Diet Selective-Backprop: Accelerating Training in Deep Learning by Pruning Examples

Yu Shen Lu
Stanford University
yushenlu@stanford.edu

Daniel Zamoshchin
Stanford University
dzam@stanford.edu

Zachary Chen
Stanford University
zachchen@stanford.edu

Abstract

In this paper, we investigated multiple methods of data pruning for training deep neural networks on the CIFAR-100 dataset. Selective-Backprop samples data based on the training loss and drops examples that are unlikely to contribute significantly to the gradient updates; the EL2N score uses early training performance across multiple model runs to prune data points that are easy for the model to learn. We observed that both methods find a similar set of data points that are not "useful" for the training process, and both methods are able to effectively prune data without much loss in accuracy. Next, we showed that Random-Backprop, where we randomly select data to backprop, performs comparatively with Selective-Backprop in most cases and uses much less compute. In addition, we combined Selective-Backprop and EL2N to further reduce computational cost while maintaining similar model performance.

1. Introduction

Training with larger datasets generally leads to better performance for trained models [12]. However, training with a large amount of data requires an enormous amount of energy, time, and resources [1]. In constrained deployments with limited compute and storage, we want to be able to train on the smallest possible dataset with as little sacrifice to model performance as possible. To do so, we must identify the most important examples for the model to train on and exclude less important examples to reduce the size of the training dataset. Various techniques are proposed to reduce the amount of data used in training neural networks to reduce the cost of training. For example, Selective-Backprop [10] reduces the number of expensive backward passes by prioritizing high-loss training examples and skipping low-loss examples, while other methods aim to score the importance of training examples and prune those with poor scores using various scoring metrics [19].

In this paper, we first reimplemented Selective-Backprop [10] as well as EL2N scoring [19] in order to reproduce the results from their respective original papers and gain additional insight into the two methods. Specifically, we first ran a baseline ResNet18 model [6] on the CIFAR-100 dataset [14] in which we aimed to take in images as input, and for each image, output a predicted class label from the 100 total classes. Then, for this same classification task, we verified that our results for Selective-Backprop and EL2N matched the results in the original papers, and we performed an in-depth analysis of the images selected to be dropped by each method. Furthermore, we calculated EL2N scores using various models and then ran them using our baseline ResNet18 model to see if EL2N scores calculated on different architectures were transferable. Finally, we explored randomly choosing examples to backprop in our Random-Backprop method, and we combined Selective-Backprop and EL2N scores in an attempt to further improve the existing data selection methods by first pruning data using EL2N scores, then subsequently applying Selective-Backprop during training.

This paper makes three primary contributions: (1) We show that using a combination of EL2N scoring and Selective-Backprop, it is possible to reduce computation cost without sacrificing performance; (2) We show that EL2N scoring can be transferable across different CNN architectures; (3) We show that randomly choosing data points per batch to backprop can speed up training without affecting the model performance on CIFAR-100.

2. Related Work

In recent work, several attempts have been made to accelerate neural network training and to reduce the computational resources required for deep learning. One idea is importance sampling, which aims to select some training examples for training with higher probability, particularly examples that may correspond to larger updates. By generating a distribution over a set of training examples then training a model on examples sampled from that distribution, previous approaches were able to reduce the number

of training steps for classification tasks within some target error rate [5, 11, 16]. Other similar approaches were also able to reduce the number of training steps in reinforcement tasks [20] as well as in object detection tasks [15,21]. However, these approaches require storing historical losses for all the training examples in order to construct a distribution, and thus require storing additional states. As this doesn't scale very well for larger datasets, other approaches that rely on additional forward passes instead of historical data for importance sampling have shown to be successful at identifying important training examples and accelerating training [10,13]. In our paper, we further explore the "Selective-Backprop" approach proposed by Jiang et al. [10], which prioritizes backpropagation on the highest loss examples during training.

