Importance Sampling based on Loss

Abstract

In this project, we conducted experiments using new and existing importance sampling methods in deep learning. Our primary importance metrics are derived from the current loss and are accordingly very cheap to compute. However, we observe from experiments in CIFAR10 that importance sampling does not appear to significantly improve or even worsen convergence. We contrast this data selection method with a data variability enhancing method (data augmentation) and observe that the latter has a much more significant impact on a model's performance.

1 Introduction

1.1 Changes w.r.t Presentation

After running experiments on other datasets, we noticed a bug in the testing accuracy calculation that invalidated the presentation results. As experiments needed to be re-run this set back our plans for further exploration. The corrected results are featured in this report.

2 Background

2.1 Importance Sampling

Importance sampling is a variance reduction technique used when sampling from a distribution [15]. Suppose we have some function $f : \mathbb{R}^K \to \mathbb{R}$ dependent on a random vector $X \in \mathbb{R}^K$ with some underlying probability mass function (PMF) p_X. Then if we sample from p_X for N times we can approximate the expectation of $f(X)$ as follows:

$$\mathbb{E}[f(X)] \approx \frac{1}{N}\sum_{i=1}^{N} f(X_i), \qquad \varepsilon = \frac{1}{N}\text{VAR}[f(X)]$$

Where ε is our approximation error, which depends on the variance. It is possible to reduce this variance by re-weighting the probability of each outcome $x \in S_X$ with another random vector $Y \in \mathbb{R}^K$ and PMF p_Y as follows:

$$\mathbb{E}[f(X)] = \mathbb{E}\left[\frac{p_X(Y)}{p_Y(Y)}f(Y)\right], \qquad \varepsilon = \frac{1}{N}\text{VAR}\left[\frac{p_X(Y)}{p_Y(Y)}f(Y)\right]$$

Where $\forall x \in S_X, p_Y(x) > 0$ whenever $p_X(x) > 0$. If we weight $p_Y(y) \propto p_X(y)f(y), \forall y \in S_Y$ then the variance reduces to 0. Unfortunately, the ideal such p_Y already requires knowledge of the exact $\mathbb{E}[f(X)]$, which is what we are trying to approximate in the first place [13]. Finding the optimal reweighting distribution is therefore a challenge.

2.2 Importance Sampling in Machine Learning

In machine learning, we generally seek to optimise a model to minimise some loss function:

$$\theta^* = \arg\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\Psi(x_i;\theta), y_i)$$

Where $\theta, \theta^* \in \mathbb{R}^m$ are the model's parameters and optimal parameters, (x_i, y_i) is a datapoint from the training set $D = \{(x_1, y_1), (x_2, y_2)..., (x_N, y_N)\}$, $\mathcal{L} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is the loss function, and $\Psi : \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}$ is the prediction function.

A common optimisation method is gradient descent [11]. However, as the size of N increases, computing the gradient on the entire dataset becomes infeasible, which is why stochastic gradient descent (SGD) or minibatch gradient descent are preferred. The parameter update of SGD is given by the following:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \mathcal{L}(\Psi(x_{I_t}; \theta_t), y_{I_t}), \qquad P[I_t = i] = \frac{1}{N}$$

Where $t \in \mathbb{N}^+$ is the training step, $\eta \in (0, 1]$ the learning rate, $\nabla_{\theta_t} \in \mathbb{R}^m$ the gradient and $I_t \in \{1, 2, ..N\}$ a random variable that yields the index of the datapoint used during that training step. As shown above, the probability of choosing any particular datapoint is usually uniform. Naturally, this does not need to be the case and we can use importance sampling to reweight this sampling as follows:

$$\theta_{t+1} = \theta_t - \eta w_{I_t} \nabla_{\theta_t} \mathcal{L}(\Psi(x_{I_t}; \theta_t), y_{I_t}), \qquad w_{I_t} = \frac{p_U(I_t)}{p_{I_t}(I_t)} = \frac{1}{N p_i^{(t)}} \tag{1}$$

$$\mathbb{E}_{P_t}\left[w_{I_t} \nabla_{\theta_t} \mathcal{L}(\Psi(x_{I_t}; \theta_t), y_{I_t})\right] = \nabla_{\theta_t} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\Psi(x_{I_t}; \theta_t), y_{I_t}) \tag{2}$$

Where we now have some distribution where each datapoint i has some arbitrary probability $P[I_t = i] = p_i^{(t)}$, which we also let depend on the training step, as we may wish to update the distribution during training. The reweighting of the gradient by w_{I_t} also ensures that importance-sampled SGD remains unbiased as shown in the second equation [16][7].

