Neural Networks with À La Carte Selection of Activation Functions

Moshe Sipper

Received: 11 August 2021 / Accepted: 18 September 2021 / Published online: 25 September 2021
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2021

Abstract
Activation functions (AFs), which are pivotal to the success (or failure) of a neural network, have received increased attention in recent years, with researchers seeking to design novel AFs that improve some aspect of network performance. In this paper we take another direction, wherein we combine a slew of known AFs into successful architectures, proposing three methods to do so beneficially: (1) generate AF architectures at random, (2) use Optuna, an automatic hyper-parameter optimization software framework, with a Tree-structured Parzen Estimator (TPE) sampler, and (3) use Optuna with a Covariance Matrix Adaptation Evolution Strategy (CMA-ES) sampler. We show that all methods often produce significantly better results for 25 classification problems when compared with a standard network composed of ReLU hidden units and a softmax output unit. Optuna with the TPE sampler emerged as the best AF architecture-producing method.

Keywords Artificial neural network - Activation function

Introduction
Activation functions (AFs) are crucial to the success (or failure) of a neural network (witness the rise in recent years of the ReLU function at the expense of the sigmoid function—see Sect. Activation Functions: Previous Work). With the growing success of deep neural networks over the past decade, AFs have been receiving increased attention, with researchers seeking to design better ones and create networks with better AF layers.

Herein, we propose to take a different route: Rather than hand-design a new AF with specific properties (mathematical or other) and then test its merits, we begin with a large “bag” of functions, some regularly (or sporadically) serving as AFs, some merely being mathematical operations on tensors. We then seek ways by which to build AF layers that can outperform a standard neural-network architecture. We call our approach à la carte, for activation layers: choose, arrange, train, et voilà.

We propose three methods by which AFs can be combined: (1) randomly, (2) through a state-of-the-art hyper-parameter optimizer with a Tree-structured Parzen Estimator (TPE) sampler, and (3) via said optimizer with a Covariance Matrix Adaptation Evolution Strategy (CMA-ES) sampler.

In the next section we survey the literature on AF design, followed by a presentation of our experimental setup in Sect. Experimental Setup. In Sect. Results and Discussion we delineate the results and discuss them. Finally, we offer concluding remarks in Sect. Concluding Remarks.

Activation Functions: Previous Work
A recent survey compared trends in AFs both in research and practice of deep learning [18]. They examined the following 21 AFs: Sigmoid, HardSigmoid, SiLU, dSiLU, Tanh, Hardtanh, Softmax, Softplus, Softsign, ReLU, LReLU, PReLU, RReLU, SReLU, ELU, PELU, SELU, Maxout, Swish, ELiSH, and HardELiSH (for precise definitions the reader is referred to [18]). Highly popular architectures—including AlexNet, ZFNet, VGGNet, SegNet, GoogleNet, SqueezeNet, ResNet, ResNeXt, MobileNets, and SeNet—were shown to have a common AF setup, with Rectified Linear Units (ReLUs) in the hidden layers and Softmax units
AF approaches can be divided into three categories: tunable or learnable AFs, ensemble methods, and theoretically motivated AFs.

**Tunable AFs.** A very recent survey focused on modern trainable AFs [3]. They proposed a two-class taxonomy comprising fixed-shape AFs and trainable AFs. The former included all the classical AFs, such as sigmoid, tanh, ReLU, and so forth, while the latter class contained AFs whose shape is learned during the training phase. The fixed-shape class was further divided into classic and rectified-based, and the trainable class was further divided into parameterized standard AFs and AFs based on ensemble methods. Finally, they looked at trainable non-standard neuron definitions, which “change the standard definition of neuron computation but are considered as neural network models with trainable activation functions in the literature” (e.g., maxout). They noted that in many cases improved performance could be gained using trainable AFs rather than fixed-shape AFs.

Tavakoli et al. [27] introduced SPLASH units (Simple Piecewise Linear and Adaptive with Symmetric Hinges), a class of learnable AFs. SPLASH units are continuous, grounded, and use symmetric hinges that are placed at fixed locations, derived from the data. Compared to nine other learned and fixed AFs, SPLASH units showed superior performance across three data sets and four architectures.

Bohra et al. [6] presented an efficient computational solution to train deep neural networks with learnable AFs, specifically focusing on deep spline networks.

Liu et al. [13] observed that small variations to activation distributions could greatly affect the semantic feature representations in 1-bit CNNs, and introduced generalized AFs with learnable coefficients. Their proposed network attained good performance at substantially lower computational cost.