A similar idea to importance sampling is data pruning, which aims to identify coresets, that is, smaller subsets of larger sets of training data, that can be used to train to approximately the same accuracy as with using the original data. Though recent work in this area has shown to be able to identify coresets for various datasets that result in very similar training error to the original datasets [2, 3, 8, 9, 18], many of these approaches require training separate smaller models on the entire dataset first or only test their methods on relatively simple datasets with few classes, such as MNIST and CIFAR-10. A more recent approach to data pruning by Toneva et al. [23] calculates a "forgetting score" that tracks how often an example goes from being correctly classified to misclassified, with the idea being that rarely forgotten examples have less impact on training accuracy and can thus be pruned. However, as the forgetting score requires the collection of statistics during training, it is generally calculated towards the middle or end of training. Addressing this, Paul et al. [19] found that by using loss gradient norms of individual examples as a heuristic for identifying important examples, they were able to prune a significant fraction of training examples early in training, even as early as initialization, with no or little drop in accuracy on the CIFAR-10 and CIFAR-100 datasets, respectively. In our paper, we will further explore the use of the EL2N score defined by Paul et al. [19] across various models, and also combine EL2N with the Selective-Backprop [10] approach previously mentioned to create an improved data selection approach.

3. Methods

3.1. Baseline

For our baseline, we trained a ResNet18 model [7] on CIFAR-100 using an existing codebase [24], with the following hyperparameters [4]:

- Epochs = 200,
- Initial Learning Rate = 0.1 divided by 5 at 60th, 120th, 160th epochs,
- Batch Size = 128,
- Loss = CrossEntropyLoss,
- Optimizer = SGD w/ Nesterov Momentum = 0.9 and Weight Decay = 5e-4,
- Warmup = 1 Epoch

We used the same parameters for all the models we present in this report. Note that the codebase we used only contains starter code for running baseline models on CIFAR-100.

3.2. Selective-Backprop (SB)

The idea behind Selective-Backprop [10] is that instead of backpropping every single example in a training batch, which can be expensive, we can put more focus on backpropping the examples that our model is not classifying as well, that is, the examples with highest loss. Ideally, this reduces the amount of examples backpropped and saves resources with minimal loss in accuracy. To focus on the higher loss examples, we choose to backprop them with higher probability than lower loss examples.

For our Selective-Backprop method, we modified our baseline model by adapting an implementation from MosaicML [17] that follows the original paper [10]. For each training batch, instead of simply running a forward pass on the entire batch then backpropogating using the entire batch, we first ran a forward pass on all examples in the batch with disabled gradient calculation in order to get their losses, then sorted the losses from lowest to highest. Then, using hyperparameters $keep$ and $batchSize$, for each example i in the batch, we found its loss percentile $perc_i$ in the batch, and calculated its probability

$$p_i = \frac{perc_i^{\frac{1}{keep}-1}}{\sum_j^{batchSize} perc_j^{\frac{1}{keep}-1}}$$

to give us a probability distribution for the examples in the batch, summing to 1. We then sampled $(keep \times batchSize)$ examples from the batch without replacement using these probabilities, ran another forward pass using only these examples, and then backpropogating. For our experiments, we ran the baseline model normally for the first n_s epochs to give the model a chance to learn on and see all the data initially, where n_s is a hyperparameter, and then turning on Selective-Backprop for all remaining epochs. We ran experiments using $keep = 0.5$ and $n_s \in \{20, 100\}$.

3.3. EL2N Score

Let $p(\mathbf{w}, x) = \sigma(f(\mathbf{w}, x))$ be the softmax output of a neural network with weight $\mathbf{w}$ for an input x. Then, we can compute the EL2N score of training example (x, y) as

Algorithm 1 Selective-Backprop Training Cycle

function TRAIN($epoch, data, bSize$)
  for $batch_{fp}$ in $data.getBatches(bSize)$ **do**
    if $epoch > n_s$ **then**
      $losses \leftarrow n.Forward(batch_{fp})$ ▷ No grad
      $percs \leftarrow percentiles(losses)$
      $probs \leftarrow percs^{(1/keep)-1} / sum(percs^{(1/keep)-1})$
      $samples \leftarrow sample(keep \times bSize, probs)$
    else
      $samples \leftarrow batch_{fp}$
    end if

    $losses \leftarrow n.Forward(samples)$ ▷ With grad
    $losses.Backward()$
    $optimizer.step()$
  end for
end function

$$\mathbb{E}||p(\mathbf{w}, x) - y||_2. \text{ [19]}$$

where y is the label of the input data x in one-hot encoding.

Paul et al. [19] devises the GraNd score that ranks examples by the expected loss gradient norm. They find that the EL2N score is a good approximation for the GraNd score as early as epoch 20 in training, is easier to compute, and experimentally provides even better information for data-pruning.

