batchboost: REGULARIZATION FOR STABILIZING TRAINING WITH RESISTANCE TO UNDERFITTING & OVERFITTING

DRAFT

Maciej A. Czyzewski
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60-965 Poznan, Poland
maciejanthonyczyzewski@gmail.com

January 21, 2020

ABSTRACT

Overfitting & underfitting and stable training are an important challenges in machine learning. Current approaches for these issues are *mixup*[1], *SamplePairing*[2] and *BC learning*[3]. In our work, we state the hypothesis that mixing many images together can be more effective than just two. *batchboost* pipeline has three stages: (a) pairing: method of selecting two samples. (b) mixing: how to create a new one from two samples. (c) feeding: combining mixed samples with new ones from dataset into batch (with ratio γ). Note that sample that appears in our batch propagates with subsequent iterations with less and less importance until the end of training. Pairing stage calculates the error per sample, sorts the samples and pairs with strategy: hardest with easiest one, than mixing stage merges two samples using *mixup*, $x_1 + (1 - \lambda)x_2$. Finally, feeding stage combines new samples with mixed by ratio 1:1. *batchboost* has 0.5-3% better accuracy than the current state-of-the-art *mixup* regularization on CIFAR-10[4] & Fashion-MNIST[5]. Our method is slightly better than SamplePairing technique on small datasets (up to 5%). *batchboost* provides stable training on not tuned parameters (like weight decay), thus its a good method to test performance on different architectures. Source code is at: https://github.com/maciejczyzewski/batchboost

***K*eywords** regularization · underfitting · overfitting · generalization · mixup

1 Introduction

In order to improve test errors, regularization methods which are processes to introduce additional information to DNN have been proposed[6]. Widely used regularization methods include *data augmentation*, *stochastic gradient descent* (SGD) [7], *weight decay* [8], *batch normalization* (BN) [9], *label smoothing*[10] and *mixup*[1]. Our idea comes from *mixup* flaws. In a nutshell, *mixup* constructs virtual training example from two samples. In term of batch construction, it simply gets some random samples from dataset and randomly mix together. The overlapping example of many samples (more than two) has not been considered in previous work. Probably because the imposition of 3 examples significantly affects the model leading to underfitting. It turned out that in many tasks, linear mixing (like *BC learning* or *mixup*) leads to underfittng (figure 3). Therefore, these methods are not applicable as universal tools.

Contribution Our work shows that the imposition of many examples in subsequent iterations (which are slowly suppressed by new overlays) can improve efficiency, but most importantly it ensures stability of training and resistance to attacks. However, it must be done wisely: that's why we implemented two basic mechanisms:

- (a) new information is provided gradually, thus *half-batch* adds new examples without mixing
- (b) mixing is carried out according to some criterion, in our case it is the best-the-worst strategy to mediate the error

The whole procedure is made in three steps to make it more understandable:

- (a) *pairing*: a method for selecting two samples
- (b) *mixing*: how to create a new one from two samples
- (c) *feeding*: to the mixed samples it supplements the batch with new examples from datasets

Note that sample that appears in our batch propagates with subsequent iterations with less and less importance until the end of training. Source code with sample implementation and experiments to verify the results we present here:

`https://github.com/maciejczyzewski/batchboost`

To understand the effects of *bootstrap*, we conduct a thorough set of study experiments (Section 3).