Agostinelli et al. [1] designed a novel form of piecewise linear AF that was learned independently for each neuron using gradient descent, and showed improvement over deep neural networks composed of static rectified linear units.

Li et al. [12] described an extreme learning machine (ELM) with a tunable AF that was tuned by means of differential evolution based on the input data. The tunable AF was of the form \( G(S, a) \), with \( S \) being the inner product of input and weights combined with bias, and \( a \) being the tunable parameter. They demonstrated improved performance over previous versions of ELM.

Ertuğrul et al. [7] presented trained AFs, where an AF was trained for each particular neuron by linear regression. A different AF was generated for each neuron in the hidden layer. The approach was used in random weight artificial neural networks (RWNs), attaining success rates surpassing RWNs that used traditional AFs.

Scardapan et al. [24] introduced a novel family of flexible AFs based on an inexpensive kernel expansion at every neuron. They showed that their kernel activation function (KAF) networks outperformed competing approaches on a number of benchmark problems.

Some older papers include [25], who introduced a multioutput neural model with a tunable, \( AF, f(S, a) \), with \( S \) being the inner product of input and weights, and \( a \) being a tunable parameter that could change during training. Piazza et al. [22] presented a Look-Up-Table (LUT) AF, offering a preliminary study of an adaptive LUT-based neuron (L-neuron) over some (now-considered) simple problems (XOR, 4-bit parity, and 8-bit parity).

**Ensemble methods.** Maguolo et al. [14] recently presented an ensemble of convolutional neural networks (CNNs) trained with different AFs, and also introduced a new AF—the Mexican Linear Unit. They showed that an ensemble of multiple CNNs that only differed in the AFs outperformed the results of the single CNNs and of naive ensembles made of ReLU networks.

Nandi et al. [17] proposed AF ensembling by majority voting, wherein a single neural network is trained with five different AFs. They showed that model accuracy could be improved for four data sets—MNIST, Fashion MNIST, Semeion, and ARDIS IV—over methods including CNN, RNN (Recurrent Neural Network), and SVM (Support Vector Machine).

**Theoretically motivated AFs.** Koçak et al. [11] introduced a number of new AFs, including: generalized swish, mean-swish, ReLU-swish, triple-state swish, sigmoid-algebraic, triple-state sigmoid, exponential swish, sinc-sigmoid, and derivative of sigmoid AFs. They compared the proposed AFs with some well-known and recently proposed AFs. They found that, “New AFs can combine the advantages of predefined ones and so they perform better than the older ones.”

Rozsa et al. [23] introduced the tent AF and showed that tents make deep learning models more robust to adversarial attacks. The constraining tent AF has the form: \( f(x; \delta) = \max(0, \delta - |x|) \); as noted by the authors, “it is basically built from two ReLUs, where a ReLU and a horizontally flipped ReLU share the same distance (\( \delta \)) from the origin.”

Ohn and Kim [19] investigated the approximation ability of deep neural networks with a broad class of frequently used AFs. They derived the required depth, width, and
sparsity of a deep neural network to approximate any Hölder smooth function up to a given approximation error.

Misra and Mish [15] proposed a novel AF called Mish, defined as \( f(x) = x \cdot \tanh(\text{softplus}(x)) \), and showed its merits over other AFs on a number of challenging data sets.

Gulcehre et al. [9] proposed a novel method to train neural networks with AFs that strongly saturate when their input is large. This was achieved by injecting noise to the AF in its saturated regime and learning the level of noise. They showed that their noisy AFs were easier to optimize, and also attained better test errors, since the noise injected to the activations also regularized the model.

Gomes et al. [8] proposed three new AFs: complementary log–log, probit, and log–log. They compared networks using these AFs to networks using classical AFs, over financial time series data sets. They concluded with recommendations of when to apply these new AFs given the type of time series data.

Having examined the literature we conclude that research into activation functions, much of it very recent, clearly shows a thriving interest in veering away from traditional, popular AFs, attempting to find better networks through improved AFs and AF layers. This line of research is mostly concerned with proposing new forms of AFs, with interesting properties that might improve network performance.

Rather than designing a new AF we take another direction, proposing to combine a slew of known AFs, and showing how this can be done advantageously.

**Experimental Setup**

We used Optuna, a state-of-the-art automatic hyper-parameter optimization software framework [2] (we also considered Hyperopt [5], but it is more dated and less potent). Optuna offers a define-by-run style user API, where one can dynamically construct the search space, and an efficient sampling offers a define-by-run style user API, where one can dynamically construct the search space, and an efficient sampling algorithm and pruning algorithm. Moreover, our experience showed that their noisy AFs were easier to optimize, and also attained better test errors, since the noise injected to the activations also regularized the model.

