An improved mini-batching technique: Sample-and-Learn

Andreas Antoniades
Department of Computer Science
University of Surrey, U.K.
Email: a.antoniades@surrey.ac.uk

Clive Cheong Took
Department of Computer Science
University of Surrey, U.K.
Email: c.cheongtook@surrey.ac.uk

Yaochu Jin
Department of Computer Science
University of Surrey, U.K.
Email: yaochu.jin@surrey.ac.uk

***Abstract*—Artificial neural networks suffer from prolonged training times, this is intensified when the volume of data is big. Mini-batching has become the standard in training neural networks. By reducing the number of data used to train each iteration the training time greatly shortens; even in a big data environment. A number of techniques such as parallel computation have rendered mini-batching a necessity for complex neural network models. Due to simplicity however, mini-batching methods have a number of inherent disadvantages that can affect the accuracy and convergence of a model. In this work, we focus on the ordering of samples presented to a neural network and propose a random sampling approach for generating mini-batches in linear time. Experimental results show that networks using our proposed Sample-and-Learn approach converge in fewer iterations while providing comparable or better accuracy.**

1. Introduction

Designing scalable neural networks for big data is a challenging task. Batch training of networks, which requires an entire pass over the data in each iteration, is no longer viable and researchers start to look at alternative training methods [1]. Online training methods use a single sample from the training set X for the learning and update processes at each iteration. The selection process can be random or from a distribution such as Gibbs sampling [2]. Online approaches tend to be faster than batch learning and particularly applicable in environments where the data becomes available through time. Another advantage of the online learning method is that it can be used to track changes in the data through time.

Mini-batching presents a compromise solution between batch and online learning approaches, that is both simple and effective. A subset S is created from the training set X of a predetermined size s; often referred to as batch size. For a mini-batching of 100 the first iteration uses $S_1 = \{x_1, x_2, .., x_{100}\}$ to train, while the second iteration uses $S_2 = \{x_{101}, x_{102}, .., x_{200}\}$ and so on. This method reduces the time taken by each iteration and is ideal for big data environments and complex models. Due to its simplicity, mini-batching has been widely accepted by the machine learning community and a number of improvements have been proposed the last few years.

A parallel implementation of mini-batching is conceptually straightforward, yet it gives rise to the problem of high communication costs. In [3], a technique based on approximate optimization is proposed to allow for an efficient distributed implementation of mini-batching with reduced communications costs.

Another issue is that a neural network using mini-batching is exposed to data from different distributions at each iteration due to the non-stationarity of the data. In this case, networks need to continuously adapt to the new data. This is often referred to as *covariance shift* [4]. An effective approach to alleviate this problem is to normalise the data of each batch independently [5].

Despite its success in improving, mini-batching suffers from one major drawback; the data is presented to the network in the same order at each iteration, increasing the chance for overfitting. Moreover, the grouping of data points in mini-batches is generally fixed; i.e. data points X_1 and X_2 have a high probability of always appearing in the same mini-batch. In [6], shuffled data was used for training and were found to be beneficial for stochastic gradient descent, which is the algorithm the majority of neural networks use.

Random sampling can be used to collect a few samples of the data set, to form a reservoir of the data for training. First, our proposed scheme circumvents the issue of overfitting due to the shuffling of data, by performing random sampling. Second, reservoir sampling involves collecting samples over the entire dataset, better reflecting the non-stationarity of data, instead of a localised batch generated by mini-batching.

A Sample-and-Learn scheme is proposed to leverage both of these benefits for the training of a neural network. As an application of big data, we consider two large classification problems, including supersymmetric and Higgs boson particle detection. Both datasets were considered in [7]. The authors empirically illustrated the advantages of deep learning for the two datasets, whereas our objective is to investigate models that use mini-batching and the proposed Sample-and-Learn scheme.

2. Reservoir Sampling against volume and velocity data

The aim of a reservoir sampling algorithm is to collect a set (or reservoir) of random samples of size s from a dataset, whose size N is very large or not known. This type of algorithm has found applications in databases [8], data streaming [9], and even in the well-known expectation maximisation learning algorithm [10], but has not been considered in the training of neural networks for big data. The processing of data streams implicitly entails the problem of volume and velocity of the data. One pass reservoir sampling addresses the problem of volume of the data by collecting only a subset of samples from the data set, whereas the online operation and the light computational complexity of reservoir sampling makes it well suited for coping with the velocity of the data.

There are several reservoir sampling algorithms. The most prominent one is known as Algorithm R [11], which selects s samples randomly (from a uniform distribution) from the data set (of size N), given that $N > s$. It has a computational complexity of O(N) and an improved implementation can be found in [12].