To prune using EL2N, we first ran a baseline model for a smaller number of epochs and scored examples using the model at an early epoch. Then, we pruned a percentage of the data at initialization on the next training run by pruning the examples with the lowest EL2N scores. We then ran the baseline model normally on the pruned dataset. We also tried averaging calculated EL2N scores across 10 different model runs, then pruning a percentage of the data using the averaged EL2N scores and running a baseline model on that data. We implemented these scoring methods in PyTorch and ran our implementation to prune 25% and 50% of the data at initialization, trying EL2N scores calculated at both Epoch 20 and Epoch 100.

3.4. Diet Selective-Backprop (Diet SB)

To try to further reduce runtime, we combined SB and EL2N score by first using calculated EL2N scores to prune a percentage of the data at initialization, then also turning on SB during the training process. For our experiments, we tried pruning 25% of the dataset at initialization using averaged EL2N scores calculated at Epoch 20, then additionally applying Selective-Backprop using $n_s = 100$ and $keep \in \{0.33, 0.5\}$.

3.5. Random-Backprop

To measure the effectiveness of the selection process in SB, instead of backpropping on examples sampled with calculated probabilities based on their loss percentiles, we tried only backpropping on a set of ($keep \times batchSize$) randomly selected examples from the batch. We call this method Random-Backprop. Similar to SB, we only use Random-Backprop after epoch n_s. For our experiments, we tried $n_s \in \{20, 100\}$ and $keep \in \{0.5, 0.75, 0.9\}$.

Algorithm 2 Random-Backprop Training Cycle

function TRAIN($epoch, data, bSize$)
  for $batch_{fp}$ in $data.getBatches(bSize)$ **do**
    if $epoch > n_s$ **then**
      $samples \leftarrow sample(p = keep, batch_{fp})$
    end if

    $losses \leftarrow n.Forward(samples)$
    $losses.Backward()$
    $optimizer.step()$
  end for
end function

4. Dataset and Features

To investigate the efficiency and efficacy of our data selection algorithms, we ran multiple experiments on the CIFAR-100 dataset [14] with various models. The CIFAR-100 dataset is a popular Computer Vision dataset consisting of 60,000 32x32 color images in 100 classes, with 6,000 images per class, split into 50,000 training images and 10,000 test images. We choose this dataset because both [19] and [10] used this dataset in their experiments, so we can match our implementation with their results to verify our correctness. We performed the following transformations on the images in the training set: random crop with padding size 4, random horizontal flip, and random rotation up to 15 degrees. The images were also normalized with the mean and standard deviation calculated from the dataset.

5. Experiments/Results/Discussion

5.1. Baseline

For our baseline, we see the expected accuracy (74.69%) for the ResNet18 model on the CIFAR-100 dataset. We used all 50,000 images in the training set.

5.2. Selective-Backprop and Random-Backprop

With Selective-Backprop, we are able to match the baseline accuracy very precisely — this is also shown in the original paper [10]. There is a drop in accuracy shortly after the pruning starts, as shown in Figure 1, but the accuracy

comes back up quickly. All the runs have final accuracy within ± 0.005 of the baseline accuracy.

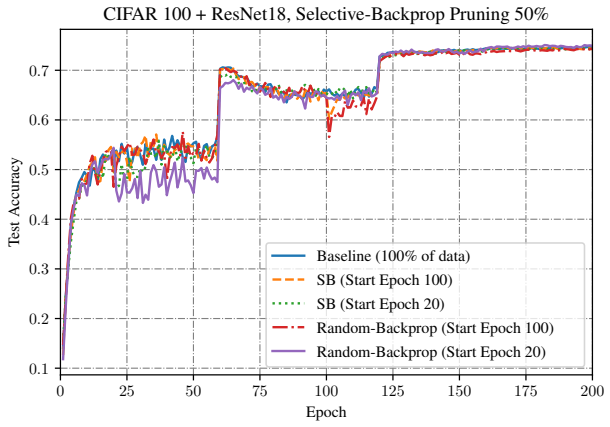

Figure 1. Validation accuracy of SB, pruning at 50%, starting the pruning process at different epochs. The final result for all runs is very close to the baseline.

An interesting discovery is Random-Backprop, although being more noisy in some parts of training, actually performs on par with SB in most cases. We also tried to more aggressively drop data with Random-Backprop, and we found that we can retain the same performance even when randomly dropping 75% of the data per batch. When we drop 90% data, the model accuracy dropped to 67.4%, and with SB, the model can retain 69.4%, this shows that while SB is useful when data is scarce, it is not necessary in most cases for CIFAR-100. In fact, we see that on average, Random-Backprop does marginally better than SB. We believe this might be because Random-Backprop actually shows the model more diverse data and backprops less on outliers, which might prevent the model from overfitting to outliers in training data. As outliers generally have high loss, SB will almost always choose to backprop on these outliers.