Most probability distributions used for importance sampling are derived from metrics such as the loss or gradient norm [12][4] [5], where for convex optimisation it has been possible to show that the gradient norm is proportional to the optimal importance sampling distribution [10]. Unfortunately, the gradient norm for a single data point is very expensive to compute which is why the much more cheaply available loss or other approximations are used instead [7] [9].

2.3 Importance Sampling in Deep Learning

In deep learning, importance sampling methods frequently make use of loss due to it being easily obtainable from the forward pass. However, single loss values are quite unstable so often entire histories of the loss for a datapoint are kept and used to calculate metrics [6][12]. In some cases, they are even used to perform meta-learning by training additional models to manage the training of the base model [17]. However, most models face scalability issues with datasets larger than MNIST [2], CIFAR10 or CIFAR100 [8], likely because the cost of updating these metrics eventually outweighs gains from faster convergence. More recent work [1] has also argued that ultimately data variability in the form of augmentations such as RICAP [14] provides more significant improvements to accuracy and convergence than importance sampling.

3 Method

With the above in mind, we were interested in cheaper importance-sampling methods where the cost of calculating sampling distributions is low. Additionally, we also wished to verify how much of an effect the data variability of augmentations has on convergence. With this in mind, we implemented a mini-batch gradient descent algorithm with an overview in Algorithm 1. The detailed components will be explained in the following sections.

3.1 Scoring Datapoints

During each mini-batch step, we use a function $f_i : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ to calculate a score $S_i^{(t+1)} \in \mathbb{R}$ for each datapoint $i \in \{1, 2..., N\}$ in the mini-batch:

$$S_i^{(t+1)} = f_i(L_i^{(t)}, S_i^{(t)})$$

Algorithm 1 Importance-Sampling Mini-batch Training

Input: $D = \{(\mathbf{X}_i, y_i)\}_{i=1}^{N}, \mathbf{X}_i \in \mathbb{R}^{h \times w \times d}, y_i \in \{1, 2, ..., k\}, \theta_0 \in \mathbb{R}^m$

Let $t = 0, \mathbf{S}^{(t)} = \mathbf{0}, epochs = 0$
while θ_t has not converged **do**
  {warm-up using uniform sampling}
  if $epochs < warmups$ **then**
    for mini-batch $B \in$ shuffle(D) **do**
      Let $W = \{w_1, w_2, ..., w_{|B|}\}$, where $w_i = \frac{1}{N}, \forall i \in B$
      $\theta_{t+1}, L^{(t)} =$ mini_grad_descent(W, θ_t, B)
      $S_i^{(t+1)} = f_i(L_i^{(t)}, S_i^{(t)}, \forall i \in B)$
      $t := t + 1$
    end for
  {using importance-sampling}
  else
    $p_i^{(t)} = \text{softmax}\left(S_i^{(t)}, \mathbf{S}^{(t)}\right), \forall i \in D$
    Get $Z \subseteq D$ samples using distribution $\pi\left(p_1^{(t)} ... p_N^{(t)}\right)$
    for mini-batch $B \in$ shuffle(Z) **do**
      Let $W = \{w_1, w_2, ..., w_{|B|}\}$, where $w_i = \frac{1}{N p_i^{(t)}}, \forall i \in B$
      $\theta_{t+1}, L^{(t)} =$ mini_grad_descent(W, θ_t, B)
      $S_i^{(t+1)} = f_i(L_i^{(t)}, S_i^{(t)}), \forall i \in B$
      $t := t + 1$
    end for
  end if
  $epochs := epochs + 1$
end while

Where f_i takes in the current loss $L_i^{(t)}$ and score $S_i^{(t)}$. We implement two variants of f_i, the *last loss score* and *running loss score*.The former is given by:

$$f_i(L_i^{(t)}) = L_i^{(t)} = \mathcal{L}(\Psi(\mathbf{X}_i; \theta_{t-1}), y_i)$$

Which is simply the last loss of the datapoint. This measure is very cheap to compute and has no extra cost to calculate, merely requiring storage. We then extend upon this metric with the *running loss score*:

$$f_i(L_i^{(t)}, S_i^{(t)}) = \beta S_i^{(t)} + (1 - \beta) L_i^{(t)}$$

Where $\beta \in [0, 1]$ is hyper-parameter. Effectively, we take the weighted sum of the current score and current loss, to have a more stable metric. Again, this does not require any additional forward or backward passes through the network.

