x^{(i)} \in \mathcal{D}$ can belong to either a signal process, with a mass hypotheses (in GeV) $m_X = \{500, 750, 1000, 1250, 1500\}$, or to a background process, where the mass feature is randomly sampled from m_X. Signal (background) samples are then labeled with class 1 (0). Moreover, the two classes are perfectly balanced, and also each $\mathcal{D}_{m_i}$ is *balanced*: containing the same amount of events for each $m_i \in m_X$. As discussed in the previous section, when training pNNs we want to pay attention to the balance of classes as well as the balanced of each m_i: this dataset avoids such issue. For further details about the data, refer to [8].

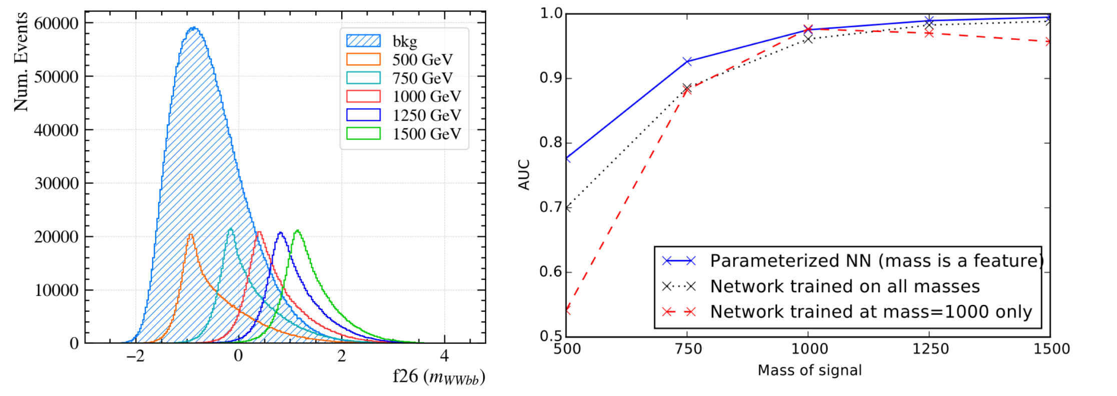
By studying the distribution of each feature we can deduce three major things:

(a) The background is unique and covers the signal, although partially (figure 3(a)).
(b) Consequently, the signal's events at $m_X = 500, 750$ GeV are the most difficult to separate out from the background, since the features distribution is mostly completely overlapped with the background's one. This explains why, in figure 3(b), the AUC is considerably lower at 500 GeV, while being almost perfect for 1500 GeV.
(c) By only considering some features (figure 4) a classifier (even simple) can easily tell which event belongs to the signal-class or not, thanks to these features being highly correlated with the class label: figure 5.

3.2. HEPMASS-IMB

Since the HEPMASS dataset is rather simple, leaving almost no room for improvement, we decided to *imbalance* the dataset by hand in order to being able to demonstrate novel methods for improving pNNs: we call this new dataset, derived from it, HEPMASS-IMB [9]. In particular the dataset is *doubly-imbalanced*: there is *class-imbalance* with respect to the class label, and *mass-imbalance* with respect the theoretical parameter (m_X), as well. A comparison between the two dataset is depicted in table 1.

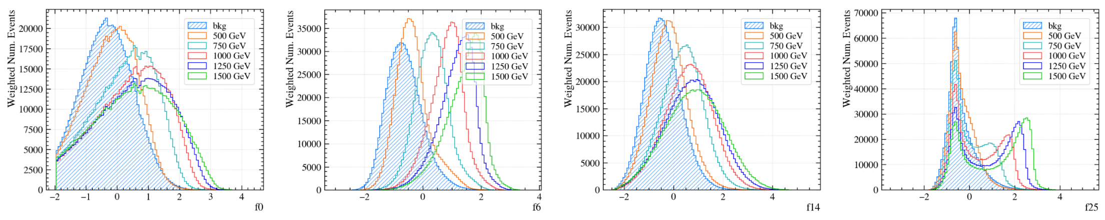
The way we imbalance the dataset is as follows. We first take all the background events (without any change), and sub-sample (without replacement) only the signal, differently at each $m_i \in \mathcal{M}$. In particular, we select: 350k (for $m_X = 500$), 140k (for $m_X = 750$), 35k (for $m_X = 1000$), 7k (for $m_X = 1250$), and lastly 2k events for $m_X = 1500$; for a total of almost 534k signal events. Indeed, we only imbalance the train-set of HEPMASS, leaving its test-set as it was provided by the authors [10]. In such way we are able to simulate a double imbalance of both class-labels and signal mass hypotheses (figure 6), that resembles more the imbalance found in real-world dataset of Monte-Carlo simulated particle decays.



(a) Distribution of the normalized feature `f26`: the whole background class is represented in blue, whereas each signal mass hypothesis with a different color.

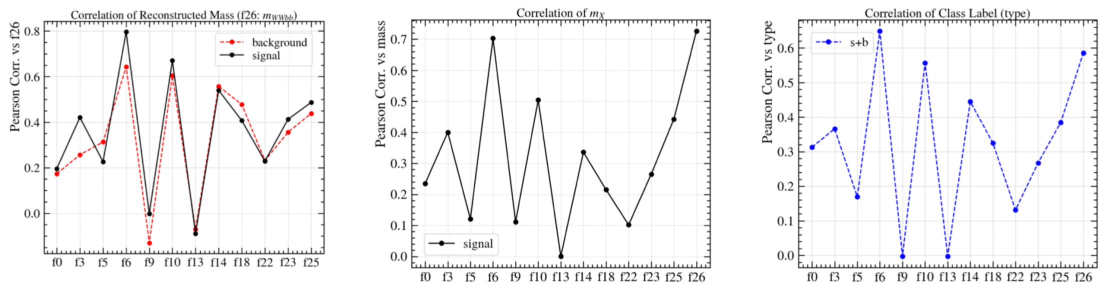
(b) Per-mass AUC (area under the curve) of the ROC (receiver operating characteristic) curve: figure from [8].

Figure 3. The plot (a) shows the *reconstructed mass* (m_{WWbb}) of the simulated decay ($X \to t\bar{t} \to W^+bW^-\bar{b}$), depicted in `HEPMASS`. We observe that the background spans the entire mass range, mostly overlapping the signal at $m_X = 500$ and 750 GeV. This fact also motivates why the authors' results (presented in plot (b), and also our own, exhibit a neat loss in AUC below 750 GeV, especially at 500 GeV. Reproduced from [8]. CC BY 4.0.



(a) Feature `f0` (b) Feature `f6` (c) Feature `f14` (d) Feature `f25`

Figure 4. Some features in which we can clearly observe how easily the distribution of each mass hypothesis drifts away from the background. In particular, the distribution of $m_X = 500$ almost completely overlaps with the background, each time telling us that these events are the most difficult to classify (as shown in figure 3(b)). The features shown here are all normalized.



(a) Correlation of both signal and background events with the reconstructed mass (m_{WWbb}) of the particle decay.

(b) Correlation of the signal events with the mass hypotheses ($\mathcal{M}$).

(c) Correlation of all the events with the class-label (`type`).

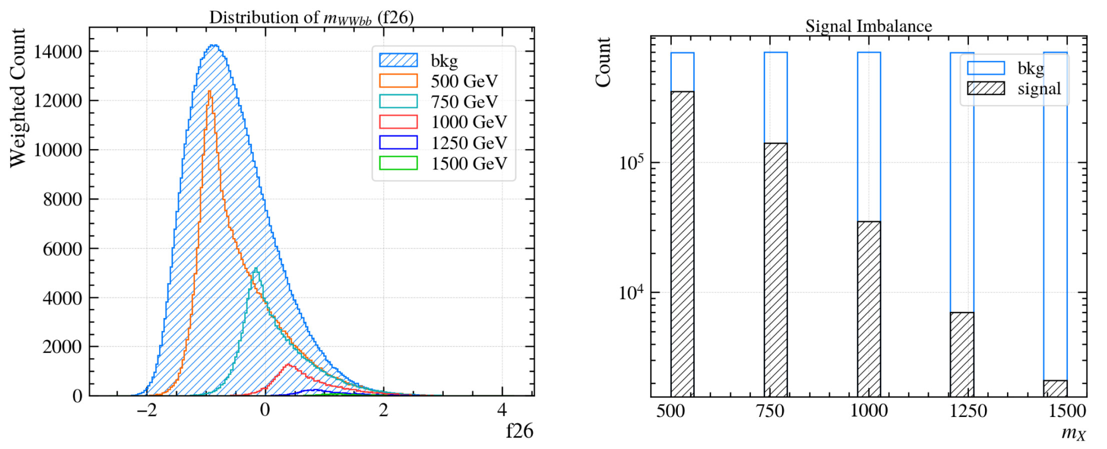
Figure 5. Pearson correlations of the training variables of `HEPMASS` (train-set).

4. Method

Having the extra *mass feature* as input gives us an additional degree of freedom for the design of classifiers. In this section we study different design decisions about *network architecture*, *background distribution*, and *training procedure*.