2 Overview

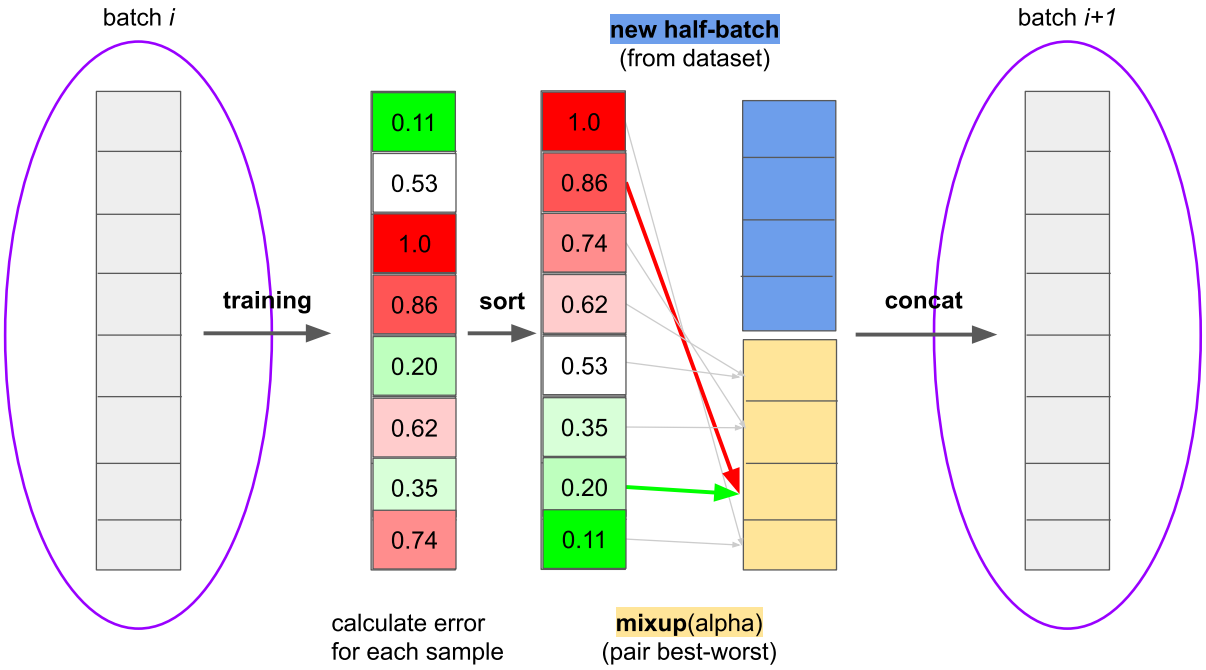

Figure 1: *batchboost* presented in three phases: (a) pairing by sorting error (b) mixing with *mixup* (c) feeding: a mixed feed-batch and new samples in half-batch by 1:1 ratio.

Batch as input for training is a combination of two different mini-batches:

- (a) *half-batch*: new samples from dataset, classical augumentation is possible here
- (b) *feed-batch* (mixup): samples mixed together (in-order presented in figure 1)

Parameter γ means the ratio of the number of samples in half-batch to feed-batch, in our work we have not considered other values than 1. However, we believe that this is an interesting topic for further research and discussion.

2.1 Pairing Method

Combining many overlapping samples may have a negative impact on our optimizer used in training. In our implementation, it calculates the error for each sample in batch. Then it sorts this vector, and pairs samples by connecting the easiest (smallest error) with the most difficult sample. The goal of this procedure is to create new artificial samples that are between classes, as described in *BC learning*.

However, in this case they are not random pairs, but those that 'require' additional work. In this way, the learning process is more stable because there are no cases when it mix only difficult with difficult or easy with easy (likely

is at the beginning or end of the learning process). In our case, the error was calculated using L2 metric between one-hot labels and the predictions (thus we analyzed *batchboost* only on classification problems like CIFAR-10[4] or Fashion-MNIST[5]). For other problems, there is probably a need to change the metric/method of error calculation. We were also thinking about using RL to pair samples. However, it turns out to be a more complicated problem thus we leave it here for further discussion.

2.2 Mixig Method

Selected two samples should be combined into one. There are three methods for linearly mixing samples: *SamplePairing*, *Mixup*, *BC Learning*. Due to the simplicity of implementation and the highest scores, we used a mixup, which looks like this:

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j, && \text{where } x_i, x_j \text{ are raw input vectors} \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j, && \text{where } y_i, y_j \text{ are one-hot label encodings} \end{aligned}$$

(x_i, y_i) and (x_j, y_j) are two examples drawn at random from our training data, and $\lambda \in [0, 1]$. Label for many samples was averaged over the last 2 labels (due to small differences in results, and large tradeof in memory).

Why it works? The good explanation is provided in BC learning research, that images and sound can be represented as waves. Mixing is an interpolation that human don't understand but machine could interpret. However, also a good explanation of this process is: that by training on artificial samples, we supplement the training data by between classes examples (visually, it fills space between clusters in UMAP/t-SNE visualization). Thus it generalizes problem more by aggressive cluster separation during training.

The question is whether linear interpolation is good for all problems. Probably the best solution would be to use a GAN for this purpose (two inputs + noise to control). We tried to use the technique described in SinGAN[11] but it failed in *batchboost*. It was unsuccessful due to the high cost of maintaining such a structure.

2.3 Continuous Feeding

The final stage is for 'feeding' new artificial samples on the model's input. In the previous researches, considered were only cases with mixing two samples along batch. *batchboost* do this by adding new samples with γ ratio to mixed ones. An interesting observation is that once we mix samples, they are in learning process till end (at each batch continuously). When applying a mixing it has only three options: (a) new sample with new sample (b) new sample with previously mixed sample (c) previously mixed sample with previously mixed sample. Pairing method cannot choose only one option for all samples because of non-zero γ ratio.

To maintain compatibility with the mixup algorithm, it chooses new λ when constructing the batch. That is why past samples have less and less significance in training process, until they disappear completely (figure 2).

