# Meta-Learning for Artificial Neural Network Hyper-Parameter Optimization for CERN CMS Offline Data Certification

**Mantas Stankevicius, Virginijus Marcinkevicius, Valdas Rapsevicius**

Vilnius University, Institute of Data Science and Digital Technologies. Akademijos str. 4, LT-08412 Vilnius, Lithuania

E-mail: mantas.stankevicius@mif.vu.lt

**Abstract.** The Compact Muon Solenoid (CMS) is one of the general-purpose detectors at the CERN Large Hadron Collider (LHC) which collects enormous amounts of physics data. Before the final physics analysis can proceed, data has to be checked for quality (certified) by passing a number of automatic (like physics objects reconstruction, histogram preparation) and manual (checking, comparison and decision making) steps. Last manual step of decision making is very important, error-prone and demands a lot of manpower. Decision making (certification) is currently under active research in computer science for automation by applying recent advancements from computer science, specifically, machine learning (ML).

Ultimately, CMS data certification is a binary classification task where various ML techniques are being investigated for applicability. Just like in any other ML task the hyper-parameter tuning is a difficult problem, there is no golden rule and each use case is different. This study explored meta-learning applicability, it is a hyper-parameters finding technique where algorithm learns hyper-parameters from previous training experiments. An Evolutionary genetic algorithm has been used to tune hyper-parameters of a neural network, like number of hidden layers, number of neurons per layer, activation functions, dropouts, training batch size and optimizer. Initially, the genetic algorithm takes manually specified set of hyper-parameters and then evolves towards the near-optimal solution. Genetic stochastic operators, crossover and mutation, were applied to avoid local optimal solutions.

This study shows that by carefully seeding the initial solution the optimal is likely to be found. The proposed solution has improved AUC score of neural network used for CERN CMS data certification. Similar algorithm can be applied for other machine learning models for hyper-parameter optimization.

## 1. Introduction

Usually, hyper parameters for a given problem are chosen conventionally and then tested experimentally. However, this requires a significant amount of experience, intuition, and trial and error. Two most common and simplest methods are exhaustive (grid) and random search. In the grid search, a pool of values for each hyper parameter is hand-picked by an expert then a full set of all value combinations is constructed for later evaluation. Grid search is easy to implement and make it multi-threaded, however the number of combinations grows exponentially with the number of hyper parameters. Quality of this method highly depends on intuition and knowledge of an expert defining pool of values. Hyper parameter areas of high importance might be under-examined, while low importance areas might become over-examined. Random search, on the other hand, tries random combinations of value combinations, usually defined as ranges. This method is also easy to implement and make multi-threaded. Empirical studies show that in high dimensional space random search is much faster than grid search and performs equally well [1]. Study below explores meta-learning applicability for hyper parameter optimization.

## 2. Previous Work

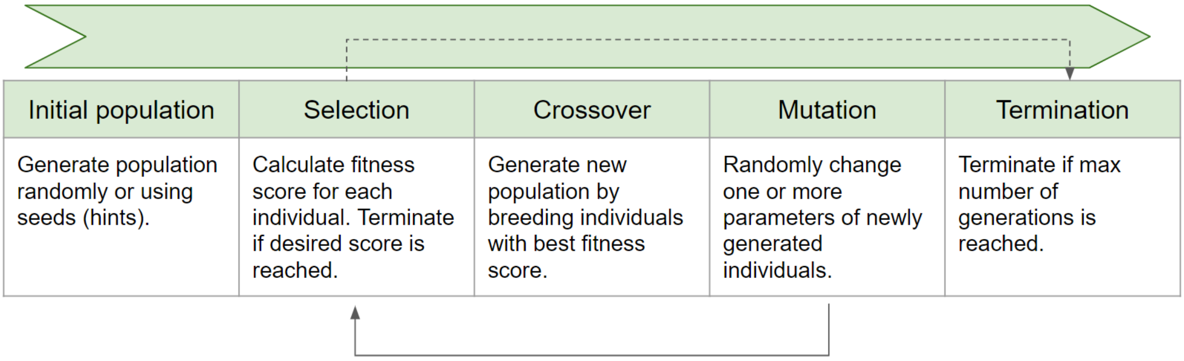
This study is a continuation of previous research [2]. Lack of knowledge about the dataset itself led to poor initialization of parameter pool, so both grid and random search techniques did not provide a decent solution. Manually tuned hyper parameter values proved to be the best choice. The aim of this work is to find a better set of hyper parameters to improve the classification power.

## 3. Dataset

The dataset which was used in this work was collected by the CERN CMS experiment during 2016 data-taking. This is a reconstructed physics data which contains values of various physics objects: photons, muons, others. Dataset contains around 160.000 observations, each representing a single lumisection (a period of approximately 23 seconds). Each observation is based on the particles of the recorded collision events and consists of 401 features like energy, eta, phi and others. Each feature is a vector of 7 numeric values - mean, RMS and five quantiles - representing specific statistical characteristics of the feature distribution during the lumisection. The class (GOOD:BAD) distribution ratio is 98:2.