Gomes et al. [8] proposed three new AFs: complementary log–log, probit, and log–log. They compared networks using these AFs to networks using classical AFs, over financial time series data sets. They concluded with recommendations of when to apply these new AFs given the type of time series data.

Having examined the literature we conclude that research into activation functions, much of it very recent, clearly shows a thriving interest in veering away from traditional, popular AFs, attempting to find better networks through improved AFs and AF layers. This line of research is mostly concerned with proposing new forms of AFs, with interesting properties that might improve network performance.

Rather than designing a new AF we take another direction, proposing to combine a slew of known AFs, and showing how this can be done advantageously.

![Fig. 1](https://example.com/fig1.png)

**Fig. 1** List of 48 AFs used herein. For full descriptions the reader is referred to pytorch.org and the respective cited papers

**PyTorch AFs:** ELU, Hardshrink, Hardtanh, LeakyReLU, LogSigmoid, PReLU, ReLU, ReLU6, RReLU, SELU, CELU, GELU, Sigmoid, Softplus, Softshrink, Softsign, Tanh, Tanhshrink, Softmin, Softmax, LogSoftmax

**PyTorch Math:** Abs, Acos, Angle, Asin, Atan, Ceil, Cos, Cosh, Digamma, Erf, Erfc, Exp, Floor, Frac, GumbelSoftmax, Log, Log10, Neg, Round, Sin, Sinh, Tan, Trunc

**New AFs culled from the literature:**

- **Mish:** \( f(x) = x \cdot \tanh(\text{softplus}(x)) \) [15]
- **Generalized Swish:** \( f(x) = x \cdot \text{sigmoid}(e^{-x}) \) [11]
- **Sigmoid Derivative:** \( f(x) = e^{-x} \cdot (\text{sigmoid}(x))^2 \) [11]
- **CLogLogM:** \( f(x) = 1 - 2 \cdot e^{-0.7 \cdot e^x} \) [8]
optimizing the number of layers as well, we chose not to increase the already-large search space, and focus on AFs alone—our main interest in this paper. We thus opted for 5- and 10-layer networks, values chosen so as to eschew shallow networks while still affording extensive testing of our approach through multiple replicates over multiple data sets.

Each network computational layer was fully connected, and performed the standard linear computation, \( y = Wx + b \), with \( W \) being the weight matrix, \( x \) being the input vector, and \( b \) being the bias (\texttt{torch.nn.Linear}). This was followed by the AF (layer) computation. We used the Adam optimizer [10] with a learning rate of 0.001, cross entropy loss, and a single batch (i.e., the entire training set). The hidden-layer sizes were set to 64 nodes, with the preceding input and succeeding output dimensions depending on the particular data set.

The pseudo-code of the experimental setup is given in Algorithm 1. Each replicate run began with a 70–30% training/test split of the data set. We fit scikit-learn’s [21] \texttt{StandardScaler} to the training set and applied the fitted scaler to the test set. This ensured that features had zero mean and unit variance (often helpful, where neural networks are concerned).

We then executed and compared four different AF architectures:

1. The standard architecture (as seen in Sect. Activation Functions: Previous Work): hidden ReLU units with a Softmax output. Train the network over the training set and record the score over the test set as the replicate’s score. The test-set score equalled prediction accuracy, with the best value being 1 and the worst value being 0.
2. Generate 1000 random networks. A random network is generated by selecting at random with uniform prob-

ability 5 or 10 AFs (for 5 or 10 layers, respectively) from the list of 48 AFs (Fig. 1). Train each random network over the training set and test it over the test set. Record the best test score of the 1000 random networks as the replicate’s score.
3. Run Optuna for 1000 trials with a TPE sampler. Set Optuna to optimize as hyper-parameters the (5 or 10) AFs (for 5 or 10 AF layers). A single trial comprises training the network with a particular set of hyper-parameters supplied by Optuna (a list of AFs) over the training set. Test the best network returned by Optuna (after 1000 trials) over the test set, and record this score as the replicate’s score.
4. As in the previous item, except use Optuna with the CMA-ES sampler.

A network was trained for up to 300 epochs, with early stopping: exit if accuracy improvement from one epoch to the next is less than 0.001 for 10 successive epochs. The code is available at github.com/moshesipper.

