VARIANCE REDUCTION IN SGD BY DISTRIBUTED IMPORTANCE SAMPLING

Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville & Yoshua Bengio*
Department of Computer Science and Operations Research
Université de Montréal
Montreal, QC. H3C 3J7
{guillaume.alain, alex.lamb, chinnadhurai.sankar,
aaron.courville}@umontreal.ca

ABSTRACT

Humans are able to accelerate their learning by selecting training materials that are the most informative and at the appropriate level of difficulty. We propose a framework for distributing deep learning in which one set of workers search for the most informative examples in parallel while a single worker updates the model on examples selected by importance sampling. This leads the model to update using an unbiased estimate of the gradient which also has minimum variance when the sampling proposal is proportional to the L2-norm of the gradient. We show experimentally that this method reduces gradient variance even in a context where the cost of synchronization across machines cannot be ignored, and where the factors for importance sampling are not updated instantly across the training set.

1 INTRODUCTION

Many of the advances in Deep Learning from the past 5-10 years can be attributed to the increase in computing power brought by specialized hardware (i.e. GPUs). The whole field of Machine Learning has adapted to this reality, and one of the latest challenges has been to make good use of multiple GPUs, potentially located on separate computers, to train a single model.

One widely studied solution is Asynchronous Stochastic Gradient Descent (ASGD), which is a variation on SGD in which the gradients are computed in parallel, propagated to a parameter server, and where we drop certain synchronization barriers to allow the algorithm to run faster. This method was introduced by Bengio et al. (2003) in the context of neural language models, and extended to model-parallelism and demonstrated on a large scale by Dean et al. (2012).

One of the important limitations of ASGD is that it requires a lot of bandwidth to propagate the parameters and the gradients. Moreover, many theoretical guarantees are lost due to the fact that synchronization barriers are removed and *stale* gradients are being used. Some theoretical guarantees can still be made in the context of convex optimization (see Agarwal & Duchi (2011), Recht et al. (2011), Lian et al. (2015)), but any result from convex optimization applied to neural networks (highly non-convex) has to be used with fingers crossed.

In this paper we present a different principle for distributed training based on importance sampling. We demonstrate many interesting theoretical results, and show some experiments to validate our ideas. The reader should view these experiments as a proof of concept rather than as an appeal to switch from Asynchronous SGD to Importance Sampling SGD. In fact, we can imagine our method being a supplement to ASGD.

Throughout this paper, we will use the word *stale* to refer to the fact that certain quantities are slightly outdated, but usually not to the point of being completely unusable. These stale values are usually gradients computed from a set of parameters θ_t when some reference model is now dealing with parameters $\theta_{t+\Delta t}$.

*Yoshua Bengio is a senior CIFAR Fellow

In section 2 we will explain our distributed Importance Sampling SGD approach. In section 3 we revisit a classical result from the importance sampling literature and demonstrate a more general result that applies to high dimensions. We also present a technique that can be used to compute efficiently the gradient norms for all the individual members of a minibatch. In section 4 we discuss our particular implementation for distributed training. In section 5 we show experiments to illustrate both the reduction in variance and the increase in performance that it can bring.

The main contribution of this paper is to open the door via theoretical and experimental results to a novel approach to distributed training based on importance sampling, to focus the attention of the learner on the most informative examples from the learning point of view.

2 Scaling Deep Learning by Distributing Importance Sampling

One of the most important constraints on ASGD is the fact that it requires a large amount of bandwidth. Indeed, all the workers connecting to the parameter server are required to regularly fetch a fresh copy of the parameters, and all their computed gradients have to be pushed to the parameter server. For every minibatch processed by a worker computing a gradient, the memory size of that gradient vector is equal to the memory size of the parameter vector for the model (i.e. every parameter value gets a gradient value). Delaying synchronization can result in "stale" gradients, that is, gradients that are computed from a set of parameters that have been fetched from the parameter server too long ago to be relevant.

The approach that we are taking in this paper is to focus on the most "useful" training samples instead of giving equal attention to all the training set. Humans can learn from a small collection of examples, and a good tutor is able to pick examples that are useful for a student to learn the current lesson. This work can therefore be seen as a follow-up on the curriculum learning ideas (Bengio et al., 2009), where the model itself is used to figure out which examples are currently informative for the learner. The method that we present in this paper will incorporate that intuition into a training algorithm that is justified by theory rooted in importance sampling (Tokdar & Kass, 2010).

The approach of calibrating the importance sampling coefficients in order to minimize variance during SGD is also presented by Bouchard et al. (2015), in their method called "Adaptive Weighted SGD", in which they adjust the coefficients by performing an intermediate gradient step to learn the best sampling proposal. They demonstrate how this can lead to improvements in convergence speed and generalization performance. In our paper, we show how an exact method can be used to get those optimal coefficients.

Compared to ASGD, our approach can be used to alleviate some of the communication costs. Instead of communicating the gradients on minibatches, the workers communicate one floating-point number per training sample. In a situation where the parameters can be of size ranging from 100 MB to 1GB, this cuts down the network transfers significantly. The parameters still have to be sent on the network to update the workers, however, but that cost can be amortized over a long period if the algorithm turns out to be robust to the use of older parameters in order to select the important samples. Our experiments confirm that hypothesis.

3 Importance Sampling in theory

3.1 Classic case in single dimension

Importance sampling is a technique used to reduce variance when estimating an integral of the form

$$\int p(x)f(x)dx = \mathbb{E}_{p(x)}\left[f(x)\right] \approx \frac{1}{N}\sum_{n=1}^{N} f(x_n) \text{ with } x_n \sim p(x)$$

through a Monte-Carlo estimate based on samples drawn from $p(x)$. Here $f(x)$ can only take on real values, but x can be anything as long as it's compatible with the probability density function $p(x)$.

It relies on a sampling proposal $q(x)$, for which $0 < q(x)$ whenever $0 < p(x)$, and the observation that

$$\mathbb{E}_{p(x)}\left[f(x)\right] = \mathbb{E}_{q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]. \tag{1}$$

Since all the quantities in the following empirical sum are independent,

$$\frac{1}{N}\sum_{n=1}^{N}\frac{p(x_n)}{q(x_n)}f(x_n) \;\text{ with }\; x_n \sim q(x)$$

we can directly verify they are unbiased and then try to minimize their variance. The unbiasedness follows directly from equation (1), and with a little work can prove that that the variance is minimized when

$$q^*(x) \propto p(x)\left|f(x)\right|. \tag{2}$$

3.2 Extending beyond a single dimension

In this section we generalize the classic importance sampling to allow the function f to take values in $\mathbb{R}^d$. The result referenced here as Theorem 1 is contained in the work of Zhao & Zhang (2014), but it is stated there without proof, and it is embedded in their specific context.

Minimizing the variance is a well-defined objective in one dimension, but when going to higher dimensions we have to decide what we would like to minimize.

For our application, a natural choice of objective function (Bouchard et al., 2015) would be the trace of the covariance matrix of the proposal distribution, $\mathrm{Tr}(\Sigma)$, because it corresponds to the sum of all the eigenvalues of Σ, which is a positive semi-definite matrix. It also corresponds to sum of all the variances for each individual component of the gradient vector. We can also imagine minimizing $\|\Sigma(q)\|_{\mathrm{F}}^2$, but in this case this would yield a different q^* for which we do not know of an analytical form.

A nice consequence of our choice is that, when $d = 1$, this $\mathrm{Tr}(\Sigma)$ will get back the classic result from the importance sampling literature. This is an pre-requisite for any general result.

