An application of ANN hyper-parameters tuning in the field of Recommender Systems

Vaios Stergiopoulos[1][0000−0003−3458−1491], Michael Vassilakopoulos[1][0000−0003−2256−5523], Eleni Tousidou[1][0000−0002−1581−0642], and Antonio Corral[2][0000−0002−0069−4642]

[1] Data Structuring & Eng. Lab., Dept. of Electrical & Computer Engineering, University of Thessaly, Volos, Greece
`{vstergiop,mvasilako,etousido}@uth.gr`
[2] Dept. of Informatics, University of Almeria, Spain
`acorral@ual.es`

Abstract. Over the past years, Artificial Neural Networks (ANNs) have achieved remarkable results for a variety of applications in Machine Learning (ML) models, including Recommender Systems (RS). The selection of the activation function, weight initialization method and training epochs number in ANNs has a major effect on the training phase and the task performance, while directly affecting the convergence of a network. Thereafter, in this work, we carry out the hyper-parameters tuning of a ML RS which utilizes an ANN, called CATA++. We have performed tuning of the activation function, weight initialization and training epochs of CATA++ in order to improve both training and performance. During the experiments, a variety of state-of-the-art activation functions have been tested: ReLU, LeakyReLU, ELU, SineReLU, GELU, Mish, Swish and Flatten-T Swish. Additionally, various weight initializers have been tested, such as: XavierGlorot, Orthogonal, He, Lecun, etc. Moreover, we ran experiments with different epochs number from 10 to 150. We have used data from CiteULike and AMiner Citation Network. The recorded metrics (Recall, nDCG) indicate that hyper-parameters tuning can reduce notably the necessary training time, while the recommendation performance is significantly improved (up to +44.2% Recall).

Keywords: Hyper-parameters tuning· Artificial neural networks· Activation function· Weight initialization· Training epochs· Recommender systems

1 Introduction

Recently, hyper-parameters (HP) tuning of Deep Neural Networks (DNN) has emerged as an important topic. DNN performance, however, is known to be highly sensitive to the HP setting; deep learning researchers often spend long hours trying to tune HP.

The activation function is an important component of neural networks because they turn an otherwise linear classifier into a non-linear one, which has proven key to the high performances witnessed across a wide range of tasks in recent years. While different activation functions seem equivalent on a theoretical level, they often show very diverse behavior in practice. Moreover, they are characterized by a variety of properties, such as ones relating to their derivatives, monotonicity, and whether their range is finite or not [17].

A proper initialization of the weights in an ANN is critical to its convergence [18]. The weights of a network are initialized and then adjusted repeatedly while training the network, till the loss converges to a minimum value and an ideal weight matrix is obtained. Thus, weight initialization directly drives the convergence of a network; the training is accelerated and performance is improved [20].

The training epochs is a form of HP which plays an integral part in the training process of a model [21]. The weights initialized in the beginning will be subject to change when the next cycle of the same dataset simulation (epoch) takes place. The epoch optimization mainly consists of two problems, namely under-fitting and over-fitting.

In this work we aim to optimize the above-mentioned HP of the neural network utilized in a RS called CATA++: A Collaborative Dual Attentive Autoencoder Method for Recommending Scientific Articles [1]. A variety of state-of-the-art activation functions have been tested: ReLU, LeakyReLU, ELU, GELU, SineReLU, Mish, Swish and Flatten-T Swish (FTS). Additionally, various weight initializers have been tested, such as: XavierGlorot Normal and XavierGlorot Uniform, Orthogonal, HeNormal and HeUniform, Lecun Normal and Lecun Uniform, VarianceScaling, RandomNormal, RandomUniform, Identity, Ones and Zeros. Moreover, we have run experiments with different numbers of training epochs in the range [10,150].

The remainder of this paper is organized in the following order. Firstly, all related work is presented in Section 2. Secondly, the essential preliminaries and theoretical background are explained in Section 3. Next, our experimental results are demonstrated thoroughly in Section 4. In Section 5, we discuss about the experimental results and the discovered good practices for HP tuning. Lastly, in Section 6 we conclude this work and discuss related future work directions.

2 Related work

There are numerous scientific publications and ongoing research in the subject of HP tuning in order to improve ANNs performance. The aim of HP optimization is to choose the HP values that return the best results in the validation phase.

To begin with, Alfarhood and Cheng [1] introduce a Collaborative Dual Attentive Autoencoder (CATA++) RS method that utilizes an item's content and learns its latent space via two parallel autoencoders. They employ the attention mechanism in the middle of the autoencoders to capture the most significant segments of contextual information, which leads to a better representation of the items in the latent space. They have utilized Matrix Factorization and Collaborative Filtering in order to improve the recommendation performance. In this work we use extensively the CATA++ RS for our experiments. We have chosen to use CATA++ because it outperformed other RS, as described in [1] and in experiments we have run (due to space limitation can't include them here).