Basirat and Roth [4] came closest to what we propose above. They suggested the use of a genetic algorithm to combine 11 AFs through the use of 8 operators: addition, subtraction, multiplication, division, exponentiation, minimum,
Table 1
Experimental results. All scores shown are those of the test sets.

| Data set | Samp  | Feat  | Cls | Lay | Standard | Random | Tpe  | Cmaes | Top  |
|----------|-------|-------|-----|-----|----------|--------|------|-------|------|
| 6        | 20000 | 16    | 26  | 5   | 0.119    | 0.924  | !!   | 0.931 | 0.923 | Tpe  |
|          |       |       |     |     | 0.04     | 0.829  | !!   | 0.896 | 0.817 | Tpe  |
| 32       | 10992 | 16    | 10  | 5   | 0.436    | 0.986  | !!   | 0.986 | 0.984 | Random|
|          |       |       |     |     | 0.103    | 0.979  | !!   | 0.982 | 0.976 | Tpe  |
| 279      | 45164 | 74    | 11  | 5   | 0.509    | 0.961  | !!   | 0.966 | 0.96  | Tpe  |
|          |       |       |     |     | 0.105    | 0.566  | !!   | 0.623 | 0.574 | Tpe  |
| 1044     | 10936 | 27    | 3   | 5   | 0.547    | 0.591  | !!   | 0.53  | 0.483 | Random|
|          |       |       |     |     | 0.387    | 0.513  | !!   | 0.374 | 0.422 | Random|
| 1459     | 10218 | 7     | 10  | 5   | 0.206    | 0.687  | !!   | 0.737 | 0.675 | Tpe  |
|          |       |       |     |     | 0.121    | 0.59   | !!   | 0.625 | 0.577 | Tpe  |
| 1471     | 14980 | 14    | 2   | 5   | 0.551    | 0.878  | !!   | 0.859 | 0.852 | Random|
|          |       |       |     |     | 0.55     | 0.721  | !!   | 0.727 | 0.723 | Tpe  |
| 1476     | 13910 | 128   | 6   | 5   | 0.964    | 0.991  | !!   | 0.991 | 0.989 | Tpe  |
|          |       |       |     |     | 0.21     | 0.983  | !!   | 0.988 | 0.983 | Tpe  |
| 1477     | 13910 | 129   | 6   | 5   | 0.975    | 0.991  | !!   | 0.992 | 0.99  | Tpe  |
|          |       |       |     |     | 0.212    | 0.982  | !!   | 0.988 | 0.983 | Tpe  |
| 1478     | 10299 | 561   | 6   | 5   | 0.964    | 0.98   | !!   | 0.974 | 0.976 | Random|
|          |       |       |     |     | 0.192    | 0.971  | !!   | 0.971 | 0.969 | Random|
| 1481     | 28056 | 6     | 18  | 5   | 0.149    | 0.637  | !!   | 0.699 | 0.642 | Tpe  |
|          |       |       |     |     | 0.127    | 0.463  | !!   | 0.549 | 0.41  | Tpe  |
| 1531     | 10176 | 3     | 5   | 5   | 0.962    | 0.966  | !    | 0.966 | 0.965 | Tpe  |
|          |       |       |     |     | 0.961    | 0.962  | !    | 0.962 | 0.962 | Tpe  |
| 1532     | 10668 | 3     | 5   | 5   | 0.964    | 0.968  | !    | 0.967 | 0.967 | Random|
|          |       |       |     |     | 0.964    | 0.964  | !    | 0.964 | 0.964 | Cmaes|
| 1533     | 10386 | 3     | 5   | 5   | 0.964    | 0.968  | !    | 0.967 | 0.967 | Random|
|          |       |       |     |     | 0.963    | 0.964  | !    | 0.964 | 0.964 | Cmaes|
| 1534     | 10190 | 3     | 5   | 5   | 0.963    | 0.967  | !    | 0.966 | 0.965 | Random|
|          |       |       |     |     | 0.962    | 0.962  | !    | 0.962 | 0.962 | Tpe  |
| 1536     | 10130 | 3     | 5   | 5   | 0.962    | 0.967  | !    | 0.966 | 0.966 | Random|
|          |       |       |     |     | 0.962    | 0.963  | !    | 0.963 | 0.962 | Tpe  |
| 1537     | 28626 | 3     | 5   | 5   | 0.974    | 0.976  | !    | 0.975 | 0.975 | Random|
|          |       |       |     |     | 0.975    | 0.975  | !    | 0.975 | 0.975 | Tpe  |
| 1568     | 12958 | 8     | 4   | 5   | 0.649    | 0.985  | !    | 0.992 | 0.984 | Tpe  |
|          |       |       |     |     | 0.329    | 0.979  | !    | 0.992 | 0.978 | Tpe  |
| 4154     | 14240 | 30    | 2   | 5   | 0.998    | 1.0    | !    | 0.999 | 0.999 | Random|
|          |       |       |     |     | 0.998    | 0.998  | !    | 0.998 | 0.998 | Standard|
| 4534     | 11055 | 30    | 2   | 5   | 0.938    | 0.958  | !    | 0.96  | 0.956 | Tpe  |
|          |       |       |     |     | 0.555    | 0.938  | !    | 0.946 | 0.936 | Tpe  |
| 40985    | 45781 | 2     | 20  | 5   | 0.062    | 0.068  | !    | 0.061 | 0.062 | Random|
|          |       |       |     |     | 0.063    | 0.067  | !    | 0.062 | 0.062 | Random|
| 41027    | 44819 | 6     | 3   | 5   | 0.52     | 0.825  | !    | 0.855 | 0.824 | Tpe  |
|          |       |       |     |     | 0.514    | 0.776  | !    | 0.795 | 0.773 | Tpe  |
| 41526    | 15547 | 60    | 2   | 5   | 0.489    | 0.511  | !    | 0.51  | 0.511 | Random|
|          |       |       |     |     | 0.486    | 0.514  | !    | 0.513 | 0.513 | Random|
| 41671    | 20000 | 20    | 5   | 5   | 0.559    | 0.569  | !    | 0.581 | 0.572 | Tpe  |
|          |       |       |     |     | 0.559    | 0.56   | !    | 0.56  | 0.56  | Random|
| 42641    | 18982 | 79    | 5   | 5   | 0.858    | 0.952  | !    | 0.955 | 0.95  | Tpe  |
|          |       |       |     |     | 0.278    | 0.904  | !    | 0.928 | 0.89  | Tpe  |
| 42670    | 18982 | 79    | 5   | 5   | 0.855    | 0.952  | !    | 0.953 | 0.951 | Tpe  |
|          |       |       |     |     | 0.276    | 0.894  | !    | 0.941 | 0.902 | Tpe  |
Results and Discussion