Table 1. Datasets comparison. As we can see, our modification of HEPMASS adds more difficulties to be addressed. This allowed us to discover many weaknesses of pNNs. (*) Design decisions about how to best distribute the mass of background events are depicted in section 4.2, going beyond what is fixed in the datasets.

	Dataset	
Characteristic	HEPMASS [10]	HEPMASS-IMB (our)
# Samples	7 M (+ 3.5 M test)	4 M (+ 3.5 M test)
Class imbalance (%)	None	13 (signal) / 87 (bkg)
Signal imbalance	None	up to 166×
Mass hypotheses	5 (equally spaced)	//
# Variables	27	//
Bkg distribution*	Identical (fixed)	//
Signal process	$X \to t\bar{t} \to W^+bW^-\bar{b} \to qq'b\ell\nu\bar{b}$	//
Bkg process	$t\bar{t} \to W^+bW^-\bar{b} \to qq'b\ell\nu\bar{b}$	//



(a) Resulting distribution of the reconstructed mass after having imbalanced HEPMASS. Compared to figure 3a, we can see how less the signal is: the background has been weighted by 1/5, for visualization purpose.

(b) Imbalance of all the $m_i \in \mathcal{M}$ signal mass hypotheses: log-scale. We can see that the imbalance ratio grows as m_i increases, for a maximum difference of few orders of magnitude.

Figure 6. Overall class and mass imbalance of HEPMASS-IMB.

4.1. The affine architecture

Baldi *et al* [8] utilized a regular *feed-forward* design for their parametric network, which we will refer to (vanilla) *pNN*, by just concatenating the mass feature m with the input features x right after the input layer. Here we propose a novel conditioning (or parametrization) scheme to better exploit m, that is based on the two conditioning mechanisms described in section 2.2, namely: *conditional scaling*, and *conditional biasing* (equivalent to concatenation-based conditioning).

We propose an *Affine Parametric Neural Network* (AffinePNN) architecture, that relies on multiple *affine-conditioning layers* (figure 7) instead of simply concatenating the mass at the beginning of the network. Such a layer takes two vectors h and m as input, where h can be the features x (if the layer is directly applied on the inputs) or the previous layer's output, and m is the mass feature. Assuming them to have dimensionality D_h and D_m (that in our case is just one), respectively, the layer applies an element-wise affine transformation (scaling and bias addition) on h, such that the output z is a function of m. Considering vectors at a generic index i of the input batch, we have:

$$z^{(i)} = h^{(i)} \odot s_\phi\left(m^{(i)}\right) + b_\psi\left(m^{(i)}\right), \tag{1}$$

where the dimensionality of $z^{(i)}$ is the same as $h^{(i)}$, i.e. D_h. The scaling and biasing operations are defined as linear functions over the mass[8]: $s_\phi = W_\phi m^{(i)}$, and $b_\psi = W_\psi m^{(i)}$, where the learned weight matrices W_ψ and

[8] In practice, these are implemented as two distinct Dense layers with linear activation.

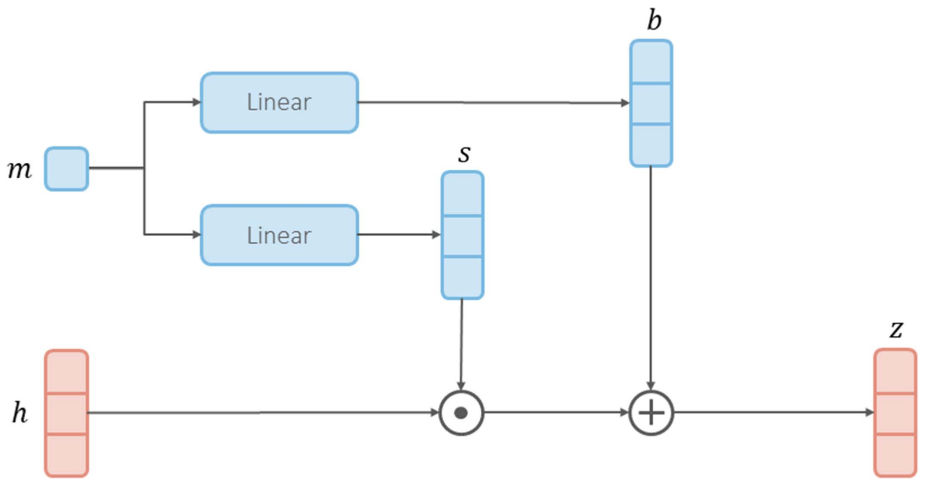


Figure 7. A schematic representation of the operations that build a single affine-conditioning layer. There are two inputs h and m. The dimensionality of m is expanded to match the dimensionality of h through linear combinations, that yield scaling (s) and biasing (b) vectors that are, respectively, multiplied and added to h in an element-wise fashion, resulting in the conditioned representation: z.

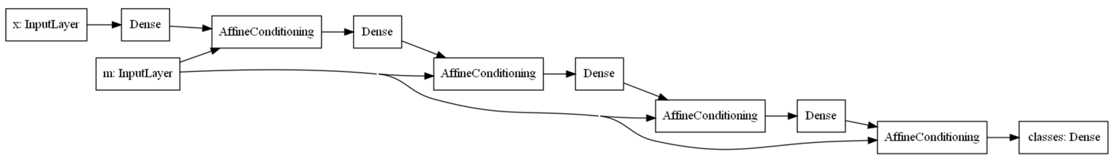


Figure 8. Diagram of the Affine architecture. In this picture the relation between the inputs and the layers can be better understood. At the beginning, the input features x are only fed to the first `Dense` layer; the mass feature m, instead, is only provided to each `AffineConditioning` layer. `Dropout` layer are omitted for clarity.

Table 2. Affine architecture. Each `AffineConditioning`(m, z_i) layer takes two inputs: the masses m input to the network, and the output z_i of the previous `Dense` layer. For better performance `Dropout` layers are also included, but only after each `AffineConditioning` layer. Basically, the architecture is made of four *blocks*, in which each block (`Dense` $\to$ `AffineConditioning` $\to$ `Dropout`) have, respectively, 300, 150, 100, and 50 hidden units, activated by ReLU non-linearity.

Layer	# Units
Input (x)	
Dense + ReLU	300
Input (m)	
AffineConditioning	300
Dropout	
Dense + ReLU	150
AffineConditioning	150
Dropout	
Dense + ReLU	100
AffineConditioning	100
Dropout	
Dense + ReLU	50
AffineConditioning	50
Dropout	
Dense + Sigmoid	1

W_ϕ have both shape $D_h \times D_m$, since the number of linear units have to match the dimensionality of $h^{(i)}$. An AffinePNN interleaves such layers with ReLU-activated[9] dense layers: in figure 8 and table 2, an overview of the architecture is shown. In principle, the affine layers can be further generalized by introducing non-linear activation functions (f and g) on the scaling s_ϕ, and biasing b_ψ, such that: $z = h \odot f\big(s_\phi(m)\big) + g\big(b_\psi(m)\big)$.

[9] $\mathrm{ReLU}(x) = \max(0, x)$, applied element-wise as usual.

In our preliminary experiments, we also evaluated some modifications of the network design shown in table 2. In particular, we tested various combinations of both activation function and weight initializer, finding that the ReLU activation paired with the default initialization scheme achieves the best performance: refer to section 6.3 for more details about the hyper-parameters. The last variation we tried, was the application of *batch normalization* [16] after each affine-conditioning layer: the resulting trained network had similar classification performance, but at the cost of a slightly longer training (due to the overhead of the additional operations). One possible explanation is that since the network is not so deep (there are just four hidden dense layers), the gradient flow is not affected by either vanishing or exploding gradients, thus making batch normalization not fully necessary. This fact was confirmed by tracking the magnitude (l_2-norm) of the gradients during training. The same modifications were also tried on the pNN architecture, showing a similar behavior. Indeed, the choice of architecture-related hyper-parameters, like the activation function, weight initialization, number of layers (or blocks) and number of units, is usually dataset and problem dependent: what we found to be working here, is not said to be equally good in other circumstances.

4.2. Background's mass distribution

In our reference work [8], the background's mass feature is *identically distributed* as the signal's mass. In other analyses, background events receive a mass that is either: (a) *randomly assigned* from the distribution of signal masses (only during training) [17], or (b) *uniform* in the interval of considered mass hypotheses. In general, we can analyze two situations: *identical* and *different* distribution of m for background events only.

(a) **Identical distribution**. The mass feature for the ith background event is assigned exactly to a value of the finite set $\mathcal{M} = \{m_0, m_1, \dots m_M\}$ of the signal mass hypotheses, selected randomly i.e. $m^{(i)} \sim \mathcal{M}$.
(b) **Different distribution**. The theoretical parameter (m_X) is used to define a probability distribution (e.g. *uniform* in the interval $[\min(\mathcal{M}), \max(\mathcal{M})]$, or *Gaussian* being centered at each $m_i \in \mathcal{M}$) that is *diverse* from (going beyond) the finite set of values that $\mathcal{M}$ encodes. In this work, we only consider a uniform distribution $U(\mathcal{M}_{\min}, \mathcal{M}_{\max})$ for the mass feature, m. Therefore, the i-th background event will be assigned a mass feature by sampling randomly from such distribution, i.e. $m^{(i)} \sim U$.