## 4. Methodology

Genetic evolutionary algorithms [3] are inspired by nature and natural selection. Evolution begins from the randomly generated initial population. All individuals are scored by the fitness function and only best ones are used for reproduction. Genetic stochastic operators are applied to generate new population by avoiding local optimal solutions. Artificial neural network hyper parameter optimization using genetic evolutionary algorithm is called neuro-evolution [4].



| Initial population | Selection | Crossover | Mutation | Termination |
|---|---|---|---|---|
| Generate population randomly or using seeds (hints). | Calculate fitness score for each individual. Terminate if desired score is reached. | Generate new population by breeding individuals with best fitness score. | Randomly change one or more parameters of newly generated individuals. | Terminate if max number of generations is reached. |

**Figure 1.** Steps of the genetic algorithm

### *4.1. Initial population*

An artificial neural network with all its parameters (optimizer, number of layers, neurons, activation function and dropout) represents an individual which is a solution to the problem. The initial population can be generated randomly using parameter pool (Table 1) as well as seeded with potential combinations where optimal solutions are likely to be found.

### *4.2. Selection*

The fitness function determines the quality of an individual and it provides the fitness score. For this algorithm, the average ROC AUC score was chosen from 3-fold cross validation. Cross validation is a technique for evaluating ML model performance. The model was trained on random data set splits and the average ROC AUC value was used for comparison. This technique makes the score independent from the dataset split. All possible pairs of individuals are created and sorted by the sum of fitness score. Pairs having the best fitness score are used to produce the new population.

### *4.3. Crossover*

Crossover is the most important part in a genetic algorithm. This process defines how new offspring is created and how it inherits genes from parent individuals. There is no golden rule, therefore crossover implementation varies between different algorithms and represents a space for innovations. The algorithm treats pairs of parents differently depending on their similarity. A pair of two neural networks is considered similar if the activation functions match layer-wise. For example: two neural networks with three hidden layers (sigmoid, sigmoid, tanh) and (sigmoid, sigmoid, tanh) are considered similar. Such pair produces only one offspring. Categorical (activation, optimizer) hyper parameter values were randomly chosen from one of its parents. Numeric hyper parameter values were treated differently - a random value was selected in the range between parent values, a picked value must exist in the initial parameter pool (see Figure 2). In order to prevent for population individuals to have the same activation function, a pair of neural networks having different activation functions would produce two offsprings, each parent was cloned and mutated with 100% of chance.

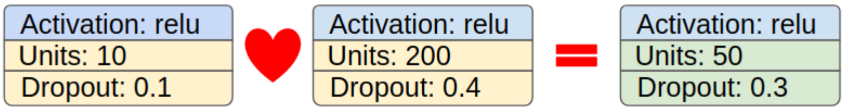


**Figure 2.** Crossover of two parents creates an offspring

### *4.4. Mutation*

The last step in offspring generation process is mutation. Occasionally, one or more parameters can be randomly altered (Figure 3). The new value was randomly chosen from initial parameter pool (Table 1). The mutation rate is a configurable parameter. The purpose of this step is to help prevent local optimal solutions. The mutation rate and amount are very sensitive parameters, high values can move the solution away from high importance areas and greatly reduce quality of the solution.

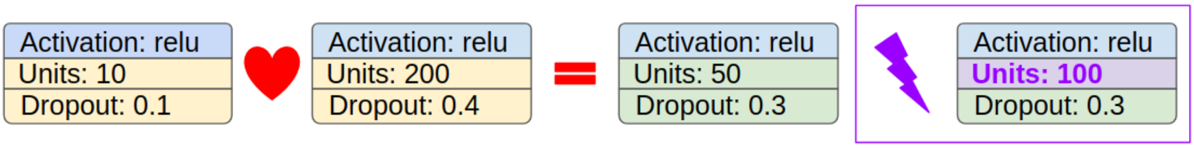


**Figure 3.** Mutation. Randomly alter one or more parameters

## 5. Experimental Results

*Experimental Setup.* Software: Python (3.6), Keras (2.1.6) [5], Tensorflow-gpu (1.8.0) [6], scikit-learn (0.19.1) [7]. Hardware: PC (4 cores 2.2 GHz, 16 GB RAM) with NVIDIA GeForce GTX

1080 Ti GPU.

**Table 1.** Hyper parameter pool used to create initial population and configuration of genetic algorithm

| Hyper Parameters | Pool |
|---|---|
| Units (# of neurons) | [10, 50, 100, 200, 500, 750, 1000, 1500, 2000] |
| Activation | [relu, elu, sigmoid, tanh] |
| Dropout | [0.0, 0.1, 0.2, 0.3, 0.4, 0.5] |
| Optimizer | [adam, rmsprop, sgd, adagrad, adadelta, adamax] |
| Genetic Algorithm Parameters | |
| Population size | 50 |
| # of generations | ∞ (while improves) |
| Mutation rate | 0.1 (10%) |

The discussed genetic algorithm implementation does not try to optimize the topology of a neural network. Topology is defined at the beginning of algorithm and does not change during evolution. We did experiments with 1-3 network layers, however we didn't observe significant improvement with deeper networks, so it was decided to continue experiments with a single layer neural network. The baseline score of ROC AUC from previous study was 0.954. Two hyper parameter search methods were tested - grid search and genetic algorithm.