5.3. EL2N Pruning

Using EL2N pruning at 25% for the ResNet18 model on CIFAR-100, as shown in Figure 2, we achieved an accuracy of 71.91% (computing the score on one independent run), resulting in a 2.78% drop in accuracy from the baseline. As shown in Figure 3, pruning 50% of the data achieved an accuracy of 57.9%, resulting in a 16.79% drop in accuracy. This is a significant drop in performance. Interestingly, pruning 50% with EL2N is worse than pruning 50% randomly. Paul et al. [19] supports this result and hypothesizes that the sharp drop in performance is because high levels of pruning with the EL2N score may lead to an incomplete coverage of the data distribution.

Additionally, Paul et al. [19] recommends averaging the EL2N score over several runs to improve the performance of pruning. We tested averaging the EL2N score on ten independent models of a ResNet18 trained to epoch 20, as shown in Figure 2. We find that pruning 25%, after averaging the score over ten runs achieved an accuracy of 73.02%, only a 1.67% drop in accuracy from the baseline.

To better compare with Selective-Backprop which begins pruning data at epoch 100, we also computed the EL2N score on a model at epoch 100. Pruning 50% of the data with the score computed at epoch 100, we achieved an accuracy of 62.27%, resulting in a 4.37% increase in accuracy over the score computed at epoch 20. This comparison is displayed in Figure 3. Notably, this is still lower than dropping 50% data randomly. We hypothesize that at epoch 100, the EL2N score has a better idea of the data distribution than at epoch 20 but pruning a percentage of the data with EL2N still skews the distribution. Therefore, pruning with information from epoch 100 would better preserve the data distribution than pruning with information from epoch 20 but still performs worse than random.

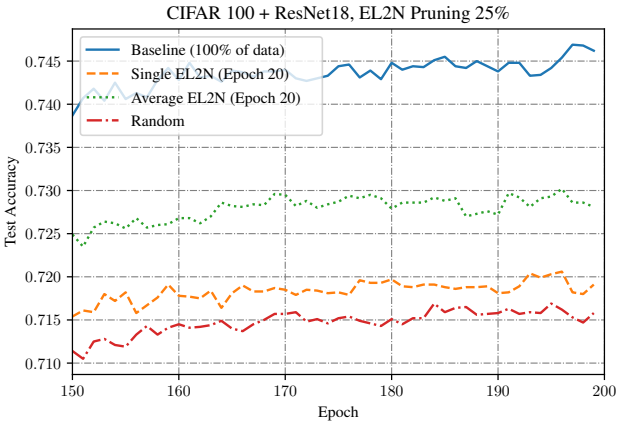

Figure 2. Validation accuracies are computed based on pruning 25% of the data with the EL2N score. This figure is zoomed in to the end of training to display the effects of averaging the EL2N score. The EL2N scores are computed at epoch 20 and 25% of the examples with the lowest scores are pruned at initialization. Final test accuracies are also compared using the EL2N score computed on one independent one versus averaging ten independent runs. Training on a random subset of the data was used as a baseline comparison to pruning with EL2N. The baseline accuracy without pruning is also displayed.

Finally, we illustrate the examples calculated as least and most valuable for training based on the EL2N score in Figures 4 and 5. It is difficult to characterize the usefulness of the examples; however, we can see a skewed class representation of sunflowers, keyboards, lawn-mowers, and trout in

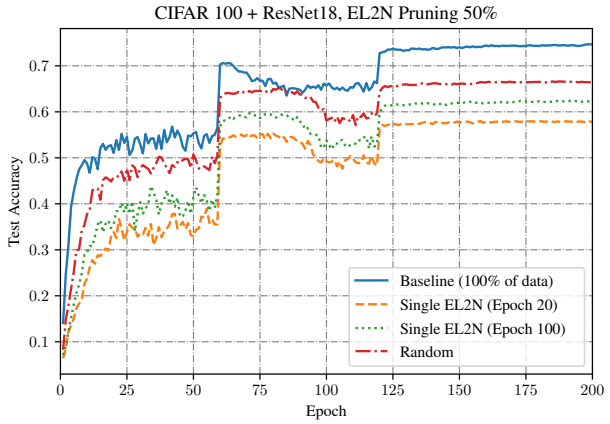

Figure 3. Validation accuracies are computed based on pruning 50% of the data with the EL2N score. The EL2N scores are computed at two different epochs and the dataset is pruned at initialization.

the lowest scoring examples and a more balanced class representation in the highest scoring examples. At least three of the keyboard images in the lowest 20 scoring examples are almost identical, so we hypothesize that the visual variety of images in these classes are low in CIFAR and it is unnecessary for the model to see all of these images to learn the class representation.