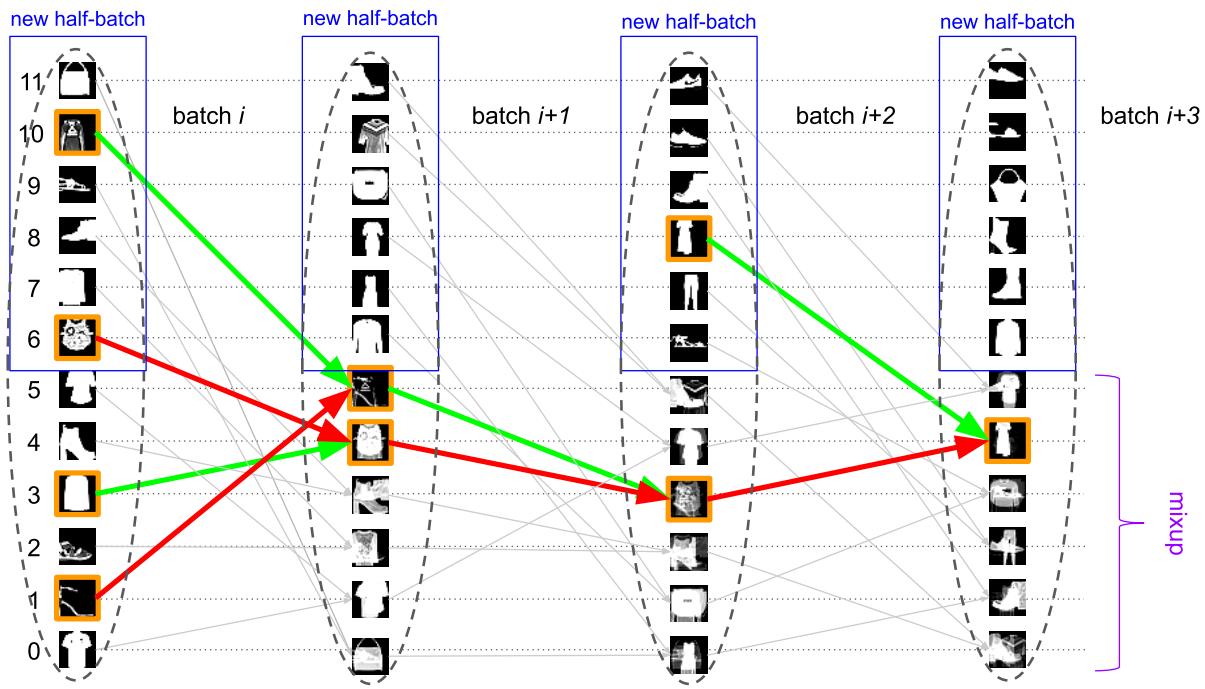

Figure 2: Orange squares indicates how information is propagated between batches in the *batchboost* method.

We found that for problems by nature not linear, for which the mixup did poorly, it was caused by the fact that model learned at the time when very low/high λ was assigned (i.e. model learned on a single example, without mixing). In *batchboost* it doesn't look much better. However, *half-batch* contains new information, and *feed-batch* has examples mixed not randomly but by pairing method. With this clues, optimizer can slightly improve the direction of optimization by better interpreting loss landscape.

3 Results

We focused on the current state-of-the-art *mixup*. The architecture we used was *EfficientNet-b0*[12] and *ResNet100k*[13] (having only 100k parameters from DAWNBench[14]). The problems we've evolved are CIFAR-10 and Fashion-MNIST. We intend to update this work with more detailed comparisons and experiments, test on different architectures and parameters. The most interesting issue which requires additional research is artificial attacks.

3.1 Underfitting & Stabilizing Training

We described this problem in the (section 2.3). The main factors that stabilize training are: (a) the appropriate pairing of samples for mixing, i.e. by error per sample (b) propagation of new information in *half-batch*.

