A Training Optimization Assistant

for Deep Learning Networks

Victor-Emil Neagoe  
Department of Applied Electronics and Information Engineering  
Polytechnic University of BucharestBucharest, Romania  
victoremil@gmail.com

Paul Diaconescu  
Department of Applied Electronics and Information Engineering  
Polytechnic University of BucharestBucharest, Romania  
paul.diaconescu@gmail.com

*Abstract*— We introduce an automated Training Optimization Assistant (TOA) for deep learning networks. By optimizing a deep learning network, we want the network to be trained to achieve an objective with given resources like time, hardware and a dataset. TOA will easily upload a dataset, will be scalable on the given hardware and with respect to a series of constrains will be able to find an optimum set of hyper-parameters and a optimum network architecture by trying different self-created scenarios, driven by a set of optimization algorithms. For this research, TOA is used for the particular case of credit risk classification. The objective has been a high accuracy of the risk classification and this was achieved, the result was at the level of the best published papers so far.

Keywords— german credit, credit risk, deep learning, automated deep learning flow

# Introduction

1. Financial Credit Risk and German Credit Dataset

The importance of credit for economic growth has been proven by several studies. As an example, the first conclusion of an important paper released by Central European Bank in 2017 is that “finance causes growth” [1].

The act of offering credit or lending money has an attached risk for the lender of not receiving back the amount of money agreed in the credit contract. To manage such risk, the lenders have different forms of assessing the borrower trustworthiness. The financial institutions can use entities or people ratings starting from AAA for high trust to BBB or CCC for lower trust. The last ratings are also called subprime or junk, having a high risk for the entity offering such credit of not getting back the funds. Errors in calculating the correct rating (or estimating the risk of incapacity to pay the credit back) has been part of cause for the systemic financial crises as the one from 2007.

This paper addresses the evaluation of credit risk associated to lending to private people and it doesn’t cover the risk associated with lending to economic entities. For the purpose of this paper, the macroeconomic context is considered stable as opposed to a context of systemic financial risk.

The credit risk is dependent on many attributes that should be taken into consideration as well as the correlation of these factors. These attributes could be demographics (at the time this article was written there was an increasing international pressure for elimination of demographic discrimination) or related to financial status. Demographic attributes mentioned above are: age, sex, job and house ownership. Financial status factors are credit history and actual financial situation.

This paper is using the German Credit dataset, a dataset containing 1,000 records with 20 attributes each. Each record is classified as good or bad for credit. There are 700 records classified as good and 300 records classified as bad.

For the usability of the original dataset in automated classifications, some of the values of the attributes have been changed, receiving numerical values. The transformation to numerical values had been done by the dataset creator, Professor Dr. Hans Hofmann (Institut fur Statistik und Okonometrie Universitat from Hamburg Germany).

Any of the records has the same structure with the record presented in Table 1.

TABLE 1: SUMMARY OF A GERMAN CREDIT RECORD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Age** | **Sex** | **Housing** | **…** | **Credit** |
| 67 | Male | Rent | … | Car |

The accuracy result of the credit risk classification is calculated by counting the results of all tests as following:

TABLE 2: COST MATRIX

|  |  |  |
| --- | --- | --- |
| **Actual vs predicted classification** | **Predicted Good** | **Predicted Bad** |
| Actual Good | 0 | 1 |
| Actual Bad | 5 | 0 |

The worst case for an entity that is lending money is to predict that a bad credit is good.

From the total number of 1,000 records 250 had been used for training, 150 records had been used for training validation and 600 records had been used for test. Between those 600 test records, 500 records are “actual good” and 100 records are “actual bad”. As an effect, the worst possible score is:

500 actual good predicted bad x 1 + 100 actual bad predicted good x 5 = 1,000

1. Description of an Automated Training Optimization Assistant

Creating a Deep Learning Network capable of very accurate classifications is a very complex process, becoming too complex for human understanding and control. Deep Networks becomes black boxes from the perspective of what is happening inside, how the information is used, how patterns are found, what weights are more important for certain classes or how an architecture can be optimized for high classification accuracy or training and test speed. The debug of issues becomes time consuming in the best case.

Selecting the optimum configuration of a high number of hyper-parameters from a continuous multi-dimensional space is an impossible task for a human. Tools and systems to reduce the complexity and automate part of the effort are mandatory for making relevant progress.