Table 1 shows the results of all 50 experiments, each row summarizing 30 replicate runs. We performed two tests to assess statistical significance: (1) a 10,000-round permutation test comparing the medians of the respective method’s scores (random AFs or Optuna-designed) with the standard network’s scores; and (2) a 10,000-round permutation test comparing the medians of the top method’s scores with the second-best network’s scores. A ‘!!’ denotes a p value < 0.001 and a ‘!’ denotes a p value < 0.05 (otherwise p value > = 0.05).

For random, tpe, cmaes, we performed a 10,000-round permutation test comparing the medians of the respective method’s scores with the standard network’s scores. For top, we performed a 10,000-round permutation test comparing the medians of the top method’s scores with the second-best network’s scores.

At least one of our newly proposed methods (random network, Optuna with TPE sampler, Optuna with CMA-ES sampler) attained statistically significant improvement in 42 of the 50 experiments. In most cases all three methods were better. Moreover, in several cases improvement was not merely statistically significant, but indeed twofold, threefold, and even more.

Table 2 Frequencies of top-10 AFs in the best networks found by the three methods discussed in the text (random, Optuna with TPE sampler, Optuna with CMA-ES sampler)

| Input      | Hidden      | Output     |
|------------|-------------|------------|
| AF         | Freq        | AF         | Freq        | AF         | Freq        |
| Softshrink | 6.7%        | Softshrink | 6.2%        | Exp        | 11.1%       |
| Sin        | 6.3%        | Abs        | 5.0%        | Sinh       | 8.1%        |
| Abs        | 5.8%        | Sinh       | 4.5%        | Cosh       | 7.5%        |
| SELU       | 4.9%        | Hardshrink | 4.5%        | LogSoftmax | 4.4%        |
| Angle      | 4.5%        | Hardtanh   | 4.5%        | Neg        | 4.0%        |
| Neg        | 4.4%        | Exp        | 4.1%        | Abs        | 4.0%        |
| Tanhshrink | 3.7%        | Sin        | 4.0%        | Softplus   | 3.7%        |
| Hardshrink | 3.5%        | Erf        | 3.8%        | PReLU       | 3.6%        |
| Exp        | 3.5%        | SELU       | 3.6%        | SELU       | 3.6%        |
| Hardtanh   | 3.4%        | ReLU6      | 3.2%        | ReLU        | 3.5%        |