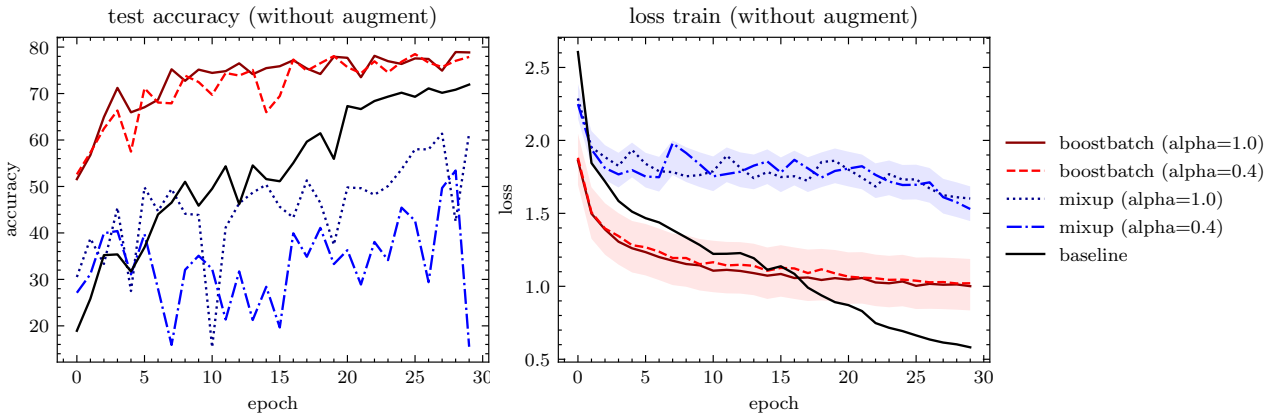

Figure 3: Evaluation on *CIFAR-10*, for *EfficientNet-b0* and *SGD(weight-decay=10e-4, lr=0.1)* (as recommended in the *mixup* research), same parameters for each model. As a result, the models behave differently, although they differ only in the method of constructing the batch.

Another problem that *mixup* often encounters is very unstable loss landscape. Therefore, without a well-chosen weight decay, it cannot stabilize in minimums. To solve this problem, we tune the optimizer parameters for *mixup*, after that it could achieve a similar result to *batchboost* (figure 4).

3.2 Overfitting (compirason to *mixup*)

The most important observation of this section is that *batchboost* retains the properties of the *mixup* (similarly to *SamplePairing* or *BC learning*). It protects against overfitting, having slightly better results.

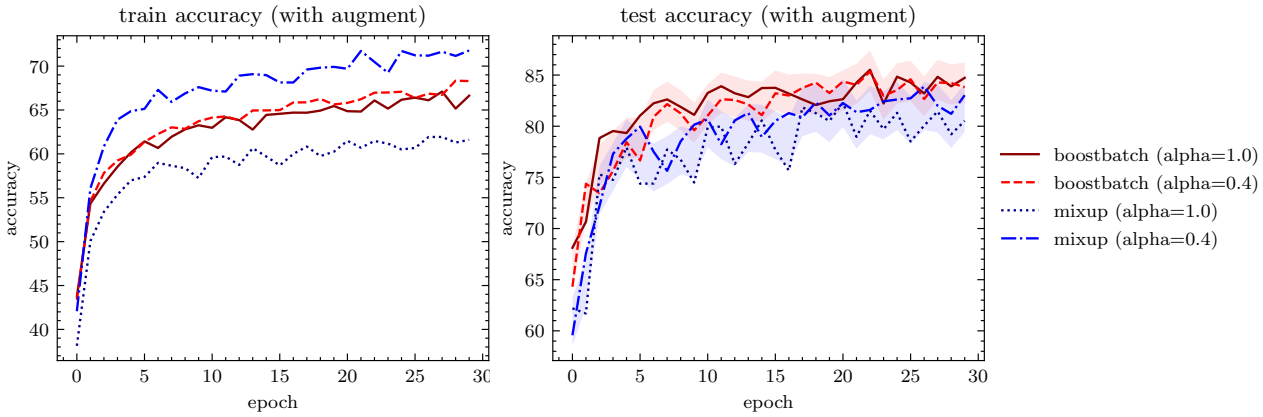

Figure 4: *batchboost* is a new state-of-the-art because it is a slightly better than *mixup* (here *mixup* has been tuned for best parameters, *batchboost* uses configuration from figure 3).

The only difference is that the α coefficient from the original *mixup* is weakened.

3.3 Accelerating Training & Adversarial Attacks

In the early stages, it learns faster than a classic *mixup*. The difference becomes significant when working on very small datasets, e.g. medical challenges on Kaggle. In this work, we have limited *Fashion-MNIST* to 64 examples we compared to the classic model and *SamplePairing*. The results were better by 5$. When the model perform well at small datasets, it means that training generalizes problem. On (figure 5) we present samples generated during this process.

Figure 5: More than two samples have been mixed.

We tried to modify *batchboost* to generate samples similar to those of adversarial attacks (by uniformly mixing all samples backward with some Gaussian noise) without any reasonable results.

4 Conclusion

Our method is easy to implement and can be used for any model as an additional BlackBox at input. It provides stability and slightly better results. Using *batchboost* is certainly more important in problems with small data sets. Thanks to the property of avoiding underfitting for misconfigured parameters, this is a good regularization method for people who want to compare two architectures without parameter tuning. Retains all properties of *mixup*, *SamplePairing* and *BC learning*.

References

[1] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[2] Hiroshi Inoue. Data augmentation by pairing samples for images classification. *arXiv preprint arXiv:1801.02929*, 2018.

[3] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5486–5494, 2018.

[4] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[5] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

[6] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[7] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[8] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[10] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[11] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4570–4580, 2019.

[12] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.