Theorem 1. *Let $\mathcal{X}$ be a random variable in $\mathbb{R}^{d_1}$ and $f(x)$ be any function from $\mathbb{R}^{d_1}$ to $\mathbb{R}^{d_2}$. Let $p(x)$ be the probability density function of $\mathcal{X}$, and let $q(x)$ be a valid proposal distribution for importance sampling with the goal of estimating*

$$\mathbb{E}_p\left[f(x)\right] = \int p(x)f(x)dx = \mathbb{E}_q\left[\frac{p(x)}{q(x)}f(x)\right]. \tag{3}$$

The context requires that $q(x) > 0$ whenever $p(x) > 0$. We know that the importance sampling estimator

$$\frac{p(x)}{q(x)}f(x) \quad \textit{with } x \sim q \tag{4}$$

has mean $\mu = \mathbb{E}_p\left[f(x)\right]$ so it is unbiased.

Let $\Sigma(q)$ be the covariance of that estimator, where we include q in the notation to be explicit about the fact that it depends on the choice of q.

Then the trace of $\Sigma(q)$ is minimized by the following optimal proposal q^* :

$$q^*(x) = \frac{1}{Z}p(x)\left\|f(x)\right\|_2 \;\; \textit{where} \;\; Z = \int p(x)\left\|f(x)\right\|_2 dx \tag{5}$$

which achieves the optimal value

$$\mathrm{Tr}(\Sigma(q^*)) = \left(\mathbb{E}_p\left[\left\|f(x)\right\|_2\right]\right)^2 - \left\|\mu\right\|_2^2.$$

Proof. See appendix section A.1. □

Note that in theorem 1 we refer to a general function f. It should be understood by the reader that we are really interested in the particular situation in which f represents the gradient of a loss function with respect to the parameters of a model to be trained. However, since our results are meant to be more general than that, we tried to avoid contaminating them with those specific details, and decided to stick with $f(x)$ instead of talking about $\nabla_\theta \mathcal{L}(x_n)$.

Also, as a side-note, some readers would feel that it is strange to be taking the integral of a vector-valued function $f(x)$, but we would like to remind them that this is always what happens when we consider the expectation of a random variable in $\mathbb{R}^2$.

From theorem 1 we can get the following corollary 1. Here we introduce the notation $\tilde{\omega}_n$ to refer to un-normalized probability weights used in importance sampling (along with their normalized equivalents ω_n), which we are going to need later in the paper. In corollary 1, we do not assume that the probability weights are selected to be norms of gradients, but this is how they are going to be used throughout section 4.

Corollary 1. *Using the context of importance sampling as described in theorem 1, let $q(x)$ be a proposal distribution that is proportional to $p(x)h(x)$ for some function $h : \mathcal{X} \to \mathbb{R}^+$. As always, we require that $h(x) > 0$ whenever $f(x) > 0$.*

Then we have that the trace of the covariance of the importance sampling estimator is given by

$$\mathrm{Tr}(\Sigma(q)) = \left(\int p(x)h(x)dx \right) \left(\int p(x) \frac{\|f(x)\|_2^2}{h(x)} dx \right) - \|\mu\|_2^2 ,$$

where $\mu = \mathbb{E}_{p(x)}[f(x)]$. Moreover, if $p(x)$ is not known directly, but we have access to a dataset $\mathcal{D} = \{x_n\}_{n=1}^{\infty}$ of samples drawn from $p(x)$, then we can still define $q(x) \propto p(x)h(x)$ by associating the probability weight $\tilde{\omega}_n = h(x_n)$ to every $x_n \in \mathcal{D}$.

To sample from $q(x)$ we just normalize the probability weights

$$\omega_n = \frac{\tilde{\omega}_n}{\sum_{n=1}^{N} \tilde{\omega}_n}$$

and we sample from a multinomial distribution with argument $(\omega_1, \ldots, \omega_N)$ to pick the corresponding element in $\mathcal{D}$.

In that case, we have that

$$\begin{aligned} \mathrm{Tr}(\Sigma(q)) &= \left(\frac{1}{N} \sum_{n=1}^{N} \tilde{\omega}_n \right) \left(\frac{1}{N} \sum_{n=1}^{N} \frac{\|f(x_n)\|_2^2}{\tilde{\omega}_n} \right) - \|\mu\|_2^2 \\ &= \left(\frac{1}{N} \sum_{n=1}^{N} \omega_n \right) \left(\frac{1}{N} \sum_{n=1}^{N} \frac{\|f(x_n)\|_2^2}{\omega_n} \right) - \|\mu\|_2^2 . \end{aligned}$$

Proof. See appendix section A.1. □

3.3 Dealing with minibatches

To apply the principles of ISSGD, we need to be able to evaluate $\|g(x_n)\|_2$ efficiently for all the elements of the training set, where g here is the gradient of the loss with respect to all the parameters of the model.

In the current landscape of machine learning, using minibatches is a fact of life. Any training paradigm has to take that into consideration, and this can be a challenge when one considers that the gradient for a single training sample is as big the parameters themselves. This fact is generally not a problem since the gradients are aggregated for all the minibatch at the same time, so the cost of storing the gradients is comparable to the cost of storing the model parameters.

In this particular case, what we need is a recipe to compute the gradient norms directly, without storing the gradients themselves. The recipe in question, formulated here as proposition 1, was

published by Goodfellow (2015) slightly prior to our work. It applies to the fully-connected layers, but unfortunately not to convolutional layers.

Proposition 1. *Consider a multi-layer perceptron (MLP) applied to minibatches of size N, and with loss $\mathcal{L} = \mathcal{L}_1 + \ldots + \mathcal{L}_N$, where $\mathcal{L}_n$ represents the loss contribution from element n of the minibatch.*

Let (W, b) be the weights and biases at any particular fully-connected layer so that $XW + b = Y$, where X are the inputs to that layer and Y are the outputs.

The gradients with respect to the parameters are given by

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}_1}{\partial W} + \ldots + \frac{\partial \mathcal{L}_N}{\partial W} \\ \frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}_1}{\partial b} + \ldots + \frac{\partial \mathcal{L}_N}{\partial b} \end{aligned}$$

where the values $\left(\frac{\partial \mathcal{L}_n}{\partial W}, \frac{\partial \mathcal{L}_n}{\partial b}\right)$ refer to the particular contributions coming from element n of the minibatch. Then we have that

$$\begin{aligned} \left\| \frac{\partial \mathcal{L}_n}{\partial W} \right\|_F^2 &= \|X[n,:]\|_2^2 \cdot \left\| \frac{\partial \mathcal{L}}{\partial Y}[n,:] \right\|_2^2 \\ \left\| \frac{\partial \mathcal{L}_n}{\partial W} \right\|_2^2 &= \left\| \frac{\partial \mathcal{L}}{\partial Y}[n,:] \right\|_2^2 \end{aligned}$$

where the notation $X[n,:]$ refers to row n of X, and similarly for $\frac{\partial \mathcal{L}}{\partial Y}[n,:]$.

That is, we have a compact formula for the Euclidean norms of the gradients of the parameters, evaluated for each N elements of the minibatch independently.

Proof. See appendix section A.2. □

Note that proposition 1 applies to MLPs that have all kinds of activation functions and/or pooling operations, as long as the parameters (W, b) are not shared between layers. We can ignore the activation functions when applying proposition 1 because the activation functions do not have any parameters, and the linear operation part (matrix multiplication plus vector addition) simply uses whatever quantities are backpropagated without knowing what comes after in the sequence of layers.

Despite the fact that convolutions are linear operations (in the mathematical sense), this formula fails to apply to convolutions because of their sparsity patterns and their parameter sharing.

