# Affine parametric neural networks for high-energy physics

**Luca Anzalone**[1,2], **Tommaso Diotalevi**[1,2] **and Daniele Bonacorsi**[1,2]
[1]Department of Physics and Astronomy (DIFA), University of Bologna, Italy
[2]Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Bologna, Italy

E-mail: luca.anzalone2@unibo.it

**Abstract.** Signal-background classification is a central problem in High-Energy Physics (HEP), playing a major role for the discovery of new fundamental particles. The recent Parameterized Neural Network (pNN) is able to leverage multiple signal mass hypotheses as an additional input feature to effectively replace a whole set of individual neural classifiers, each providing (in principle) the best response for the corresponding mass hypothesis. PNNs have the potential to overcome the burden of training a multitude of independent classifiers, enabling interpolation as well as some degree of extrapolation, but also achieving better generalization and data-efficiency as a single model is trained on all the available data. In this work we discuss several design choices, with a particular focus on the conditioning mechanisms, to enable parametric neural networks for real-world physics analyses.

## 1. Introduction

The problem of signal-background classification represents a crucial part of a particle physics analysis. It is of fundamental interest the development of methods able to provide an increasingly better selection efficiency. The aim of this process is to discard how much *background* events as possible, representing everything that is already well-known, while retaining the largest amount of *signal* events: the theorized particle decay(s). Preserving enough signal in a statistically plausible manner is fundamental for physicists, in order to give relevance to their findings. The very classical approach to this problem is to rely on expert knowledge to derive hand-designed threshold or cut rules, applied to one or more manually-selected variables. Such *cut-based approach* has been (partially) replaced by machine learning (ML) methods, such as decision trees (DTs) [1] and neural networks (NNs) [2]. ML methods have the benefit of being able to find higher-order relations (or patterns) in the data that often lead to superior discrimination performance, without the need of either an extensive study of the data nor the design of hand-made rules or features.

Despite such benefits, when physicists want to study their theoretical hypotheses on multiple mass points, even the ML approach starts to be difficult to apply in practice: assuming we want to study our hypothetical signal(s) on $M$ mass values, we would be required an effort that scales as a function of the number of mass hypotheses to test: as $O(M)$, i.e. *linearly*. In practical terms, this means that the *development*, *training*, hyper-parameter *tuning*, *management*, and eventual *deployment* effort is almost multiplied by $M$: the number of chosen mass points. In addition, generating a sufficient amount of data samples for each mass point can be challenging, as *Monte-Carlo simulations* required to generate such samples are notoriously computationally

intensive to run. Fortunately, the introduction of parameterized neural networks (pNNs) [3, 4] provides an elegant solution to such problems.

## 2. Improving parameterized neural networks

Parameterized neural networks [3, 4] are designed to exploit the domain knowledge of one or more *physics parameters*, resulting in an architectural variation of the common neural network design [5]. The physics parameter(s), like the *mass* of the hypothetical particle, is treated as an additional input that is fed to the neural network along with the feature vectors. Let the (scalar) physics parameter be $m$ (since, in our particular case, it represents the signal mass hypotheses), $x$ the vector of particle's feature, and $\theta$ a set of learnable weights (or neural network's parameters); a pNN can be defined as $f_\theta(x, m)$, being a learnable function of the physics parameter, $m$, whereas a canonical neural network would be denoted as $f_\theta(x)$. The dependence among $f_\theta$, $x$, and $m$ is the core idea behind pNNs, that, in practice, is implemented by a proper *conditioning mechanism* [6].