Figure 4. Examples with the 20 lowest EL2N scores from a ResNet18 trained on CIFAR-100. The EL2N score is based on an average of EL2N scores, calculated at epoch 20, over 10 independent runs.

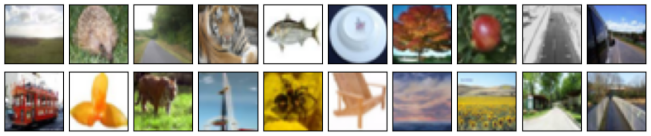

Figure 5. Examples with the 20 highest EL2N scores from a ResNet18 trained on CIFAR-100. The EL2N score is based on an average of EL2N scores, calculated at epoch 20, over 10 independent runs.

5.3.1 Different Model Architectures

We compared which examples are pruned when computing the EL2N score for two different CNN architectures: the ResNet and GoogLeNet [7, 22] architectures. We chose GoogLeNet because it has comparable performance to a ResNet18 with relatively similar training time and fewer parameters (6.2M compare to ResNet18's 11.2M parameters). In addition, it has inception modules that are not present in ResNet, which means that the two models are not just scaled versions of each other.

We hypothesized that if many of the same examples are pruned, then the EL2N scores can be transferred between different architectures. This means that we can train smaller models to calculate EL2N scores and then use the pruned dataset to train larger models.

To test this, we averaged the EL2N score for 10 independent runs of the ResNet18 model and 10 independent runs of the GoogLeNet model. In Figure 6, we compared the overlap of examples pruned for both architectures. When pruning the lowest 25% of examples respectively, we find that GoogLeNet and ResNet18 prune the same 94.62% of points. At 50%, both architectures prune the same 96.81% of points.

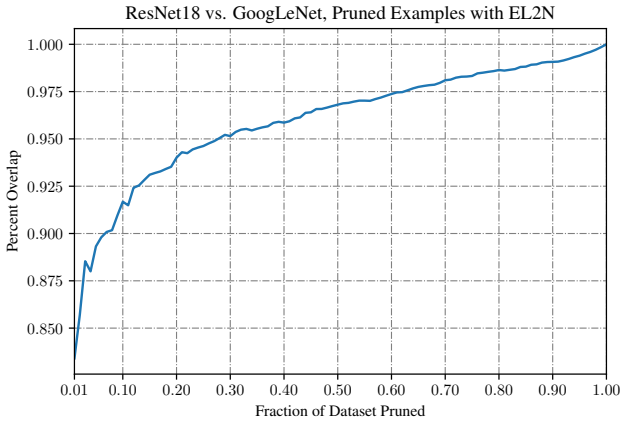

Figure 6. We compared the overlap of examples pruned at different fractions of the dataset between two different CNN architectures: GoogLeNet and ResNet18, base on EL2N scores.

Since the percentage of shared dropped examples between both architectures is very high (greater than 90% after pruning 10%), we can conclude that the EL2N score is transferable between both the GoogLeNet and ResNet architectures.

5.4. Comparing EL2N and SB

We want to know if the two methods select the same set of data. First, we want a way to visualize if there is any

propensity of data selection throughout the training process for SB. To do so, we visualized the 200 least selected points for SB, as seen in Figure 7. We see that although the points are sparse, they seem to be used throughout the training process rather than only for specific parts of training.

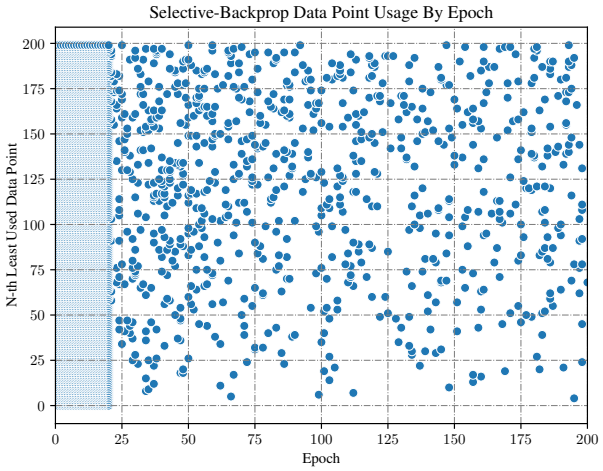

Figure 7. Usage of least used points in SB starting at epoch 20. There is no significant trend for SB's point selection.