In a situation where we face convolutional layers along with fully-connected layers, proposition 1 applies to the fully-connected layers. For our purpose of performing importance sampling, this is not satisfying because we would have to find another way to compute the gradient norms for all the parameters. One might suggest to abandon the plan of achieving *optimal* importance sampling and simply ignore the contributions of sparsely-connected layers, but we do not investigate this strategy in this paper.

4 Distributed implementation of ISSGD

4.1 Using an oracle to train on a single machine

Assume for a moment that we are training on a single machine, and that we have access to an oracle that could instantaneously evaluate $\tilde{\omega}_n = \|g(x_n)\|_2$ on all the training set, then it is easy to implement importance sampling in an exact fashion. These $\tilde{\omega}_n$ depend on the model parameters currently sitting on the GPU, but we assume the oracle is nevertheless able to come up with the values.

We compose minibatches of size M based on a re-weighting of the training set by sampling (with replacement) the values of x_n with probability proportional to $\tilde{\omega}_n$. Let $(i_1, \ldots, i_M)$ be the indices sampled to compose that minibatch, that is, we are going to use samples $(x_{i_1}, \ldots, x_{i_M})$ to perform

forward-prop, backward-prop, and update the parameters. We have to scale the loss accordingly, as prescribed by importance sampling, so we end up using the loss

$$\mathcal{L}_\theta(\text{MINIBATCH}) = \left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n\right)\frac{1}{M}\sum_{m=1}^{M}\frac{1}{\tilde{\omega}_{i_m}}\mathcal{L}_\theta(x_{i_m}).$$

As as sanity check, we can see that this falls back to the usual value of $\frac{1}{M}$ if we find ourselves in a situation where all the $\tilde{\omega}_n$ are equal, which corresponds to the situation where the minibatch is composed from samples from the training set selected uniformly at random.

We can see from corollary 1 that the expected trace of the covariance matrix over the whole training set is given by

$$\text{Tr}(\Sigma(q)) = \left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n\right)\left(\frac{1}{N}\sum_{n=1}^{N}\frac{\|g(x_n)\|_2^2}{\tilde{\omega}_n}\right) - \|g^{\text{TRUE}}\|_2^2. \tag{6}$$

The constant $\|g^{\text{TRUE}}\|_2^2$ does not depend on the choice of q so we will leave it out of the current discussion. Refer to section B.2 for more details about it.

The oracle allows us to achieve the *ideal* Importance Sampling SGD, and this quantity becomes

$$\text{Tr}(\Sigma(q_{\text{IDEAL}})) = \left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n\right)^2 - \|g^{\text{TRUE}}\|_2^2. \tag{7}$$

In this situation, we are using q_{IDEAL} as notation instead of q^*. This is because we will want to contrast this situation with q_{UNIF} and q_{STALE} that we will define shortly.

When performing SGD training with q_{IDEAL}, we can plot those values of equation (7) as we go along, and we can compare at each time step the $\text{Tr}(\Sigma(q_{\text{IDEAL}}))$ with the value of $\text{Tr}(\Sigma(q_{\text{UNIF}}))$ that we would currently have if we were using uniform sampling to construct the minibatches. The latter are given by

$$\text{Tr}(\Sigma(q_{\text{UNIF}})) = \frac{1}{N}\sum_{n=1}^{N}\|g(x_n)\|_2^2 - \|g^{\text{TRUE}}\|_2^2. \tag{8}$$

In figure 4 we will see those quantities compared during an experiment where we do not have access to an oracle, but where we can still evaluate what would have been the $\text{Tr}(\Sigma(q_{\text{IDEAL}}))$ that we would have had if we had an oracle. This is relatively easy to evaluate by using equation (7).

4.2 Implementing the oracle using multiple machines

In practical terms, we can implement a close approximation to that oracle by throwing more computational resources at the problem. One machine is selected to be the *master*, running ISSGD, and it will query a database in order to read the probability weights $\tilde{\omega}_n$. Computing those probability weights and pushing them on the database is the job of a collection of *workers*. The workers spend all their lifetimes doing two things : getting recent copies of the parameters from the database, and updating the values of $\tilde{\omega}_n$ to keep them as fresh as possible. The master communicates its current model parameters to the database as regularly as possible, but it tries to do a non-trivial amount of training in-between.

Both the master and the workers have access to a GPU to process minibatches, but only the master will update the parameters (through ISSGD). The master is almost oblivious to the existence of the workers. It communicates its parameters to the database, and it gets a set of probability weights $\{\tilde{\omega}_n\}_{n=1}^N$ whenever it asks for them.

The presence of the database between the master and the workers allows the master to "fire and forget" its parameter updates. They are pushed to the database, and the workers will retrieve them when they are ready to do so. The same goes for the workers pushing their updates for the $\tilde{\omega}_n$. They are not communicating directly with the master. Among other things, this also allows for us to potentially use any database tool to get more performance (e.g. sharding), but we currently are not doing anything fancy in that regards.

Because of the various costs and delays involved in the system, the master should not expect the values of $\tilde{\omega}_n$ to be perfectly up-to-date. That is, the master has parameters $\theta_{t+\Delta_t}$ on its GPU, but it is receiving weights $\tilde{\omega}_n$ that are based on parameters θ_t. We refer to those weights as being *stale*.

There are degrees of staleness, and the usefulness of a weight computed 5 minutes ago differs greatly from that of a weight computed 2 seconds ago. We refer to q_{STALE} as the proposal that is based on all the weights from the previous iteration. It serves its role as pessimistic estimator, which is generally worse than what we are actually using. It is also easier to compute because we can get it from values stored in the database without having to run the model on anything more.

Our evaluation of $\text{Tr}(\Sigma(q_{\text{STALE}}))$ is based on assuming that all the probability weights come from the previous set of parameters, so they are certain to be outdated. Now it becomes a question of how much we are hurt by staleness. Without trying to introduce too much notation, for the next equation we will let $\tilde{\omega}_n^{\text{OLD}}$ refer to the outdated weights at a given time. Then we have that

$$\text{Tr}(\Sigma(q_{\text{STALE}})) = \left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n^{\text{OLD}}\right)\left(\frac{1}{N}\sum_{n=1}^{N}\frac{(\tilde{\omega}_n)^2}{\tilde{\omega}_n^{\text{OLD}}}\right) - \|g^{\text{TRUE}}\|_2^2 . \tag{9}$$

We know for a fact that $\text{Tr}(\Sigma(q_{\text{IDEAL}}))$ is the lower bound on all the possible $\text{Tr}(\Sigma(q))$. When the weights are not in a horrible state due to excessive staleness, we generally observe experimentally that the following inequality holds:

$$\text{Tr}(\Sigma(q_{\text{IDEAL}})) \leq \text{Tr}(\Sigma(q_{\text{STALE}})) \leq \text{Tr}(\Sigma(q_{\text{UNIF}})).$$

This has proven to be verified for the all the practical experiments that we have done, and it does not even depend on the training being done by importance sampling. Again, this is not an equality that holds all the time, and setting the probability weights $\tilde{\omega}_n$ to be randomly generated values will break that inequality.

4.3 Exact implementation vs relaxed implementation

To illustrate the whole training mechanism, we start in figure 1 by showing a *synchronized* version of ISSGD. In that diagram, we show the database in the center, and we have horizontal dotted lines to indicate where we would place synchronization barriers. This would happen after the master sends the parameters to the database, because it can decide then to wait for the workers to update all the probability weights $\tilde{\omega}_n$. The workers themselves could work their way through all the training set before checking for recent parameter updates present on the database. This is rather excessive, which is why we use those synchronization barriers only to perform sanity checks, or to study the properties that ISSGD if it was performed in the complete absence of staleness.

This kind of relaxation is analogous to how ASGD discards the synchronization barriers to trade away correctness to gain performance. However, in the case of ISSGD, stale probability weights may lead to more variance but we will always get an unbiased estimator of the true gradient, even when we get rid of all the synchronization barriers.