With such an elegant formulation, a single model (the pNN) is trained on all the data and so on all signal mass hypotheses at once (in general, on all combinations of the physics parameters), being an effective replacement for $M$ independent classifiers, each trained on a single mass point. In this way, the effort required for the entire workflow is now *not* a function of the number of mass hypotheses, anymore. Indeed, there is *no free lunch*; there are practical challenges to address, such as: (1) finding the best way to condition on the physics parameter, (2) how to correctly assign the parameter to background events (since it is, in principle, well defined for the signal only), and (3) how to correctly evaluate the pNN. Fortunately, if such problems are correctly solved the pNN offers even more benefits compared to the classical approach, such as: *interpolation* and *extrapolation* of intermediate and boundary values of the physics parameter, and a better *data-efficiency* (since all the data is consumed by a single model, instead of training multiple classifiers on subsets of the dataset), which usually leads to improved generalization.

### *2.1. Conditioning mechanisms*

The conditioning operation can be applied to any kind of data (e.g. vectors, images) and domains. It consists of altering the output of whatever neural network model (the conditional-GAN [7] is a good example) as $z$, called the *conditioning* or *task representation*, varies. The representation $z$ can take various forms: an embedding, a one-hot encoded vector, or a single or multi-dimensional discrete or continuous variable. Simple yet effective mechanisms are [6]:

- **Concatenation-based conditioning**: the physics parameter $m$ (our mass) is first concatenated along the last dimension (axis) of the input features $x$, and the result is then passed through a linear layer. The overall operation is

$$z = W[x\ m] + b, \tag{1}$$

  where $z$ is the *conditioned* representation, and $(W, b)$ are the weights and biases parameters of the linear combination[1]. The concatenation along the last axis is simply denoted by $[\cdot]$, which, in TensorFlow [8], corresponds to `tf.concat([x, m], axis=-1)`.
- **Conditional biasing**: it turns out that the concatenation-based conditioning can be expressed in an equivalent form, in terms of *element-wise addition* (denoted by $\oplus$):

$$z = (Wm + b) \oplus x, \tag{2}$$

  where the conditioning representation ($m$) is first expanded by a linear layer (whose output dimension must match $x$) to yield a *bias vector*, which is then added to the inputs $x$.

[1] In practice, such linear combination corresponds to the application of a `Dense` layer without activation function.

- **Conditional scaling**: an alternative mechanism to both concatenation and biasing, can be built from *element-wise multiplication* (or Hadamard product, $\odot$):

$$z = x \odot (Wm + b) \tag{3}$$

This time the parameter is expanded to a *scaling vector*, which is then multiplied element-by-element to the vector $x$.

*2.2. The affine architecture*

It is not yet clear when one of such mechanisms should be preferred. So, to avoid trying each one at a time, we can design a novel *affine-conditioning* mechanism [6, 4] by combining biasing with scaling, as depicted in figure 1. Such conditioning operations can be implemented as *layers*,

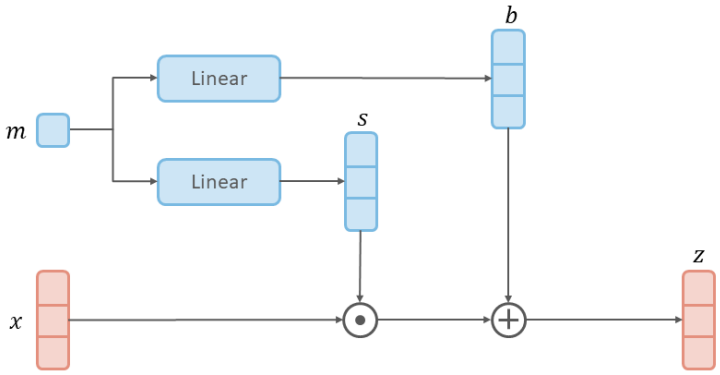


**Figure 1.** The affine-conditioning mechanism: the physics parameter, $m$, is first independently expanded twice, by means of two linear layers, respectively to scaling $s$ and biasing $b$ vectors, which are subsequently multiplied to the inputs, $x$, and finally added to the resulting vector. The resulting conditioned representation, $z$, is yield as follows: $z = x \odot s(m) \oplus b(m)$.

and exchanged together to easily build different neural network architectures. For example, the Baldi's pNN [3] makes use of concatenation-based conditioning to combine $x$ with $m$, right after the input layer, and the result of the conditioning is then subject to various non-linearities until the sigmoid output. Another degree of freedom, when designing parametric architectures, regards the point of the layer hierarchy at which is best to perform the conditioning: at the beginning, right before the output, or in the middle? Intuitively, to make the conditioning stronger we can think of applying the mechanism repeatedly, at multiple levels of the hierarchy. This is how the *affine architecture* [4] is designed, applying the affine-conditioning on the mass parameter after each non-linearity. In practice, the first conditioning is performed on a non-linear combination of the input features (after the first `Dense` layer), then the output is subject to another non-linearity and conditioned again, repeating this until reaching the output layer.

*2.3. Assignment of the physics parameter*

The mass feature (in general, the physics parameters) are only well defined for the signal samples. This can be, for example, because the value is not meaningful when associated to background events. Nonetheless, the assignment of the parameter to the background samples is necessarily required to train the pNN at all. In practice, there are two general strategies to follow [4]:

(i) **Identical distribution**: the parameter for the $i$-th background event is assigned by randomly sampling a value from the finite set $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}$ of unique signal mass hypotheses, i.e., $m^{(i)} \sim \mathcal{M}$. For a generic parameter, $p$, that follows a probability distribution, $D_p$, we just sample from the same distribution: $p^{(i)} \sim D_p$.

(ii) **Different distribution**: in general terms, the idea here is to define a probability distribution from the parameter, $p$, that is different from $D_p$, which is the original law that $p$ follows. Taking the mass parameter as an example, we can define a *uniform distribution*

that covers the entire mass range, i.e., $U(m_1, m_M)$. Also in this case, we sample to assign the mass parameter: $m^{(i)} \sim U(m_1, m_M)$.

It should be noticed that if the parameter is differently distributed between signal and background, as in the second strategy, information about the true class label can be leaked to the model: this may cause an increase in classification performance due to learning from spurious correlations, which should be avoided unless proper regularization is applied.

### *2.4. Balanced training procedure*

To further improve our parametric models we can leverage the domain knowledge about our own data. In general, our background events can belong to $P$ different processes (in this particular case, $P = 1$), and the physics parameter, if discrete[2] or enumerable, can take at most $M$ distinct values. This results in $P \times M \times C$ possible combinations of background process, parameter value, and class labels (for binary classification, like our case, $C = 2$.) These possibilities allow us to organize (or *balance*) the samples within each training *mini-batch* in multiple ways [4]:

- **Class balance**: The class labels ($C$) are balanced within each mini-batch, regardless the signal and background processes. Considering two classes, we balance them by *sampling* events such that half the batch represents the positive class (i.e., the signal), and the other half with background samples.
- **Signal balance**: We build mini-batches such that there is an equal amount of signal events for each distinct value of $M$, implying that each $m \in M$ is present in a batch. In this case, the batch size $B = (M \times B_s) + B_b$ would contain a total of $M \times B_s$ signal samples (where $B_s$ denotes the number of events to consider each time), and $B_b$ background events.
- **Background balance**: An equal amount of background samples are allocated in a mini-batch per process, $P$. This results in a batch size $B = B_s + (B_b \times P)$, where $B_s$ is the proportion of the (unbalanced) signal, and $B_b \times P$ the total number of background events. In this case, all the background processes are contained in a mini-batch.
- **Full balance**: The mini-batches are build considering an equal proportion of samples given all the possible combinations $C \times M \times P$. For two classes the batch size, $B$, is divided in two balanced halves such that $B = (B_s \times M) + (B_b \times P)$, where $B_s$ and $B_b$ have to be chosen such that $B_s \times M$ and $B_b \times P$ are approximately equal.

Adopting the balanced training procedure can be helpful at mitigating the bias that arise from *class frequencies*, especially if some of them are particularly more frequent than others.

## 3. Results

We conduct our evaluation on the `HEPMASS-IMB` [9, 4] dataset, a more challenging version of `HEPMASS` [10, 3] in which a large portion of the training signal events have been removed to create an *imbalance* among mass points, and between class labels as well. The dataset depicts the search of an hypothetical particle $X$, at five mass points: 500, 750, 1000, 1250, and 1500 GeV. There are a total of 26 features (a combination of both low- and high-level variables) that describe each event, plus the 27-th *mass feature* $m_X$: by default a random value from the five available have been assigned to the background events.

### *3.1. The significance ratio metric*

The approximate median significance (AMS) [11, 12] is a formula used for hypothesis testing, which can also provide insights about classification performance since it relates the amount of *true* classified signal, $s_t$, with how much background events, $b_t$, are retained for a given

[2] If continuous, it can be, for example, binned to yield $N_{bins}$ disjoint intervals.

classification *threshold*, $t$. The issue regards the dependency it has on the total number of signal and background events contained in our dataset. Without any weighting, the scale of the AMS of each mass points is different, therefore not comparable. For this reason, the *significance ratio* [4] introduces a *normalization constant* that depends on the mass point, such that:

$$\sigma_{ratio}(m) = \frac{\max_t AMS(t)}{s^m_{max}/\sqrt{s^m_{max}}}, \tag{4}$$

where the ratio $s^m_{max}/\sqrt{s^m_{max}}$ represents the *ideal significance* value for the mass point $m$: assuming a perfect classification scenario in which all the background is rejected, and all the signal is correctly predicted (i.e., $s^m_{max}$). Dividing by the constant enables each $\sigma_{ratio}(m)$ to have the same numerical scale for all mass points, $m$, allowing them to be comparable against each other. Furthermore, the metric is normalized in $[0, 1]$, providing also an intuitive interpretation of classification performance where the best models approaches a significance ratio of 1.

### *3.2. Impact of conditioning mechanism*

When designing pNNs the choice of the conditioning mechanism is an important one. In particular, we evaluated the *type* (concatenation, biasing, scaling, and affine conditioning) and the *position*, i.e. the place where the conditioning happens, such as: right after the input layer (`begin`), placed after every non-linearity (`all`), and just before the output layer (`end`). Table 1 shows the results for two tasks: classification, and interpolation [4]. For both tasks we evaluated the Area Under the Receiver Operating Characteristic and Precision-Recall curves (respectively denoted by `AUROC` and `AUPR`), and the proposed significance ratio ($\sigma_{ratio}$) but computed on the best threshold value. As we can see, conditioning at the `end` usually leads to the worst performance, whereas the biasing and affine conditioning are the best in both scenarios: the latter also being more robust regardless the position at which it happens. Finally, concatenation (at `begin`) attains competitive results.

All the models considered for comparisons have four layers, respectively with 300, 150, 100, and 50 units (resulting in about 70k learnable parameters), with ReLU [13] activation and a `Dropout` [14] probability of 25%. The minimization of the binary cross-entropy loss was conducted by the `Adam` optimizer [15], with a learning rate of $5 \times 10^{-4}$, on a batch size of 1024. The trainable parameters were initialized following the `he_uniform` scheme [16] (and `zeros` for biases), and further regularized by an $l_2$-penalty of $10^{-5}$ and $10^{-6}$ for weights and biases, respectively. Finally, the physics parameter were assigned by following the identical distribution strategy, and no mini-batch balancing occurred to isolate the contribution of the conditioning.

## 4. Conclusions

In this paper we discussed several design choices to improve parameterized neural networks: a recent kind of classifier that is particularly promising for signal-background classification in HEP. We assessed the impact of the conditioning mechanism, a crucial design choice, finding that (1) although concatenation and biasing have an equivalent formulation, in practice performance are expected to differ, and (2) that the affine conditioning yields, on average, the best and more robust performance when facing both classification and interpolation tasks.