| 10 layers | input      | Hidden     | Output     |
|-----------|------------|------------|------------|
| AF        | Freq       | AF         | Freq       | AF         | Freq       |
| Neg       | 3.8%       | SELU       | 3.7%       | GumbelSoftmax | 9.1%       |
| Sin       | 3.8%       | Sin        | 3.5%       | Exp        | 5.4%       |
| Exp       | 3.7%       | Neg        | 3.2%       | LogSoftmax | 4.2%       |
| Angle     | 3.5%       | Erf        | 3.2%       | CELU       | 4.0%       |
| Sin       | 3.4%       | Hardtanh   | 3.1%       | ELU        | 4.0%       |
| SELU      | 3.3%       | Exp        | 3.0%       | PReLU       | 3.8%       |
| Hardtanh  | 3.3%       | CELU       | 3.0%       | Cosh       | 3.7%       |
| Erf       | 3.2%       | Tanh       | 3.0%       | SELU       | 3.7%       |
| Tanh      | 3.1%       | Atan       | 3.0%       | Neg        | 3.5%       |
| GeneralizedSwish | 2.9% | Sinh | 3.0% | RReLU | 3.3% |

Shown by category: AFs in input layer, AFs in hidden layers, and AFs in output layer.
Table 3 Frequencies of topmost AF in the best networks found by all three methods, by data set

| ds | lay | AF Input | Freq | AF Hidden | Freq | AF Output | Freq |
|----|-----|----------|------|-----------|------|-----------|------|
| 6  | 5   | Sinh     | 13.3%| Exp       | 11.9%| Exp       | 24.4%|
| 10 | Tanh| 10.0%    |      | SELU      | 5.7% | CELU      | 12.2%|
| 32 | 5   | Exp      | 18.9%| Exp       | 12.2%| Sinh      | 14.4%|
| 10 | Sinh| 10.0%    |      | Exp       | 5.3% | CELU      | 11.1%|
| 279| 5   | Sinh     | 7.8% | Sinh      | 8.1% | Exp       | 25.6%|
| 10 | Sin | 5.6%     | Digamma| 5.3% | SigmoidDerivative | 11.1%|
| 1044| 5  | Round    | 17.8%| Abs       | 15.2%| CELU      | 11.1%|
| 10 | ReLU| 6.7%     | SELU  | 4.4% | GumbelSoftmax | 28.9%|
| 1459| 5  | Angle    | 43.3%| Exp       | 9.6% | Softplus  | 20.0%|
| 10 | Angle| 7.8%    | Angle | 6.8% | CELU      | 8.9% |
| 1471| 5  | Angle    | 13.3%| Abs       | 11.9%| Atan      | 11.1%|
| 10 | ReLU6| 5.6%    | Abs   | 5.1% | GumbelSoftmax | 40.0%|
| 1476| 5  | Abs      | 28.9%| Erfc      | 10.0%| Exp       | 16.7%|
| 10 | LeakyReLU| 11.1% | SELU   | 6.2% | Exp       | 11.1%|
| 1477| 5  | Abs      | 31.1%| Abs       | 9.6% | Exp       | 18.9%|
| 10 | Erf  | 7.8%    | ELU    | 5.1% | ELU       | 11.1%|
| 1478| 5  | Cos      | 30.0%| Cos       | 6.3% | Exp       | 15.6%|
| 10 | Cos | 12.2%    | Abs    | 6.4% | GumbelSoftmax | 11.1%|
| 1481| 5  | Sinh     | 23.3%| Sin       | 10.7%| Exp       | 16.7%|
| 10 | CELU| 7.8%    | Exp    | 4.6% | Cosh      | 15.6%|
| 1531| 5  | Softshrink| 13.3%| Softshrink| 11.9%| SELU      | 8.9% |
| 10 | Trunc| 6.7%   | Digamma| 3.9% | CELU      | 5.6% |
| 1532| 5  | Tanhshrink| 12.2%| Softshrink| 11.5%| Neg       | 8.9% |
| 10 | Sinh| 8.9%    | Digamma| 5.1% | Mish      | 7.8% |
| 1533| 5  | Tanhshrink| 14.4%| Hardshrink| 10.0%| Abs       | 6.7% |
| 10 | Exp | 5.6%    | Ceil   | 3.5% | ELU       | 6.7% |
| 1534| 5  | Tanhshrink| 11.1%| Softshrink| 12.6%| PReLU     | 10.0%|
| 10 | Cosh| 7.8%    | Digamma| 3.8% | Softshrink | 6.7% |
| 1536| 5  | Tanhshrink| 12.2%| Hardshrink| 10.7%| LogSoftmax| 10.0%|
| 10 | Cosh| 6.7%    | Digamma| 5.1% | Softshrink | 6.7% |
| 1537| 5  | Sinh     | 13.3%| Softshrink| 9.6% | ReLU      | 8.9% |
| 10 | Exp | 8.9%    | Digamma| 3.6% | ReLU      | 6.7% |
| 1568| 5  | Hardtanh| 8.9% | Abs       | 13.3%| Sinh      | 27.8%|
| 10 | Exp | 8.9%    | Atan   | 4.7% | Exp       | 11.1%|
| 4154| 5  | Softshrink| 16.7%| Hardshrink| 12.2%| GumbelSoftmax| 20.0%|
| 10 | Abs | 11.1%   | Neg    | 3.8% | Floor     | 5.6% |
| 4534| 5  | Abs      | 40.0%| Abs       | 11.9%| Abs       | 12.2%|
| 10 | RReLU| 5.6%   | Sin    | 6.0% | GumbelSoftmax | 35.6%|
| 40985| 5  | Tan      | 22.2%| Hardtanh  | 8.1% | GumbelSoftmax | 23.3%|
| 10 | Digamma| 11.1% | Digamma| 5.6% | GumbelSoftmax | 18.9%|
| 41027| 5  | Exp      | 21.1%| Softshrink| 8.9% | Neg       | 11.1%|
| 10 | Hardtanh| 8.9%  | Erf    | 4.7% | GumbelSoftmax | 40.0%|
| 41526| 5  | Softshrink| 34.4%| Softshrink| 22.6%| Softmin   | 5.6% |
| 10 | Softshrink| 13.3%| Softshrink| 12.2%| GumbelSoftmax | 11.1%|
| 41671| 5  | SELU     | 12.2%| SELU      | 9.6% | GumbelSoftmax | 16.7%|
| 10 | Cosh| 6.7%    | GumbelSoftmax| 6.1% | GumbelSoftmax | 8.9% |
| 42641| 5  | Softshrink| 10.0%| Abs       | 6.7% | Exp       | 28.9%|
| 10 | Angle| 8.9%   | Erf    | 6.1% | Exp       | 12.2%|
| 42670| 5  | Softshrink| 11.1%| Abs       | 8.9% | Exp       | 22.2%|
| 10 | Angle| 10.0%  | SELU   | 7.4% | PReLU     | 11.1%|