In the appendix we discuss three aspects of how training can be adapted to be more practical and robust. In section B.1 we discuss the possibility of using only a subset of the probability weights, filtering them based on how recently they have been updated. In section B.2 we discuss how to approximate $\|g^{\text{TRUE}}\|_2^2$, which is not a quantity that we absolutely need to compute for perform training, but which is something that we like to monitor to assess the benefits of using ISSGD instead of regular SGD. In section B.3 we add a smoothing constant to the probability weights in order to make training more robust to sudden changes in gradients.

5 Experimental results

5.1 Dataset and model

We evaluated our model on the Street View House Numbers (SVHN) dataset from Netzer et al. (2011). We used the cropped version of the dataset (sometimes referred to as SVHN-2), which contains about 600,000 32x32 RGB images of house number digits from Google Street View.

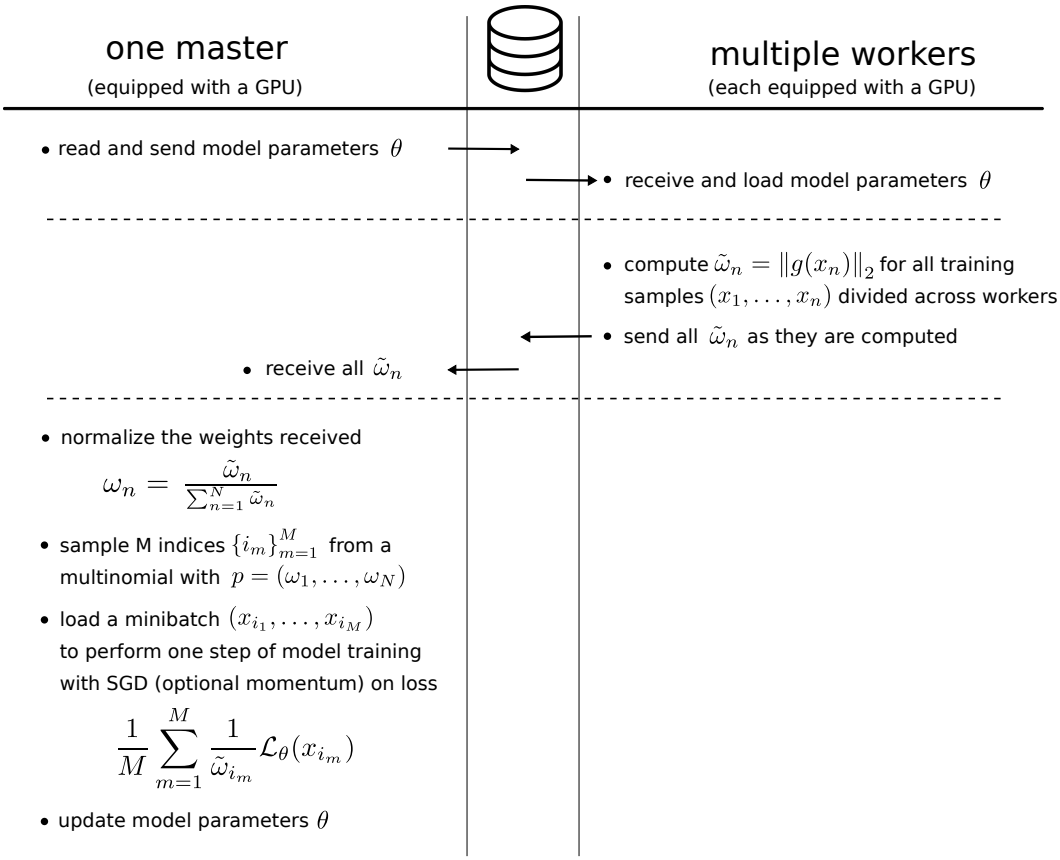


Figure 1: The actual distributed training experiment that we run relies on 3 kinds of actors. We have one *master* process that is running ISSGD. We have one *database* process in charge of storing and exchanging all kinds of measurements, as well as the parameters when they are communicated by the master to the workers. We have multiple *worker* processes, each with one GPU, in charge of evaluating the quantities necessary for the master to do importance sampling. The master has to read the model parameters from the GPU before sending them to the database, and the workers also have to load them unto the GPU after receiving them. The horizontal dotted lines represent synchronization barriers that we can enforce to have an exact method, or that we can drop to have faster training in practice.

Since there is no standard validation set, we randomly split 5% of the data to form our own validation set. Since our per-example gradient norm computation (from section 3.3) does not work with parameter sharing models (such as RNNs and Convnets), we consider the permutation invariant version of the SVHN task, in which the model is forced to discard the spatial structure of the pixels. While the permutation-invariant task is not practically relevant (as the spatial structure of the pixels is useful), it is commonly used as a testbed for studying fully connected neural networks (Goodfellow et al., 2013; Srivastava et al., 2013).

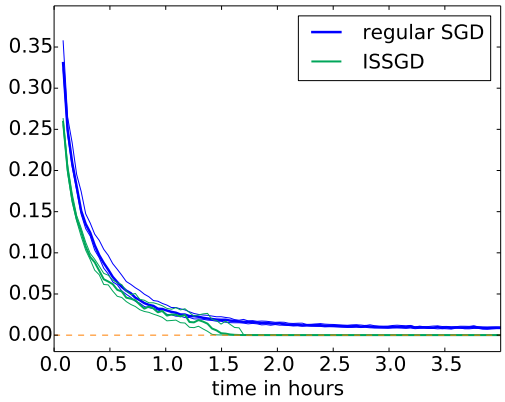
This is not meant to be a paper about exploring a variety of models, and since we stick with the permutation-invariant task this already limits our ability to use more interesting models. In any case, we picked an MLP with 4 hidden layers, each with 2048 hidden units and with a ReLU at its output (except for a softmax at the final layer). We are very much aware that a convolutional model would perform better.

We have used Theano ((Bergstra et al., 2010; Bastien et al., 2012)) to implement the model, and Redis as a database solution. The master and workers are each equipped with a k20 GPU.

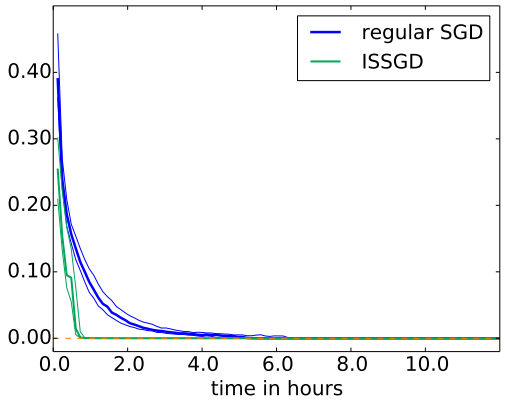
5.2 REDUCED TRAINING TIME AND BETTER PREDICTION ERROR

We compare in figure 2 the training loss for a model trained with ISSGD (in green) and regular SGD (in blue). We used 3 workers to help with the master. In the case of regular SGD, we also used a worker in the background to be able to compute statistics as we go along without imposing that burden on the process training the model. To make sure that the results are not due to the random initialization of parameters, we ran this experiment 50 times. We report here the median (thicker line), and the quartiles 1 and 3 above and below (thinner lines). This represents a "tube" into which half of the trajectories fit.