In the context of our proposed Sample-and-Learn scheme for the training of neural networks, Algorithm R has been considered. Being a sampling algorithm without replacement, Algorithm R has a linear computational complexity, scaling better with big data unlike more expensive, two-pass sampling algorithms; making it a viable choice. The pseudo code for Algorithm R can be found in Alg.1. Big data often implies that no prior knowledge on the data set is available; hence it is more appropriate to select a subset of the data from a uniform distribution rather than from a skewed distribution.

Algorithm 1 Algorithm R

```
1. Define reservoir size s
2. Repeat
3. Read the k^th sample
4. If(k < s)
6.    Add k^th sample to reservoir
7. Else
8.   Generate a random number r between 1
     and k
9.  If(r < s)
10.     Replace the r^th sample in the
          reservoir with k^th sample
12. Until k=N+1
```

3. The proposed Sample-and-Learn Scheme

A benefit of big data is the availability of data for learning, which minimises the risk of overfitting; this cannot be exploited when a small data set is used. The underlying idea behind our proposed method, Sample-and-Learn, is to provide exposure to a large spectrum of the data, but perform the learning of the neural network on data at much smaller scale. The proposed learning process can be found in Fig. 1.

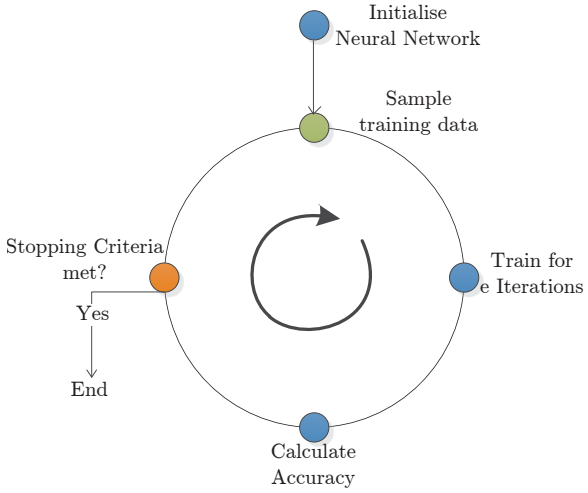


Figure 1: Diagram of the proposed learning process.

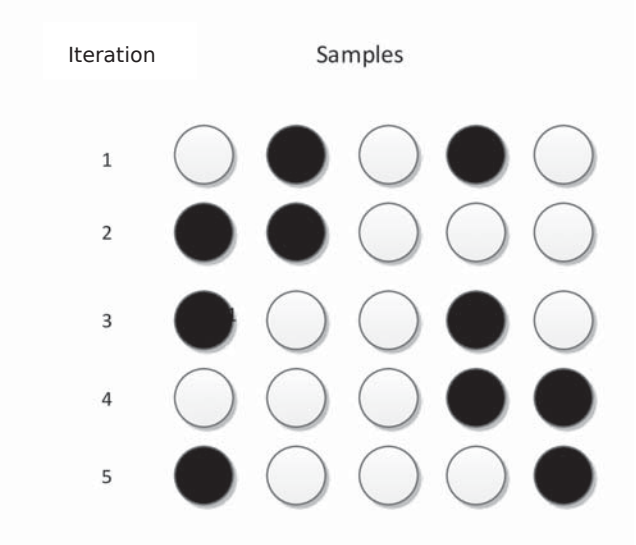


Figure 2: Reservoir Sampling over a number of iterations. Shaded circles represent the samples chosen by the reservoir at each iteration. This is a scenario where five iterations were allowed, the reservoir size is two and the number of samples is five.

For the purpose of illustration, consider a data set comprising of a total of five samples $\{s1, s2, s3, s4, s5\}$ and a reservoir of size two. The aim is to allow most of the samples from the data set into the reservoir at least once throughout the many iterations. If we allow five iterations for learning, then a possible set of reservoirs generated for the five iterations is shown in Fig. 2. The best case scenario would be for all samples, or at least the majority, to pass through the reservoir, which is the case in Fig. 2. The addition of reservoir sampling for each iteration increases

the computational complexity of learning by O(N).

Remark #1 The example in Fig. 2 indicates that the input to the neural network may be sampled at different periods. In other words, the neural network may learn from the input of $\{s2, s4\}$. In fact, Algorithm R can provide a *shuffled* input of $\{s4, s2\}$ to the neural network.

