Learn What You Don't Know—Adaptive Focus Learning

Clayton A. Harper*
Southern Methodist University
Dallas, TX

Eli J. Laird*
Southern Methodist University
Dallas, TX

***Abstract*—Overall classification performance is typically the objective when training neural networks; however, this global objective is often attained at the expense of the performance for difficult classes in the data. We propose a modified gradient-based adaptive neural network training procedure in order to improve classification performance on the worst performing classes in a given dataset. Using a minimax-inspired protocol during each training epoch, we find a more stable optimization procedure for worst performing classes that additionally limits the effects of over-fitting.**

I. Introduction

During the training process of a typical neural network, the objective is to achieve the best overall performance on the training and validation sets, where best is defined by a metric of performance. A common procedure during the training of a neural network is to monitor the loss on the validation set at the end of each epoch. After the validation loss has not improved over some number of epochs, the training procedure is considered finished, as the model has either converged on a solution or is no longer improving. Although this process can find a solution for the dataset a whole, per-class performance can often be overlooked.

Overall performance can be misleading when considering classification on a per-class basis. For example, consider validation loss. If we have many classes that are easy to classify correctly and a few classes that are difficult to classify, a neural network may learn to exploit the easily classified classes to minimize the overall loss; however, performance for the difficult classes may not necessarily improve. In traditional pattern recognition tasks, this problem has been addressed with a variety of algorithms.

Certain classification tasks may associate a larger cost for incorrect classifications of a particular class. This is especially common in biometric systems where false positives mean granting access to non-authorized individuals [1]. This issue can also arise when dealing with imbalanced datasets, in which the classifier learns to only classify the overly represented class. To incentivize machine learning models to avoid these issues, a cost matrix is often used to penalize different types of incorrect classifications as seen in [2]–[5].

Another approach used to improve performance across classes, specifically the worst performing class, is the Minimax Algorithm [6]. The Minimax Algorithm aims to minimize the maximum cost. In [7], Lanckriet et al. propose a method of binary classification that minimizes the probability of misclassifying unseen data points with Minimax Probability Machines (MPMs). An extension of Minimax Probability Machines was proposed by [8] which seeks to improve performance on imbalanced datasets; this approach and traditional MPMs have been shown in [9] to work well in face recognition applications.

[10] employs the Minimax Algorithm in the process of training multi-layer perceptrons across a suite of 8 different datasets. Among the datasets chosen, there is a maximum number of 8 classes, $4,177$ observations, and a 57-dimensional input. Two multi-layer perceptron architectures are discussed—each have one hidden layer with 2 and 7 hidden units respectively. The models used the sigmoid activation function. For each epoch during training, the weights are modified according the the gradients found with the current model weights and the loss over the training dataset. Then, the training dataset is subset down to the worst performing class in the training dataset. Once the subset is defined, the weights are then modified according to the gradients with respect to the subset, instead of the entire training set. The goal of [10] was to minimize the uncertainty between class priors in the training dataset and class priors in real scenarios through this minimax procedure.

Although not intended to improve per-class performance, the Adaptive Boosting or AdaBoost Algorithm [11] is an adaptive procedure that utilizes the observations that the classifier had the most difficult time classifying. AdaBoost is used in ensemble learning techniques [2] to improve overall classification performance. Essentially, the training process places a larger importance on observations that were misclassified and a smaller importance on observations that were correctly classified, effectively "boosting" the gradient updates for poor performers.

In order to improve worst-class performance and balance validation performance between classes in a neural network environment, we propose an adaptive procedure based on the worst performing classes at the end of each training epoch, similar to [10]. *Adaptive Focus Learning* is a modified training procedure for neural networks seen in Algorithm 1.

By evaluating the per-class validation performance after a traditional training process, we aim to leverage the worst K performing classes. We focus the learning process on

*Author names are colored in red and green. Which one is which?

the worst performing classes to adjust the loss space by nudging the network weights in a direction that improves the classification of the worst performing classes. During each epoch, these worst performers may change, hence the *adaptive* characteristic of the algorithm.

II. Proposed Training Procedure

Our proposed training procedure is shown in Algorithm 1. We note that *Adaptive Focus Learning* is similar to the approach defined in [10]; however, our proposed approach contains a few key advantages in the modified procedure and applications in modern neural network architectures.