Our Training Optimization Assistant contains the following components

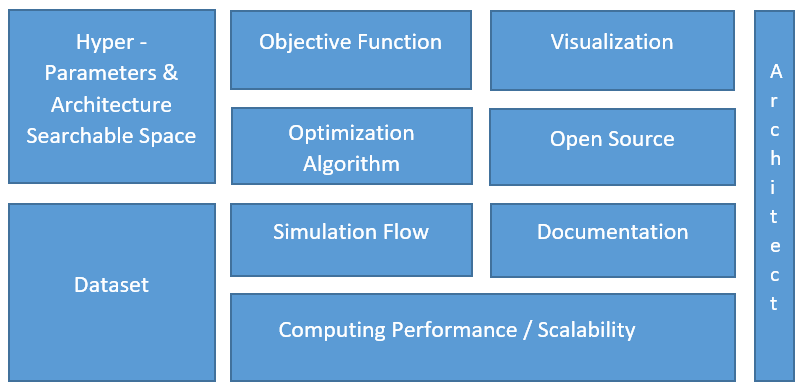


Figure 1. TOA Architecture

The most known open source simulation flows with optimization algorithms for finding solutions are: Hyperopt, BayesianOptimization, Optuna, HpBandSter, Sherpa, Tpot, Scikit-optimize. They present some of the required elements, still no one of them has all these elements.

In an effort for building a better architecture for automated training flow, the following components have been researched:

1. Computing Performance / Scalability
   1. Parallel Run

A Deep Learning simulation must be scalable in terms of computing capacity. It should allow parallel computations in the same process, multiple processes and scalability to multiple machines. In our application the parallel computations in the same process are enabled by the Nvidia GPU DataParallel libraries, the multi process run is supported by the Scikit framework or torch.multiprocessing library.

* 1. Correct environment

The simulations have been run in 3 environments: Ubuntu 18.04, Nvidia Container with Ubuntu inside Ubuntu and Windows 10.

The performances obtained with the same code had been the following:

TABLE3: TRAINING AND TEST PERFORMANCE IN DIFFERENT ENVIRONMENTS

|  |  |
| --- | --- |
| **Environment** | **Time (less is better)** |
| Ubuntu 18.04 | 100% |
| Nvidia Container | 126% |
| Windows 10 | 234% |

* 1. Code performance

The code effect on performance was also analyzed with cProfile tool for taking out time consuming sequences. cProfile can be activated or disabled during runs of training and test.

1. Multiple optimization algorithms and capability to define a custom optimization algorithm

An optimization algorithm should be able to check multiple results of an objective function obtained from different combinations of hyper-parameters and be able to select new values for hyper-parameters that will minimize or maximize the objective function.

For this application, the following optimization algorithms have been used:

* Decision trees: the space of possible values is divided into distinct and non-overlapping regions based on hyperparameters used by the optimization function; every observation is used to build a mean of responses for the corresponding region.
* Gradient Boosted Regression Trees (GBRT). This model is starting an initial set of simulations based on randomly chosen hyper-parameters from a given range, then it tries to find a better combination of hyper-parameters by building trees from the results, then adding new branches on the most interesting nodes.
* Bayesian optimization. The posterior probability of a model based on existing evidence is proportional with the likehood of that evidence multiplied by prior probability. The main difference of this optimization model compared with the previous two is that all exercised points matter, the final solution depends on all of them. This optimization algorithm works as following:

1) initially observing a number of simulations (a priori) which are considered existing data

2) generating an objective function of possible results based on existing data (the objective function is associated to high uncertainty)

3) finally updating the function (combine the estimated function with evidence) by adding new real results which reduce the uncertainty.

The final step described above is a combination of exploration (of areas with high uncertainty) and exploitation (of areas with best observations)

* Uniform sampling within the given bounds of the hyperparameters. The space was split in a number of sub-spaces given by the number of algorithm iterations (ex: 300).
* Gradient Descent Optimization algorithms from Pytorch libraries as ADAM, Adagrad, SGD, Adadelta and others. These algorithms are starting with a single random point in the n-dimensional space of all solutions and based on the error size of the error between the result and the objective, are moving the selection of the parameters less or more in different directions.

1. Ability to define a searchable space given by hyper-parameters

With only 2 hyper-parameters we can say the objective function is searched in 2 dimensions; with n hyper-parameters we search for the objective in a n-dimensional space. Custom values should be given for hyper-parameters in different formats (integer, range, float, list, enumeration…).

For this work, the application is automatically selecting values for fourteen hyper-parameters, from pre-defined ranges.

1. Ability to define and use an objective function (should be moved ??)

The flow should be capable to receive an objective (Ex: classification accuracy score). Multiple runs will return multiple results for the objective function.

The objective used for the credit risk classification is the credit cost. The training of the network is the function and its result, the credit cost minimization is the function objective. The optimization algorithm used is running multiple simulations, basically running the network training with different hyper-parameters. The simulations are run in two phases: a random phase when hyper-parameters are randomly chosen and results kept, then the second phase, when the optimization algorithm is looking at the stored results and run new simulations with hyper-parameters “similar” with those that had the best results. The similarity depends on optimization algorithm.