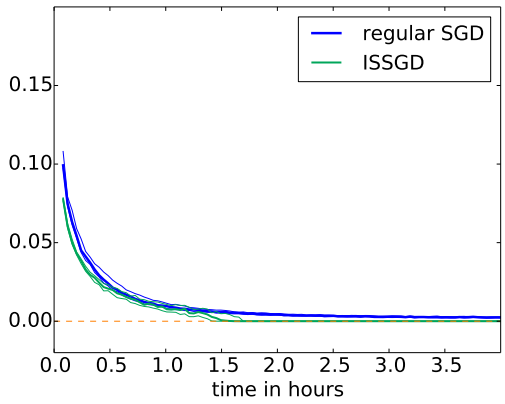
In all the figures from this section, we always compare the same two sets of hyperparameters. On the left we always have a setting where the learning rate is higher (0.01) and where we smoothe the probability weights by adding a constant (+10.0) to them (see section B.3 in the appendix for more explanations on this technique). On the right we always have a setting where the learning rate is smaller (0.001) and where the smoothing constant is also smaller (+1.0).



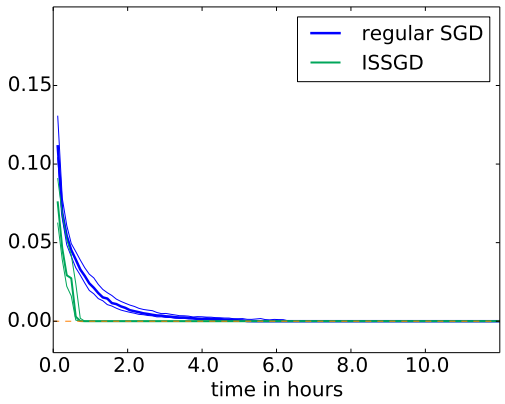
(a) train loss, learning rate 0.01



(b) train loss, learning rate 0.001



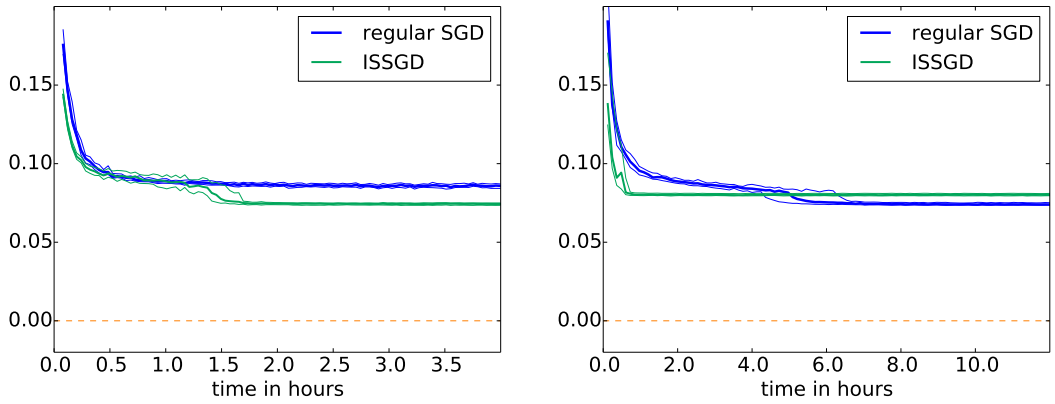
(c) train prediction error, learning rate 0.01



(d) train prediction error, learning rate 0.001

Figure 2: Here in the top two plots we compare the training loss optimized with two sets of hyperparameters. On the top-left we use a higher learning rate, but also a higher smoothing of the importance weights to stabilize the algorithm. In the two top plots, these are the actual quantities that are getting minimized by our procedure. We can see that, in both cases, ISSGD minimizes the loss more quickly than regular SGD, and it actually reaches 0.0. Those results are the median quantities reported during 50 runs for each set of hyperparamters, using a different random initialization. We also show the quartiles 1 and 3 in thinner lines to get an idea of the distributions. In the two bottom plots we also report the prediction error on the training set for each method. Note the different time scale between the left and the right.

In figure 2 we can see that in both cases ISSGD minimizes the train loss more quickly than regular SGD, and it actually reaches 0.0. This obviously corresponds to overfitting, but since we are pre-



(a) prediction error test, learning rate 0.01

(b) prediction error test, learning rate 0.001

Figure 3: Here we report the prediction error on the test set. Just like in figure 2, we report the median results over 50 runs with the same two sets of hyperparameters. In a fairly consistent way, we have that one setup has a better generalization error for ISSGD (on the left plot), and the opposite happens in the other scenario (right plot). We believe that this can be explained by ISSGD converging quickly to a configuration that minimizes the loss perfectly, after which it just gives up trying to do better. Regular SGD, on plot (b), would appear to experience some kind of regularization due to its variance, and it would continue to optimize over the course of 6 hours instead of only one hour (as shown on plot 2b).

Model	Test Error
SGD (Davis & Arel, 2013)	9.31
SGD (ours)	0.0754
Importance Sampling SGD	**0.0756**

Table 1: Test Error on Permutation Invariant SVHN Datasets with different methods. Our results are aggregated from 50 runs with random initialization, and we report the average prediction error (as percentage) over the final 10% iterations. In this case, the measured results are similar when using early stopping on validation set.

senting here an optimization method, it seems natural to celebrate the fact that it can minimize the objective function faster and better.

In figure 3 we show the test prediction error. These results are not so easy to interpret, and we see that faster convergence does not always lead to a better generalization error. This suggests that regular SGD benefits here from a kind of regularization effect.

We also report in table 1 what are the final prediction errors for both methods (averaged over the last 10% of the timesteps plotted). We picked the set of hyperparameters that had the best validation prediction error and reported the test prediction errors. Unsurprizingly, this corresponds to using the result from figure 3a for ISSGD and figure 3b for regular SGD. The final values are very similar for the two methods.

5.3 Variance reduction

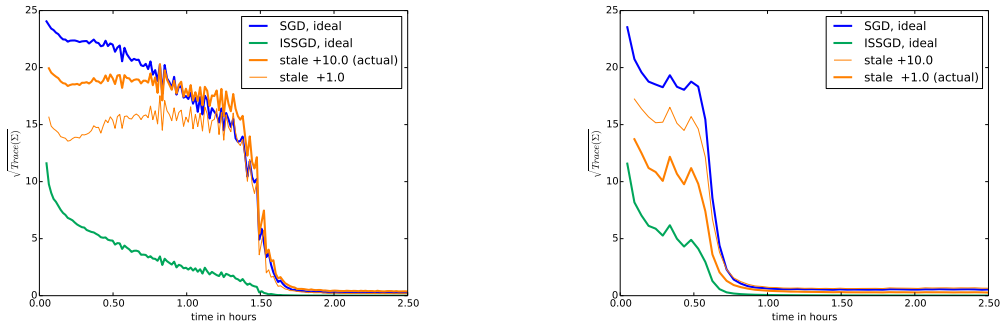
Here we look at the values of values of $\mathrm{Tr}(\Sigma(q))$ during the ISSGD training from the previous section (which led to figure 2 and figure 3).

We would like to compare the values of $\mathrm{Tr}(\Sigma(q))$ for $(q_{\text{IDEAL}}, q_{\text{STALE}}, q_{\text{UNIF}})$. Note that q_{UNIF} does not mean here that we trained with the regular SGD (that assigns the same probability to each training example). It means that, during ISSGD training, we can report the value of $\mathrm{Tr}(\Sigma(q))$ that we *would* get if we performed the next step with regular SGD. In figure 4, we refer to this as "SGD, ideal".

We compare it to "ISSGD, ideal", which corresponds to the best possible situation for our method, $\mathrm{Tr}(\Sigma(q_{\mathrm{IDEAL}}))$, which is not necessarily achieved in practice.

In section B.3 of the Appendix we describe how we add a constant to the probability weights in order to make the method more robust. We are trading away potential gains to make training more stable.