Moreover, lots of research on selection and comparison of activation functions has been conducted in [19], [22], [23] and [24] since it has been proved as one of the most valuable HP in ANNs. Afaq and Rao [21] state that the number of epochs is a form of HP which plays an integral part in the training process of a model and prove the significance of epochs on ANN training. Neary [25] tried to implement an automatic algorithm for HP tuning in Deep Convolutional Neural Networks. Li et al [26] presented parallel ways of tuning more than one HP simultaneously, by introducing a simple and robust HP optimization algorithm called ASHA (Asynchronous Successive Halving Algorithm), which exploits parallelism and aggressive early-stopping. For HP optimization of general machine learning problems, numerous automated solutions have been developed; some of the most popular solutions are based on Bayesian Optimization [27], [28].

Motivated by the ideas in the above-mentioned related work and the lack of optimization for some HP in CATA++, here we aim to tune them and optimize its recommendation performance.

3 Background

Major gains have been recently made in RS due to advances in deep neural networks. However, choosing the proper network architecture for a given problem is still a struggle in deep learning. In the rest of Section 3 we analyze the different choices for each HP we wish to tune in the CATA++ model.

3.1 Activation Functions

Activation functions play a key role in neural networks; therefore it becomes fundamental to understand their advantages and disadvantages in order to achieve reduced training time and better recommendation performance.

Rectifier Linear Unit (ReLU) It is the standard activation function, widely used since it was introduced by Nair and Hinton in 2010 [4], due to its positive impact on different ML tasks. The ReLU activation function has the form:

$$relu(x) = max(0, x) \tag{1}$$

Pedamonti in [5] states that ReLU comes with the aim to solve the vanishing gradient and exploding gradient problems: while using Sigmoid and working on shallower layers doesn't give any problem, some issues arise when the architecture becomes deeper because the derivative terms that are less than 1 will be multiplied by each other many times that the values will become smaller and smaller until the gradient tends towards zero (vanishing). On the other hand, if the values are bigger than 1 then the opposite happens, with numbers being multiplied becoming bigger and bigger until they tend to infinity and explode the gradient. A good solution would be to keep the values to 1, so even when they are multiplied, they don't change. This is exactly what ReLU does: it has gradient 1 for positive inputs and 0 for negative ones. The fact that the gradient is zero might be seen as an issue at first, but it actually helps to make the network sparse, keeping the useful links. Sparsity helps to keep the network less dense and decreases the computation, however once the gradient is zero the corresponding nodes don't have any influence on the network anymore, so they can't contribute to the improvement of the learning. In short, ReLU treats all negative values as unimportant representation. Consequently, Deep Neural Networks have not been benefited from the negative representations. This is called the "dying ReLU problem" and gave origin to many variants of the ReLU, which are trying to solve this issue.

Leaky-ReLU (LReLU) was introduced by Maas et al. [6] and it is an improved version of the ReLU function, where for negative values of x, instead of defining the ReLU value as zero, it is defined as extremely small linear component of x. It can be expressed mathematically as:

$$lrelu(x) = ax, \quad \text{if } x < 0 \quad \text{and} \quad lrelu(x) = x, \quad \text{if } x \geq 0 \tag{2}$$

To overcome the "dying ReLU problem", an alpha parameter (usually $\alpha \in [0.01, 0.1]$) has been added which is indeed the "leak", so the gradient will be small but not zero. This parameter makes the gradient more robust for optimization since now the weight will be adjusted for those nodes that were not active with ReLU. After some performance tests, we selected $\alpha = 0.1$ for our tuning experiments.

Exponential Linear Unit (ELU) [7] is another variant of ReLU which is given as:

$$elu(x) = a \cdot (\exp(x) - 1), \quad \text{if } x \leq 0 \quad \text{and} \quad elu(x) = x, \quad \text{if } x > 0 \tag{3}$$

The ELU HP α controls the value to which an ELU saturates for negative net inputs. The vanishing gradient problem is alleviated because the positive part of these functions is the identity, therefore their derivative is one.

SineReLU [8] is probably the newest variant of ReLU, which is also trying to solve the "dying ReLU problem" that appears when x < 0. SineReLU is described mathematically as:

$$sinerelu(x) = \varepsilon \cdot (sin(x) - cos(x)), \text{ if } x \leq 0 \text{ and } sinerelu(x) = x, \text{ if } x > 0 \tag{4}$$

Since SineReLU is a sinusoidal wave when $x \leq 0$, it is a differentiable function for all input values of x. Also, ε works as a HP, used to control the wave amplitude; there are some default values to be used, usually $\varepsilon \in [0.002, 0.025]$.