First, we enable the worst K classes, during the *Focus Stage* as opposed to just the single worst performer, where K is a hyperparameter that is tuned appropriately for each application. This is advantageous for a few reasons. In modern machine learning algorithms, the training process consists of navigating a "loss space" to find a point at which the loss is minimum, or at least close to minimum. The loss space of a particular problem, however, changes for each set of data you train on. In the case of *Adaptive Focus Learning* with $K = 1$, i.e. focusing on the single worst performing class, the focus-training procedure would update with respect to only observations that are a member of that class; this might bias the weights of the model too much on that particular class while consequently "unlearning" the representations of other classes. Increasing K to be greater than 1 would allow the model to focus on the worst performing class while also retaining a better representation of the other classes.

Also, our approach monitors the validation loss, not the training loss, during the evaluation stage to obtain the worst K classes. By monitoring the validation loss, we aim to avoid over-fitting on the training dataset so the model does not memorize information on the training set and believe it has reached a stopping point.

We also expand on the research of [10] by applying *Adaptive Focus Learning* on a modern dataset and neural network architecture. For example, [10] applied the minimax procedure on a 12 dimensional Gaussian input with one 7-node hidden layer and a 7-node output layer for a total of 147 trainable parameters. In our evaluation, we use the architecture defined in Table II with the CIFAR-10 dataset [12]. We aim to illustrate the *Adaptive Focus Learning* Algorithm can be used at scale with tunable parameters for modern applications.

We also allow for different gradient penalties on top of the learning rate to improve worst-case performance without sacrificing overall classification performance. As mentioned, we adjust the "loss space" quite a bit during the *Focus Stage*. We do not want the model to unlearn what it has generalized for the dataset as a whole during *Focus Stage*, where we only look at a small subset of the original training dataset. However, we also want to use the small subset consisting of the K worst classes to minimize the loss of the worst performers. Balancing this trade-off can depend on the application; however, in this work we aim to achieve parity between the overall and worst performing loss. With our algorithm, we enable a gradient penalty, $penalty$, as an input (see Algorithm 1) to be applied to the gradients in the *Focus Stage*. Using a penalty, we can effectively increase or decrease the learning rate during the *Focus Stage* as seen in Table I.

TABLE I
Focus Gradient penalties used in experiments.

Focus Gradient Penalty	Learning Rate	Effective Focus Learning Rate
0	1e-4	0
1e-2	1e-4	1e-6
0.5	1e-4	5e-5

Algorithm 1 Adaptive Focus Learning Algorithm

inputs :
Training dataset, D_{Train}
Validation dataset, $D_{Validation}$
Number of worst performing classes to monitor, K
Worst K gradient penalty to be applied, $penalty$

output:
A trained model, $M(D_{Train})$ based on training dataset D_{Train}

$D_{Total} \leftarrow D_{Train} + D_{Validation}$
foreach $epoch \in epochs$ **do**
  Stage 1 (Traditional training procedure):
  foreach *minibatch* $m_{Train} \in D_{Train}$ **do**
    $\nabla_{m_{Train}} \leftarrow$ gradients with respect to m_{Train}
    Apply gradient update to weights in $M(D_{Train})$
  Stage 2 (Evaluation):
    Evaluate $M(D_{Train})$ with $D_{Validation}$ for each class in $D_{Validation}$
    Compute the worst K performing classes based on the evaluation
    $D_{Worst_k} \leftarrow$ subset D_{Train} to worst K performing classes
  Stage 3 (Focus):
  foreach *minibatch* $m_{Worst_k} \in D_{Worst_k}$ **do**
    $\nabla_{m_{Worst_k}} \leftarrow$ gradients with respect to m_{Worst_k}
    $\nabla_{m_{Worst_k}} \leftarrow penalty * \nabla_{m_{Worst_k}}$
    Apply gradient update to weights in $M(D_{Train})$
  return $M(D_{Train})$

III. Dataset

In this work, we focus our experiments on the CIFAR-10 dataset [12] which consists of 10 classes with $50,000$ training observations and $6,000$ validation observations equally balanced across all classes. The classes are *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck*. Each observation is a 32x32x3 image where the first two dimensions represent the 2D spatial dimensions and the last dimension represents three color channels: red, green, and blue. We use the same train-validation split across all experiments in order to fairly compare the performance across all models.