Since there is no significant trends we can draw from SB's selection, we use the frequency of points included in SB as metric for how valuable SB found the points to be. We then compared the points that are least used in SB with the lowest scoring examples from EL2N. The result is shown in Figure 6. The correlation here is not as strong as the score between different runs of EL2N , but all the SB settings still show a much higher correlation than random. It is evident that there is significant overlap between points that are considered not "not useful for training" by the two methods. An interesting side note is that the correlation between different runs of SB is of the similar trend, this is lower than that between different EL2N runs due to the innate randomness in SB and throughout neural networks' training processes.

We report the top 20 images that are least useful for training as decided by both EL2N and SB in Figure 9. We see similar class representation and many repeated images as in EL2N scoring alone. There is a little more diversity in class, which might be because SB considers all images that the model does well on throughout the entire training process rather than just in the early stages. Similar to the images with highest scores in EL2N, there is a lot more diversity of the class for the images that are deemed "useful" for both methods, as shown in Figure 10.

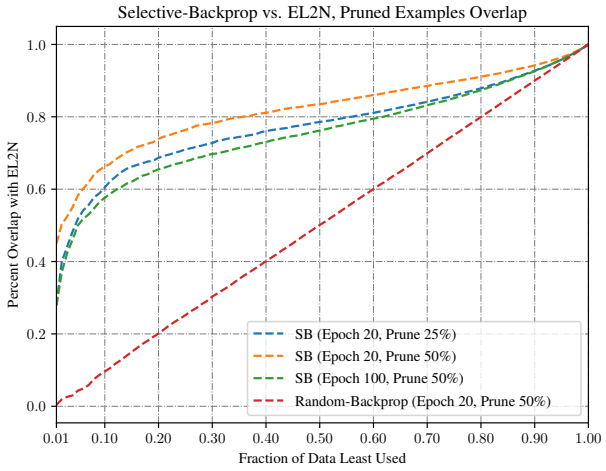

Figure 8. Images that are least used in SB are often have low score in EL2N. This graph shows that for the top N% of images that are least used in SB, how many are also in the N% lowest rated set in EL2N. We see a much higher correlation than random for all the SB methods.

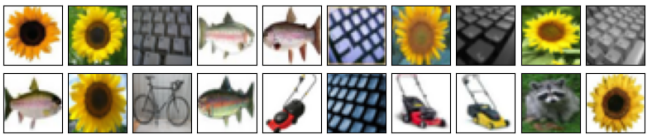

Figure 9. Examples with the lowest cumulative rank for EL2N and SB. The cumulative rank was computed by summing the rank from sorting the EL2N score and the rank of examples least used by SB.

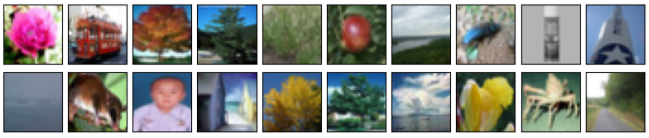

Figure 10. Examples with the highest cumulative rank for EL2N and SB.

5.5. Diet Selective-Backprop

Base on our observation, EL2N and SB drops similar set of data points, so combining them should be similar to just increasing the percent of data pruned by each method. However, this is still valuable since the two methods reduces computation cost at different stage, EL2N helps with hardware usage by removing data at initialization, whereas SB reduces arithmetic cost during training while preserving the

model performances.

The result of combined EL2N and Selective-Backprop, which we call Diet Selective-Backprop is shown in Figure 11. We dropped 25% of the data with EL2N at initialization and use SB during the training process. The final accuracy of Diet SB with approximately half of the data pruned is 72.29%, the final accuracy of Diet SB with 62.5% total data dropped is 72.24%. This shows that it is possible to combine the two methods; however, the accuracy seems to be limited by EL2N's performance, which has a higher impact to performance since it removes the data completely from the dataset before the training process. As we are unable to find the exact limit of the number of data that can be dropped by SB without influencing the performance, it is unclear if Diet SB can lead to better computation/performance trade-off, but we see that the performance of Diet SB is similar to that of EL2N despite Diet SB dropped half of the samples during training.