In both cases, we must also consider how to initialise all $S_i^{(0)} \in \mathbf{S}^{(0)}$. As loss is non-negative, an easy choice is $\mathbf{S}^{(0)} = \mathbf{0}$. However, in the context of the above metrics, this makes the sampler less inclined to sample unseen datapoints as they effectively have the lowest possible score from the beginning as opposed to already seen ones, which will usually have some loss. Instead, we implemented support for warmup episodes where we use uniform sampling and only update scores without generating probability.

3.2 Probability Calculation & Sampling

The probability generation from the scores can be done in several ways. We have opted for softmax as emphasises higher scores more than low ones, which should yield a less uniform distribution:

$$p_i^{(t)} = \text{softmax}\left(S_i^{(t)}, \mathbf{S}^{(t)}\right) = \frac{e^{S_i^{(t)}}}{\sum_{j=1}^{N} e^{S_j^{(t)}}}$$

We then parameterise a distribution $\pi \left(p_1^{(t)} ... p_N^{(t)} \right)$ that we can then sample from. Regarding whether to replace elements, we decided to leave this as a hyper-parameter as there was no clear advantage. In theory, sampling with replacement is faster, requiring $O(\log n)$ operations, whereas with replacement requires $O(n)$ as the probabilities of other elements have to be updated for each sample removed. However, sampling the same datapoint multiple times may overweight it when calculating the gradient and importance.

3.3 Data Augmentation

To compare the relevance of importance sampling with augmentation, we also apply 2 augmentations to each datapoint as it is sampled for training. As our datasets focus on images, we opted for a random crop operation and random flip, which should increase data variability.

4 Experiments

4.1 Setup

We ran our implementations on the CIFAR10 image dataset [8]. It is comprised of 10 classes, evenly divided among 50000 training examples and 10000 testing examples. For our model, we used a standard implementation of Resnet18 [3]. During training, we used minibatch gradient descent with batch size 128, with a learning rate of 0.01 and a step-wise decay of the learning rate by 0.1 at epochs 15 and 30. A full table of training hyper-parameters is provided in Appendix 1.

To choose these hyper-parameters, we tuned the model with a uniform sampler as a baseline. Then we tuned for each type of importance sampling's hyper-parameters, while keeping the hyper-parameters shared with the uniform case the same. This was done specifically to isolate the effect of using importance sampling on what should otherwise be the same training process. Additionally, we ran each tuned variant with and without data augmentation as a further comparison. Each run was conducted 3 times on the same 8-core 3.3GHz CPU and 8GB GPU to ensure some consistency.

4.2 Results

We compare the training loss, training accuracy and test accuracy of these 6 run configurations in Figures 1, 2, 3 respectively. What we can observe is that the *last loss score*, *running loss score* and uniform sampling have almost equivalent training loss and accuracy, which at first seems odd, but further investigation suggests that these importance sampling methods begin to approach uniform sampling within a few epochs of training (see Appendix B).

A more noticeable distinction can be made when we compare these metrics based on whether the methods use data augmentation. Training runs that use the augmentation experience slower convergence with training loss and training accuracy (reaching around 99% in the latter case) but achieve a better overall test accuracy (91% for augmented vs. 84% for non-augmented). The data variability induced by augmentation seems to have a more significant effect than data selection via importance sampling.

5 Evaluation

From our experiments, we can observe that importance sampling is, ironically, not that important and that our results concur with those of [1]. Of course, it should be mentioned that our chosen importance metrics are not nearly as advanced or impressive as some others [7] and that we could definitely experiment with some more expensive ones, given that *last loss score* and *running loss score* have no real computational footprint (see Appendix C). However, given that the more advanced importance sampling metrics seem to be unable to scale and basic ones do not have much impact, it begs the question of whether it makes sense to pursue a middle-ground solution. After all, there appear to be much simpler and more practical methods to make data more effective during training.

Figure 1: Depicts training loss of importance sampling methods using augmented vs. non-augmented. Generally, loss-based importance sampling methods seem to have a higher loss during training. Given that the distribution is biased towards such examples, this makes sense.