TABLE II
MODEL ARCHITECTURE USED FOR ALL EXPERIMENTS.

Layer	Output Dimensions
Input	32 x 32 x 3
Conv2D	28 x 28 x 100
MaxPool2D	14 x 14 x 100
Conv2D	12 x 12 x 200
MaxPool2D	6 x 6 x 200
Flatten	7200
Dense/ReLU	100
Dense/ReLU	100
Dense/ReLU	100
Dense/Softmax	10

IV. MODEL ARCHITECTURE

To compare the performance of different training procedures, we use the same model architecture, seen in Table II, across all experiments in order to directly compare the performance of all classifiers. By utilizing the same architecture across all experiments, no modified training procedure has a pre-defined advantage before training. This architecture contains approximately $930,000$ trainable parameters. We do note that the architecture employed was not tuned for the dataset and there are likely other architectures better suited for the classification task; however, we use this architecture due to its simplicity and relatively low computational cost.

All models use the Adam optimizer with a learning rate of $1e-4$ for optimization [13]. To ensure that the models are not solely fine-tuning weights and achieving higher performance at random, we keep this learning rate consistent throughout the entire experiment. The ultimate goal of this work is to achieve a balance between the overall and worst-case performance of the classifier. We keep this learning rate fixed as a control on the experiments.

V. EXPERIMENTATION AND RESULTS

When evaluating the efficacy of the different training procedures, there are three objectives we consider. First, we want to maintain stability in the worst performing class across epochs. That is, we want the performance metric of the worst performing class to improve steadily, or remain consistent, across epochs and not fluctuate erratically. Second, we want to improve the per-class performance of the classifier. Third, the overall performance of the classifier, considering all classes, also achieves a good result.

To evaluate the proposed method, we measure the method's performance on four different parameter configurations. We use gradient penalty parameters of 0.5, 1e-2, and 0.0, where a 0.0 penalty is used as a baseline that reflects traditional training procedures. From non-baseline models, we take the model that resulted in the lowest validation loss for the worst performing class, which in our case was the model with the 1e-2 penalty, and trained a separate model using the same configuration but with a reversed modified training procedure. The reversed procedure shifts *Stage 1* to occur after *Stage 3* in Algorithm 1.

First, we observe the method's effect on the validation loss for the worst performing class. From Figure Figure 1, we

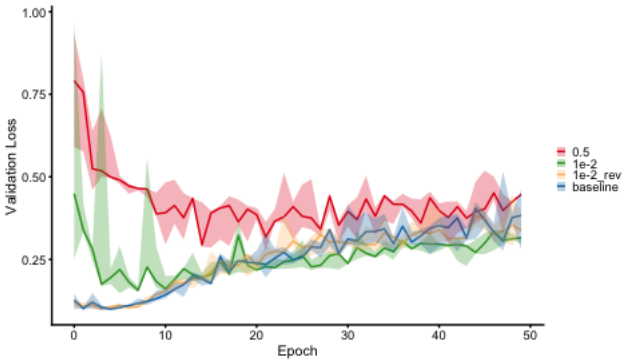

Fig. 1. Validation loss for worst performing class for each model configuration. Configurations include: gradient penalty of 0.5 (Red), 1e-2 (Green), 1e-2 with focusing first (Yellow), and the baseline with penalty of 0.0 (Blue). The solid line denotes the median loss of the 3 runs of each model, while the maximum and minimum loss is represented by the shading.

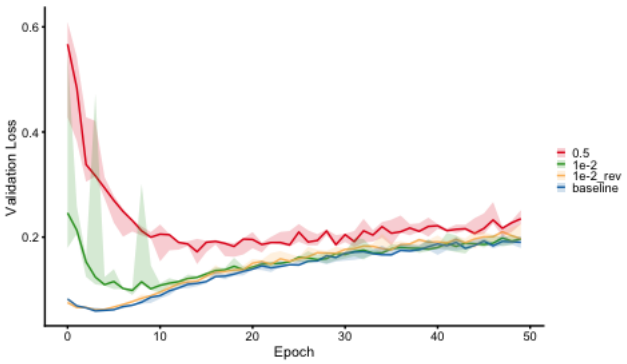

Fig. 2. Validation loss over all classes for each model configuration. Configurations include: gradient penalty of 0.5 (Red), 1e-2 (Green), 1e-2 with focusing first (Yellow), and the baseline with penalty of 0.0 (Blue). The solid line denotes the median loss of the 3 runs of each model, while the maximum and minimum loss is represented by the shading.