In both cases, we can *fix* the values of the mass feature (only for background) in the dataset (e.g. by sampling m only one time, and writing them to disk), or we can *sample* them during training, repeatedly and differently at each mini-batch. So, we have two degrees of freedom (distribution type, and assignment strategy) that lead us to a total of four unique combinations: (a) *identical fixed*, (b) *identical sampled*, (c) *uniform fixed*, and lastly (d) *uniform sampled*. In our experiments we noticed that, without proper regularization, having a uniform mass feature for the background allowed the network to almost perfectly fit both the training and validation sets: this may be due to the introduction of an artificial correlation between the mass feature and the class label, that was exploited during training. Nevertheless, by regularizing the model enough generalization is still ensured.

We may further discuss an additional assignment strategy where the mass feature m for the background can be also determined by means of *mass intervals* (or bins), based on the underlying *reconstructed mass* of the selected decay products. For example, if we consider $(150, 250)$ to be a specific mass interval centered around 200 GeV, we can assign $m = 200$ as mass feature for each background event x whose reconstructed mass is within the mass interval. In this work, such third assignment strategy have not been taken into account.

4.3. Training procedure

Beyond the architecture and regularization of the parametric network, as well as the distribution of the mass feature m, we can make some further considerations about how to properly train such kind of neural networks in light of what we already know about the structure of our own data. In general, we known that the signal is arranged in $|\mathcal{M}|$ groups (one for each m_i), and that the background is (eventually) composed of different processes. Therefore, beyond class labels, our data is naturally divided in *sub-classes*: in terms of the signal generating mass (i.e. the various $m_i \in \mathcal{M}$), and background processes[10]. We can exploit such domain-knowledge to design a training procedure that embeds such *inductive biases*.

In particular, we can further notice that each sub-class may have its own unique *frequency*, in terms of how much data samples fall into each sub-class, e.g. due to data imbalance: as shown in figures 3(a) and 6(b). Such frequencies may bias the (parametric) network towards certain sub-classes, resulting in an overall sub-optimal fitting of the data.

[10] We can think of sub-classes as additional labels in our data.

We propose to mitigate this simply by *balancing* each sub-class in a way that an equal number of events belongs to each of them. We call such approach *balanced training*, that, without discarding or generating new data, can be easily implemented by balancing each *mini-batch* during training, e.g. by sampling each sub-class in equal proportion. Specifically, we have:

- **No balance.** The usual training procedure, in which the network is trained by experiencing the data as it is. This will be our baseline for comparison.
- **Class-only balance.** Only the class labels are balanced within each mini-batch. Considering two classes, we can balance them in two ways: (a) by associating *sample or class weights* to each event, resulting in a weighted loss function, or (b) by sampling the events for a mini-batch such that half the batch is populated with samples belonging to the positive class (i.e. signal), and the other half with background samples (the negative class). We implement class-balancing by following the second option, as if the class weights were *implicitly* provided to weight the loss function.
- **Signal balance.** Since the entire signal is generated at different values of m_X, we can build mini-batches such that there is an equal number of events for each $m_i \in \mathcal{M}$. In practice, we take half the size B of a batch, i.e. $B_s = B/2$, and then split that in even parts such that each $m_X = m_i$ is represented by exactly $B_s/|\mathcal{M}|$ events. This implies that, at each mini-batch, all the signal mass hypotheses are always represented: this may not occur, especially if the batch size is small, for the case of no class and background balancing.
- **Background balance.** Similarly to signal balancing, mini-batches are divided into equally-sized parts such that each part contains events that belong to a certain background process. Of course, such balancing strategy is only meaningful if our background comprises more than one process: not the case of `HEPMASS-IMB`. Also in this case, each mini-batch will contain samples coming from all background processes.
- **Full balance**[11]**.** A combination of class, signal, and background balance. A batch is divided into two halves (each of size $B/2$) to ensure class balance. Then, one half will be signal balanced, and the other one is balanced according to the background. In this way, the network will always experience mini-batches that comprise all signal hypotheses, as well as all background processes.

4.3.1. Model selection

For every balancing procedure (even when the training is not balanced) and regardless the background's mass feature distribution, we always perform *validation* in the same way on a 25% split of the *original* (not batch-balanced) data. The AUC of the ROC is used as validation metric. The way we perform validation resembles the way the model is evaluated on the respective test-set (see section 6). In particular, for each $m_i \in \mathcal{M}$ we take the corresponding signal samples s, and all the background b; for the latter only, we sample their mass feature m from $\mathcal{M}$: as in the identical (sampled) option, described in the previous section. Thus, the mass feature m, will be $m^{(s)} = m_i$ for the signal events, and $m^{(b)} \sim \mathcal{M}$ for the background events. This is important to do, since: (a) balancing during validation will alter the value of the validation metric(s), as the original distribution of the data will be changed, and (b) regarding the mass feature distribution for the background, validating in a different way would lead to sub-optimal generalization performance of the selected model.

4.4. Preprocessing and regularization

In general, for our models we found out regularization to be beneficial for improved performance, and also (as we will see later) crucial for good interpolation. We utilize two well known regularization techniques: *dropout* [18], and *l*2-*weight decay*. For the affine architecture (section 4.1) we insert a `Dropout` layer after each affine-conditioning layer, thus zeroing random elements of the conditioned internal representation z: refer to equation (1). Instead, for the standard pNN architecture the `Dropout` layer is inserted after each ReLU activation. In both cases, we use a drop probability of 25%. Lastly we apply $l2$-regularization on all learnable parameters of the network, but with different coefficients for weights and biases respectively.

Classification performances can be usually further boosted by properly normalizing the data input to the network. Recall that our data have the form (x, m, y), in which: x is a multi-dimensional vector of features, m (the mass feature) corresponds to a one-dimensional vector of values that make the network be 'parametric', lastly y is a vector of class-labels (either 1 for signal or 0 for background, in our case). Discarding y, we can normalize (or preprocess, in general) both x and m in the same way, as done by [8] by means of *min-max*

[11] In our case, with only one background process, the signal-only and fully balanced training (in which half the batch size is reserved for signal, and the other half for background events) options are equivalent; not true, in general.

normalization, or differently. In our case, as HEPMASS is already standardized, we only normalize m by just *dividing* it by 1000.

5. Properties of PNNs

We believe that parametric networks have many interesting properties beyond interpolation. In this section we attempt a first characterization of them, trying to better understand such kind of models.

5.1. Interpolation

The interpolation capability of a pNN is its ability to generalize towards novel mass points that lie between two known mass hypotheses, resulting in effective interpolation of events between them: we also use the term *extrapolation* when the missing or novel mass points lie at the two extremes of the mass range, or even beyond them. In this case the generalization capability should be *twofold*: the network is requested (a) to perform well on novel samples belonging to the known masses, and also (b) to correctly classify new events that belong to the missing hypotheses. This means that a pNN capable of good interpolation should provide more accurate outcomes compared to the ones that would be obtained by interpolating the results of M individual classifiers, instead. We want our pNN to perform well even if some hypotheses are missing. So, how to be sure and ensure that our model has acquired such capability?

5.1.1. Factors

We investigated several potential factors, including *per-mass features distribution*, *mass imbalance*, *background distribution*, *network regularization*, *batch size*, and *training procedure*, that may affect the interpolation capability of pNNs. Here we describe the most impactful:

- **Per-mass features distribution**. Recalling from figure 4, a shift in the feature distribution has been observed. This tells us that some mass points are more difficult to classify than others. In fact such behavior is directly reflected on *single-mass interpolation* and *extrapolation* (i.e. when we train our model on just one mass less). In figure 9 we can observe how extrapolating $\mathcal{D}_{500}$ (plot d) is way more difficult than the other masses. This also suggests us that evaluating our model on only one mass less does not necessarily imply that our network will correctly interpolate or extrapolate everywhere, in general.
  Another way to understand how much the similarity among masses helps (or avoids) our network at interpolating or extrapolating them, is to stress our model at *extrapolating*: we train a pNN on just *one* mass hypothesis, requesting it to extrapolate all the remaining ones. In figures 10(a) and (b), we observe how easier is to predict the missing masses, being the model only trained on $m_X = 750$ or $m_X = 1000$. This fact seems to be (at least, partially) independent from the AUC achieved on such mass points: although on $\mathcal{D}_{1500}$ the highest AUC is obtained (plot 10(e)), average extrapolation performance are not the highest among the others mass points.
- **Background distribution and regularization**. The impact of background distribution (section 4.2) goes beyond classification performance, as it may also affect interpolation. Figure 14(c) denotes a pNN that hardly interpolates; such network was trained on a *uniformly distributed* background, without regularization. During training on all the mass points, we noticed that the same network were able to almost classify perfectly both the training and validation sets: clearly overfitting them. As discussed previously, having a uniformly distributed mass feature for the background introduces an additional correlation with the class label, making training ‘easy’. Indeed, by regularizing the model enough and increasing the batch size, generalization as well interpolation can be achieved with success.