Gaussian Error Linear Unit (GELU) is another variant of ReLU proposed by Hendrycks and Gimpel [9], where they state that GELU is $x \cdot \Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function. It can be expressed as:

$$gelu(x) = x \cdot \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right] \tag{5}$$

Mish [10] is a novel, smooth and non-monotonic neural activation function, which can be defined as:

$$mish(x) = x \cdot \tanh(\text{softplus(x)}), \text{ where: softplus}(x) = ln(1 + e^x) \tag{6}$$

Misra [10] states that the properties of Mish, like being unbounded above, bounded below, smooth and non-monotonic, all play a significant role in the improvement of the performance.

Swish is a relatively new activation function introduced by Ramachandran et al. [11] which can be expressed mathematically as:

$$swish(x) = x \cdot sigmoid(x), \text{ where: } sigmoid(x) = (1 + exp(-x))^{-1} \tag{7}$$

Flatten-T Swish (FTS) [12] Despite the fact that ReLU has been popular, its hard zero property has been heavily hindering the negative values from propagating through the network. Chieng et al. [12] proposed FTS that leverages the benefit of the negative values. To construct the FTS activation function, multiply the linear identity part of the original ReLU function (when $x \geq 0$) with the Sigmoid activation function. So, the idea can be expressed by FTS(x) = ReLU(x) * Sigmoid(x) + T, or:

$$fts(x) = x/(1 + e^{-x}) + T, \text{ if } x \geq 0 \text{ and } fts(x) = T, \text{ if } x < 0 \tag{8}$$

The default value of T = 0, but usually is set to be less than zero (T < 0) in order to benefit the network with the representations in the negative form.

3.2 Weight Initialization

This section covers various weight initialization techniques that determine the algorithm by which weights are initially selected for a neural network. We have used various state-of-the-art weight initialization algorithms, that are available online, in the official Keras library[3].

Glorot and Bengio [13] have examined the effect of different HP on training and have also studied the propagation of gradients. The authors commented that

[3] https://keras.io/api/layers/initializers/

the variance of the gradients decreases during backpropagation. An initialization strategy was proposed where the network is initialized with weights from uniform random distribution in a specified range and are later scaled by a scaling factor, which depends on the number of neurons in the previous layer. In our experiments we test both versions: Glorot-Normal (Gaussian distribution) and Glorot-Uniform (Uniform distribution). In the remainder of this article these initializations are referred as GloN and GloU, respectively.

He et al. [14] have modified the scaling factor for weights given by Glorot and Bengio [13] to consider the rectifier non-linearities. This proposed method worked well for the ReLU activation function. Also, a comparison with the weight initialization technique proposed by Glorot and Bengio [13] for a 22-layer model was described. The results have shown that this method performed well, even as the network layers were increased. In this work we test both versions: He-Normal (Gaussian distribution) and He-Uniform (Uniform distribution). These initializations are referred as HeN and HeU, respectively.

Orthogonal weight initialization [15] is a new class of random orthogonal initial conditions on weights that, like unsupervised pre-training, enjoys depth independent learning times. Also, these initial conditions lead to faithful propagation of gradients even in deep nonlinear networks. Moreover, Lecun Normal and Lecun Uniform [16] (referred as LecN and LecU respectively) have been used during our experiments.

Finally, we have experimented with some of the weight initialization classes implemented in the Keras library: Random-Normal, Random-Uniform, Identity, Zeros, Ones. However, these classes are not included in this work because of their significantly poor overall performance.

4 Experiments

In this section, we will try to optimize a neural network by performing HP tuning in order to obtain a high-performing model of CATA++[4]. In the experiments we use different CATA++ versions by changing: activation function, weight initialization and epochs number. We compare the results of the tuned-CATA++ against the default CATA++ (the paper's original version: ReLU-150 epochs). We have used PyCharm Professional IDE, Tensorflow and Keras libraries to run our experiments, and Matplotlib, which is a graph plotting library in python, that serves as a data visualization utility.

4.1 Datasets

In this work, two scientific article datasets are used for experiments. The first dataset, called citeulike-a, was gathered from the CiteULike website which is currently unavailable. CiteULike was a web service that let users create their own library of academic publications. Thus, the user-article interaction data

[4] Code available at https://www.github.com/jianlin-cheng/CATA

came from users that bookmarked the articles they liked. Tags are single-word keywords generated by users when they add an article to their library.