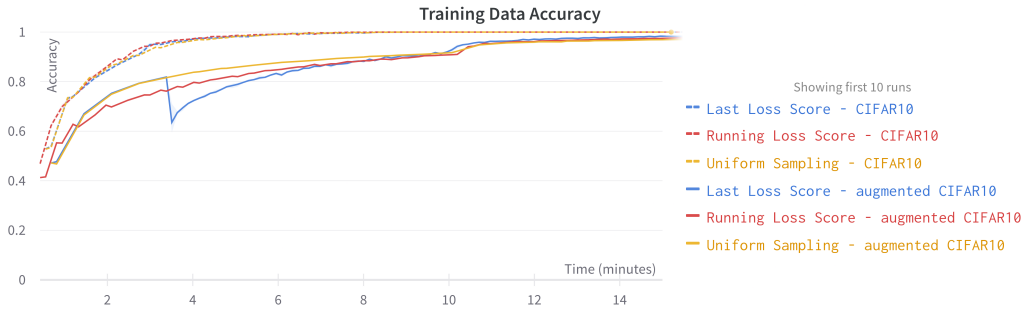

Figure 2: Depicts the accuracy vs. the training dataset. When CIFAR10 is not augmented with randomisation it appears that all methods achieve 100% accuracy, which is a sign that they are probably overfitting towards it.

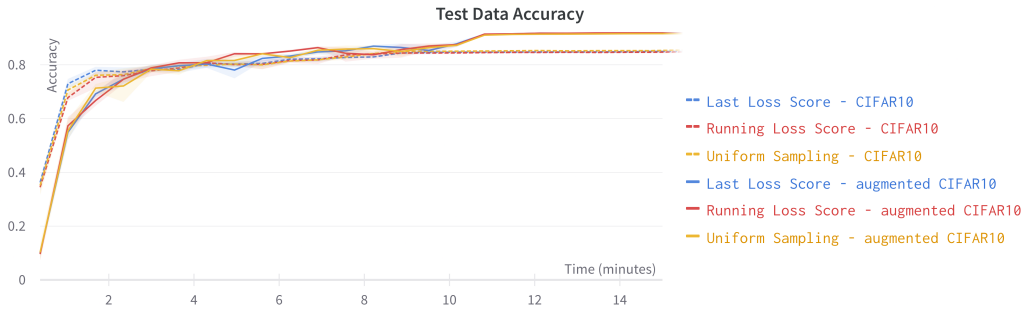

Figure 3: Depicts accuracy vs. the test dataset. Initially non-augmented CIFAR10 methods converge to a high accuracy more quickly, but do not end up generalising as well as the augmented CIFAR10 ones.

References

[1] E. Arazo, D. Ortego, P. Albert, N. E. O'Connor, and K. McGuinness. How important is importance sampling for deep budgeted training? *CoRR*, abs/2110.14283, 2021.

[2] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] A. H. Jiang, D. L. Wong, G. Zhou, D. G. Andersen, J. Dean, G. R. Ganger, G. Joshi, M. Kaminsky, M. Kozuch, Z. C. Lipton, and P. Pillai. Accelerating deep learning by focusing on the biggest losers. *CoRR*, abs/1910.00762, 2019.

[5] T. B. Johnson and C. Guestrin. Training deep models faster with robust, approximate importance sampling. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[6] A. Katharopoulos and F. Fleuret. Biased importance sampling for deep neural network training. *CoRR*, abs/1706.00043, 2017.

[7] A. Katharopoulos and F. Fleuret. Not all samples are created equal: Deep learning with importance sampling. *CoRR*, abs/1803.00942, 2018.

[8] A. Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[9] I. Loshchilov and F. Hutter. Online batch selection for faster training of neural networks. *CoRR*, abs/1511.06343, 2015.

[10] D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm, 2013.

[11] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[13] M. TABOGA. *Lectures on probability theory and Mathematical Statistics - 3rd Edition*. CREATESPACE, 2017.

[14] R. Takahashi, T. Matsubara, and K. Uehara. Ricap: Random image cropping and patching data augmentation for deep cnns. In J. Zhu and I. Takeuchi, editors, *Proceedings of The 10th Asian Conference on Machine Learning*, volume 95 of *Proceedings of Machine Learning Research*, pages 786–798. PMLR, 14–16 Nov 2018.

[15] S. T. Tokdar and R. E. Kass. Importance sampling: a review. *WIREs Computational Statistics*, 2(1):54–60, 2010.

[16] L. Wang, Y. Yang, M. R. Min, and S. T. Chakradhar. Accelerating deep neural network training with inconsistent stochastic gradient descent. *CoRR*, abs/1603.05544, 2016.

[17] J. Zhang, H. Yu, and I. S. Dhillon. Autoassist: A framework to accelerate training of deep neural networks. *CoRR*, abs/1905.03381, 2019.