5.1.2. Select mass hypotheses

Another practical aspect to consider is how to select the mass hypotheses to drop for a fair measure of interpolation. As described earlier, the goodness of training data can mislead us when quantifying interpolation. So, we suggest to drop almost half of the mass hypotheses, in the following way: let us assume we have hypotheses $[m_0, m_1, m_2, m_3, m_4, m_5]$, we may drop $[m_0, m_2, m_4]$ or $[m_1, m_3]$. Sometimes, it is interesting (also useful) to see what a parametric network can achieve when trained on *only one mass*, and evaluated for extrapolation on all the others: as shown in figure 10. Such a test may resemble the training of individual classifiers, being different in that also the mass feature is provided. This kind of test can be useful to better understand the contribution of both network architecture and training data to the quality of the resulting extrapolation: in this case, we can expect the parametrized network (trained on only one mass) to perform well on the only training mass, but not too worse on immediately *close* hypotheses also maintaining a reasonable accuracy on *far* masses. This can be an easy way to asses the similarity of features among masses

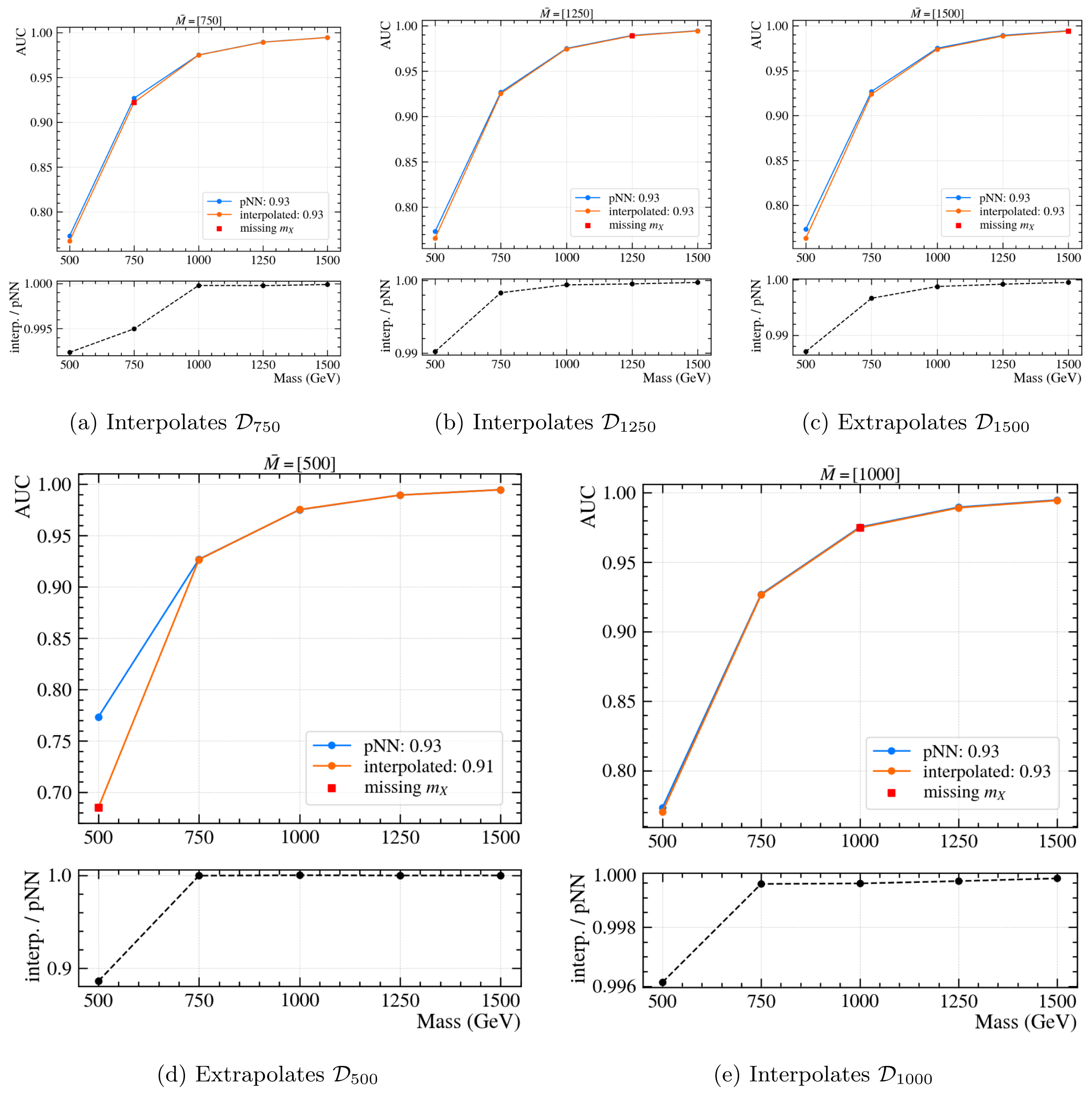


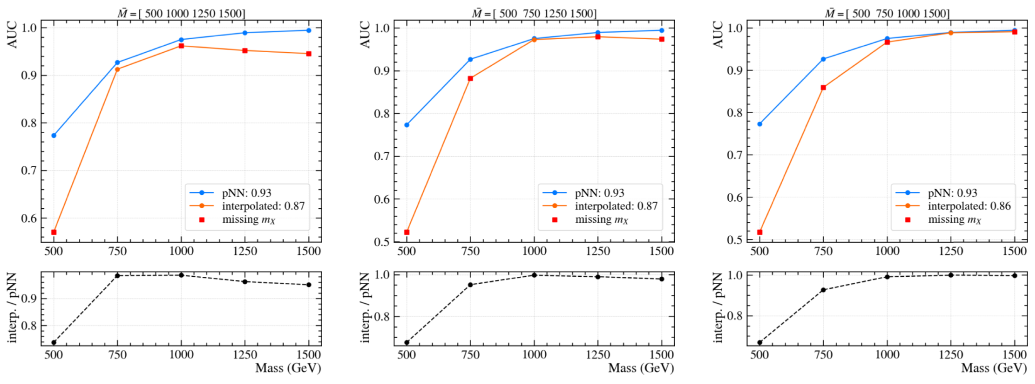
Figure 9. Single-mass interpolation and extrapolation on HEPMASS. The blue curve (same in all plots) represents the ROC's AUC of a pNN trained on all hypotheses. The orange curve depicts AUC performance for a pNN trained on a subset of the masses. The missing signal mass hypotheses are denoted by a red square. Lastly, below each plot is depicted the ratio between the orange and blue curves.

(which is an intrinsic property of the training data): if masses are similar, the network should perform almost the same on each unseen mass.

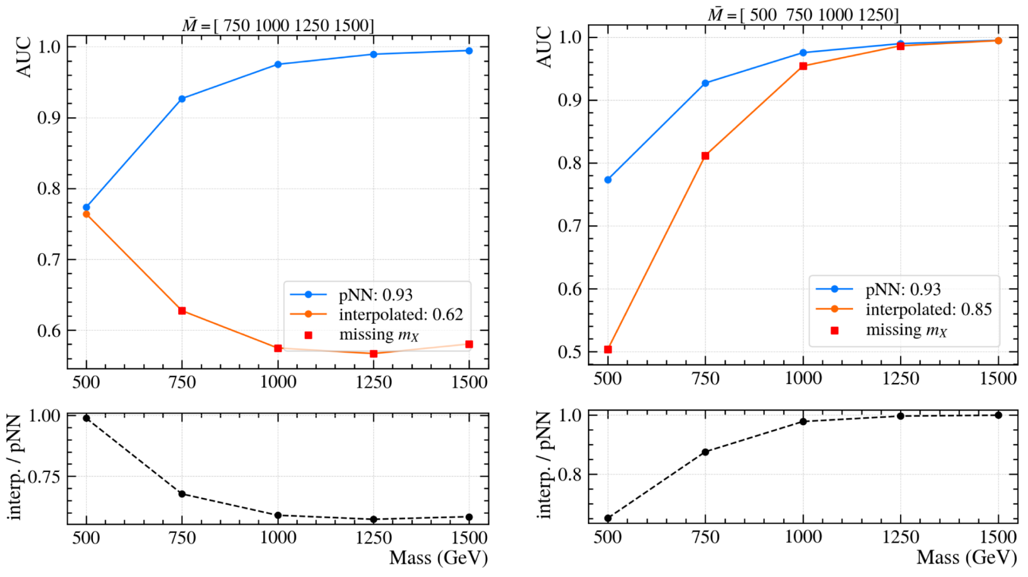
5.2. Learned mass representation

From section 1, in our signal-background classification problem, we actually consider $\mathcal{M}$ mass hypotheses for the signal. This means that our original task can be broken down into $|\mathcal{M}|$ smaller classification problems, each of them considering only a specific mass $m_i \in \mathcal{M}$. In fact, approaches before the parametric network [4] used to solve each sub-task by training a neural network solely on $\mathcal{D}_{m_i}$ (i.e. a slice of the original dataset $\mathcal{D}$, that selects events whose mass feature is m_i), thus obtaining a (disjoint) set of $\mathcal{M}$ individual networks, which we will call g_{m_i} (or g_i for short). Somehow each individual network g_i, despite being trained solely on one mass, is able to *implicitly* relate the input features to the signal hypothesis m_i they truly belong to (or even to the underlying invariant mass). This fact seems to be confirmed by the visualization in figure 11, in which each intermediate representation h_i (of network g_{m_i} for all $m_i \in \mathcal{M}$) has a precise and nicely clustered spatial arrangement, that also relates well to the learned class label.