The grid search method evaluated 1080 combinations of hyper parameter values. Evaluation is a 3-fold cross validation and it alone took around 6 minutes. The evaluation mechanism was same for both search methods. Grid search managed to find a very good solution and improved the ROC AUC score from 0.954 to 0.969, however computations took about 108 hours (4.5 days).

The first population of genetic algorithm was generated randomly. A significantly improved parameter pool allowed a genetic algorithm to reach baseline score after the first evolution (Figure 4). The score kept improving and after 4-5 evolutions (200-250 trainings) the ROC AUC score stabilized around 0.968 and did not show significant improvements. Computations took 20 to 25 hours

Both methods improved the baseline score and found similarly good solutions, however computation time was highly reduced by the genetic algorithm implementation. See Table 2.

**Table 2.** Neural network classification results and parameters

| Method | AUC | ± | Trainings | Optimizer | Network Layers |
|---|---|---|---|---|---|
| Hand picked [2] | *0.954* | 0.005 | unknown | Adam | 3 x [Relu (200) DO (0.5)] |
| Grid search | **0.969** | 0.002 | 1080 | Adagrad | Sigmoid (2000) DO (0.2) |
| Meta learning | 0.968 | 0.002 | **200** | Adagrad | Sigmoid (1000) DO (0.2) |

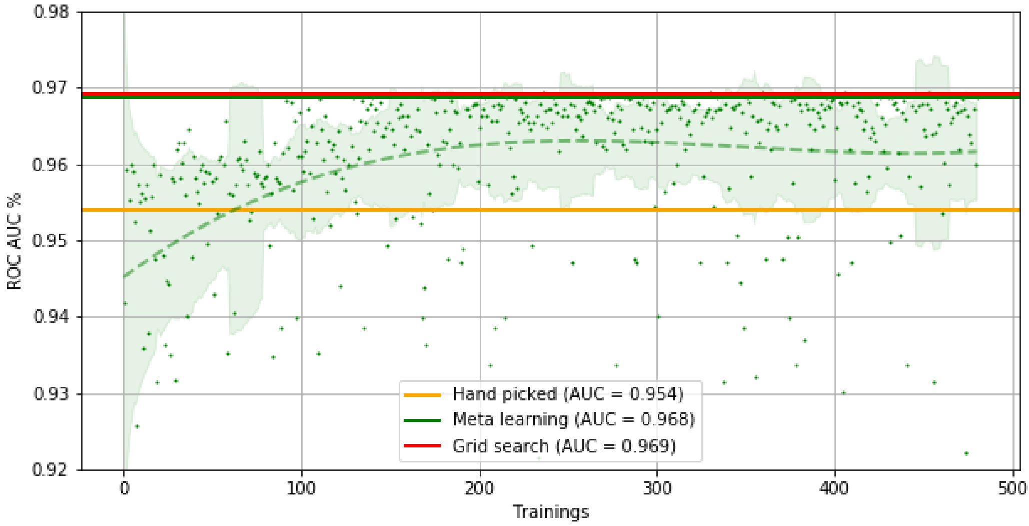


**Figure 4.** Comparison of different hyper parameter optimization methods. Each point is a fitness function score of each training using meta learning method, shaded region visualizes distribution of those points

## 6. Conclusions

This studies the use of genetic algorithms for hyper parameter tuning, and observes that with careful seeding the initial population the optimal solution is likely to be found faster than trying all possible combinations of hyper parameter values. The proposed solution has improved AUC score of neural network used for CERN CMS data certification. A similar algorithm can be applied for other machine learning models for hyper-parameter optimization.

## References

[1] Bergstra J and Bengio Y 2012 *The Journal of Machine Learning Research* **13** 281–305

[2] Stankevicius M, Marcinkevicius V and Rapsevicius V 2018 *CEUR* **Vol-2158** 170–176

[3] Xin Yao 1999 *Proceedings of the IEEE* **87** 1423–1447 ISSN 0018-9219

[4] Lehman J and Miikkulainen R 2013 *Scholarpedia* **8** 30977 revision #137053

[5] Chollet F *et al.* 2015 Keras https://keras.io

[6] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado G S, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y and Zheng X 2015 TensorFlow: Large-scale machine learning on heterogeneous systems software available from tensorflow.org URL `https://www.tensorflow.org/`

[7] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M and Duchesnay E 2011 *Journal of Machine Learning Research* **12** 2825–2830

[8] Such F P, Madhavan V, Conti E, Lehman J, Stanley K O and Clune J 2017 *CoRR* **abs/1712.06567** (*Preprint* `1712.06567`) URL `http://arxiv.org/abs/1712.06567`

[9] Hinz T, Navarro-Guerrero N, Magg S and Wermter S 2018 *International Journal of Computational Intelligence and Applications* **17** 1850008 (*Preprint* `https://doi.org/10.1142/S1469026818500086`) URL `https://doi.org/10.1142/S1469026818500086`