3.1. Statistical Analysis of Sample-and-Learn Scheme

For rigour, we provide a statistical analysis of our proposed Sample-and-Learn method. In particular, we illustrate how the data size and the number of iterations of the training neural network affect its accuracy. It is shown that the probability of the k^{th} sample being in the reservoir of Algorithm R in one-pass through the entire data set is computed as s/N. Recall that s denotes the size of the reservoir and N denotes the size of the entire data set. Therefore, the probability of **not** getting k^{th} sample after E iterations can be expressed as:

$$P_{not}(k) = \left[1 - \frac{s}{N}\right]^E \tag{1}$$

Probability of getting k^{th} sample at least once after E iterations can be computed as:

$$P_r(k) = 1 - \left[1 - \frac{s}{N}\right]^E \tag{2}$$

Remark #2: Equation (2) shows that increasing either s or E results in an increase in the probability of obtaining the k^{th} sample $P_r(k)$. The greater the probability of getting the k^{th} sample $P_r(k)$ to train the neural network, the better the accuracy of the neural network.

However, it is not clear from Equation (2) whether the size of training data s or the number of iterations E has a greater impact on $P_r(k)$. To this end, the rate of change of $P_r(k)$ with respect to both s and E is derived as follows.

Let

$$X = 1 - \frac{s}{N} \tag{3}$$

From (2), its derivatives can be obtained as

$$\frac{\partial P_r(k)}{\partial s} = \frac{E}{N}\left[X\right]^{E-1} \tag{4}$$

$$\frac{\partial P_r(k)}{\partial E} = -X^E \ln(X) \tag{5}$$

Remark #3: Given that $N \gg s$, Equations (4) and (5) indicate that a change in the number of iterations ΔE has a greater impact on the probability $P_r(k)$ in Equation (2) than a change in data size Δs.

For clarity, it is also instructive to determine the relationship between Δs and ΔE, which can be expressed as

$$\frac{\partial P_r(k)}{\partial E} = -\frac{N}{E}\frac{\partial P_r(k)}{\partial s} X \ln(X) \tag{6}$$

Notice that X in Equation (3) is a fraction, hence ln(X) is a negative number. By letting $\ln(X) = -\lambda$, Equation (6) becomes

$$\frac{\partial P_r(k)}{\partial E} = \lambda X \frac{N}{E}\frac{\partial P_r(k)}{\partial s} \tag{7}$$

Using the approximation $\partial y/\partial x \approx \Delta y/\Delta x$, where Δ is the change in variable x or y and taking the inverse of (7), the relationship between a change in number of iterations ΔE and a change in data size Δs can be approximated as

$$\Delta E \approx \frac{E}{\lambda X N}\Delta s \tag{8}$$

Remark #4: To achieve the same level of accuracy, Equation (8) implies that training a neural network with a smaller data size does not cause a significant increase in the number of iterations. In other words, a small change in data size corresponds to a much smaller change in the number of iterations, in order to have the same effect on the probability $P_r(k)$ in Equation (2).

3.2. Comparison to mini-batching

In this section we provide a comparison between our proposed method, Sample-and-Learn, and mini-batching. Recall that our objective is to provide a learning scheme that allows a randomly sampled set of datapoints to be presented to the network. We compare the two methods in terms of the probability of each datapoint to be selected and the computational cost.

3.2.1. Probability of datapoint selection.

We have already calculated the propability of a datapoint to be selected at least once through a number of iterations to be:

$$P_r(k) = 1 - \left[1 - \frac{s}{N}\right]^E \tag{9}$$

For mini-batching, since the selection is deterministic, the probability of selecting a datapoint is proportional to the batch size s and the number of iterations E, and inversely proportional to the size of the data set N.

$$P_b(k) = \begin{cases} 1 & \text{if} \quad sE \geq N \\ \frac{sE}{N} & \text{if} \quad sE < N \end{cases} \tag{10}$$

Assuming that we are dealing with big data, where N is extremely large and using all the datapoints is not viable, we define $P_b(k) = \frac{sE}{N}$. For small N, $P_b(k) > P_r(k)$, however as N increases, $P_b(k) \approx P_r(k)$. Fig. 3 depicts how $\delta(P(k)) = P_b(k) - P_r(k)$ approaches zero as N increases. Given a large enough datasets, we estimate that $P_b(k) \equiv P_r(k)$. As a result we can conclude that Sample-and-Learn and mini-batching are equal in terms of the probability of presenting a specific sample to the network for big datasets.