Secondly, experiments were conducted using the DBLP-Citation-network-V13 (2021-05-14)[5] from AMiner [2] available at the time of authoring this work, consisting of 5,354,309 papers and 48,227,950 citation relationships. Using this enormous dataset and executing the algorithms for data preprocessing described in [3], we created three datasets and the necessary input files for CATA++.

In reality, data sparsity is one of the real problems facing RS. Data sparsity for the datasets is calculated based on the user-item interaction. The datasets we used with their characteristics are described in Table 1.

Table 1. Datasets

dataset	#users	#items	#tags	#sparsity	Recommendation type (item)
citeulike-a	5,551	16,980	7,386	99.78%	Scientific publications
dblp13_collection1	6,959	27,025	6,137	99.95%	Scientific publications
dblp13_venues	14,687	2,389	7,187	99.92%	Publication venues
dblp13_people	16,213	21,630	8,144	99.99%	People (collaboration)

4.2 Evaluation Metrics

The evaluation of our experiments is accomplished using two metrics: Recall & normalized Discounted Cumulative Gain (nDCG). Recall (Sensitivity or True Positive Rate) is the number of relevant documents retrieved by a search divided by the total number of existing relevant documents and can be computed as:

$$recall@K = \frac{|\text{Test Articles } \cap \text{ K Recommended Articles}|}{|\text{Test Articles}|} \tag{9}$$

However, Recall does not measure the ranking quality within the top-K list. Therefore, nDCG is used to show the ability of a model to recommend articles at the upper part of the recommendation list. nDCG is computed as:

$$nDCG@K = \frac{1}{|U|}\sum_{u=1}^{|U|}\frac{DCG@K}{IDCG@K} \tag{10}$$

where:

$$DCG@K = \sum_{i=1}^{K}\frac{\alpha(i)}{\log_2(i+1)} \tag{11}$$

$$IDCG@K = \sum_{i=1}^{min(R,K)}\frac{1}{\log_2(i+1)} \tag{12}$$

[5] https://www.aminer.cn/citation

Where U refers to the users number, i is the article rank, R is the number of relevant articles and $\alpha(i)$ is a variable that takes the value of 1 if the article is relevant, and 0 otherwise.

4.3 CiteULike experiments in recommendation of scientific publications

Firstly, using the citeulike-a dataset we ran experiments with the default parameters of CATA++, followed by experiments where different weight initialization techniques where used. During this first experimental phase no other HP was modified. The observed Recall results are recorded in Table 2, where we can compare the performance of the different weight initializers used in the tuned CATA++, to the Default CATA++ performance. We present the Recall values for a different number of recommendations, from K=10 to K=300. The best performance for each column (number K) is highlighted. As someone can realize from Table 2, the best performance is recorded when we are using CATA++ with Glorot-Uniform or He-Normal weight initializations; for the case of K=300, the achieved Recall performance for CATA++ with He-Normal is more than 2% improved.

Table 2. Recall performance @K= 10, 50, 100, 150, 200, 250, 300 recommendations

Weight Init	10	50	100	150	200	250	300
GlorotN	0.0399	0.1245	0.1848	0.2275	0.2606	0.2876	0.3111
GlorotU	**0.0415**	0.1277	0.1905	0.2339	0.2675	0.2945	0.318
Orthogonal	0.041	0.1265	0.188	0.2309	0.2638	0.2902	0.3129
HeNormal	0.0408	**0.1284**	**0.1917**	**0.2351**	**0.2696**	**0.2963**	**0.3198**
HeUniform	0.0403	0.1264	0.1873	0.2302	0.2637	0.2905	0.3137
LecunN	0.0402	0.1265	0.1886	0.2321	0.2655	0.2922	0.3153
LecunU	0.0399	0.1257	0.1875	0.2309	0.2645	0.2918	0.3153
VarianceScaling	0.0379	0.1221	0.1834	0.2272	0.2603	0.2879	0.311
Default CATA++	0.0409	0.1274	0.1892	0.2318	0.2655	0.2936	0.3134

In Fig. 1 we can see the Recall performance of Default CATA++ and the best weight initialization variants: CATA++ with GlorotU and CATA++ with HeNormal. The performance is similar due to the relatively small size and sparsity of the dataset, compared to the other datasets we used. The only difference in performance comes up when we increase the number of recommendations over 250. Similar results are observed for the nDCG metric; all results are shown in Table 3.

In the second experimental phase we tested the range [10,150] of training epochs number for each activation function. The results are surprising, as most of the newest activation functions perform really well with much fewer training epochs than ReLU (default activation for CATA++) which needs 150 training epochs. For example, as can be seen in Fig. 2, the Swish activation records the optimum Recall performance at 60 epochs only. We observe this fast learning because Swish is one of the activations that permits the negative values to influence the network training, so they contribute to the improvement of the learning phase (confronting the "dying ReLU problem").

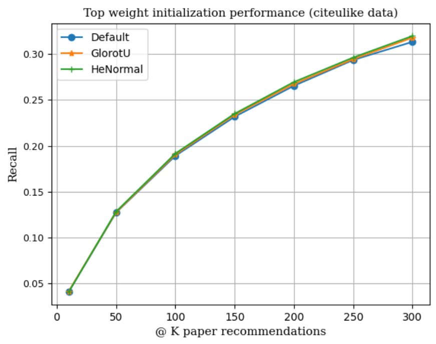

Fig. 1. Top weight init.

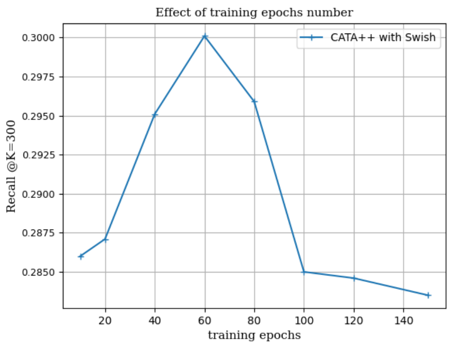

Fig. 2. Effect of training epochs

The full Recall results for all tested activations in CATA++ and for K=300 recommendations, are presented in Table 4. The best performance for each activation is highlighted. In Table 4, we can see that Swish, Mish, GELU, SineReLU, ELU and LeakyReLU perform best for a number of epochs in the range [40,80], as they all utilize the negative values to influence the network training. On the contrary, FTS and ReLU need 150 epochs to achieve their best score (due to the "dying ReLU problem"). The slow training of FTS (150 epochs) was expected, because we used the default value of T = 0; so the representations in the negative form could not benefit the network's training. Therefore, the top 3 recorded performances on Recall @K=300 are: FTS-150, ReLU-150, GELU-40 & SineReLU-60 (both at 3rd position).

After the above-mentioned experiments with citeulike-a, we have created seven tuned versions of CATA++: one version for each activation, along with the most suitable weight initialization and number of epochs. The HP of these seven tuned-CATA++ models are presented in Table 5; *xx* is the epochs number in the [40,80] range.

4.4 AMiner Citation Network experiments

In the present section we continue the experiments with the datasets created by AMiner, in order to further tune the versions of CATA++ described in Table 5.

Recommendation of scientific publications Using the dblp13_collection1 and tuned-CATA++ as a RS for scientific publications we ran the tests of Table 6. The best performance for each K (column in Table 6) is highlighted. We

Table 3. nDCG performance @K= 10, 50, 100, 150, 200, 250, 300 recommendations

Weight Init	10	50	100	150	200	250	300
GlorotN	0.0905	0.111	0.1338	0.15	0.163	0.1734	0.1822
GlorotU	**0.0948**	0.1145	0.1381	0.1547	0.1678	0.1782	0.1871
Orthogonal	0.0932	0.1137	0.1368	0.1531	0.1659	0.1761	0.1846
HeNormal	0.0935	**0.1146**	**0.1385**	**0.1549**	**0.168**	**0.1786**	**0.1874**
HeUniform	0.0922	0.1129	0.1357	0.1519	0.1649	0.1752	0.1839
LecunN	0.0926	0.1131	0.1365	0.1531	0.1661	0.1764	0.1851
LecunU	0.0917	0.1123	0.1356	0.1522	0.1653	0.1758	0.1847
VarianceScaling	0.0884	0.1088	0.1322	0.1485	0.1616	0.1719	0.1808
Default CATA++	0.0933	0.1139	0.1372	0.1537	0.1669	0.1773	0.186

ran many tests in order to select the training epoch number for each model version. In Table 6, each version is recorded with the epochs number that provided the best Recall. Therefore, the top 3 recorded performances on Recall@K=300 are: SReLU-heN-20, Mish-heN-20 and GELU-gloU-40. We can note here that SineReLU and Mish have recorded their best performance at only 20 epochs of training. We had similar results regarding nDCG for the different CATA++ versions, but due to space limitations we can't include them here.

Recommendation of publishing venues The same experiments for the tuned CATA++ versions and the default one, were executed with the dblp13_venues dataset to recommend publishing venues. The Recall performance for all alternatives is presented in Table 7. The nDCG results are similar, so they are not presented here due to article's length limitation. Therefore, the top 3 recorded performances on Recall @K=300 are: LReLU-heN-80, SReLU-heN-40 and Mish-heN-20.

Recommendation of people (collaboration RS) Finally, the same experiments for the tuned CATA++ versions and the default one, were executed with the dblp13_people dataset to recommend researchers for collaboration. The Recall performance for all the alternatives is presented in Table 8. The nDCG results are similar, so they are not presented here due to article's length limitation. Therefore, the top 3 recorded performances on Recall @K=300 are: FTS-heN-150, LReLU-heN-80, SReLU-heN-60 and ELU-gloU-60 (both at 3rd position).

In Fig. 3 (paper RS), Fig. 4 (venue RS) and Fig. 5 (people RS) we present the performance of the top-3 variants compared to default CATA++. As someone can see, there is a significant difference in Recall performance to default CATA++, as HeN and GloU offer better results in these bigger datasets with increased sparsity (compared to citeulike-a) and at the same time SReLU, LReLU, Mish, GELU and ELU use the negative values in order to improve the network's training. Also, FTS benefits from HeN initialization and outperforms all other activations in Fig. 5.

Table 4. Recall performance for the tested activation @K=300 (citeulike-a dataset)

Activation vs Epochs	10	20	40	60	80	100	120	150
FTS	0.2898	0.2907	0.2926	0.3069	0.31	0.314	0.3144	**0.3196**
Swish	0.286	0.2871	0.2951	**0.3001**	0.2959	0.285	0.2846	0.2835
Mish	0.284	0.2907	0.2963	0.297	**0.2984**	0.293	0.2922	0.2913
GELU	0.2995	0.301	**0.3019**	0.3001	0.3014	0.2952	0.2941	0.2933
SineReLU	0.2892	0.2933	0.2975	**0.3018**	0.2994	0.2981	0.297	0.2952
ELU	0.2865	0.2901	**0.2925**	0.2916	0.2892	0.2873	0.286	0.2846
LeakyReLU	0.2918	0.293	0.2941	0.2953	**0.3012**	0.2946	0.2893	0.2887
ReLU (default)	0.2843	0.2898	0.3031	0.3049	0.3087	0.3098	0.3101	**0.3134**

Table 5. The hyper-tuned versions of CATA++

CATA++ Version	activation	weight init.	training epochs
FTS-heN-150	Flatten-T Swish	He-Normal	150
Swish-gloU-xx	Swish	Glorot-Uniform	40-80
Mish-heN-xx	Mish	He-Normal	40-80
GELU-gloU-xx	GELU	Glorot-Uniform	40-80
SReLU-heN-xx	SineReLU	He-Normal	40-80
ELU-gloU-xx	ELU	Glorot-Uniform	40-80
LReLU-heN-xx	LeakyReLU	He-Normal	40-80

Table 6. Recall performance for the different CATA++ versions (Paper RS)

CATA++ Version	10	50	100	150	200	250	300
FTS-heN-150	0.0582	0.1429	0.189	0.2203	0.2461	0.2664	0.2837
Swish-gloU-60	0.0719	0.1757	0.2301	0.2659	0.2921	0.3144	0.3325
Mish-heN-20	0.1135	**0.2393**	**0.2995**	0.3368	0.364	0.386	0.4045
GELU-gloU-40	0.0724	0.1767	0.2329	0.2707	0.2985	0.3223	0.3417
SReLU-heN-20	**0.1144**	0.2391	**0.2995**	**0.3379**	**0.3656**	**0.3884**	**0.407**
ELU-gloU-40	0.0582	0.1559	0.2107	0.2475	0.2758	0.2975	0.3163
LReLU-heN-80	0.0454	0.1303	0.1805	0.2134	0.2384	0.26	0.2783
Default CATA++	0.0495	0.1338	0.1832	0.2166	0.2431	0.2643	0.2823

Table 7. Recall performance for the different CATA++ versions (Venue RS)

CATA++ Version	10	50	100	150	200	250	300
FTS-heN-150	0.2375	0.45	0.5389	0.585	0.6159	0.6393	0.6578
Swish-gloU-40	0.2247	0.4497	0.54	0.5916	0.6237	0.6485	0.6689
Mish-heN-20	0.2153	0.4481	0.5458	0.5947	0.6273	0.6528	0.6736
GELU-gloU-40	0.2234	0.4481	0.5387	0.5872	0.6173	0.64	0.6605
SReLU-heN-40	0.2304	**0.4686**	**0.5675**	0.6174	0.6506	0.6779	0.6976
ELU-gloU-40	0.197	0.4414	0.5407	0.5897	0.6243	0.65	0.6707
LReLU-heN-80	0.2361	0.4645	0.5671	**0.6197**	**0.6543**	**0.6787**	**0.6991**
Default CATA++	**0.2397**	0.4635	0.5555	0.5977	0.6292	0.6515	0.6709

Table 8. Recall performance for the different CATA++ versions (People RS)