Such kind of visualizations may provide further insights about the relation existent between a parametric network and a set of individual networks. Intuitively, we may want the intermediate representation of our

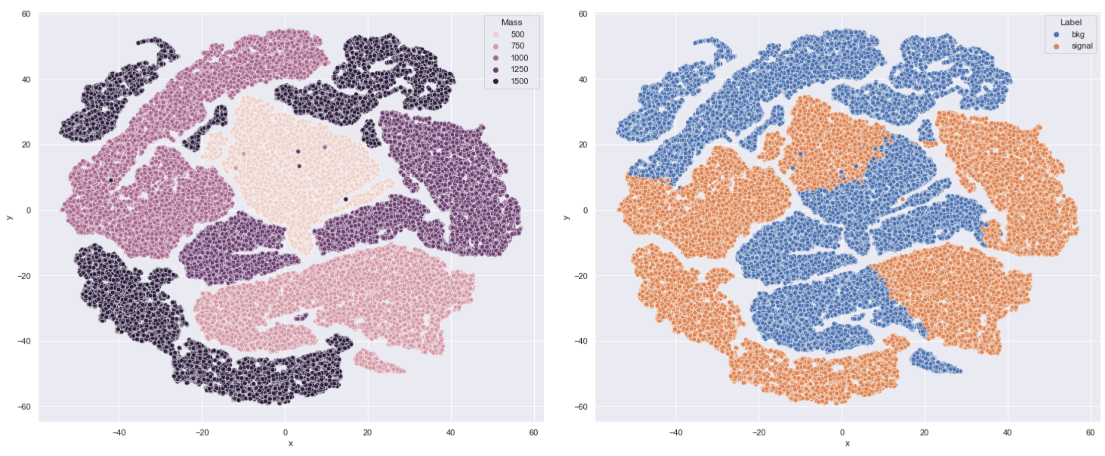


(a) Trained on $\mathcal{D}_{750}$ (b) Trained on $\mathcal{D}_{1000}$ (c) Trained on $\mathcal{D}_{1250}$



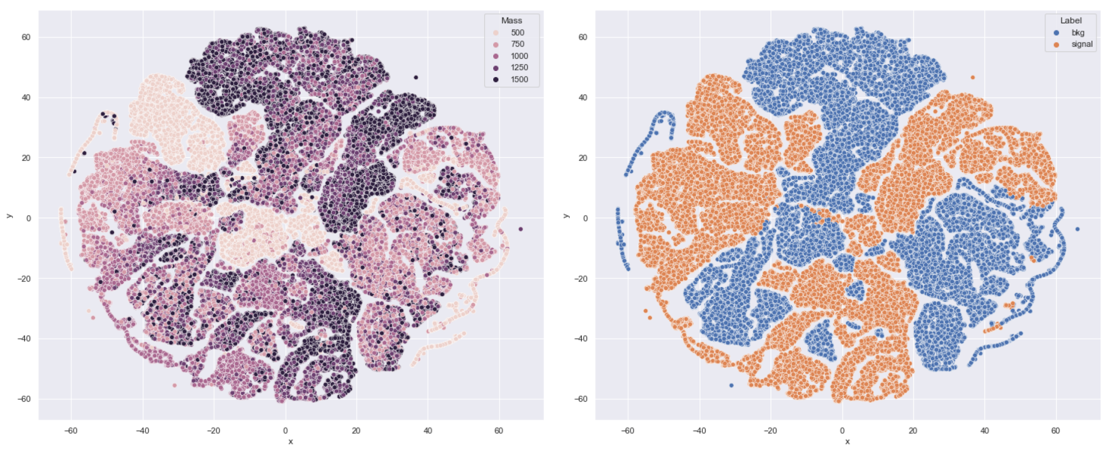
(d) Trained on $\mathcal{D}_{500}$ (e) Trained on $\mathcal{D}_{1500}$

Figure 10. Extrapolation on HEPMASS. We can clearly notice how different mass 500 is from all the others, in fact its extrapolation performance is really poor: the (mean) AUC decreases by more than 30%. The other plots present a mean loss in AUC that is at most 8%.



(a) Mass label (b) Class label

Figure 11. Visualization of the intermediate learned representation (i.e. last hidden layer) of each individual network by means of t-SNE [19, 20] on the HEPMASS dataset. By coloring the points by the mass label (left plot), we notice that representations related to the same mass are clustered together, meaning that each network g_i has indirectly acquired knowledge about its underlying signal mass hypothesis m_i (although not given as input). Also a structured part of them clearly depicts the learned class label (right plot).



(a) Mass label (b) Class label

Figure 12. Visualization of the intermediate learned representation of a pNN trained on the `HEPMASS` dataset. Compared to figure 11, some structure is present although more convoluted, and less clear in general.

pNN to be disentangled along the *mass 'axis'* (in the underlying manifold), as seen in figure 11 for individually trained neural networks (considered as a whole). The situation for the pNN is similarly structured (figure 12): some mass are well clustered (e.g. at 500 GeV) and for others we can observe a smooth 'shading' among them. This means that the pNN has *partially* recovered the underlying structure about the individual masses, but there is still some confusion about representing datapoints coming from higher values of the signal mass hypotheses.

6. Results

Since the datasets we have for signal-background classification can be divided into $|\mathcal{M}|$ groups (in order to be able to 'parametrize' a neural network), also evaluation metrics have to be considered in terms of the available mass hypotheses for the signal. In particular, the models are evaluated on each m_i separately. So we consider the signal events generated at a certain $m_i \in \mathcal{M}$, along with the whole background: i.e. the background events that spans the entire mass range. Indeed, the results provided in this section were all computed only on the test-set of the respective datasets. Moreover, we *weight* both signal and background samples (only for evaluation) such that the weighted count of signal events is equal to the weighted count of background events, i.e:

$$\sum_i w_s^{(i)} = \sum_j w_b^{(j)}. \tag{2}$$

In particular, for both datasets we set the signal weight to one, $w_s = 1$, and for the background as $w_b = 1/5$; since we have five mass hypotheses for the signal. For such reason, each time we test a particular mass hypothesis $m_i \in \mathcal{M}$, we select the corresponding signal (i.e. all the signal events that have m_i as mass feature) and the *whole* background: i.e. the mass feature for all background samples b, is set equal to m_i; thus, $m^{(b)} = m_i$. This is to account for the fact that, in `HEPMASS`, the original background's mass is assigned randomly, being sampled from the set $\mathcal{M}$ (i.e. the *identical fixed* strategy, discussed in section 4.2).

6.1. Metrics

We consider standard evaluation metrics for classification tasks, such as the *AUC* (area under the curve) of the *ROC* (receiving operating characteristic) and *Precision-Recall* curves. In particular, the ROC curve can be interpreted for HEP as comparing the *signal efficiency* (y-axis) against the *background efficiency* (x-axis): in terms of how much signal is retained when considering a certain fraction of the background. Otherwise, we can consider the *background rejection* (i.e. 1 − background efficiency): how much signal is retained at a certain discard of background. The Precision-Recall curve instead, compares the signal efficiency (*recall*) with what we call the *purity* (precision): the number of true signal divided by the number of events classified as signal (which also contains misclassification of the background).

Along usual classification metrics, we also consider the *Approximate Median Significance* (AMS) [21, 22] but in the following form:

$$\mathrm{AMS}(t) = \frac{s_t}{\sqrt{s_t + b_t}}, \tag{3}$$

in which s_t and b_t is the (weighted[12]) number of *true* signal and true background events, respectively, that passed the classification threshold t. This quantity is useful to determine an optimal classification threshold for our networks, called the *best cut* $t^\star$, that is the threshold that maximizes the significance: $t^\star = \arg\max_t \mathrm{AMS}(t)$. Since the value of the AMS depends on the number of events from which is calculated, we propose a new metric the *significance ratio* (σ_{ratio}) that is normalized in $[0,1]$, thus being very intuitive to interpret. The significance ratio is defined as the ratio between the best (maximum) AMS by the largest possible significance (only achievable ideally, by means of a perfect classification when $s_t = s_{\max}$, i.e. equal to all the true signal, and $b_t = 0$):

$$\sigma_{\text{ratio}} = \frac{\max_t \mathrm{AMS}(t)}{s_{\max}/\sqrt{s_{\max}}} = \max_t \left\{ \frac{s_t \cdot \sqrt{s_{\max}}}{s_{\max} \cdot \sqrt{s_t + b_t}} \right\} = \frac{s_\star \cdot \sqrt{s_{\max}}}{s_{\max} \cdot \sqrt{s_\star + b_\star}}, \tag{4}$$

where $s_\star = s_{t^\star}$, and $b_\star = b_{t^\star}$. Such metric can be also used to compare how well the same model classifies different mass hypotheses: this is now possible since the number of events belonging to a certain m_i does not affect the scale of the metric (as happens for the regular AMS, instead), anymore. We can further say that the best cut $t^\star$, apart from telling us which classification threshold is the best to determine the positive class, is also an useful quantity to monitor because it can provide additional information about the goodness of the classification. In particular, by plotting the best cut versus the mass we may observe failure cases in which $t^\star$ is either 0 or 1, depicting a situation in which the network is unable to correctly separate out the background ($t^\star = 0$) or to retain a significant amount of signal ($t^\star = 1$). A special failure case can be observed when $s_\star = s_{\max}$ and $s_\star = b_\star$, i.e. the signal is equal in number to all the true signal, which is also equal to the true classified background (e.g. due to applied weights). In this case we would obtain $\sigma_{\text{ratio}} = 1/\sqrt{2} \approx 0.707$, since:

$$\frac{s_\star \cdot \sqrt{s_{\max}}}{s_{\max} \cdot \sqrt{s_\star + b_\star}} = \frac{\sqrt{s_\star}}{\sqrt{s_\star + s_\star}} = \frac{1}{\sqrt{2}}. \tag{5}$$

Indeed, measuring $\sigma_{\text{ratio}} \approx 0.707$ also implies having an AUC of 0.5, corresponding to nonsense classification. Moreover, if the best cut $t^\star$ is such that $s_\star = s_{\max}$ but $b_\star > 0$, then $\sigma_{\text{ratio}} = \frac{\sqrt{s_\star}}{\sqrt{s_\star + b_\star}}$ decreases toward zero (in the limit), as $b_\star$ approaches $b_{\max}$ (i.e. the weighted count of all true background events).