On both plots of figure 4 we show the "ideal" measurements that we would get with exact probability weights, and we compare with the "stale" measurements that we get with probability weights used in the actual experiments, which are all stale to varying degrees. On those stale curves, we show the effects of using the actual additive constant to the probability weights, and the effects of using an alternate one. Bear in mind that, in both cases, the validation loss reached its minimum in around 30 minutes, and these plots are shown for 2.5 hours. Also, these are the $\mathrm{Tr}(\Sigma(q))$ with respect to the gradient on the training set. One naturally expects that gradient to converge to 0.0 during the overfitting regime.



(a) learning rate 0.01, add smoothing $+10.0$

(b) learning rate 0.001, add smoothing $+1.0$

Figure 4: Square root of trace of covariance for different proposals q. We show here the median results aggregated over 50 runs of ISSGD. These plots come from the same hyperparameters used for figure 2. On the left plot, we use a higher learning rate in the hopes of making convergence faster. This required the probability weights to be smoothed by adding a constant (+10.0) to all the probability weights, and this washed away a part of the variance-reduction benefits of using ISSGD. On the right plot, we used a smaller learning rate, and we still got comparably fast convergence. However, because of the additive constant +1.0 used, these runs were closer to the ideal ISSGD setting. The point of these plots is to show that with ISSGD we can a smaller measurement of $\mathrm{Tr}(\Sigma(q))$. This happens clearly on the right plot, but not as convincingly on the left.

Note that in figure 4 we report the square root of those values in order to have it be on the same scale and the gradients themselves (this is analogous to reporting σ instead of σ^2).

6 Future work

One of the constraints that we are facing is that proposition 1 works with models with only fully-connected layers. This rules out all the convolutional neural networks, which are very popular and very useful.

One alternative would be to use an approximate formula for the individual gradient norms for convolutional layers. Either something naive (such as applying proposition 1 without proper justification), or possibly even ignoring the contributions from those layers. This would yield an importance sampling scheme that would be of lesser quality, but it would also be hard to evaluate how much we actually suffer for that.

We have avoided direct comparisons with ASGD in this paper because we are not currently in possession of a good production-quality ASGD implementation. We would certainly like to see how ASGD could be combined with ISSGD, whether this would create positive interactions or whether the two methods would impede each other.

Note that there are alternative ways to combine our method with ASGD, and they are not equally promising. Our recommendation would be to get rid of the master/workers distinction and have only workers (or "peers") along with a parameter server (or shared memory, or whatever synchronization method is used to aggregate the gradients and parameters). Whenever a gradient contribution is computed, the importance weights can be obtained at the same time. These can be shared in the same way that the gradients are shared, so that all the workers are able to use the importance weights to run ISSGD steps.

7 CONCLUSION

We have introduced a novel method for distributing neural network training by using multiple machines to search for the most informative examples to train on. This method led to significant improvements in training time on permutation invariant SVHN. Our results demonstrated that importance sampling reduced the variance of the gradient estimate, even in the distributed setting where the importance weights are not exact. One area for future work is extending this method to models that use parameter sharing (such as convnets and RNNs), either by finding a new formula for per-example gradient norms or by finding an approximation to the gradient norm that is easy to compute. Finally, much of the most successful work on data parallel distributed deep learning has used a variant of Asynchronous SGD. It would be useful to understand exactly how our method compares with Asynchronous SGD and to see if further improvement is gained by using both approaches simultaneously.

ACKNOWLEDGMENTS

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR.

The authors would like to acknowledge the stimulating discussions with Ian Goodfellow, Zack Lipton, Kari Torkkola, Daniel Abolafia, and Ethan Holly.

We would also like to thank the developers of Theano. [1]

REFERENCES

Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pp. 873–881, 2011.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *JMLR*, 3:1137–1155, 2003.

Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML'09*, 2009.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Guillaume Bouchard, Théo Trouillon, Julien Perez, and Adrien Gaidon. Accelerating stochastic gradient descent via online learning to sample. *CoRR*, abs/1506.09016, 2015. URL `http://arxiv.org/abs/1506.09016`.

A. Davis and I. Arel. Low-Rank Approximations for Conditional Feedforward Computation in Deep Neural Networks. *ArXiv e-prints*, December 2013.

[1] http://deeplearning.net/software/theano/

J. Dean, G.S Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M.A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS'2012*, 2012.

I. Goodfellow. Efficient Per-Example Gradient Computations. *ArXiv e-prints*, October 2015.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. Technical Report Arxiv report 1302.4389, Université de Montréal, February 2013. URL http://arxiv.org/abs/1302.4389.

Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 2719–2727, 2015.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. Deep Learning and Unsupervised Feature Learning Workshop, NIPS, 2011.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS'2011*, 2011.

Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26*, pp. 2310–2318. 2013. URL http://media.nips.cc/nipsbooks/nipspapers/paper_files/nips26/1109.pdf.

Surya T. Tokdar and Robert E. Kass. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):54–60, 2010. ISSN 1939-0068. doi: 10.1002/wics.56. URL http://dx.doi.org/10.1002/wics.56.

P. Zhao and T. Zhang. Stochastic Optimization with Importance Sampling. *ArXiv e-prints*, January 2014.

A IMPORTANCE SAMPLING IN THEORY

A.1 EXTENDING BEYOND A SINGLE DIMENSION

Theorem 1. *Let $\mathcal{X}$ be a random variable in $\mathbb{R}^{d_1}$ and $f(x)$ be any function from $\mathbb{R}^{d_1}$ to $\mathbb{R}^{d_2}$. Let $p(x)$ be the probability density function of $\mathcal{X}$, and let $q(x)$ be a valid proposal distribution for importance sampling with the goal of estimating*

$$\mathbb{E}_p\left[f(x)\right] = \int p(x)f(x)dx = \mathbb{E}_q\left[\frac{p(x)}{q(x)}f(x)\right]. \tag{10}$$

The context requires that $q(x) > 0$ whenever $p(x) > 0$. We know that the importance sampling estimator

$$\frac{p(x)}{q(x)}f(x) \quad \textit{with } x \sim q \tag{11}$$

has mean $\mu = \mathbb{E}_p\left[f(x)\right]$ so it is unbiased.

Let $\Sigma(q)$ be the covariance of that estimator, where we include q in the notation to be explicit about the fact that it depends on the choice of q.

Then the trace of $\Sigma(q)$ is minimized by the following optimal proposal q^* :

$$q^*(x) = \frac{1}{Z}p(x)\left\|f(x)\right\|_2 \quad \textit{where } Z = \int p(x)\left\|f(x)\right\|_2 dx \tag{12}$$

which achieves the optimal value

$$\mathrm{Tr}(\Sigma(q^*)) = \left(\mathbb{E}_p\left[\left\|f(x)\right\|_2\right]\right)^2 - \left\|\mu\right\|_2^2.$$

Proof. This proof is almost exactly the same as the well-known result in one dimension involving Jensen's inequality. Everything follows from the decision to minimize $Tr\left(\Sigma\right)$ and the choice of q^*. Nevertheless, we include it here so the reader can get a feeling for where q^* comes into play.