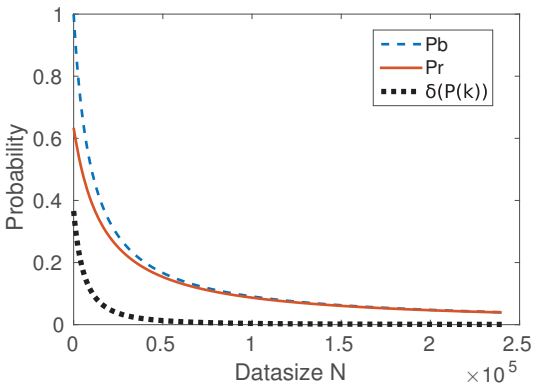


Figure 3: The probability of not selecting a specific sample for different data set sizes N. For this simulation $E = 100$ and $s = 100$.

3.2.2. Computational Complexity.

The total computational complexity $\mathbb{C}$ of a network with a single hidden layer and a logistic regressor can be approximated as:

$$\mathbb{C} = E(N(dc + h)) \tag{11}$$

where N denotes the number of datapoints, d is the dimensionality of the data, c is the number of weights of the hidden layer, h is the number of hidden neurons and E is the number of iterations run. which corresponds to the big Oh notation of:

$$O(ENdc) \tag{12}$$

We expect that standard mini-batching is faster than our Sample-and-Learn scheme. This is because every iteration is followed by the reservoir sampling process with computational complexity $O(N)$, see Fig. 1. To improve the computational complexity of Sample-and-Learn we have designed two improvements to reservoir sampling, multi reservoir sampling and asynchronous sampling.

The first improvement, takes into consideration the volume of N for big datasets. To address the sheer size of big data (e.g. 11×10^6 samples as considered in our simulations), a number of reservoirs L are generated for each pass over the data. That would enable us to accelarate the selection process of samples to be considered for the training of our models. Note that this approach is memory intensive since more reservoirs will be stored in memory. Yet, given enough memory we can generate hundreds of reservoirs for training in a single pass over the data. Recall that algorithm R is a sampling algorithm without replacement, therefore a particular sample can only exist once in each reservoir. However, the same sample can exist in two different reservoirs.

The second improvement comes in the form of asynchronous sampling. Meaning that, an iteration of training E_n does not need to be completed for the sampling process of the reservoir S_{n+1} for the next iteration to be generated. This insight has led us to implement reservoir sampling on a standalone thread that asynchronously creates reservoirs of the data and stores them in a shared queue. When the neural network finishes an iteration, it simply queries the shared queue for the next available reservoir.

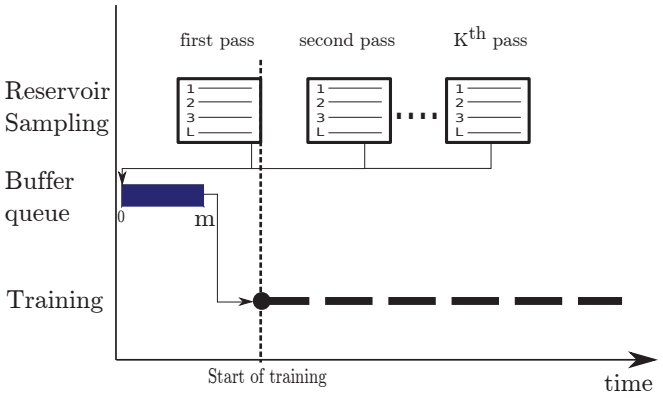


Figure 4: Timeline of the Sample-and-Learn scheme

In this work, we consider s to be both the batch and the reservoir size since both of these serve identical roles, i.e. how much data is used per iteration for training. Integrating multi-reservoir and asynchronous sampling reduces the computational cost of training with Sample-and-Learn to approximately $\mathbb{C}_{SandL} \approx \mathbb{C}_b$. The pseudocode for our Sample-and-Learn algorithm with the improved algorithm R can be found in Alg. 2. A timeline of the suggested scheme is illustrated in Fig. 4. The first sampling pass over the data needs to be completed before training can commence. After each pass the L generated reservoirs are put into a buffer queue that supplies the learning algorithm with data. Sampling continues asynchronously with training constantly providing new reservoirs until the buffer limit m is reached.