6.2. Baselines

To first assess the advantages brought by pNNs we should compare them to their 'non-parametric' counterparts, namely *single* and *individual* neural networks. What we call a single-NN is just a neural network that is trained *without* the mass feature (m) as input but on all $\mathcal{M}$ hypotheses at the same time; so, it has only one input: the features x. Instead, the individual networks are, as the name suggests, a set of single networks, g_{ϕ_i}, each of them trained to target a specific mass hypothesis $m_i \in \mathcal{M}$. Since each g_{ϕ_i} is trained in isolation on one m_i against the whole background, we expect the individual networks to easily beat the single network as it should face a harder learning problem. Moreover, both kind of networks can provide baseline performance for interpolation: the single network is trained to fit all data regardless of the mass (not given as input), whereas each individual network g_{ϕ_i} will interpolate by means of the *similarity* between the hypothesis m_i it was trained on, and the m_j to interpolate.

Finally, to assess the effectiveness of the various decision choices described in section 4, we apply them (whether possible) to the non-parametric baselines, and also (of course) to the parametric baseline: a *vanilla pNN*, as intended by Baldi *et al* [8]. Results for classification performance are shown in figure 13.

[12] In general, the weights w_s and w_b are introduced with the only aim of balancing the occurrences of the two classes when testing for a particular m_i: they have no physical meaning, as we want to keep our study as general as possible, without assuming any luminosity and signal cross section weights during training and evaluation.

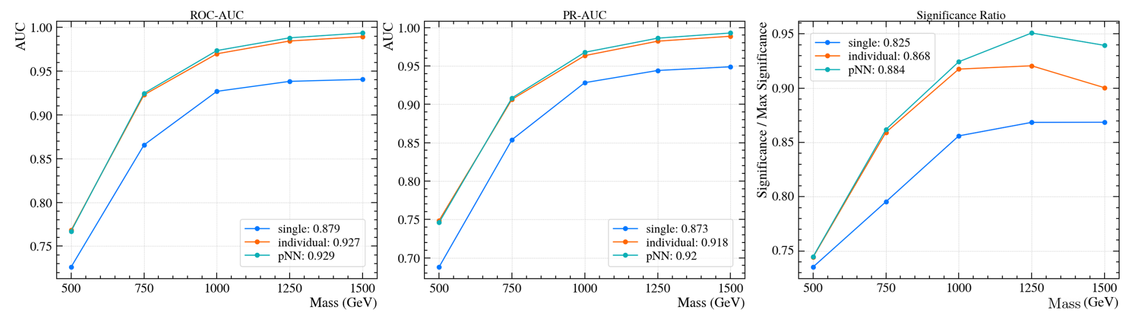


Figure 13. Comparison among (tuned) baselines on `HEPMASS-IMB`. As we can see, the best classification performance (in terms of AUC of both ROC and PR curves, and σ_{ratio} as well; for each $m_i \in \mathcal{M}$) are achieved by the individual NNs, as well as the parametric baseline (with identical-fixed distribution for the background's mass feature).

Table 3. Per-mass ROC's AUC metric computed on the test-set of `HEPMASS`: all AUC values are in percentage, higher is better. Best results are shown boldface. Options for mass distribution are described in section 4.2. We have also included a third baseline, a parametric network with linear activation: as we can see it underperforms even the single-NN, despite leveraging the additional information provided by the input mass feature.

Model		Mass (GeV)					
Kind	Mass distribution	500	750	1000	1250	1500	AUC (average)
Single-NN	None	68.55	89.37	96.38	98.07	98.69	90.21%
Individual-NNs	None	**77.35**	**92.59**	**97.42**	**98.89**	**99.45**	**93.14%**
pNN (linear)	Identical (*fixed*)	63.42	89.36	96.08	97.90	98.58	89.07%
pNN	Uniform (*fixed*)	70.95	91.80	97.26	98.79	99.37	91.63%
Affine	Uniform (*fixed*)	71.24	90.81	96.94	98.64	99.26	91.38%
pNN	Uniform (*sampled*)	71.63	91.71	97.25	98.82	99.39	91.76%
Affine	Uniform (*sampled*)	70.16	91.61	97.16	98.75	99.34	91.40%
pNN	Identical (*fixed*)	76.78	92.56	97.46	98.92	99.46	93.04%
Affine	**Identical (*fixed*)**	**77.34**	**92.80**	**97.55**	**98.96**	**99.49**	**93.23%**
pNN	Identical (*sampled*)	76.77	92.60	97.48	98.92	99.46	93.05%
Affine	**Identical (*sampled*)**	**77.31**	**92.77**	**97.55**	**98.96**	**99.49**	**93.22%**

6.3. Evaluation

6.3.1. Hyperparameters

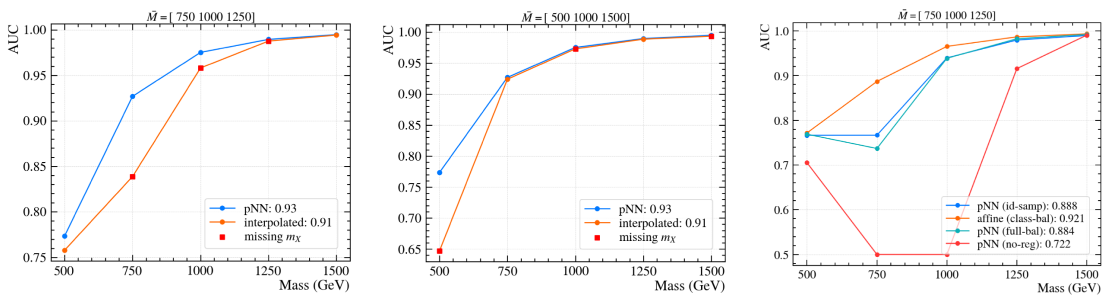
All the neural networks used for comparison are built using the TensorFlow 2.X [23] framework along with the Keras [24] library for Python. To improve reproduce our results we fix the *random seed* to be 42. All the networks use the same hyperparameters: ReLU activation, $[300, 150, 100, 50]$ units for each hidden layer, sigmoid output, binary-crossentropy loss, Adam optimizer [25], batch size of 1024 (except when told otherwise), and default initialization: `glorot_uniform` [26] for weights, and constant `zero` initializer for biases. It results in about 70k learnable parameters. The learning rate is never decayed, and set to 3×10^{-4}. In general, we always use regularization by means of both dropout (with drop rate of 25%), and *l*2-weight decay. The weight decay is applied differently, with a strength of 10^{-4} for weights (or 10^{-5}), and 10^{-5} for biases (or 10^{-6}). Also, the same hyperparameters are kept for both datasets, as well as the training budget fixed at 25 epochs. In general, the hyperparameters we use were initially tuned for the vanilla pNN architecture on `HEPMASS`.

6.3.2. HEPMASS

Results about classification performance (with baseline models), background's mass feature distribution, and model architecture are presented in table 3. Whereas further results for interpolation are showed in figures 14(a) and (b).

6.3.3. HEPMASS-IMB

Results on interpolation are presented both in table 4 and figure 14(c). Furthermore, an exhaustive comparison among baseline models, parametric and affine architectures, background's mass distribution,



(a) Interpolation on `HEPMASS`. (b) Interpolation on `HEPMASS`. (c) Interpolation on `HEPMASS-IMB`.

Figure 14. In figure (a) and (c) the models have been trained solely on two mass hypotheses: at 500 and 1500 GeV. Instead, the networks depicted in plot (b), were trained on mass 750 and 1250 GeV. The title of each plot denotes the missing masses by $\bar{M}$. In particular, in figure (c) we can seen how the lack of proper regularization prevents the network (depicted by the red line) to interpolate the signal at the missing mass hypotheses. Moreover, the affine network outperforms by a large margin the two competent pNNs.