When sampling from $q(x)$ instead of $p(x)$, we are looking at how the unbiased estimator

$$\mathbb{E}_{q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$$

which has mean μ and covariance $\Sigma(q)$. We make use of the fact that the trace is a linear function, and that $\mathrm{Tr}(\mu\mu^T) = \|\mu\|_2^2$. The trace of the covariance is given by

$$\begin{aligned}
\mathrm{Tr}\left(\Sigma(q)\right) &= \mathrm{Tr}\left(\mathbb{E}_{q(x)}\left[\left(\frac{p(x)}{q(x)}f(x)-\mu\right)\left(\frac{p(x)}{q(x)}f(x)-\mu\right)^T\right]\right) \\
&= \mathrm{Tr}\left(\mathbb{E}_{q(x)}\left[\left(\frac{p(x)}{q(x)}f(x)\right)\left(\frac{p(x)}{q(x)}f(x)\right)^T\right]-\mu\mu^T\right) \\
&= \mathbb{E}_{q(x)}\left[Tr\left(\left(\frac{p(x)}{q(x)}f(x)\right)\left(\frac{p(x)}{q(x)}f(x)\right)^T\right)\right]-\|\mu\|_2^2 \\
&= \mathbb{E}_{q(x)}\left[\left\|\frac{p(x)}{q(x)}f(x)\right\|_2^2\right]-\|\mu\|_2^2 . \qquad (13)
\end{aligned}$$

There is nothing to do about the $\|\mu\|_2^2$ term since it does not depend on the proposal $q(x)$. Using Jensen's inequality, we get that

$$\mathbb{E}_{q(x)}\left[\left\|\frac{p(x)}{q(x)}f(x)\right\|_2^2\right] \geq \mathbb{E}_{q(x)}\left[\left\|\frac{p(x)}{q(x)}f(x)\right\|_2\right]^2 = \left(\int q(x)\frac{p(x)}{q(x)}\left\|f(x)\right\|_2 dx\right)^2 = \left(\mathbb{E}_{p(x)}\left[\|f(x)\|_2\right]\right)^2 .$$

This means that, for any proposal $q(x)$, we cannot do better than $\left(\mathbb{E}_{p(x)}\left[\|f(x)\|_2\right]\right)^2 - \|\mu\|_2^2$. All that is left is to take the proposal q^* in the statement of the theorem, to evaluate $Tr(\Sigma(q^*))$ and to show that it matches that lower bound.

We have that

$$\begin{aligned}
\mathrm{Tr}(\Sigma(q^*)) &= \mathbb{E}_{q^*(x)}\left[\left\|\frac{p(x)}{q^*(x)}f(x)\right\|_2^2\right]-\|\mu\|_2^2 \\
&= \int q^*(x)\left(\frac{p(x)}{q^*(x)}\right)^2\|f(x)\|_2^2\,dx-\|\mu\|_2^2 \\
&= \int \frac{p(x)^2}{q^*(x)}\|f(x)\|_2^2\,dx-\|\mu\|_2^2 \qquad (14) \\
&= \int \frac{p(x)^2 Z}{p(x)\left\|f(x)\right\|_2}\|f(x)\|_2^2\,dx-\|\mu\|_2^2 \;\text{ where }\; Z=\int p(x)\left\|f(x)\right\|_2 dx \\
&= \left(\mathbb{E}_{p(x)}\left[\|f(x)\|_2\right]\right)^2-\|\mu\|_2^2
\end{aligned}$$

which is the minimal value achievable, so q^* is indeed the best proposal in terms of minimizing $Tr(\Sigma(q))$. □

Note also that the single-dimension equivalent, mentioned in section 3.1, is a direct corollary of this proposition, because the Euclidean norm turns into the absolute value.

Corollary 1. *Using the context of importance sampling as described in theorem 1, let $q(x)$ be a proposal distribution that is proportional to $p(x)h(x)$ for some function $h : \mathcal{X} \to \mathbb{R}^+$. As always, we require that $h(x) > 0$ whenever $f(x) > 0$.*

Then we have that the trace of the covariance of the importance sampling estimator is given by

$$\mathrm{Tr}(\Sigma(q)) = \left(\int p(x)h(x)dx\right)\left(\int p(x)\frac{\|f(x)\|_2^2}{h(x)}dx\right) - \|\mu\|_2^2\,,$$

where $\mu = \mathbb{E}_{p(x)}[f(x)]$. Moreover, if $p(x)$ is not known directly, but we have access to a dataset $\mathcal{D} = \{x_n\}_{n=1}^{\infty}$ of samples drawn from $p(x)$, then we can still define $q(x) \propto p(x)h(x)$ by associating the probability weight $\tilde{\omega}_n = h(x_n)$ to every $x_n \in \mathcal{D}$.

To sample from $q(x)$ we just normalize the probability weights

$$\omega_n = \frac{\tilde{\omega}_n}{\sum_{n=1}^{N}\tilde{\omega}_n}$$

and we sample from a multinomial distribution with argument $(\omega_1, \ldots, \omega_N)$ to pick the corresponding element in $\mathcal{D}$.

In that case, we have that

$$\begin{aligned}\mathrm{Tr}(\Sigma(q)) &= \left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n\right)\left(\frac{1}{N}\sum_{n=1}^{N}\frac{\|f(x_n)\|_2^2}{\tilde{\omega}_n}\right) - \|\mu\|_2^2 \\ &= \left(\frac{1}{N}\sum_{n=1}^{N}\omega_n\right)\left(\frac{1}{N}\sum_{n=1}^{N}\frac{\|f(x_n)\|_2^2}{\omega_n}\right) - \|\mu\|_2^2\,.\end{aligned}$$

Proof. We start from equation (13), which applies to a general proposal q. In fact, we make it to equation (14) still without making assumptions on q. At that point we can look at the normalizing constant of q, which is equal to

$$Z = \int p(x)h(x)dx = \frac{1}{N}\sum_{n=1}^{N}h(x_n) = \frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n.$$

Then we have that

$$\mathrm{Tr}(\Sigma(q)) = \int \frac{p(x)^2 Z}{p(x)h(x)}\|f(x)\|_2^2\,dx - \|\mu\|_2^2 \quad \text{where} \quad Z = \frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n \tag{15}$$

$$= (Z)\left(\int p(x)\frac{\|f(x)\|_2^2}{h(x)}dx\right) \tag{16}$$

$$= \left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\omega}_n\right)\left(\frac{1}{N}\sum_{n=1}^{N}\frac{\|f(x_n)\|_2^2}{\tilde{\omega}_n}\right) \tag{17}$$

The equivalent formula for ω_n instead of $\tilde{\omega}_n$ follows from dividing the left expression by $\sum_{n=1}^{N}\tilde{\omega}_n$ and multiplying the expression on the right by that constant. □

A.2 Dealing with minibatches

Proposition 1. *Consider a multi-layer perceptron (MLP) applied to minibatches of size N, and with loss $\mathcal{L} = \mathcal{L}_1 + \ldots + \mathcal{L}_N$, where $\mathcal{L}_n$ represents the loss contribution from element n of the minibatch.*

Let (W, b) be the weights and biases at any particular fully-connected layer so that $XW + b = Y$, where X are the inputs to that layer and Y are the outputs.

The gradients with respect to the parameters are given by

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial W} &= \frac{\partial\mathcal{L}_1}{\partial W} + \ldots + \frac{\partial\mathcal{L}_N}{\partial W} \\ \frac{\partial\mathcal{L}}{\partial b} &= \frac{\partial\mathcal{L}_1}{\partial b} + \ldots + \frac{\partial\mathcal{L}_N}{\partial b}\end{aligned}$$

where the values $\left(\frac{\partial \mathcal{L}_n}{\partial W}, \frac{\partial \mathcal{L}_n}{\partial b}\right)$ *refer to the particular contributions coming from element* n *of the minibatch. Then we have that*

$$\begin{aligned}\left\|\frac{\partial \mathcal{L}_n}{\partial W}\right\|_F^2 &= \|X[n,:]\|_2^2 \cdot \left\|\frac{\partial \mathcal{L}}{\partial Y}[n,:]\right\|_2^2 \\ \left\|\frac{\partial \mathcal{L}_n}{\partial W}\right\|_2^2 &= \left\|\frac{\partial \mathcal{L}}{\partial Y}[n,:]\right\|_2^2,\end{aligned}$$

where the notation $X[n,:]$ *refers to row* n *of* X*, and similarly for* $\frac{\partial \mathcal{L}}{\partial Y}[n,:]$.