1. An early stopping mechanism

One of the best predictors of the training performance is the validation error, which TOA is monitoring during training. When no reduction is validation error is found for 21 epochs, the training is stopped. We found that is better to eliminate also very small (<10^-5) reductions of validation error, as in most of these cases no training improvement is made.

1. Visualization

Many time Deep Learning training is similar with a black box, being very hard to understand if and how the network was trained or how far is an optimum configuration.

In the following image is easy to see how the optimization function returned a series of 30 results with the purpose of finding the minimum. Immediately after a new minimum is found, that results became the optimum result. Figure 1 is the result of using visualization libraries of Scikit-optimize framework.

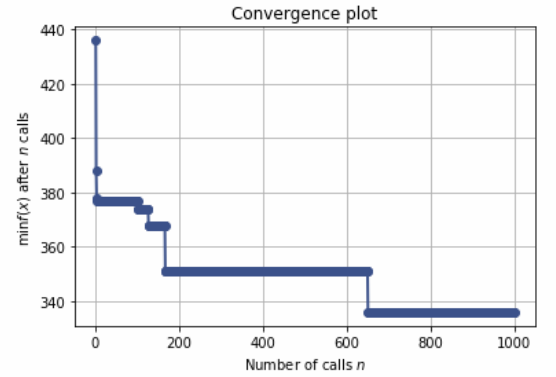


Figure 2. Visualization of the optimization function results

1. Documentation

Documentation should be updated and available, should contain enough explanation and examples

1. Reproducibility mechanism

AI field is confronting a reproducibility crisis. This is not a new problem, for example medicine had the same problem for the past decades.

A mechanism should be in place, ready to keep the configuration for the runs in such a way that the run can be re-done at a later time for result reproducibility benefit.

1. Open Source

Open Source development brings the ideas of the crowd, reliability (as verified by more eyes), transparency of the results and reduced maintenance costs.

# Proposed Method

To prove the power of this application, a high number of hyper-parameters have been chosen to be used by the optimization algorithm.

TABLE 3. HYPER-PARAMETERS

|  |  |
| --- | --- |
| **Hyper-Parameter** | **Range** |
| Architecture | ReLu: True or False (layer exists or doesn’t exist)  DropOut: True or False (layer exists or doesn’t exist) |
| Training Optimizer | Adam, SGD, Adadelta, Adagrad, AMSGrad, AdamW |
| SGD Momentum | 0 .. 0.99 |
| Training Loss | CrossEntropyLoss, NLLLoss, MultiMarginLoss |
| Learning Rate | 1e\*10^-5 … 1e10^-1 |
| Number of convolutional filters | 2 .. 64 |
| Regularization R2 | 0.0001, 0.001, 0.01 |
| DropOut | 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 |
| Training Mini Batch Size | 32, 64, 90, 100, 128, 192, 256 |
| Bias | True, False |
| Momentum for Normalization layer | 0 .. 0.99 |
| Number of epochs | Up to 10,000 or 20,000 |

1. How complex should be the deep learning network?

The network architecture is based on parallel flows of convolutional layers and is presented in Figure 3.

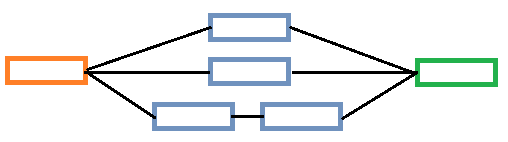


Figure 3 The Deep Learning Network Architecture. Orange: Input Image. Blue: Convolutional – Relu - DropOut layers, Green: Linear – Relu – DropOut – Linear – SoftMax layers

The layers used in the presented deep learning architecture are shown in Table 3.

TABLE 4. NETWORK ARCHITECTURE

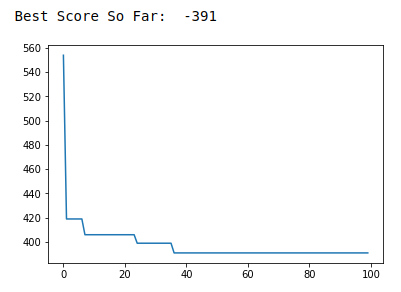
|  |  |
| --- | --- |
| **Layer number** | **Layer** |
| 1 - 3 | Conv-Relu - DropOut |
| 4 - 6 | Conv-Relu - DropOut |
| 7 - 12 | 2x Conv-Relu - DropOut |
| 13 - 18 | Linear – Relu – DropOut – Linear – SoftMax |

# EXPERIMENTAL RESULTS

## Results

Due to the counting system for the credit cost, the results are different for The minimum credit cost offered by our deep learning architecture is presented in Table 5.