observe that the 1e-2 penalty configuration started with a lower validation loss than the 0.5 penalty. During training, both the 1e-2 and 0.5 configurations appear to have reached a steady-state, whereas the baseline and the reversed procedure continue to increase. The increase in validation loss for the baseline and reversed configurations is likely an effect of over-fitting and is further explored in Figure 2 and Figure 3. The steady increase in the validation loss in the baseline and reversed configurations might also be evidence that these models trade worse performance for a minority of classes for the benefit of better overall performance. However, our proposed method appears to mitigate this effect by allowing the loss for the worst performing class to reach a steady-state.

Evidence of over-fitting can be seen more clearly when observing the overall loss in both training and validation sets. In Figure 2, we see that all configurations, except the 0.5 penalty, start with low validation loss and steadily increase as the model over-fits to the training data. As shown in Figure 3, we see that each model's training loss quickly decreases. The steady increase in validation loss paired with the consistent decrease in training loss supports the evidence of over-fitting.

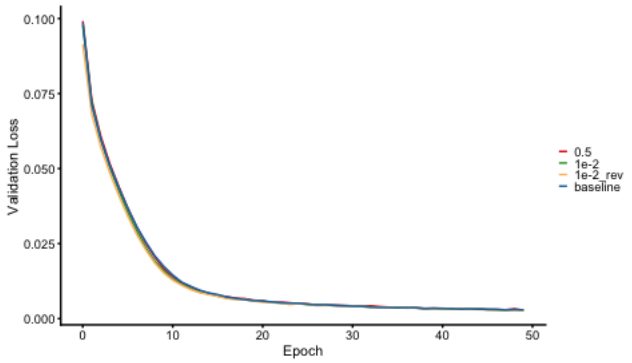

Fig. 3. Training loss for each model configuration. Configurations include: gradient penalty of 0.5 (Red), 1e-2 (Green), 1e-2 with focusing first (Yellow), and the baseline with penalty of 0.0 (Blue). Solid lines denote the mean training loss of 3 runs per configuration.

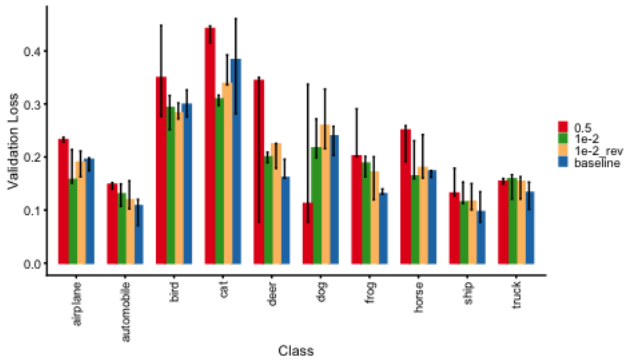

Fig. 4. Validation loss for each class on the final epoch for each model configuration. Configurations include: gradient penalty of 0.5 (Red), 1e-2 (Green), 1e-2 with focusing first (Yellow), and the baseline with penalty of 0.0 (Blue). Black error bars represent maximum and minimum loss values of the 3 runs and the bar itself represents the median.

While the model with the baseline configuration is shown to consistently have lower overall validation loss than all other methods, this divergence from the worst performing class is not as prevalent in methods with a focused gradient penalty.

As an unexpected result, we noticed that the 0.5 penalty approached a steady-state in the loss of the worst performing class and the overall validation loss. This result suggests that the higher penalty has a regularization effect on the training, keeping the model from over-fitting, and merits further exploration in future works.

In Figure 4, we see the validation loss for each of the ten classes in the CIFAR-10 dataset. Across all configurations the 'automobile' and 'ship' classes exhibit the lowest validation losses, while the 'cat' and 'bird' classes exhibit the highest validation losses. For 5 out of the 10 classes, the model with the baseline configuration performed best with respect to validation loss. The model with the 1e-2 penalty performed best for 3 out of the 10 classes and the reversed and 0.5 configurations performed best for 1 out of 10 each.

To compare the results of the computed models with each configuration, we ran pair-wised McNemar tests [14] to determine whether the predictions were statistically significant.