That is, we have a compact formula for the Euclidean norms of the gradients of the parameters, evaluated for each N *elements of the minibatch independently.*

Proof. The usual backpropagation rules give us that

$$\frac{\partial \mathcal{L}}{\partial W} = X^T \frac{\partial \mathcal{L}}{\partial Y} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b} = \sum_{n=1}^{N} \frac{\partial \mathcal{L}}{\partial Y}[n,:].$$

All the backpropagation rules can be inferred by analyzing the shapes of the quantities involved and noticing that only one answer can make sense. If we focus on $\mathcal{L}_n$ for some $n \in \{1, \dots, N\}$, then we can see that

$$\frac{\partial \mathcal{L}_n}{\partial W} = X[n,:]^T \frac{\partial \mathcal{L}}{\partial Y}[n,:] \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial Y}[n,:]. \tag{18}$$

Note here that $X[n,:]^T \frac{\partial \mathcal{L}}{\partial Y}[n,:]$ is the outer product of two vectors, which yields a rank-1 matrix of the proper shape for $\frac{\partial \mathcal{L}_n}{\partial W}$. Similarly, we have that $X[n,:]X[n,:]^T = \|X[n,:]\|_2^2$ is a 1x1 matrix, which can be treated as a real number in all situations.

The conclusion for $\left\|\frac{\partial \mathcal{L}_n}{\partial b}\right\|_2^2$ follows automatically from taking the norm of the corresponding expression in equation (18). Some more work is required for $\left\|\frac{\partial \mathcal{L}_n}{\partial W}\right\|_2^2$. We will make use of the three following properties of matrix traces.

- $\|A\|_{\mathrm{F}}^2 = \mathrm{Tr}(AA^T)$
- $\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA) = \mathrm{Tr}(CAB)$
- $\mathrm{Tr}(kA) = k\,\mathrm{Tr}(A) \qquad \text{for } k \in \mathbb{R}$

We have that

$$\begin{aligned}\left\|\frac{\partial \mathcal{L}_n}{\partial W}\right\|_{\mathrm{F}}^2 &= \mathrm{Tr}\left(X[n,:]^T \frac{\partial \mathcal{L}_n}{\partial Y}[n,:] \left(X[n,:]^T \frac{\partial \mathcal{L}_n}{\partial Y}[n,:]\right)^T\right) \\ &= \mathrm{Tr}\left(X[n,:]^T \frac{\partial \mathcal{L}_n}{\partial Y}[n,:] \frac{\partial \mathcal{L}_n}{\partial Y}[n,:]^T X[n,:]\right) \\ &= \mathrm{Tr}\left(X[n,:]X[n,:]^T \frac{\partial \mathcal{L}_n}{\partial Y}[n,:] \frac{\partial \mathcal{L}_n}{\partial Y}[n,:]^T\right) \\ &= \|X[n,:]\|_2^2 \cdot \left\|\frac{\partial \mathcal{L}}{\partial Y}\right\|_2^2.\end{aligned}$$

□

One might wonder why we are interested in computing the Frobenius norm of the matrix $\frac{\partial \mathcal{L}_n}{\partial W}$ instead of its L2-norm. The reason is that when running SGD we serialize all the parameters as a flat vector, and it is the L2-norm of that vector that we want to compute. We flatten the matrices, and the following equality reveals why this means that we want the Frobenius norms of our matrix-shaped parameters :

$$\|A\|_{\mathrm{F}}^2 = \|A.\text{flatten}(\,)\|_2^2\,.$$

B DISTRIBUTED IMPLEMENTATION OF ISSGD

B.1 USING ONLY A SUBSET OF THE WEIGHTS

Of the many hyperparameters that are available to adjust the behavior of the master and workers, we have the possibility the use a staleness threshold that filters out all the x_n candidates whose corresponding $\tilde{\omega}_n$ have not been updated sufficiently recently.

For many of the experiments that we ran on SVHN, where we used a training set of roughly 570k samples, with 3 workers, a staleness threshold of 4 seconds leads to 15% of the probability weights to be kept. The other 85% are filtered out. This filtering is rather fair in that it does not favor any sample a priori. Every $\tilde{\omega}_n$ stands equal chances of having been recomputed last.

We have tried training without that staleness threshold and it is hard to see a difference. Adding more workers naturally lowers the average staleness of probability weights, because more workers can update them more frequently. If it were not the cost of communicating the model parameters, we could argue that a sufficiently large number of workers would simulate an oracle perfectly.

B.2 APPROXIMATING $\|g^{\text{TRUE}}\|_2^2$

To report values of $\text{Tr}(\Sigma(q))$, we need to be able to compute the actual expected gradient over the whole training set. We refer to that quantity as the *true gradient* $g^{\text{TRUE}} = \frac{1}{N}\sum g(x_n)$, but we never really compute it due to practical reasons. This would entail reporting the gradient for each chunk of the training set and aggregating everything. This is precisely the kind of thing that we avoid with ISSGD.

Instead we compute the gradients of the parameter for each minibatch, and we report the L2-norm of those. We then average those values. This produces an upper-bound to the actual value of $\|g^{\text{TRUE}}\|_2$.

One important thing to note is that the equations (7), (8) and (9) each have the $\|g^{\text{TRUE}}\|_2^2$ term, so any approximation of that term, provided that it is the same for all three, will not alter the respective order of $\text{Tr}(\Sigma(q_{\text{IDEAL}}))$, $\text{Tr}(\Sigma(q_{\text{STALE}}))$, $\text{Tr}(\Sigma(q_{\text{UNIF}}))$

Moreover, when are getting close to the end of the training, we should have that $\|g^{\text{true}}\|_2$ is getting close to zero. That is, the gradient is zero when we are close to an optimum. This does not meant that the individual gradients are all zero, however. But when our upper-bound on $\|g^{\text{TRUE}}\|_2$ is getting close to being insignificant, then we can tell for sure that our values computed for the three $\text{Tr}(\Sigma(q))$ are very close to their exact values.

B.3 SMOOTHING PROBABILITY WEIGHTS

Sometimes we can end up with probability weights that fluctuate too rapidly. This can lead to some problems in a situation where one training sample x_n is assigned a small probability weight ϵ, when compared to the other probability weights. Things normally balance out because x_n has a probability proportional to ϵ, and when it gets selected its gradient contribution $g(x_n)$ gets scaled by $1/\epsilon$. The resulting contribution is a gradient of norm ≈ 1.

However, when that gradient changes quickly (and probability weight along with it), it is possible to get into a situation where the gradient computed on the master is now much larger (due to the parameters having changed), and it still gets divided by ϵ when selected. This does not affect the bias, but it affects the stability of the method in the long term. When running for an indefinitely long period, it is dangerous to having a time bomb in the algorithm that has a small probability of ruining everything.

To counter this effect, or just to make the training a bit smoother, we decided to add a smoothing constant to all the probability weights $\tilde{\omega}_n$ before normalizing them. The larger the constant, the more this will make ISSGD resemble regular SGD. In the limit case where this constant is infinite, this turns exactly into regular SGD.

We had some ideas for using an adaptive method to compute this smoothing constant, but this was not explored due to the large number of other hyperparameters to study. One suggestion was to look at the entropy of the distribution of the $\{\omega_n\}_{n=0}^{\infty}$ that determine which training sample are going to

be used. With a smoothing constant sufficiently large, we can bring this entropy down to any target level (or down to regular SGD when that constant is infinite).