Table 4. Per-mass and averaged, ROC's AUC and significance ratio (equation (4)) metrics computed on `HEPMASS-IMB`, showing interpolation capabilities. The missing signal mass hypotheses are $\bar{\mathcal{M}} = \{750, 1000, 1250\}$. Best results are shown in boldface. The *identical (sampled)* distribution strategy is described in section 4.2, and the *balanced training* is detailed in section 4.3.

Model		Mass (GeV)					Average (%)	
Kind	Mass distribution	500	750	1000	1250	1500	AUC	σ_{ratio}
pNN	Identical (*sampled*)	76.65	76.66	93.93	97.92	98.93	88.82	
		74.38	74.78	88.38	93.40	91.52		84.49
pNN (class)	Identical (*sampled*)	77.41	71.25	94.60	98.48	99.29	88.21	
		74.81	73.23	89.16	94.55	96.32		85.61
pNN (full)	Identical (*sampled*)	76.88	73.68	93.85	98.21	99.17	88.36	
		74.56	75.38	88.39	93.67	93.82		85.16
Affine	Identical (*sampled*)	76.49	88.79	96.39	98.44	99.13	91.85	
		74.33	82.66	91.07	93.86	89.90		86.36
Affine (class)	**Identical (*sampled*)**	77.16	88.67	96.51	98.63	99.31	**92.05**	
		74.70	83.10	91.30	94.80	96.43		**88.07**
Affine (full)	Identical (*sampled*)	76.20	84.68	94.87	97.87	98.89	90.50	
		74.26	80.28	89.04	92.91	89.84		85.27

and training procedure is detailed in table 5. Finally, outcomes about classification performance in terms of class separation are detailed in figure 15.

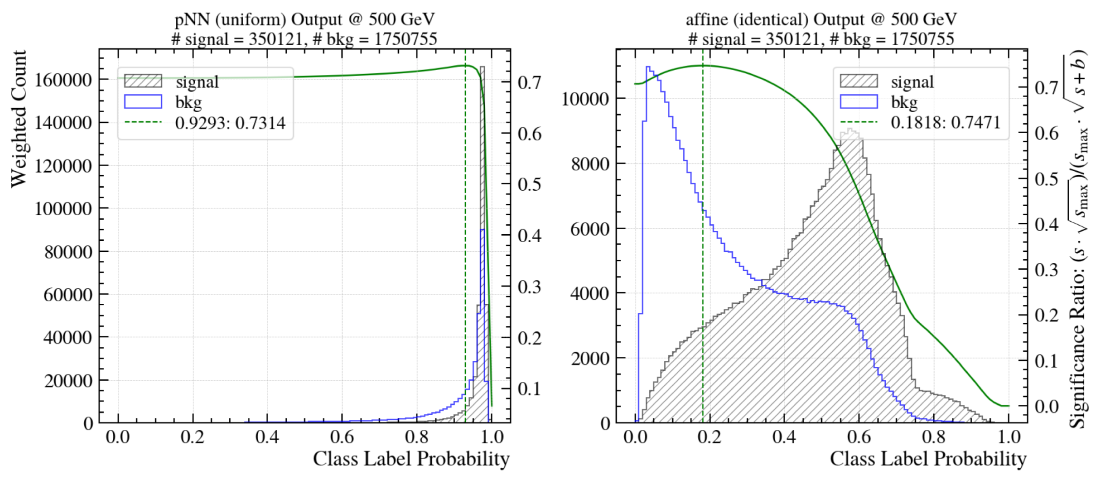
6.4. Discussion

In our empirical comparison among network architectures, background distribution, and training procedure, we can conclude that:

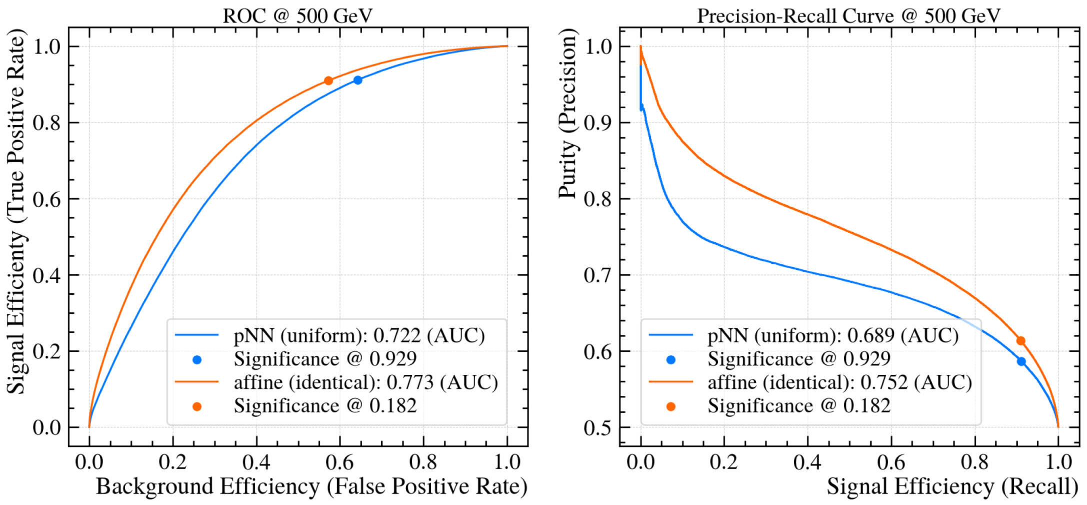
(a) The affine-conditioning mechanism is able to better exploit the information brought by the mass feature, resulting in improved classification performance.
(b) The balanced training procedure, that yields balanced mini-batches, can further improve performance, even without changing the network architecture.
(c) The way the mass feature is distributed has a profound impact on how the model classifies and interpolates the missing masses. In general, the uniform distribution tends to easily overfit resulting in lower performance.
(d) Finally, the right combination of network architecture, background distribution, and balanced training, allowed us to greatly improve on both imbalanced classification and interpolation, almost recovering the classification performances achieved on the original, full, and not-imbalanced dataset.

Table 5. Comparison of baselines, model architecture, background's mass feature distribution, and training procedure on `HEPMASS-IMB`. Performances are evaluated in terms of ROC-AUC and significance ratio. Options for mass distribution are described in section 4.2. The words *class* and *full*, refer, respectively, to class- and fully-balanced training (section 4.3). Best results are shown boldface.

Model		Mass (GeV)					Average (%)	
Kind	Mass distribution	500	750	1000	1250	1500	AUC	σ_{ratio}
Single-NN	None	72.62	86.56	92.68	93.83	94.06	87.95	
		73.52	79.51	85.59	86.85	86.86		82.47
Individual-NNs	None	76.81	92.28	96.98	98.44	98.93	92.69	
		74.43	85.91	91.75	92.07	90.04		86.84
pNN	Identical (*fixed*)	76.71	92.46	97.35	98.80	99.35	92.93	
		74.46	86.17	92.43	95.09	93.94		88.42
pNN (class)	Identical (*fixed*)	76.93	92.56	97.41	98.85	99.39	93.03	
		74.67	86.34	92.55	95.32	96.64		89.11
pNN (full)	Identical (*fixed*)	76.29	92.33	97.33	98.79	99.34	92.82	
		74.31	86.14	92.44	95.17	96.47		88.91
Affine	Identical (*fixed*)	77.19	92.62	97.43	98.86	99.40	93.10	
		74.70	86.30	92.52	95.31	95.61		88.89
Affine (class)	**Identical (*fixed*)**	77.19	92.64	97.46	98.88	99.42	**93.12**	
		74.76	86.38	92.62	95.37	96.77		**89.18**
Affine (full)	Identical (*fixed*)	76.45	92.46	97.38	98.78	99.31	92.88	
		74.42	86.24	92.50	95.07	95.07		88.66
pNN	Identical (*sampled*)	76.73	92.44	97.34	98.80	99.35	92.93	
		74.47	86.11	92.37	94.96	93.76		88.34
pNN (class)	Identical (*sampled*)	76.88	92.51	97.40	98.86	99.40	93.01	
		74.61	86.29	92.51	95.33	96.70		89.09
pNN (full)	Identical (*sampled*)	76.38	92.39	97.34	98.79	99.34	92.85	
		74.35	86.17	92.47	95.16	96.52		88.93
Affine	Identical (*sampled*)	77.26	92.64	97.43	98.87	99.40	93.12	
		74.71	86.34	92.55	95.33	95.74		88.93
Affine (class)	**Identical (*sampled*)**	77.24	92.66	97.46	98.88	99.42	**93.13**	
		74.78	86.39	92.60	95.39	96.77		**89.19**
Affine (full)	Identical (*sampled*)	76.41	92.43	97.37	98.80	99.34	92.87	
		74.37	86.19	92.47	95.10	95.23		88.67
pNN	Uniform (*fixed*)	76.67	92.41	97.33	98.79	99.34	92.91	
		74.48	86.15	92.39	95.15	94.70		88.57
pNN (class)	Uniform (*fixed*)	71.00	91.56	97.14	98.75	99.32	91.55	
		72.71	85.64	92.22	95.09	96.45		88.42
pNN (full)	Uniform (*fixed*)	72.19	91.41	97.11	98.70	99.25	91.73	
		73.06	85.42	92.16	94.93	96.23		88.36
Affine	Uniform (*fixed*)	77.26	92.62	97.43	98.86	99.41	93.11	
		74.73	86.31	92.52	95.25	94.65		88.69
Affine (class)	**Uniform (*fixed*)**	77.29	92.68	97.47	98.89	99.43	**93.15**	
		74.82	86.42	92.64	95.40	96.80		**89.22**
Affine (full)	Uniform (*fixed*)	66.67	91.15	97.08	98.70	99.30	90.58	
		70.75	85.46	92.05	94.88	96.34		87.90
pNN	Uniform (*sampled*)	72.21	91.48	97.17	98.72	99.29	91.77	
		73.14	85.55	92.15	94.88	95.17		88.18
pNN (class)	Uniform (*sampled*)	71.48	91.68	97.16	98.75	99.32	91.68	
		72.80	85.61	92.20	95.10	96.47		88.44
pNN (full)	Uniform (*sampled*)	71.35	91.53	97.18	98.73	99.31	91.62	
		72.57	85.44	92.20	94.98	96.44		88.33
Affine	Uniform (*sampled*)	67.52	91.13	97.01	98.64	99.26	90.71	
		72.73	85.47	91.90	93.90	90.49		86.78
Affine (class)	Uniform (*sampled*)	66.97	90.99	97.06	98.69	99.28	90.60	
		71.60	85.29	92.01	94.84	96.29		88.01
Affine (full)	Uniform (*sampled*)	66.08	90.37	96.82	98.51	99.14	90.18	
		72.46	84.87	91.57	94.41	95.91		87.84