TABLE III
MCNEMAR COMPARISON RESULTS SHOWING THE NUMBER OF COMPARISONS FOUND TO BE SIGNIFICANT OVER THE NUMBER OF COMPARISONS.

Procedure	Baseline	1e-2	0.5	1e-2_rev
Baseline	0/3	0/9	9/9	7/9
1e-2		0/3	9/9	7/9
0.5			2/3	7/9
1e-2_rev				1/3

Table III shows the breakdown of the pairwise McNemar tests where each entry is the number of comparisons found to be statistically significant over the total number of comparisons. Values along the diagonal had 3 different comparisons since we ran each training procedure 3 times. Off-diagonal values had 9 comparisons because each of the 3 models were compared to the 3 models from the alternative procedure. All statistically significant results were determined with at least 95% confidence.

We can see that many of the comparisons were statistically significant when comparing results from models trained on different procedures. This is especially true for the 0.5 penalty which we found to have a regularizing effect during training. Because of the McNemar results, we can claim that the 0.5 penalty procedure on average creates a model that has predictions that are statistically different from the other procedures; however, this was not the case for the 1e-2 procedure.

VI. CONCLUSION

In this work we have explored a minimax-inspired training procedure for neural networks. Through the use of different gradient penalties during the *Focus Stage* we found that higher gradient penalties exhibited regularizing effects on the model and limited over-fitting. We also observed that modifying the procedure by moving the *Focus Stage* before the *Traditional* learning stage performed similarly to the baseline method, suggesting that keeping the *Focus Stage* after the *Traditional Stage* is necessary to appropriately focus on the worst performing classes.

Although we found the traditional baseline model to have the lowest overall validation loss and per-class validation loss, the proposed method, *Adaptive Focus Learning*, exhibited more stability with respect to the worst performing class during the training process. That is, the worst performing class validation loss achieves a steady-state whereas the baseline and reversed modified procedure per-class validation loss diverges during training. The baseline and reversed procedure methods showed evidence of over-fitting by observing the decrease in training loss paired with the increase in validation losses. We noted that this divergence from the worst performing classes could be evidence that the baseline and reversed modified procedure exploit the best performers to achieve higher overall performance at the expense of the worst performers.

REFERENCES

[1] Anil Jain, Lin Hong, and Sharath Pankanti, "Biometric identification," *Communications of the ACM*, vol. 43, no. 2, pp. 90–98, 2000.

[2] Jun Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.

[3] Minlong Lin, Ke Tang, and Xin Yao, "Dynamic sampling approach to training neural networks for multiclass imbalance classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 4, pp. 647–660, 2013.

[4] Bartosz Krawczyk, MichaÅ WoÅºniak, and Gerald Schaefer, "Cost-sensitive decision tree ensembles for effective imbalanced classification," *Applied Soft Computing*, vol. 14, pp. 554–562, 2014.

[5] Vaibhava Goel and William J Byrne, "Minimum bayes-risk automatic speech recognition," *Computer Speech Language*, vol. 14, no. 2, pp. 115–135, 2000.

[6] Willem Michel, *Minimax Theorem*, Birkhäuser, 1996.

[7] Gert Lanckriet, Laurent E Ghaoui, Chiranjib Bhattacharyya, and Michael I Jordan, "Minimax probability machine," in *Advances in neural information processing systems*, 2001, pp. 801–807.

[8] Kaizhu Huang, Haiqin Yang, I. King, and M.R. Lyu, "Learning classifiers from imbalanced data based on biased minimax probability machine," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, 2004, vol. 2, pp. II–II.

[9] Johnny K.C. Ng, Yuzhuo Zhong, and Shiqiang Yang, "A comparative study of minimax probability machine-based approaches for face recognition," *Pattern Recognition Letters*, vol. 28, no. 15, pp. 1995–2002, 2007.

[10] RocÃo Alaiz-RodrÃguez, Alicia Guerrero-Curieses, and JesÃºs Cid-Sueiro, "Minimax Classifiers Based on Neural Networks," *Pattern Recognition*, vol. 38, no. 1, pp. 29–39, 2005.

[11] Yoav Freund, Robert Schapire, and Naoki Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, pp. 1612, 1999.

[12] Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Tech. Rep., 2009.

[13] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2017.

[14] Quinn McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, June 1947.