*ds*, OpenML data set id; *lay*, number of network layers

Shown by category: AFs in input layer, AFs in hidden layers, and AFs in output layer
A statistically significant number-one ranking (when compared with the second-best) was attained by Optuna using a TPE sampler 17 times, and by the random network 5 times. AF layout design with Optuna would seem to be an excellent choice when one wishes to improve a deep learner’s performance.

In addition to the results given in Table 1, we were now in possession of 4500 architectures (25 data sets × 30 replicates × 2 layer counts × 3 non-standard methods). Given the large search space of architectures (48^5 for 5 layers and 48^{10} for 10 layers) it was not surprising that almost all 4500 architectures were unique (the only exceptions were 2 5-layer networks that appeared twice). We could, however, take a closer at the 33750 AFs within these architectures, and examine their frequencies—which are given in Table 2.

Examining Table 2, we can make several interesting observations. First, we note what is not there: the commonly used ReLU and Softmax functions (they do appear lower down the list, not shown for brevity). We note the preponderance of usually overlooked mathematical functions, which would seem to be a good choice for AF candidates, e.g., sine, hyperbolic sine, negative, and exponential. The diversity of AFs chosen by the methods presented herein is also apparent. Providing a large set of AF candidates as grist for the AF layout-design mill is beneficial and can be fruitfully applied.

Of the four newly proposed AFs (Fig. 1) only Generalized Swish made it into one of the top-10 rankings. We asked whether for particular data sets specific AFs appeared more frequently, an inquiry that led to Table 3. We note that for some data sets certain AFs were selected with a notably higher frequency. In addition, once again we note the preponderance of AFs not often used.

The freedom to choose from a large set of possible AFs and compose AF layouts can produce better-performing networks. It would seem to help reduce error from a model by avoiding overfitting to the training set. As such, it might be acting as a regularizer, performing a beneficial tradeoff by reducing variance significantly without increasing bias (substantially). This regularization effect of à la carte is a conjecture at this point.

Concluding Remarks

Presenting deep networks à la carte, we showed that it is possible to significantly improve performance by randomly choosing non-standard AF architectures. In addition, even better results can be obtained using a state-of-the-art hyperparameter optimizer.

One might inquire as to the cost of searching for better AF layouts. Deep networks come with a plethora of hyperparameters that need to be tuned, and adding AFs into the mix could, therefore, be cost-effective when compared with the possible benefits. Our experience has shown that Optuna is efficient at hyper-parameter optimization in general, and at optimizing AF layouts in particular. Ultimately, whether the cost of à la carte is worthwhile or not depends on how important it is for the user to improve performance for a particular task.