CATA++ Version	10	50	100	150	200	250	300
FTS-heN-150	**0.627**	**0.6982**	**0.7259**	**0.7434**	**0.7558**	**0.7636**	**0.7708**
Swish-gloU-60	0.5499	0.6606	0.6978	0.7178	0.7323	0.7431	0.7528
Mish-heN-60	0.5406	0.6639	0.6974	0.7183	0.7322	0.7427	0.7513
GELU-gloU-60	0.521	0.6567	0.697	0.7175	0.7312	0.7421	0.7518
SReLU-heN-60	0.5371	0.6621	0.701	0.7231	0.743	0.7516	0.761
ELU-gloU-60	0.5913	0.6794	0.711	0.7286	0.7421	0.7511	0.76
LReLU-heN-80	0.581	0.6747	0.7101	0.7307	0.745	0.7557	0.7647
Default CATA++	0.5814	0.6622	0.6931	0.712	0.7249	0.7368	0.7459

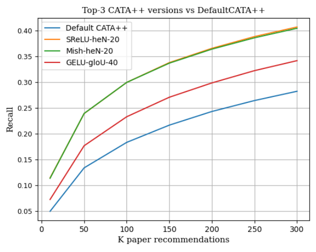

Fig. 3. Top-3 paper RS

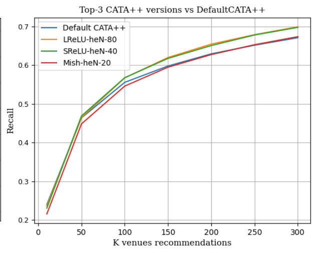

Fig. 4. Top-3 venue RS

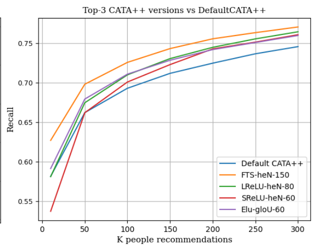

Fig. 5. Top-3 people RS

5 Discussion of the experimental results

Our findings suggest that the weight initializations that provide the best performance are He-Normal and Glorot-Uniform, offering an improvement of more than 2% in the citeulike-a dataset experiments.

In Table 9, the CATA++ versions with the top-3 Recall performances for each experiment are shown, along with the improvement compared to the default CATA++.

One of our major findings is that SineReLU is a really stable activation performing really well in different datasets (always in the top-3 performances in Table 9). A very important new activation is FTS which gave the top performance in two out of four datasets-experiments. While ReLU remains competitive, we have found other activations that usually outperform it; SineReLU, FTS, Leaky-ReLU and Mish are definitely in this category.

Finally, we found that except for ReLU and FTS, the other activation functions produce their highest performance in fewer training epochs, namely in the range [20,80]; the exact number depends on the dataset's and model's characteristics. If FTS is used with $T < 0$, it will probably need fewer epochs too.

6 Conclusion and Future work

We have conducted a large-scale comparison of activation functions and weight initializations using CATA++, a ML RS, and different datasets. Activation func-

Table 9. Top-3 Recall performances for the different experiments (datasets)

dataset-experiment	CATA++ version (Top-3)	Improvement
	FTS-150	**2%**
citeulike-a	Default CATA++	0
(Table 4)	GELU-40 & SReLU-60	0
	SReLU-heN-20	**44.2%**
dblp13_collection1	Mish-heN-20	43.3%
(Table 6)	GELU-gloU-40	21%
	LReLU-heN-80	**4.2%**
dblp13_venues	SReLU-heN-40	4%
(Table 7)	Mish-heN-20	0.4%
	FTS-heN-150	**3.3%**
dblp13_people	LReLU-heN-80	2.5%
(Table 8)	SReLU-heN-60 & ELU-gloU-60	2%

tion, weight initialization and training epochs are among the most important and determinant factors. An activation function turns an otherwise linear classifier into a non-linear one and affects its training, a proper weight initializer is critical for the convergence of the ANN, while the proper number of training epochs helps avoid under-fitting and over-fitting. We showed that the training time (epochs number) can be reduced, while at the same time the recommendation performance can increase up to 44.2%. Regarding future work, our experiments could be expanded to cover other HP, like hidden layers number or latent factors dimension, or other neural networks RS.