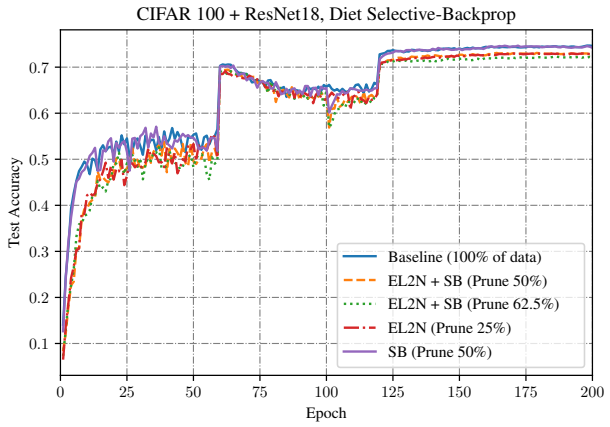

Figure 11. Validation accuracies for Diet SB are computed based on pruning 25% of the data with the EL2N score at initialization, and dropping additional data dynamically throughout the training process.

5.6. Runtime

Although reducing runtime was not the primary goal of this study, the reduction in computation can be seen in the change in runtime between different methods. Due to the implementation of the PyTorch models used in the experiment, there is additional overhead of an extra forward pass for all the Selective-Backprop based methods ran in this project. This can potentially be improved if PyTorch is not used. We run our experiments with AWS g4dn.xlarge on a Tesla T4, and the runtime for a typical training epoch for ResNet18 is reported in Table 1. Theoretically, the speed up for each of the methods will be much more significant with large models, since those models spend more time in back-propagation.

Method	**Total Pruned (Pre-Training)**	**Runtime per Epoch**
Baseline	0% (0%)	45.77s
SB	50% (0%)	43.37s
Random-Backprop	50% (0%)	**27.19s**
EL2N	25% (25%)	35.97s
Diet SB	**62.5% (25%)**	32.66s
Diet SB	50% (25%)	37.86s

Table 1. Typical runtime per epoch when training with different methods. All percentage are with respect to the total training dataset.

6. Conclusion/Future Work

In this project, we replicated results from two papers focusing on reducing computational cost of deep neural network training and compared the points that are pruned by each of the methods. We then found two ways to further reduce the computation cost: Random-Backprop, where only a random subset of the data in each batch is back-propped, and Diet Selective-Backprop, where we use calculated EL2N scores to prune a portion of the dataset at initialization and then use Selective-Backprop during training.

Our experiments showed that there is a strong correlation between EL2N scores calculated using different model architectures, meaning that EL2N scores calculated using different models may be transferable to other models and a new model may not have to recalculate EL2N scores. We also showed that on the CIFAR-100 dataset, there was a large overlap between data pruned by EL2N and data frequently excluded by SB. In particular, some classes like sunflowers, keyboards, and lawn-mowers were often pruned and excluded, possibly because images within those classes lack diversity. Furthermore, all of our methods showed speedup in runtime per epoch, with Random-Backprop providing the most significant speedup while seeing no drop in accuracy compared to the baseline. SB and Random-Backprop performed similarly in terms of accuracy when dropping 50% of the data, though as the amount of data dropped increases, SB outperforms Random-Backprop. Finally, we saw that combining EL2N and SB helped improve computation cost, although the performance of our Diet SB was largely bounded by EL2N, especially when we do not prune aggressively with SB.

In the future, it would be interesting to explore how a combination of EL2N and Random-Backprop would perform, as well as how well our methods work on other

datasets, such as ImageNet or SVHN, and models of different architectures or sizes, such as ResNet20, WideResNet, or GoogleNet. We also would like to account for randomness of the experiments since most of the change in accuracy are small, so it would be helpful to have an error estimation associated with the final result to see whether the change in accuracy is significant.

7. Appendices

7.1. CIFAR-10 result with ResNet20

In addition to training ResNet18 on CIFAR-100, we also trained ResNet20 [7] model since those are designed specifically for CIFAR. Since ResNet20 has fewer parameters, there is less time used in the forward and backward propagation process, and the time saving from SBP is not significant.

