# Improved Training Speed, Accuracy, and Data Utilization Through Loss Function Optimization

**Santiago Gonzalez**[1,2] and **Risto Miikkulainen**[1,2]
[1]Cognizant Technology Solutions, *San Francisco, California, USA*
[2]Department of Computer Science, University of Texas at Austin, *Austin, Texas, USA*
Email: slgonzalez@utexas.edu, risto@cs.utexas.edu

## Abstract

As the complexity of neural network models has grown, it has become increasingly important to optimize their design automatically through metalearning. Methods for discovering hyperparameters, topologies, and learning rate schedules have lead to significant increases in performance. This paper shows that loss functions can be optimized with metalearning as well, and result in similar improvements. The method, Genetic Loss-function Optimization (GLO), discovers loss functions *de novo*, and optimizes them for a target task. Leveraging techniques from genetic programming, GLO builds loss functions hierarchically from a set of operators and leaf nodes. These functions are repeatedly recombined and mutated to find an optimal structure, and then a covariance-matrix adaptation evolutionary strategy (CMA-ES) is used to find optimal coefficients. Networks trained with GLO loss functions are found to outperform the standard cross-entropy loss on standard image classification tasks. Training with these new loss functions requires fewer steps, results in lower test error, and allows for smaller datasets to be used. Loss function optimization thus provides a new dimension of metalearning, and constitutes an important step towards AutoML.

## 1 Introduction

Much of the power of modern neural networks originates from their complexity, i.e. number of parameters, hyperparameters, and topology. This complexity is beyond human ability to optimize, and metalearning methods are needed; based on various methods such as gradient descent, simulated annealing, reinforcement learning, Bayesian optimization, and evolutionary computation (EC) [5].

While a wide repertoire of work now exists for optimizing many aspects of neural networks, the dynamics of training are still usually set manually without concrete, scientific methods. Particularly, loss functions affect the training and final functionality of a neural network. Perhaps they could also be optimized through metalearning?

The goal of this paper is to verify this hypothesis. A general framework for loss function metalearning, covering both novel loss function discovery and optimization, is developed and evaluated experimentally. This framework, Genetic Loss-function Optimization (GLO), leverages Genetic Programming to build loss functions represented as trees, and subsequently a Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to optimize their coefficients. EC methods were chosen because EC is arguably the most versatile of the metalearning approaches. It is a population-based search method; allowing for extensive exploration, which often results in creative, novel solutions that are not obvious at first [16]. EC has been successful in hyperparameter optimization and architecture design in particular [18, 26, 22, 17]. It has also been used to discover mathematical formulas to explain experimental data [23]. It is, therefore, likely to find creative solutions in the loss function optimization domain as well.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Indeed, in the MNIST image classification benchmark, GLO discovered a surprising new loss function, named Baikal for its shape. This function performs very well, presumably by establishing an implicit regularization effect. Baikal outperforms the standard cross-entropy loss in terms of training speed, final accuracy, and data requirements. Furthermore, Baikal was found to transfer to a more complicated classification task, CIFAR-10, while carrying over its benefits.

## 2 Related Work

In addition to hyperparameter optimization and neural architecture search, new opportunities for metalearning have recently emerged. In particular, learning rate scheduling and adaptation can have a significant impact on a model's performance. Learning rate schedules determine how the learning rate changes as training progresses. This functionality tends to be encapsulated away in practice by different gradient-descent optimizers, such as AdaGrad [4] and Adam [12]. While the general consensus has been that monotonically decreasing learning rates yield good results, new ideas, such as cyclical learning rates [24], have shown promise in learning better models in fewer epochs.

Metalearning methods have also been recently developed for data augmentation, such as AutoAugment [3], a reinforcement learning based approach to find new data augmentation policies. In reinforcement learning tasks, EC has proven a successful approach. For instance, in evolving policy gradients [11], the policy loss is not represented symbolically, but rather as a neural network that convolves over a temporal sequence of context vectors. In reward function search [20], the task is framed as a genetic programming problem, leveraging PushGP [25].

In terms of loss functions, a generalization of the L2 loss was proposed with an adaptive loss parameter [2]. This loss function is shown to be effective in domains with multivariate output spaces, where robustness might vary across between dimensions. Specifically, the authors found improvements in Variational Autoencoder (VAE) models, unsupervised monocular depth estimation, geometric registration, and clustering.

Notably, no existing work in the metalearning literature automatically optimizes loss functions for neural networks. As shown in this paper, evolutionary computation can be used in this role to improve neural network performance, gain a better understanding of the processes behind learning, and help reach the ultimate goal of fully automated learning.

## 3 The GLO Approach

The task of finding and optimizing loss functions can be framed as a functional regression problem. GLO accomplishes this through the following high-level steps (shown in Figure **??**): **(1) loss function discovery:** using approaches from genetic programming, a genetic algorithm builds new candidate loss functions, and **(2) coefficient optimization:** to further optimize a specific loss function, a covariance-matrix adaptation evolutionary strategy (CMA-ES) is leveraged to optimize coefficients. Information on implementation details is provided in the supplementary material.

GLO uses a population-based search approach, inspired by genetic programming, to discover new optimized loss function candidates. Under this framework, loss functions are represented as trees, due to their hierarchical nature, within a genetic algorithm. The loss function search space is defined by the following tree nodes: *Unary:* $\log(\ldots), \ldots^2, \sqrt{\ldots}$, *Binary:* $+, *, -, \div$, *Leaf Nodes:* $x, y, 1, -1$, where $x$ represents a true label, and $y$ represents a predicted label.

The search space is further refined by automatically assigning a fitness of 0 to trees that don't contain both at least one $x$ and one $y$. Generally, a loss function's fitness within the genetic algorithm is the validation performance of a network trained with that loss function. To expedite the discovery process, and encourage the invention of loss functions that enable faster learning, training does not proceed to convergence. Unstable training sessions that result in NaN values are assigned a fitness of 0. Fitness values are cached to avoid needing to retrain the same network twice. Information on the initial population, recombination, and mutations is presented in the supplementary material.

Loss functions found by the above genetic algorithm can all be thought of having unit coefficients for each node in the tree. This set of coefficients can be represented as a vector with dimensionality equal to the number of nodes in a loss function's tree. The coefficient vector is optimized independently and iteratively using a covariance-matrix adaptation evolutionary strategy (CMA-ES). [8] The specific

variant of CMA-ES that GLO uses is $(\mu/\mu, \lambda)$-CMA-ES [9], and incorporates weighted rank-$\mu$ updates [7] to reduce the number of objective function evaluations that are needed. The presented GLO implementation uses an initial step size $\sigma = 1.5$. As in the discovery phase, the objective function is the network's performance on a validation dataset after a shortened training period.

## 4 Experimental Evaluation

This section provides an experimental evaluation of GLO, on the MNIST and CIFAR-10 image classification tasks. Baikal, a GLO loss function found on MNIST, is presented and evaluated in terms of its resulting testing accuracy, training speed, training data requirements, and transferability to CIFAR-10.

Experiments on GLO are performed using two popular image classification datasets, MNIST Handwritten Digits [15] and CIFAR-10 [13]. Both datasets are well understood, and relatively quick to train, allowing rapid iteration in the development of GLO and time for more thorough experimentation. The selected model architectures are simple, since achieving state-of-the-art accuracy on MNIST and CIFAR-10 is not the focus of this paper, rather the improvements brought about by using a GLO loss function are. More information on the datasets, along with their corresponding architectures and experimental setup is provided in the supplementary material.

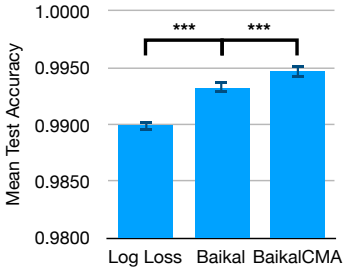


Figure 1: Mean testing accuracy on MNIST, $n = 10$. Both Baikal and BaikalCMA provide statistically significant improvements to testing accuracy over the cross-entropy loss.

Both of these tasks, being classification problems, are traditionally framed with the standard cross-entropy loss (sometimes referred to as the log loss): $\mathcal{L}_{\text{Log}} = -\frac{1}{n}\sum_{i=0}^{n} x_i \log(y_i)$, where $x$ is sampled from the true distribution, $y$ is from the predicted distribution, and $n$ is the number of classes. The cross-entropy loss is used as a baseline in this paper's experiments.

### 4.1 The Baikal loss function

The most notable loss function that GLO discovered against the MNIST dataset (with 2,000-step training for candidate evaluation) is the Baikal loss (named as such due to its similarity to the bathymetry of Lake Baikal when its binary variant is plotted in 3D):

$$\mathcal{L}_{\text{Baikal}} = -\frac{1}{n}\sum_{i=0}^{n} \log(y_i) - \frac{x_i}{y_i},$$

where $x$ is from the true distribution, $y$ is from the predicted distribution, and $n$ is the number of classes. Additionally, after coefficient optimization, GLO arrived at the following version of the Baikal loss (simplified):

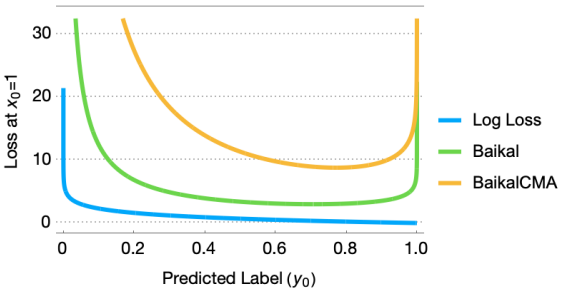


Figure 2: Binary classification loss functions at $x_0 = 1$. Correct predictions lie on the right side of the graph. The log loss is shown to be monotonically decreasing, while Baikal and BaikalCMA present counterintuitive, sharp increases in loss as predictions, approach the true label. This provides regularization by preventing the model from being too confident in its predictions.

$$\mathcal{L}_{\text{BaikalCMA}} = -\frac{1}{n}\sum_{i=0}^{n} 2.7279\left(0.9863 * \log(1.5352 * y_i) - 1.8158\frac{x_i}{y_i}\right).$$

This loss function, BaikalCMA, was selected for having the highest validation accuracy out of the population. The Baikal and BaikalCMA loss functions had validation accuracies at 2,000 steps equal to 0.9838 and 0.9902, respectively. For comparison, the cross-entropy loss had a validation accuracy at 2,000 steps of 0.9700. Models trained with the Baikal loss on MNIST and CIFAR-10 (to test transfer) are the primary vehicle for validating GLO's efficacy, as detailed in subsequent sections.

**Testing accuracy** Figure 1 shows the increase in testing accuracy that Baikal and BaikalCMA provide on MNIST over models trained with the cross-entropy loss. Over 10 trained models each, the

mean testing accuracies for cross-entropy loss, Baikal, and BaikalCMA were 0.9899, 0.9933, and 0.9947, respectively. This increase in accuracy from Baikal over cross-entropy loss is statistically significant, with a $p$-value of $2.4 \times 10^{-11}$, in a heteroscedastic, two-tailed T-test, with 10 samples from each distribution. With the same significance test, the increase in accuracy from BaikalCMA over Baikal was statistically significant, with a $p$-value of $8.5045 \times 10^{-6}$.

**Training speed** Training curves for networks trained with the cross-entropy loss, Baikal, and BaikalCMA are shown in Figure 3. Each curve represents 80 testing dataset evaluations spread evenly (i.e., every 250 steps) throughout 20,000 steps of training on MNIST. Networks trained with Baikal and BaikalCMA both learn significantly faster than the cross-entropy loss. Interestingly, the Baikal and BaikalCMA training curves are both smoother than the cross-entropy loss curve, implying that their loss surfaces have fewer or less detrimental local minima. These phenomena make Baikal a compelling loss function for fixed time-budget training, where the improvement in accuracy over the cross-entropy loss becomes most evident.