Appendix A Hyper-parameter Table

Name	Notation	Uniform Sampler	Last Loss Sampling	Running Loss Sampling
number of samples	$\|Z\|$	50000 (N)	8192	30000
replacement	-	False	False	False
Running loss weight	β	-	-	0.2
warm-up epochs	$warmup$	-	5	0
optimiser	-	Mini-batch SGD	Mini-batch SGD	Mini-batch SGD
Minibatch size	$\|B\|$	128	128	128
Momentum	α	0.9	0.9	0.9
Weight decay	λ	0.0005	0.0005	0.0005
Learning rate	η	0.01	0.01	0.01
Learning rate decay Step-wise at epochs	γ	0.1 [15,30]	0.1 [15,30]	0.1 [15,30]
Early stopping after k epochs with no improvement	-	15	15	15

Table 1: Hyper-parameters specific to training process and importance sampling.

Appendix B Importance Sampling Scores

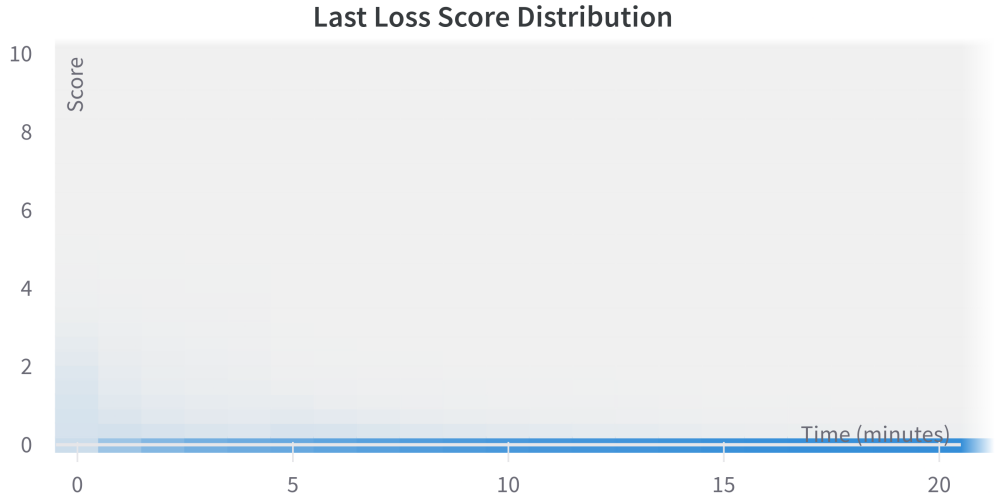

Figure 4: Distribution of scores under last loss. A higher opacity indicates a higher concentration of values. Note here that even though initially volatile, this metric begins to approach uniform scores and therefore uniform distribution eventually.

Figure 5: Distribution of scores under running loss. A higher opacity indicates a higher concentration of values. Here the scores start in a more uniform state than last loss and approach a uniform distribution more quickly.

Appendix C Cost of Sampling

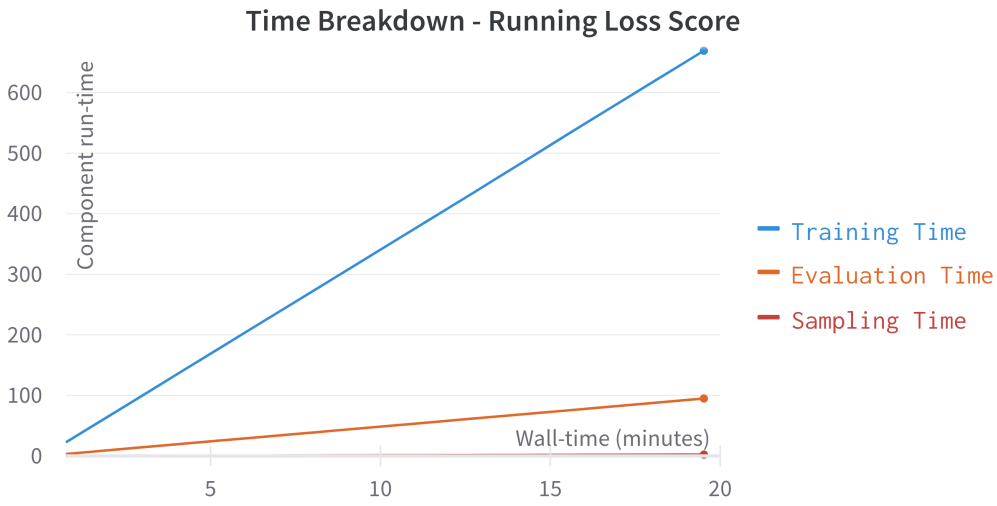

Figure 6: This is a breakdown in process time needed by different parts of a training run. Here, "training" signifies all forward passes, backward passes and gradient updates, "evaluation" only forward passes on evaluation data and "sampling" only the updating of scores and sampling of datapoints based on these scores.