Method	Data Pruned	Final Accuracy
Baseline	0%	91.65%
Random-Backprop	50%	91.76%
SB	50%	91.73%

7.2. Additional Images Comparing SP and EL2N

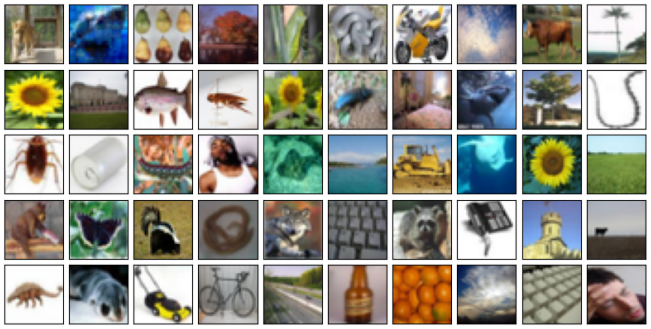

Figure 12. We found the top 50 images with the lowest absolute ranking difference between EL2N and Selective-Backprop. This may include images that are both likely and unlikely to be dropped by both methods. This shows that images in classes found in Figure 9 like sunflower, raccoon, trout, lawn-mower, and keyboard are closely ranked for both methods.

7.3. Correlation of Data Selected Between Different Runs of SB

See Figure 13.

8. Contributions and Acknowledgements

Advisor: Jonathan Frankle (Chief Scientist at MosaicML, PhD at MIT, new faculty at Harvard).

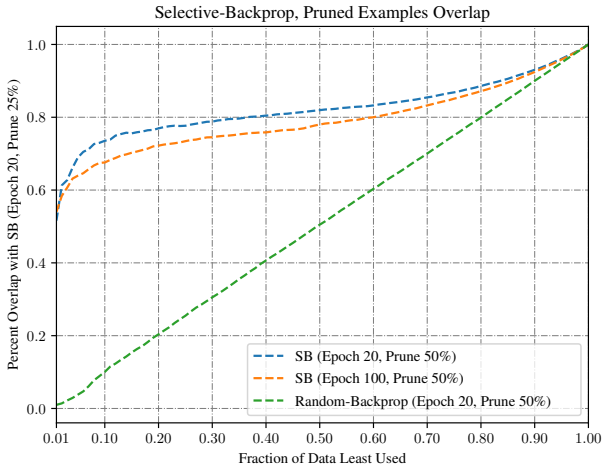

Figure 13. As mentioned in 5.4, the correlation of data dropped for different SB run is not as strong as that of EL2N due to random sampling, however the overlap is still significant.

Jonathan Frankle introduced us to the Selective-Backprop and EL2N papers [10, 19], pointed us to the MosaicML improvements on Selective-Backprop and answered all of our questions for training ResNet. We appreciate his guidance and advice throughout this project.

Link to codebase/starter code: https://github.com/weiaicunzai/pytorch-cifar100

Y.S.L. and Z.C. implemented and ran experiments for all variants of Selective-Backprop, D.Z. implemented and ran experiments for EL2N, Y.S.L., Z.C., and D.Z. wrote the paper.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020. 1

[2] Trevor Campbell and Tamara Broderick. Bayesian coreset construction via greedy iterative geodesic ascent, 2018. 2

[3] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning, 2019. 2

[4] Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 2

[5] Jinyang Gao, H. V. Jagadish, and Beng Chin Ooi. Active sampler: Light-weight accelerator for complex data analytics at scale. *CoRR*, abs/1512.03880, 2015. 2

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2, 5, 8

[8] Jonathan H. Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression, 2016. 2

[9] Myunggwon Hwang, Yuna Jeong, and Wonkyung Sung. Data distribution search to select core-set for machine learning. In *The 9th International Conference on Smart Media and Applications*, SMA 2020, page 172–176, New York, NY, USA, 2020. Association for Computing Machinery. 2

[10] Angela H. Jiang, Daniel L. K. Wong, Giulio Zhou, David G. Andersen, Jeffrey Dean, Gregory R. Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C. Lipton, and Padmanabhan Pillai. Accelerating deep learning by focusing on the biggest losers, 2019. 1, 2, 3, 8

[11] Tyler B Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 2

[12] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

[13] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. *CoRR*, abs/1803.00942, 2018. 2

[14] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 1, 3

[15] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017. 2

[16] Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. *CoRR*, abs/1511.06343, 2015. 2

[17] MosaicML. Selective backprop - mosaicml documentation. 2

[18] Alexander Munteanu, Chris Schwiegelshohn, Christian Sohler, and David P. Woodruff. On coresets for logistic regression, 2018. 2

[19] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. *CoRR*, abs/2107.07075, 2021. 1, 2, 3, 4, 8

[20] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. 2

[21] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016. 2

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 5

[23] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning, 2018. 2

[24] weiaicunzai. pytorch-cifar100. https://github.com/weiaicunzai/pytorch-cifar100, 2022. 2