Comparison of optimization strategies with graphs



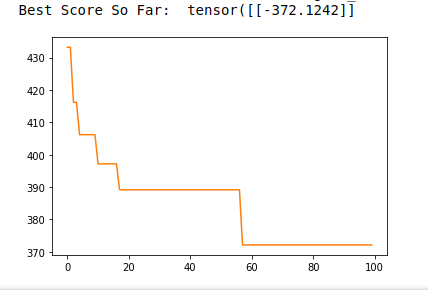


Figure 4. The performance of different optimization strategies

TABLE 5. RESULTS OF DEEP LEARNING CLASSIFICATION ACCURACY FOR 220 TRAINING ELEMENTS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Epochs | Optimization | LR | DL Optimization/ DL Loss Type | Correct Classification | Credit Cost |
| 1 | 1003 | forest | 1e-05 | Adam/CEL | 49% | 481 |
| 2 | 5000 | forest | 1e-05 | Adadelta/NLL | 78% | 643 |
| 3 | 5060 | forest | 1e-05 | SGD/CEL | 54% | 470 |
| 4 | 50 | forest | 0.02 | ADAM/MMR | 50% | 367 |
| 5 | 31 | uniform | 0.05 | ADAM/ CEL | 68% | 405 |
| 6 | 86 | uniform | 0.001 | ADAM/ CEL | 69% | 399 |
| 7 | 20 | gbs | 0.2 | Adagrad/NLL | 70% | 352 |
| 8 | 31 | gbs | 0.5 | Adagrad/NLL | 71.91% | 348 |
| 9 | 18 | gbs | 0.05 | Adam/NLL | 73.43% | 318 |

The most important classification score is the credit cost (last column). Another classification is (correct classification column) the percent of test elements correctly classified independent of the element cost and class.

The optimum values are presented in the following table:

TABLE 6: OPTIMUM VALUES OF HYPER-PARAMETERS

|  |  |
| --- | --- |
| **Hyper-Parameter** | **Optimum values** |
| Architecture | Relu True, Dropout True |
| Training Optimizer | ADAM |
| SGD Momentum | Not relevant for ADAM |
| Training Loss | CrossEntropyLoss |
| Learning Rate | 0.17 |
| Number of convolutional filters | 98 |
| Regularization R2 | 0.003 |
| DropOut | 0.2 |
| Training Mini Batch Size | 4 |
| Bias | True |
| Momentum for Normalization layer | 0.9 |
| Number of epochs | 22 |

# CONCLUSIONS

The present paper presents a Training Optimization Assistant and explains how it works in case of a classification problem.

The main benefits from using TOA are the following: the available hardware is better used, the network manual optimization for a better training is replaced by a fast and automated optimization of hyper-parameters driven by advanced mathematical algorithms.

As a novelty element, we have proposed two frameworks, Maltlab and PyTorch, for comparing their capabilities of supporting a deep learning architecture. We have also reviewed the training speeds on different processing hardware (CPU, one GPU or two GPUs).

In terms of classification score, the proposed architecture achieved an 89.8% correct classification score, more than 51.7% score of the SVM algorithm. Both methods, deep learning and SVM have used non-enhanced training samples.

The training time is very competitive considering the complexity of the classification and the hardware used and this makes the proposed architecture interesting for the industries that needs almost real time results.

##### References

1. Alexander Popov, " Evidence on finance and economic growth", *European Central Bank – Working Paper Series, No 2115 / December 2017*
2. Ricardas Mileris, Vytautas Boguslauskas, “Credit Risk Estimation Model Development Process: Main Steps and Model Improvement”, *Inzinerine Ekonomika-Engineering Economics, 2011, 22(2), 126-133*
3. Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, … Alexander Fabisch. (2018, March 25). scikit-optimize/scikit-optimize: v0.5.2 (Version v0.5.2). Zenodo. http://doi.org/10.5281/zenodo.1207017
4. Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, *eprint arXiv:1502.03167, February 2015*
5. Andreas Merentitis, Christian Debes, “Many Hands Make Kight Work – On Ensemble Learning Techniques for Data Fusion in Remote Sensing”, *IEEE Geoscience and Remote Sensing Magazine · September 2015*
6. Peter I. Frazier, “A Tutorial on Bayesian Optimization”, *arXiv:1807.02811v1 [stat.ML] 8 Jul 2018*
7. Javier Gonzales, “Introduction to Bayesian Optimization”, *Masterclass Lancaster University, 7-February 2017*
8. Eric Brochu, Vlad M. Cora and Nando de Freitas, “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning”, *arXiv:1012.2599v1 [cs.LG] 12 Dec 2010*
9. James Bergstra, Yoshua Bengio “Random Search for Hyper-Parameter Optimization”, *Journal of Machine Learning Research 13 (2012) 281-305*
10. Jasper Snoek, Hugo Larochelle, Ryan P. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms”