Supplementary Material

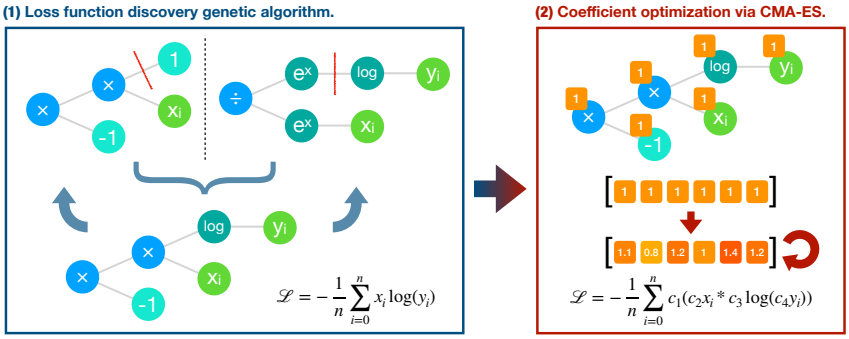

Figure 4: Genetic Loss Optimization (GLO) overview. A genetic algorithm constructs candidate loss functions as trees. The best loss functions from this set then has its coefficients optimized using CMA-ES. GLO loss functions are able to train models more quickly and more accurately.

Initial population, recombination, and mutation

The initial population is composed of randomly generated trees with a maximum depth of 2. Recursively starting from the root, nodes are randomly chosen from the allowable operator and leaf nodes using a weighting (where $\log(...), x, y$ are three-times as likely and $\sqrt{...}$ is two-times as likely as $+, *, -, \div, 1, -1$), this can impart a bias and prevent, for example, the integer 1 from occurring too frequently. The genetic algorithm has a population size of 80, incorporates elitism with 6 elites per generation, and uses roulette-sampling.

Recombination is accomplished by randomly splicing two trees together. For a given pair of parent trees, a random element is chosen in each as a crossover point. The two subtrees, whose roots are the two crossover points, are then swapped with each other. Both resultant trees become part of the next generation. Recombination occurs with a probability of 80%.

To introduce variation into the population, the genetic algorithm has the following mutations, applied in a bottom-up fashion: integer scalar nodes are incremented or decremented with a 5% probability, nodes are replaced with a weighted-random node with the same number of children with a 5% probability, nodes (and their children) are deleted and replaced with a weighted-random leaf node with a $5\% * 50\% = 2.5\%$ probability, leaf nodes are deleted and replaced with a weighted-random element (and weighted-random leaf children if necessary) with a $5\% * 50\% = 2.5\%$ probability.

MNIST experimental setup

The first target task used for evaluation was the MNIST Handwritten Digits dataset [15], a widely used dataset where the goal is to classify 28×28 pixel images as one of ten digits. The MNIST dataset has 55,000 training samples, 5,000 validation samples, and 10,000 testing samples.

A simple CNN architecture with the following layers is used: (1) 5×5 convolution with 32 filters, (2) 2×2 stride-2 max-pooling, (3) 5×5 convolution with 64 filters, (4) 2×2 stride-2 max-pooling, (5) 1024-unit fully-connected layer, (6) a dropout layer [10] with 40% dropout probability, and (7) a softmax layer. ReLU [19] activations are used. Training uses stochastic gradient descent (SGD) with a batch size of 100, a learning rate of 0.01, and, unless otherwise specified, for 20,000 steps.

CIFAR-10 experimental setup

To further validate GLO, the more challenging CIFAR-10 dataset [13] (a popular dataset of small, color photographs in ten classes) was used as a medium to test the transferability of loss functions found on a different domain. CIFAR-10 consists of 50,000 training samples, and 10,000 testing samples.

A simple CNN architecture, taken from [6] (and itself inspired by AlexNet [14]), with the following layers is used: (1) 5×5 convolution with 64 filters and ReLU activations, (2) 3×3 max-pooling

with a stride of 2, (3) local response normalization [14] with $k = 1, \alpha = 0.001/9, \beta = 0.75$, (4) 5×5 convolution with 64 filters and ReLU activations, (5) local response normalization with $k = 1, \alpha = 0.001/9, \beta = 0.75$, (6) 3×3 max-pooling with a stride of 2, (7) 384-unit fully-connected layer with ReLU activations, (8) 192-unit fully-connected, linear layer, and (9) a softmax layer.

Inputs to the network are sized $24 \times 24 \times 3$, rather than $32 \times 32 \times 32$ as provided in the dataset; this enables more sophisticated data augmentation. To force the network to better learn spatial invariance, random 24×24 croppings are selected from each full-size image, which are randomly flipped longitudinally, randomly lightened or darkened, and their contrast is randomly perturbed. Furthermore, to attain quicker convergence, an image's mean pixel value and variance are subtracted and divided, respectively, from the whole image during training and evaluation. CIFAR-10 networks were trained with SGD, L^2 regularization with a weight decay of 0.004, a batch size of 1024, and an initial learning rate of 0.05 that decays by a factor of 0.1 every 350 epochs.

Implementation details

Due to the large number of partial training sessions that are needed for both the discovery and optimization phases, training is distributed across the network to a cluster of dedicated machines that use Condor [27] for scheduling. Each machine in this cluster has one NVIDIA GeForce GTX Titan Black GPU and two Intel Xeon E5-2603 (4 core) CPUs running at 1.80GHz with 8GB of memory. Training itself is implemented with TensorFlow [1] in Python. The primary components of GLO (i.e., the genetic algorithm and CMA-ES) are implemented in Swift. These components run centrally on one machine and asynchronously dispatch work to the Condor cluster over SSH. Code is available at: `http://bit.ly/neurips2019_GLO_code`

Analysis

This section presents a symbolic analysis of the Baikal loss function, followed by experiments that attempt to elucidate why Baikal works better than the cross-entropy loss. A likely explanation is that Baikal results in implicit regularization.

Binary classification

Loss functions used on the MNIST dataset, being a 10-dimensional classification problem, are difficult to plot and visualize graphically. In this section, loss functions are analyzed in the context of binary classification; where $n = 2$, the Baikal loss expands to:

$$\mathcal{L}_{\text{Baikal2D}} = -\frac{1}{2}\left(\log(y_0) - \frac{x_0}{y_0} + \log(y_1) - \frac{x_1}{y_1}\right) \propto -\log(y_0) + \frac{x_0}{y_0} - \log(1-y_0) + \frac{1-x_0}{1-y_0}, \quad (1)$$

since vectors x and y sum to 1, by consequence of being passed through a softmax function, for binary classification $x = [x_0, 1-x_0]$ and $y = [y_0, 1-y_0]$. This constraint simplifies the binary Baikal loss to a function of two variables (x_0 and y_0). This same methodology can be applied to the cross-entropy loss and BaikalCMA.

In practice, true labels are assumed to be correct with certainty, thus, x_0 is equal to either 0 or 1. The specific case where $x_0 = 1$ is plotted in Figure 5 for the cross-entropy loss, Baikal, and BaikalCMA. The cross-entropy loss is shown to be monotonically decreasing, while Baikal and BaikalCMA counterintuitively show an increase in the loss value as the predicted label, y_0, approaches the true label x_0. Section 5 provides reasoning for this unusual phenomenon.

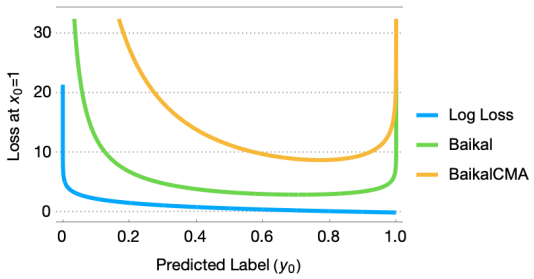

Figure 5: Binary classification loss functions at $x_0 = 1$. Correct predictions lie on the right side of the graph, and vice versa. The log loss is shown to be monotonically decreasing, while Baikal and BaikalCMA present counterintuitive, sharp increases in loss as predictions, approach the true label.

As also seen in Figure 5, the minimum for the Baikal loss where $x = 1$ lies around 0.71, while the minimum for the BaikalCMA loss where $x = 1$ lies around 0.77. This, along with the more pronounced slope around $y = 0.5$ is likely

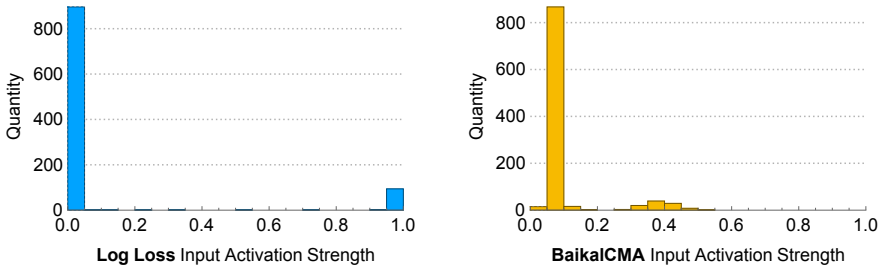

Figure 6: Loss function input activation strength histograms for cross-entropy loss and BaikalCMA. The peaks are likely shifted with BaikalCMA due to implicit regularization. These histograms match those from a network trained with a confidence regularizer [21].

a reason why BaikalCMA performs better than Baikal.

Implicit regularization

The Baikal and BaikalCMA loss functions are unusual in that they are not monotonically decreasing (see the previous section for more details). At first glance, this behavior may seem undesirable; however, this may be an advantageous trait that implicitly provides a form of regularization (enabling better generalization). This is strongly supported by [21], where researchers built a confidence regularizer, on top of cross-entropy loss, that penalizes low entropy prediction distributions. The bimodal distribution of output probabilities that the researchers found on MNIST is nearly identical to that which can be found on a network trained with Baikal or BaikalCMA.

Histograms of the output probability distributions of network trained with the cross-entropy loss and BaikalCMA on the test dataset, after 15,000 steps of training on MNIST, are shown in Figure 6. Note that the abscissae in Figures 5 and 6 correspond with each other, thus one can qualitatively see how the channel-shaped curves for BaikalCMA may contribute to the shift in histogram peaks.

Furthermore, the improved behavior under small-dataset conditions described in Section 4.1 backs this theory of implicit regularization, since less overfitting was observed when using Baikal and BaikalCMA.