## References


[1] S. Chatrchyan *et al.*, "Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC," *Phys. Lett. B*, vol. 716, pp. 30–61, 2012.

[2] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for Exotic Particles in High-Energy Physics with Deep Learning," *Nature Commun.*, vol. 5, p. 4308, 2014.

[3] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, "Parameterized Neural Networks for High-Energy Physics," *The European Physical Journal C*, vol. 76, no. 5, pp. 1–7, 2016.

**Table 1.** Impact of conditioning mechanisms. For interpolation the missing signal mass hypotheses are $\{750, 1000, 1250\}$. Best results are shown in boldface.

| **Conditioning** | | **Classification (%)** | | | **Interpolation (%)** | | |
|---|---|---|---|---|---|---|---|
| Type | Position | AUROC | AUPR | $\sigma_{ratio}$ | AUROC | AUPR | $\sigma_{ratio}$ |
| Concat | begin | 93.09 | 92.20 | 88.84 | 89.20 | 88.13 | 85.67 |
| Concat | all | 90.19 | 88.67 | 87.10 | 84.37 | 83.81 | 83.58 |
| Concat | end | 88.45 | 87.58 | 83.40 | 70.93 | 72.57 | 71.90 |
| Biasing | begin | 92.98 | 92.08 | 88.80 | 87.25 | 86.07 | 84.58 |
| **Biasing** | **all** | **93.12** | **92.23** | **88.95** | 91.44 | **90.36** | 86.99 |
| Biasing | end | 88.57 | 87.68 | 83.69 | 69.89 | 72.36 | 71.86 |
| Scaling | begin | 93.01 | 92.10 | 88.74 | 85.42 | 85.53 | 82.35 |
| Scaling | all | 92.93 | 91.99 | 88.73 | 87.95 | 86.80 | 84.81 |
| Scaling | end | 92.03 | 90.76 | 87.64 | 87.12 | 86.84 | 83.84 |
| Affine | begin | 92.41 | 91.54 | 88.14 | 90.34 | 88.98 | 86.21 |
| **Affine** | **all** | 93.05 | 92.14 | 88.92 | **91.50** | 90.33 | **87.61** |
| Affine | end | 91.98 | 90.79 | 87.45 | 86.78 | 86.40 | 83.97 |


[4] L. Anzalone, T. Diotalevi, and D. Bonacorsi, "Improving parametric neural networks for high-energy physics (and beyond)," *Machine Learning: Science and Technology*, vol. 3, p. 035017, oct 2022.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[6] V. Dumoulin, E. Perez, N. Schucher, F. Strub, H. d. Vries, A. Courville, and Y. Bengio, "Feature-wise Transformations," *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations.

[7] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *CoRR*, vol. abs/1411.1784, 2014.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "TensorFlow: A System for Large-scale Machine Learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

[9] L. Anzalone, T. Diotalevi, and D. Bonacorsi, "HEPMASS-IMB," Apr. 2022. https://doi.org/10.5281/zenodo.6453048.

[10] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, "HEPMASS Dataset - UCI Machine Learning Repository," 2015. http://archive.ics.uci.edu/ml/datasets/HEPMASS.

[11] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl, and D. Rousseau, "The Higgs Boson Machine Learning Challenge," in *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, pp. 19–55, PMLR, 2015.

[12] G. Cowan, K. Cranmer, E. Gross, and O. Vitells, "Asymptotic Formulae for Likelihood-based Tests of New Physics," *The European Physical Journal C*, vol. 71, no. 2, pp. 1–19, 2011.

[13] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011* (G. J. Gordon, D. B. Dunson, and M. Dudík, eds.), vol. 15 of *JMLR Proceedings*, pp. 315–323, JMLR.org, 2011.

[14] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1026–1034, IEEE Computer Society, 2015.