We believe this work may have broad implications as it provides an automated means to attain better deep-learning models, with little input from the user (except, possibly, adding AF candidates to our own plethora). We can further suggest a number of avenues for future research:

- Devise other methods for designing AF architectures.
- Seek a better understanding of the dynamics involved. For example, does the approach indeed perform some form of regularization, as suggested in Sect. Results and Discussion?
- While we noted that virtually all AF layouts were unique, and were able to perform an analysis of AF frequencies, it would be of interest to carry this further. No AF is an island: possibly, certain AFs perform better in the presence of certain others.
- There might be additional functions that could be beneficially added to the “bag of AFs” (Fig. 1).
- Apply à la carte to other types of networks, e.g., convolutional neural networks and recurrent neural networks.
- Go beyond AF selection—to AF construction. As we saw in Sect. Activation Functions: Previous Work, researchers have suggested tailored combinations of previous AFs (or functions that are not AFs), and [4] showed that evolving simple, new AFs is possible. Perhaps this AF-design process could be further enhanced, as we did for difficult design problems, such as designing game strategies [26] and designing novel statistics [16].

Acknowledgements

I thank Zvika Haramaty, Raz Lapid, and Snir Vitrack Tamam for helpful comments.

Declarations

Conflict of Interest The author declares that he has no conflict of interests.

References

1. Agostinelli F, Hoffman MD, Sadowski PJ, Baldi P. Learning activation functions to improve deep neural networks. In: Bengio Y, LeCun Y (eds) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings. pp. 1–9 (2015)
2. Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: A next-generation hyperparameter optimization framework. In:
Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 2623–2631 (2019)

3. Apicella A, Donnarumma F, Isgro F, Prevece R. A survey on modern trainable activation functions. Neural Netw (2021)

4. Basirat M, Roth PM. The quest for the golden activation function (2018). arXiv preprint arXiv:1808.00783

5. Bergstra J, Yamins D, Cox DD, et al. Hyperopt: a python library for optimizing the hyperparameters of machine learning algorithms. In: Proceedings of the 12th Python in science conference, vol. 13, p. 20. Citeseer (2013)

6. Bohra P, Campos J, Gupta H, Aziznejad S, Unser M. Learning activation functions in deep (spline) neural networks. IEEE Open J Signal Process. 2020;1:295–309.

7. Ertugrul ÖF. A novel type of activation function in artificial neural networks: trained activation function. Neural Netw. 2018;99:148–57.

8. Gomes GSDS, Ludermir TB, Lima LM. Comparison of new activation functions in neural network for forecasting financial time series. Neural Comput Appl. 2011;20(3):417–39.

9. Gulcehre C, Moczulski M, Denil M, Bengio Y. Noisy activation functions. In: International Conference on Machine Learning, pp. 3059–3068. PMLR (2016)

10. Kingma D.P., Ba J, Adam: A method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980

11. Koçak Y, Şiray GÜ. New activation functions for single layer feedforward neural network. Expert Syst Appl. 2021;164:113977.

12. Li B, Li Y, Rong X. The extreme learning machine learning algorithm with tunable activation function. Neural Comput Appl. 2013;22(3):531–9.

13. Liu Z, Shen Z, Savvides M, Cheng KT. ReActNet: Towards precise binary neural network with generalized activation function. In: European Conference on Computer Vision, pp. 143–159. Springer (2020)

14. Maguolo G, Nanni L, Ghidoni S. Ensemble of convolutional neural networks trained with different activation functions. Expert Syst Appl. 2021;166:110448.

15. Misra D., Mish: A self regularized non-monotonic neural activation function, 4 (2019). arXiv preprint arXiv:1908.08681

16. Moore JH, Olson RS, Chen Y, Sipper M. Automated discovery of test statistics using genetic programming. Gen Program Evol Mach. 2019;20(1):127–37.

17. Nandi A, Jana N.D, Das S. Improving the performance of neural networks with an ensemble of activation functions. In: 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1–7. IEEE (2020)

18. Nwankpa C, Ijomah W, Gachagan A, Marshall S., Activation functions: Comparison of trends in practice and research for deep learning (2018). arXiv preprint arXiv:1811.03378

19. Ohn I, Kim Y. Smooth function approximation via deep neural networks with general activation functions. Entropy, 2019;21(7):627.

20. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, et al., PyTorch: An imperative style, high-performance deep learning library (2019). arXiv preprint arXiv:1912.01703

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.