(a) Weighted class separation (histograms), significance ratio (solid green curve), and best-cut (vertical dashed green line), at $m_X = 500$ GeV on HEPMASS-IMB. The value of the best cut (classification threshold), and the corresponding value of the significance ratio are shown in the legend at the top.



(b) Comparison of ROC and PR curves at $m_X = 500$ GeV.

Figure 15. Comparison of classification performance on HEPMASS-IMB, between a pNN (*uniform*, sampled) and affine model (*identical*, sampled). We can notice how the two models opt for rather different classification thresholds.

7. Conclusions

In this study we first discussed the concept of 'parametrization' which is really a re-brand of the widely used conditioning mechanisms in deep learning. Establishing such connection allows us to brought ideas and methods from such area, to improve pNNs in HEP. Another proposed intuition is about the structure of the data we use for signal-background classification: we know the contribution of the background(s), and at which mass the signal is generated. In fact we leverage the latter information to build a mass feature that parametrizes a neural network, allowing the model to replace a set of individual classifiers, as well as to interpolate beyond events seen during training. By studying the general structure of the data, we can exploit the inductive biases it provides by embedding them in the network design, and training as well. Lastly we demonstrated that pNNs are able to interpolate under real-world assumptions. We hope the ideas proposed here to be inspirational for further work about parametric networks, but also to be useful in other fields beyond HEP that have a similar problem setting and requirements.

7.1. Open questions

Our work is a first step towards a full understanding of parametric networks. We believe more properties and extensions to what is presented here to exist. In particular, we may suggest further research directions:

- Real-world datasets are imbalanced, so either *self-supervised learning*, class- or mass-specific *data augmentation*, or (parametric) *generative models* may provide a major improvement in classification performance.
- The signal is generated at few discrete mass hypotheses, what about parametrizing on the whole, *continuous* mass range?
- The output of a pNN is a single number, why not letting the network output or discover a *classification rule* that can be easily interpreted by physicists to further increase their knowledge about a certain phenomena?

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://zenodo.org/record/6453048. The code used to produce our experiements is openly available on GitHub: https://github.com/Luca96/affine-parametric-networks.

Acknowledgments

The authors gratefully acknowledge the CMS Bologna analysis team - in particular Federica Primavera, Stefano Marcellini and Gianni Masetti - for the constructive discussions, and Andrea Perrotta for the valuable feedback and support.

ORCID iDs

Luca Anzalone https://orcid.org/0000-0002-0399-8836
Tommaso Diotalevi https://orcid.org/0000-0003-0780-8785
Daniele Bonacorsi https://orcid.org/0000-0002-0835-9574

References

[1] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (Cambridge, MA: MIT press)
[2] Friedman J *et al* 2001 *The Elements of Statistical Learning* (*Springer Series in Statistics* vol 1) (New York: Springer)
[3] Chatrchyan S *et al* 2012 Observation of a New Boson at a mass of 125 GeV with the CMS experiment at the LHC *Phys. Lett.* B **716** 30–61
[4] Baldi P, Sadowski P and Whiteson D 2014 Searching for exotic particles in high-energy physics with deep learning *Nat. Commun.* **5** 4308
[5] Evans L and Bryant P 2008 LHC machine *J. Instrum.* **3** S08001
[6] Sirunyan A M *et al* 2020 Search for a charged Higgs boson decaying into top and bottom quarks in events with electrons or muons in proton-proton collisions at $\sqrt{s} = 13$ TeV *J. High Energy Phys.* **01** 096
[7] Sirunyan A M *et al* 2018 Search for resonant and nonresonant Higgs boson pair production in the $b\bar{b}\ell\nu\ell\nu$ final state in proton-proton collisions at $\sqrt{s} = 13$ TeV *J. High Energy Phys.* **01** 054
[8] Baldi P, Cranmer K, Faucett T, Sadowski P and Whiteson D 2016 Parameterized neural networks for high-energy physics *Eur. Phys. J.* C **76** 1–7
[9] Anzalone L, Diotalevi T and Bonacorsi D 2022 HEPMASS-IMB (https://doi.org/10.5281/zenodo.6453048)
[10] Baldi P, Cranmer K, Faucett T, Sadowski P and Whiteson D 2015 HEPMASS dataset—UCI machine learning repository (available at: http://archive.ics.uci.edu/ml/datasets/HEPMASS)
[11] Codevilla F, Müller M, López A M, Koltun V and Dosovitskiy A 2018 End-to-end driving via conditional imitation learning *2018 IEEE Int. Conf. on Robotics and Automation (ICRA 2018)* (*Brisbane, Australia, 21–25 May 2018*) (IEEE) pp 1–9
[12] Finn C, Abbeel P and Levine S 2017 Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks *Proc. 34th Int. Conf. on Machine Learning (ICML 2017)* (*Sydney, NSW, Australia, 6–11 August 2017*) (*Proc. Machine Learning Research*) vol 70, ed D Precup and Y W Teh (PMLR) pp 1126–35
[13] Eysenbach B, Gupta A, Ibarz J and Levine S 2019 Diversity is all you need: learning skills without a reward function *7th Int. Conf. on Learning Representations (ICLR 2019)* (*New Orleans, LA, USA, 6–9 May 2019*) (OpenReview.net)
[14] Mirza M and Osindero S 2014 Conditional generative adversarial nets arXiv:1411.1784
[15] Dumoulin V, Perez E, Schucher N, Strub F, Vries H d, Courville A and Bengio Y 2018 Feature-wise transformations *Distill* (available at: https://distill.pub/2018/feature-wise-transformations)
[16] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift *Proc. 32nd Int. Conf. on Machine Learning (ICML 2015) (Lille, France, 6–11 July 2015) (JMLR Workshop and Conf. Proc.)* vol 37, ed F R Bach and D M Blei (JMLR.org) pp 448–56
[17] Aad G *et al* 2021 Search for charged Higgs bosons decaying into a top quark and a bottom quark at $\sqrt{s} = 13$ TeV with the ATLAS detector *J. High Energy Phys.* **06** 145
[18] Srivastava N, Hinton G E, Krizhevsky A, Sutskever I and Salakhutdinov R 2014 Dropout: a simple way to prevent neural networks from overfitting *J. Mach. Learn. Res.* **15** 1929–58
[19] Van der Maaten L and Hinton G 2008 Visualizing data using t-SNE *J. Mach. Learn. Res.* **9** 11
[20] Wattenberg M, Viégas F and Johnson I 2016 How to use t-SNE effectively *Distill* **1** e2
[21] Adam-Bourdarios C, Cowan G, Germain C, Guyon I, Kégl B and Rousseau D 2015 The Higgs Boson Machine Learning Challenge *NIPS 2014 Workshop on High-Energy Physics and Machine Learning* (PMLR) pp 19–55

[22] Cowan G, Cranmer K, Gross E and Vitells O 2011 Asymptotic formulae for likelihood-based tests of new physics *Eur. Phys. J.* C **71** 1–19

[23] Abadi M *et al* 2016 TensorFlow: a system for large-scale machine learning *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* pp 265–83

[24] Chollet F *et al* 2015 Keras: Deep Learning for Humans *Keras.io*

[25] Kingma D P and Ba J 2015 Adam: a method for stochastic optimization *3rd Int. Conf. on Learning Representations (ICLR 2015)* (*San Diego, CA, USA, 7–9 May 2015*) (*Conf. Track Proc.*) ed Y Bengio and Y LeCun

[26] Glorot X and Bengio Y 2010 Understanding the difficulty of training deep feedforward neural networks *Proc. 13th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2010)* (*Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010*) (*JMLR Proc.*) vol 9, ed Y W Teh and D M Titterington (JMLR.org) pp 249–56