References

1. Alfarhood M, Cheng J: CATA++: A Collaborative Dual Attentive Autoencoder Method for Recommending Scientific Articles. IEEE Access 8: 183633-183648 (2020)
2. Tang et al.: ArnetMiner: Extraction and Mining of Academic Social Networks. Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp.990-998 (2008)
3. Stergiopoulos, V., Tsianaka, T. and Tousidou, E.: AMiner Citation-Data Preprocessing for Recommender Systems on Scientific Publications. Proc. 25th Pan-Hellenic Conf. on Informatics, pp. 23-27, ACM (2021)
4. Nair, V. and Hinton, G.E.: Rectified Linear Units Improve Restricted Boltzmann Ma-chines. Proc. 27th Int. Conf. on Machine Learning, pp.807-814, Haifa (2010)
5. Pedamonti, D.: Comparison of non-linear activation functions for deep neural networks on MNIST classification task. CoRR:1804.02763 (2018)
6. Maas, A. L., Hannun, A. Y., and Ng, A. Y.: Rectifier nonlinearities improve neural network acoustic models. Proc. 30th Int. Conf. on Machine Learning (2013)
7. Clevert, D., Unterthiner, T., & Hochreiter, S.: Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). arXiv:1511.07289v5 and ICLR (Poster) (2016)
8. Rodrigues, W.: SineReLU – An Alternative to the ReLU Activation Function, https://wilder-rodrigues.medium.com/sinerelu-an-alternative-to-the-relu-activation-function-e46a6199997d (2018) last accessed 2022/03/05

9. Hendrycks, D., and Gimpel, K.: Gaussian error linear units (GELUS). arXiv:1606.08415v4 (2020)
10. Misra, D.: Mish: A Self Regularized Non-Monotonic Neural Activation Function. CoRR:1908.08681 (2019)
11. Ramachandran, P., Zoph, B.& Le, Q.: Searching for Activation Functions. CoRR:1710.05941 (2017) and ICLR'18 (Workshop) (2018)
12. Chieng, H., Wahid, N., Pauline, O., & Perla, S.: Flatten-T Swish: a thresholded ReLU-Swish-like activation function for deep learning. Int. Journal of Advances in Intelligent Informatics, 4(2), 76-86 (2018)
13. Glorot, X. and Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks, In Proc. of the 13th Int. Conf. on Artificial Intelligence and Statistics, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9 (2010)
14. He, K. et al: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. IEEE Int. Conf. on Computer Vision, pp.1026-1034 (2015)
15. Saxe, A.M., McClelland, J.L. and Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv:1312.6120 and ICLR'14 (2014)
16. LeCun, Y., Bottou, L., Orr, G.B., Muller, K.R.: Efficient BackProp, Neural Networks: tricks of the trade, Springer (1998)
17. Eger, S., Youssef, P., & Gurevych, I.: Is it time to swish? Comparing deep learning activation functions across NLP tasks. CoRR:1901.02671 (2019) and EMNLP'18, pp. 4415-4424 (2018)
18. Kumar, S.K.: On weight initialization in deep neural networks. CoRR :1704.08863 (2017)
19. Hayou, S., Doucet, A., Rousseau, J.: On the Impact of the Activation function on Deep Neural Networks Training. Proc. of the 36th Int. Conf. on Machine Learning in Proc. of Machine Learning Research, 97:2672-2680 (2019)
20. Narkhede, M.V., Bartakke, P.P., & Sutaone, M.S.: A review on weight initialization strategies for neural networks. Artificial Intelligence Review, 55, 291-322 (2022)
21. Afaq, S., & Rao, S.: Significance Of Epochs On Training A Neural Network. Int. Journal of Scientific & Technology Research, 9, 485-488 (2020)
22. Nwankpa, C., Ijomah, W.L., Gachagan, A., & Marshall, S.: Activation Functions: Comparison of trends in Practice and Research for Deep Learning. CoRR:1811.03378 (2018)
23. Sharma, S., Sharma, S., & Athaiya, A.: Activation Functions In Neural Networks, Int. J. of Engineering Applied Sciences and Technology, Vol. 4, Issue 12, pp 310-316 (2020)
24. Hayou, S., Doucet, A., & Rousseau, J.: On the Selection of Initialization and Activation Function for Deep Neural Networks. CoRR:1805.08266v2 (2018)
25. Neary, P.: Automatic Hyperparameter Tuning in Deep Convolutional Neural Networks Asynchronous Reinforcement Learning. IEEE Int. Conf. on Cognitive Computing (2018)
26. Li, L., Jamieson, K.G., Rostamizadeh, A., Gonina, E., Bentzur, J., Hardt, M., Recht, B., & Talwalkar, A.S.: A System for Massively Parallel Hyperparameter Tuning. arXiv:1810.05934v5 and MLSys'20 (2020)
27. Victoria, A.H., & Maragatham, G.: Automatic tuning of hyperparameters using Bayesian optimization. Evolving Systems, 12, 217-223 (2021)
28. Cho, H. et al: Basic Enhancement Strategies When Using Bayesian Optimization for Hyperparameter Tuning of Deep Neural Networks, IEEE Access, vol. 8 (2020)