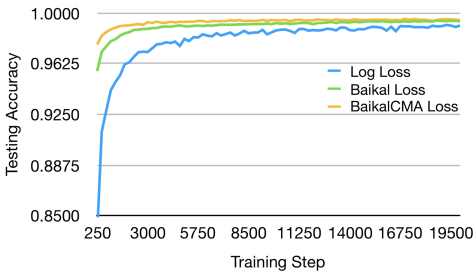


Figure 3: Training curves for different loss functions on MNIST. Baikal and BaikalCMA result in faster and smoother training compared to the cross-entropy loss.

**Training data requirements** Experiments were performed to ascertain the effects of dataset size on networks trained with cross-entropy loss, Baikal, and BaikalCMA. For each training dataset portion size, five individual networks were trained for each loss function. The degree by which Baikal and BaikalCMA outperform cross-entropy loss increases as the training dataset becomes smaller. This provides evidence of less overfitting when training a network with Baikal or BaikalCMA. As expected, BaikalCMA outperforms Baikal at all tested dataset sizes. The size of this improvement in accuracy does not grow as significantly as the improvement over cross-entropy loss, leading to the belief that the overfitting characteristics of Baikal and BaikalCMA are very similar.

**Loss function transfer to CIFAR-10** The Baikal loss was applied to CIFAR-10, as a means to test transferability. In a collection of 18 separate tests on CIFAR-10, Baikal outperforms cross-entropy across all training durations, with the difference becoming more prominent for shorter training periods. These results present an interesting use case for GLO, where a loss function that is found on a simpler dataset can be transferred to a more complex dataset while still maintaining performance improvements. This provides a particularly persuasive argument for using GLO loss functions in fixed time-budget scenarios.

## 5 Conclusion

This paper proposes Genetic Loss-function Optimization (GLO) as a general framework for discovering and optimizing loss functions for a given task. A surprising new loss function, Baikal, was discovered in the experiments, and shown to outperform the cross-entropy loss on MNIST and CIFAR-10 in terms of accuracy, training speed, and data requirements. Further analysis (provided in the supplementary material) suggests that Baikal's improvements result from implicit regularization that reduces overfitting to the data. GLO can be combined with other aspects of metalearning in the future, paving the way to robust and powerful AutoML.

In the future, GLO can be applied to other machine learning datasets and tasks. The approach is general, and could result in discovery of customized loss functions for different domains, or even specific datasets. GLO could also leverage co-evolution, where multiple interacting solutions are developed simultaneously. For instance, GLO could be combined with techniques like CoDeepNEAT [18] to learn jointly-optimal network structures, hyperparameters, learning rate schedules, data augmentation, and loss functions simultaneously. Such approaches require significant computing power, but they may also discover and utilize interactions between the design elements that result in higher complexity and better performance than is currently possible.

## References


[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, 2016. USENIX Association.

[2] J. T. Barron. A general and adaptive robust loss function. *arXiv preprint arXiv:1701.03077*, 2017.

[3] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[4] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[5] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[6] S. Gonzalez, J. Landgraf, and R. Miikkulainen. Faster training by selecting samples using embeddings. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019.

[7] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In *International Conference on Parallel Problem Solving from Nature*, pages 282–291. Springer, 2004.

[8] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317. IEEE, 1996.

[9] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[11] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. J. Ho, and P. Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems*, pages 5400–5409, 2018.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[15] Y. LeCun, C. Cortes, and C. Burges. The MNIST dataset of handwritten digits, 1998.

[16] J. Lehman et al. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *arXiv preprint arXiv:1803.03453*, 2018.

[17] I. Loshchilov and F. Hutter. CMA-ES for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

[18] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[19] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[20] S. Niekum, A. G. Barto, and L. Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.

[21] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[23] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[24] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[25] L. Spector, E. Goodman, A. Wu, W. B. Langdon, H. m. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, E. Burke, and M. Kaufmann Publishers. Autoconstructive evolution: Push, pushgp, and pushpop. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 05 2001.

[26] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[27] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency and computation: practice and experience*, 17(2-4):323–356, 2005.