Automating Autoencoder Architecture Configuration: An Evolutionary Approach

Francisco Charte(✉), Antonio J. Rivera, Francisco Martínez, and María J. del Jesus

Andalusian Research Institute in Data Science and Computational Intelligence, Computer Science Department, Universidad de Jaén, Campus Las Lagunillas, s/n, 23071 Jaén, Spain
{fcharte,arivera,fmartin,mjjesus}@ujaen.es

Abstract. Learning from existing data allows building models able to classify patterns, infer association rules, predict future values in time series and much more. Choosing the right features is a vital step of the learning process, specially while dealing with high-dimensional spaces. Autoencoders (AEs) have shown ability to conduct manifold learning, compressing the original feature space without losing useful information. However, there is no optimal AE architecture for all datasets. In this paper we show how to use evolutionary approaches to automate AE architecture configuration. First, a coding to embed the AE configuration in a chromosome is proposed. Then, two evolutionary alternatives are compared against exhaustive search. The results show the great superiority of the evolutionary way.

Keywords: Deep learning · Autoencoder · Optimization · Evolutionary

1 Introduction

The performance of many machine learning methods mostly depends on the quality of the data patterns. Hence the prevalence of feature engineering (FE) [7] techniques in late years. Feeding the training model with good features greatly improves its predictive ability. This is specially important with high-dimensional and other nonstandard problems [4]. The subset of features can be picked up from the original set of attributes through feature selection [10] procedures. A new reduced set of features holding more information can also be obtained [11], e.g. relying on algorithms such as *Principal Component Analysis* (PCA) [13].

Representation learning [2] is an inherent capability of numerous artificial neural networks (ANNs). Many of them generate this representation as an intermediate step in the full learning process, such is the case of two Deep Learning (DL) models, *Convolutional Neural Networks* and *Deep Belief Networks*. There

This work is supported by the Spanish National Research Project TIN2015-68454-R.

© Springer Nature Switzerland AG 2019
J. M. Ferrández Vicente et al. (Eds.): IWINAC 2019, LNCS 11486, pp. 339–349, 2019.
https://doi.org/10.1007/978-3-030-19591-5_35

are a plethora of DL applications in the neuroscience field, and the high dimensionality problem is usually present in them [17].

In this context *Autoencoders* (AEs) [5] are an interesting tool, since they are mostly devoted to learn new data representations. AEs work in unsupervised fashion, trying to reconstruct the input into the output the best they can while preserving certain restrictions in the coding (hidden) layer. The benefits of AEs compared to classic alternatives such as PCA, specifically in brain disease diagnosis, have been also demonstrated [15].

As usually happens with most ANNs, adjusting the architecture of an AE is not an easy work. There are too many options to perform an exhaustive search of parameters. Therefore, the design frequently is entrusted to the experience of the practitioner or researcher. However, there is no a best AE architecture to all cases as the traits of the data to be processed vary.

Our proposal is to lean on evolutionary approaches (EAs) [1], that usually provide good results in many optimization problems, to design the best AE for every specific data. The main contributions of this paper are the introduction of an scheme to represent the AE architecture as a chromosome and the conducted experiments. These demonstrate that evolutionary methods are able to find good AE configurations in acceptable time.

The rest of this paper is structured as follows. Basic concepts related to FE, AEs and EAs are provided in Sect. 2. Section 3 describes the proposal, detailing the chromosome codification, the EAs to be used and their configuration. The conducted experimentation and its results are covered in Sect. 4. Some final thoughts in Sect. 5 close this work.

2 Preliminaries

This section provides a concise introduction to a few essential concepts, including how FE has been faced until now, what AEs are and the foundations of EAs. Some basic references useful for further study in these fields are supplied.

2.1 Feature Engineering

Feature engineering is a manual or automated task aimed to obtain a set of features better than the original one. Feature selection [10] consists in choosing a subset of attributes while maintaining most useful information in the data. It can be manually performed by an expert in the field, but mostly is faced with automated methods based on feature correlation [12] and mutual information [16]. By contrast, feature extraction methods transform the original data features to produce a new, usually reduced, set of attributes. Popular algorithms to do this are PCA and LDA, whose mathematical foundations are relatively easy to understand.

More advanced studies work with the hypothesis that the distribution of variables in the original data lies along a lower-dimensional space, usually known as *manifold*. A manifold space works with the parameters that produce the data

points in the original high-dimensional space. Finding this embedded space is the task of manifold learning [3] algorithms. Unlike PCA or LDA, manifold methods apply non-linear transformations, so they fall into the non-linear dimensionality reduction [14] category.

2.2 Autoencoders

Autoencoders, as detailed in [5], are ANNs having a symmetric architecture, as shown in Fig. 1. The input and output layers have as many units as features there are in the data. Inner layers usually have fewer units, so that a more compact representation of the information hold in the data is produced. The goal is to reconstruct the input patterns into the output as faithfully as possible.

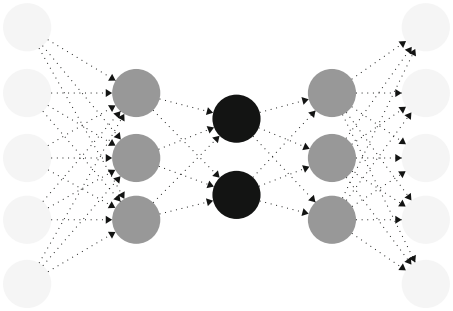

Fig. 1. Classic architecture for an AE. Black nodes denote a 2-variable encoding layer.

Although AEs have many practical applications, the most common one is to perform feature fusion [5], searching the manifold in which the parameters to rebuild the data are found. AEs can be configured with a variable amount of inner layers, each of them having different lengths. The proper architecture will mostly depend on the complexity of the patterns to be reconstructed and the restrictions imposed by the codification layer.

2.3 Evolutionary Optimization

Finding the best parameters to tune a machine learning model is an uphill battle. Performing a grid search through an internal validation process is an usual approach. However, it is useful only for limited sets of parameters taking known ranges of values. Evolutionary algorithms [8] have been used to optimize hyper-parameters for many years, for instance for support vector machines [9] and more recently for deep learning networks [18].

Even though EAs have been also used to optimize ANNs, and even AEs, most of the proposals have been focused on learning the weights linked to each connection. By contrast, the proposal described in the following section is based on EAs to evolve the AE architecture, while weights are learned through the usual back-propagation algorithm.

3 Learning AE Configuration Through Evolution

Next, the approach used to code an AE configuration in a chromosome and the evolutionary methods used to search good configurations are described.

3.1 Coding AE Structure in a Chromosome

Evolutionary algorithms usually work with binary or real-valued genes. A set of genes builds a chromosome or individual of the population. In our case each chromosome will code the configuration of an AE. However, an integer gene representation is used rather than binary or real-valued genes.

The chromosome will be made up of 14 genes, as shown in Fig. 2. The number of each gene is shown above, their names inside and just below the range of values that can be assigned to them. The purpose of each gene, as well as the meaning of its values, are portrayed in Table 1.

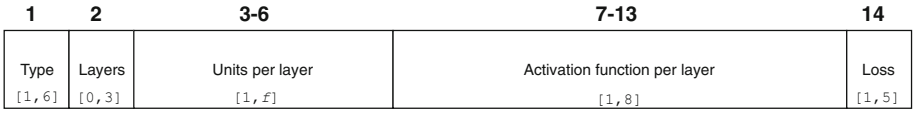

Fig. 2. Chromosome genes, name and interval of values they can get.

Table 1. Purpose of each gene and description of their values.

Name	Purpose	Values
Type	Sets the type of AE to be used	(1) Basic, (2) Denoising, (3) Contractive, (4) Robust, (5) Sparse, (6) Variational
Layers	Number of additional layers in coder/decoder	(0) Only a coding layer, (1–3) Additional layers in both coder and decoder
Units	Set the number of units per layer, with f being the amount of features in the dataset	The first integer (gen 3) configures the number of units in the outer layer, while the last one (gen 6) sets the coding length
Activ.	Activation function to use in each layer, both for the coder and decoder	(1) linear, (2) sigmoid, (3) tanh, (4) relu, (5) selu, (6) elu, (7) softplus, (8) softsign
Loss	Loss function to evaluate during fitting	(1) Mean squared error, (2) Mean absolute error, (3) Mean absolute percentage error, (4) Binary crossentropy, (5) Cosine proximity

As can be easily seen, the search space is huge. Excluding genes 3–6, whose values would vary depending on the number of features in the dataset, there are more than 250 million combinations: $Type \times Layers \times Activation^7 \times Loss$. For small datasets having only a few dozens of attributes, this number will grow to several billions, reaching the trillions of solutions or even more for high-dimensional datasets. Evaluating all those solutions to find the best one is currently unfeasible. Therefore, searching the optimal AE configuration will be not always possible by brute force. However, we could find good enough solutions through optimization mechanisms based on evolution strategies.

3.2 Evolutionary Approaches

We propose two different evolutionary ways of attacking the outlined problem. Both of them will use the former chromosome representation. These two approaches are:

- **Genetic algorithm (GA).** A classical genetic algorithm, in which a population of individuals evolves through a crossover operator, to give rise to new ones, and to which a mutation operator is applied with a certain probability.
- **Evolution strategy (ES).** An aggressive solution-seeking procedure, working with a few individuals who give rise to new ones exclusively through mutation.

Table 2 summarizes the main parameters used to run these methods. Each gene in the chromosome is mutated with a probability of 1/15, value based on the chromosome length itself. Elitism is used in the GA to preserve the tenth percent of individuals having better fitness.

For the GA, crossover points have been established according to the diagram in Fig. 2. This allows the two individuals acting as parents to interchange several of their genes to produce childhood.

Table 2. Main parameters of the evolutionary algorithms.

Parameter	GA	ES
Population size	50	4
Iterations	100	500
Prob. mutation	1/15	1/15
Elitism (individuals)	5	NA
Termination cost	0	0

Regarding the fitness function that will decide the quality of the solutions, it will be computed as shown in (1), where *trainloss* is the reconstruction loss produced by the AE with training data, *Layers* is the number of additional hidden layers (gen 2), *Unitscode* is the size of the codification layer (gen 6),

and α is a coefficient setting the level of penalization applied according to the complexity of the AE.

$$fitness = trainloss + \alpha(Layers \times Unitscode) \tag{1}$$

4 Experiments

In order to test the ability of evolutionary algorithms to find good solutions in a reasonable time, two experiments have been carried out. The first one is a small-scale experiment with the `sonar` dataset, while the second is a large-scale one using the well-known `MNIST` dataset. Exactly the same hardware[1] and software[2] configuration has been used in both cases. The high-level interface provided by the `ruta` [6] package has been used to configure the AEs. The penalization factor has been set as $\alpha = 1 \times 10^{-4}$, so that simpler architectures are preferred over complex ones for AEs with similar performance.

Three runs are made for each experiment. One will use the GA to look up for a solution, other will rely on the ES approach, and the last one will try an exhaustive search. In the latter case the experiment is run for 24 hours, since it is impossible to evaluate all existing configurations. Publicly available training/test partitions were used.

4.1 Small-Scale Case Study

As it happens with all neural networks having several layers, adjusting AE parameters through training is a time consuming process. The more units there are in these layers the longer it will take. Because of this our first experiment is small-scale with the `sonar` dataset, having only 60 input features. This will be the number of units in the input and output layers, as well as it would be the maximum amount of units in the hidden layers. The goal is to obtain a lower-dimensional codification preserving enough information to reconstruct the original data.

Having at most 60 units per layer implies that there will be 3.26×10^{15} possible AE configurations. Assuming we were able to evaluate one AE per millisecond in a machine running 24/7, we would need $> 100\,000$ years to find the optimal solution.

The plots in Fig. 3 give a clear glimpse of each algorithm behavior. The top row shows the quality[3] (Y axis) of evaluated solutions by each approach through the running time (X axis). The GA tests some bad solutions at the beginning (left), but it quickly focuses on the better ones. ES only uses mutation and it produces some bad solutions from time to time, but most of them are quite good. By contrast, the exhaustive search is not guided by any strategy, so quite heterogeneous results are obtained.

[1] 1 PC, CPU Core i5, 16 GB RAM, GPU Nvidia RTX-2080.
[2] GNU/Linux, Tensorflow and Keras.
[3] Measured by the reconstruction mean squared error expressed as percentage.

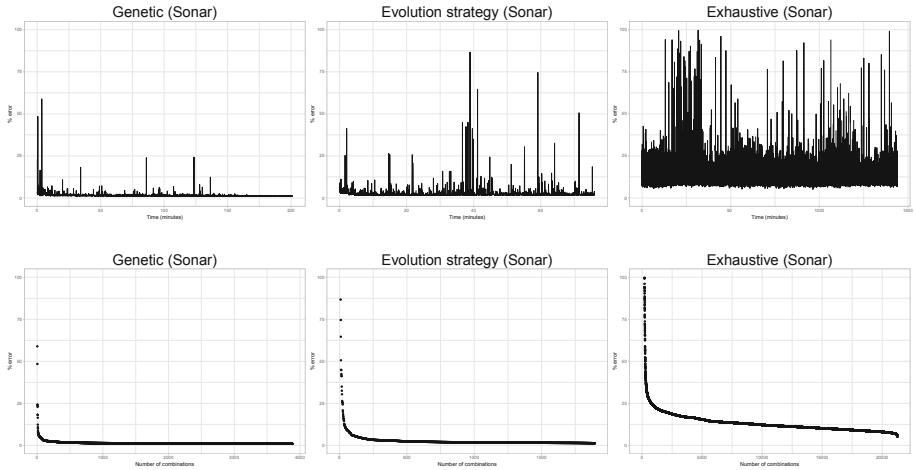

Fig. 3. Analysis of results from the Sonar dataset.

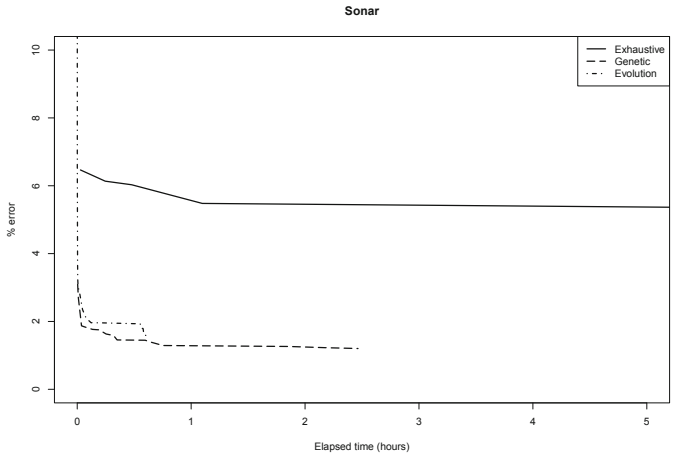

Fig. 4. Comparative convergence of three approaches on the Sonar dataset.

The bottom row in Fig. 3 shows the same data but with all tested configurations sorted by committed error. As can be seen, both the GA and the ES have most of the configurations close to the baseline of the Y axis (% error), while the exhaustive search is not able to reach this point.

In addition to the quality of the solutions, it is also interesting to know how quickly each method converges to the best solution they are able to find. This is represented in Fig. 4. It can be noted that ES is the first method to complete its work. It is quite fast at the beginning, mutating the initial bad solutions into others much better. The GA takes a little more time and it is able to find solutions a little more precise than the ES. The convergence of the exhaustive

search is too slow, and the improvement achieved from the 5th hour (period presented in this plot) until the end of the 24th hour was negligible.

4.2 Large-Scale Case Study

The MNIST dataset, containing handwritten digit images, was used for the large-scale study. The images are 28×28 pixels, so it has 784 input features. Following the former reasoning for `sonar`, in this case we have 9.51×10^{19} potential configurations. It will take more than 3 billion years of computation time to evaluate all of them.

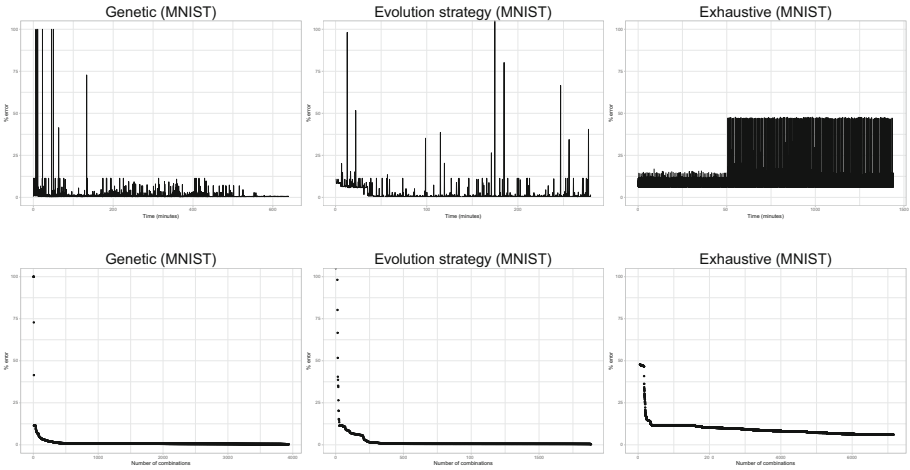

Fig. 5. Analysis of results from the MNIST dataset.

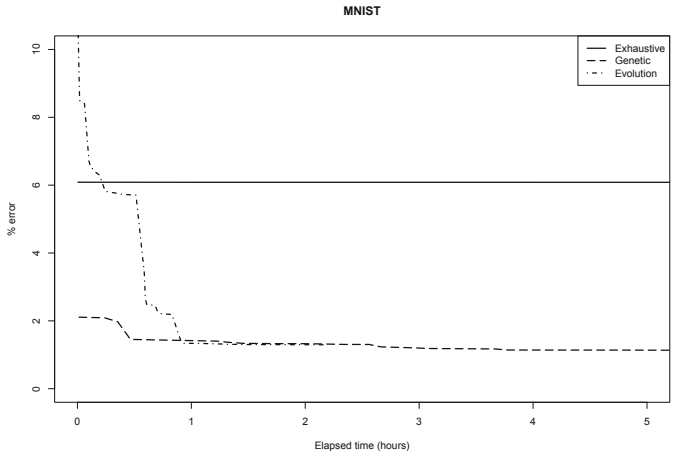

Fig. 6. Comparative convergence of three approaches on the MNIST dataset.

Solutions tested through time and all solutions sorted by quality have been represented in Fig. 5. Once again, the GA forgets bad solutions faster than the ES does. Moreover, it seems that the ES takes longer to reach the same baseline than the GA. The exhaustive approach, with its non-guided search, explores lots of bad solutions through time.

The convergence plot for this dataset (see Fig. 6) is quite similar to that of `sonar`. Once again, the ES quickly reduces the error and it achieves better solutions than the GA in less time. The GA spends more time, and eventually it seems to reach the same performance as the ES but several hours later. As it would be expected after analyzing its behavior in Fig. 5, the exhaustive search almost stalled in the same error level for all the running time.

4.3 Summary of Results

The exact running time, number of evaluated configurations and error percentage for the best one are provided in Table 3. From the analysis of these results the following consequences can be drawn:

- The non-guided exhaustive search evaluates a larger amount of AE configurations, but it is not able to reach the reconstruction performance level of GA and ES.
- GA and ES almost achieve the same error levels. The GA returned a slightly better configuration for `sonar`, while the ES found the best one for `MNIST`.
- Although it starts with worse solutions than the GA, the ES takes less time to hit at the same level.

Table 3. Summary of results.

	Approach	Running time	Configurations	Error (%)
Sonar	Exhaustive	24 h	21 262	7.093
	Genetic algorithm	3 h 21 m	4 505	1.180
	Evolution strategy	1 h 16 m	4 001	1.553
MNIST	Exhaustive	24 h	7 153	6.084
	Genetic algorithm	10 h 39 m	4 505	0.607
	Evolution strategy	4 h 40 m	4 001	0.560

Overall, if we are willing to sacrifice a minimal performance advantage in some cases, the ES seems the best approach to find a good AE configuration for any dataset, even in cases as `MNIST` with hundreds of features and several dozens of thousands of instances.

5 Final Thoughts

AEs are a useful tool to perform manifold learning, but setting the most appropriate AE architecture for every case is a difficult task. Internal cross-validation is an usual approach for tuning hyper-parameters. However, when the structure of the AE has to be adjusted the search space is huge. Therefore, more powerful ways of facing this problem are needed.

In this paper we propose the use of EAs to find the best AE architecture for each dataset. This may not be the optimal, but it is the best that can be found in a reasonable time. The conducted experiments demonstrate that ES and GA are competitive methods to accomplish the job.

References

1. Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evol. Comput. **1**, 1–23 (1993)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013)
3. Cayton, L.: Algorithms for manifold learning. Technical report, University of California at San Diego (2005)
4. Charte, D., Charte, F., García, S., Herrera, F.: A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations. Prog. Artif. Intell. **8**(1), 1–14 (2018). https://doi.org/10.1007/s13748-018-00167-7
5. Charte, D., Charte, F., García, S., del Jesus, M.J., Herrera, F.: A practical tutorial on autoencoders for nonlinear feature fusion: taxonomy, models, software and guidelines. Inf. Fusion **44**, 78–96 (2018)
6. Charte, D., Herrera, F., Charte, F.: Ruta: implementations of neural autoencoders in R. Knowl.-Based Syst. **174**, 4–8 (2019, in press). https://doi.org/10.1016/j.knosys.2019.01.014
7. Domingos, P.: A few useful things to know about machine learning. Commun. ACM **55**(10), 78–87 (2012)
8. Freitas, A.A.: A review of evolutionary algorithms for data mining. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 371–400. Springer, Boston (2009). https://doi.org/10.1007/978-0-387-09823-4_19
9. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple svm parameters. Neurocomputing **64**, 107–117 (2005)
10. García, S., Luengo, J., Herrera, F.: Data Preprocessing in Data Mining. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10247-4
11. Guyon, I., Elisseeff, A.: An introduction to feature extraction. In: Guyon, I., Nikravesh, M., Gunn, S., Zadeh, L.A. (eds.) Feature Extraction, pp. 1–25. Springer, Heidelberg (2006). https://doi.org/10.1007/978-3-540-35488-8_1
12. Hall, M.A.: Correlation-based feature selection for machine learning. Ph.D. thesis, University of Waikato Hamilton (1999)
13. Hotelling, H.: Analysis of a complex of statistical variables into principal components. J. Educ. Psychol. **24**(6), 417 (1933)
14. Lee, J.A., Verleysen, M.: Nonlinear Dimensionality Reduction. Springer, New York (2007). https://doi.org/10.1007/978-0-387-39351-3

15. Martinez-Murcia, F.J., et al.: Deep convolutional autoencoders vs PCA in a highly-unbalanced Parkinson's disease dataset: a DaTSCAN study. In: Graña, M., et al. (eds.) SOCO'18-CISIS'18-ICEUTE'18 2018. AISC, vol. 771, pp. 47–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-94120-2_5
16. Peng, H., Long, F., Ding, C.H.Q.: Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans. Pattern Anal. Mach. Intell. **27**, 1226–1238 (2005)
17. Segovia, F., Górriz, J., Ramírez, J., Martinez-Murcia, F., García-Pérez, M.: Using deep neural networks along with dimensionality reduction techniques to assist the diagnosis of neurodegenerative disorders. Logic J. IGPL **26**(6), 618–628 (2018)
18. Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.H., Patton, R.M.: Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, p. 4. ACM (2015)