Algorithm 2 Sample-and-Learn

```
1. Define Global queue q, reservoir size s
2. Define error E, previous error E_p
3. Launch new thread(Algorithm3)
   E = ∞
4. Repeat
     Ep = E
5.   Read the next reservoir
     S_n = q.dequeue()
6.   Train for one iteration
7.   Calculate validation error E
8. Until E>E_p to avoid overfitting
9. Calculate test error
```

Algorithm 3 Multi-reservoir Algorithm R

```
1. Define number of reservoirs per pass
   L and queue of size m × s
2.  Repeat
3.    if (q.size() < m)
4.    Repeat
5.      Read the k^th sample
6.      l = 0
7.      Repeat for reservoir l
8.        If(k < s)
9.          Add k^th sample to reservoir l
10.       Else
11.         Generate a random number r
    between 1 and k
12.           If(r < s)
13.             Replace the r^th sample in
    reservoir l with k^th sample
14.       l=l+1
15.     Until l=L
16.   Until k=N+1
18.   q.add(L reservoirs)
17. Until True
```

4. Simulations

To illustrate the efficacy of our Sample-and-Learn scheme, a number of simulations using real world data were considered. The datasets used include the the supersymmetric particles dataset (Susy) and the Higgs boson dataset [7]. The Theano [13] framework was used for the realisation of the neural networks. Simulations were run on an Nvidia GTX980 with an I7 3990K clocked at 4.2GHz and 28GB of ram. We used early stopping, see Algorithm (2), to moderate the training of our neural networks and used a 70/15/15 split for the training/validation/testing sets respectively. For each iteration of training, the neural network used s samples to train and for every 50 iterations the network was validated against an independent validation set. A number of simulations, for $s \in [10, 50, 100, 300, 500]$, were carried out to demonstrate the impact of volume of the training data on the computational cost. For each simulation set, the results were averaged over 10 trial runs to alleviate the probabilistic behaviour of Sample-and-Learn as well as to allow for different weight initialisations.

4.1. Susy dataset

The Susy dataset was created using Monte Carlo simulations for discriminating a process where new supersymmetric particles are produced and a background process [7]. The first 8 features are kinetic properties measured by particle detectors in a particle accelerator. The last 10 are functions of the first 8 features, handcrafted by physicists to help discriminate between the two classes. The complete dataset was used with all features, a total of 5×10^6 samples where 46% are positive. The machine learning problem is to classify whether the kinetic properties and high level features belong to a process that generates supersymmetric particles or a background process.

TABLE 1: Training parameters for all the considered methods

Parameter	Susy	Higgs
Hidden layers	3	4
Hidden layer order	$[200, 50, 10]$	$[200, 50, 20, 5]$
Learning rate	0.001	0.03
Validation frequency	50	50

4.2. Higgs dataset

The Higgs dataset was produced using Monte Carlo simulations for discriminating a signal process that theoretically produces Higgs bosons from a background process [7]. The dataset is divided in two parts, the first contains kinetic properties measured by particle detectors in a particle accelerator and makes up the first 21 features. The second part contains 7 handcrafted high level features created by physicists. The complete dataset was used with all features, a total of 11×10^6 samples where 53% are positive. The problem is to classify whether the kinetic properties and high level features belong to a process that theoretically creates Higgs boson particles or a background process.

4.3. Network parameters

Two multi layer perceptron networks were utilised for the comparison between Sample-and-Learn and mini-batching. Initial simulations were run to optimise the initial parameters of the models. Due to the size of the datasets, the initial optimisation of parameters was not exhaustive. The networks used for both datasets and their parameters can be found in Table 1. For both networks, the cross entropy cost function was considered:

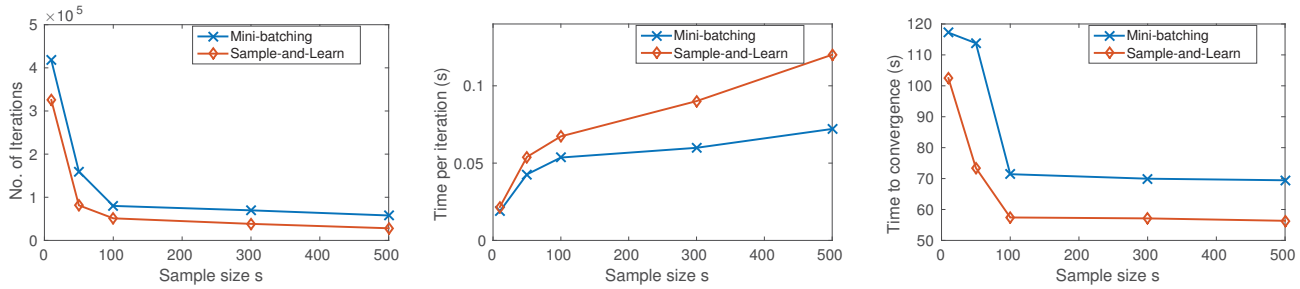
$$J = -y \log(\phi(z)) + (1 - y) \log(1 - \phi(z)) \qquad (13)$$

where y is the desired class, ϕ the activation function and z the output. The hyperbolic tangent function was used as the activation function for all networks:

$$\phi(\theta) = \tanh(\theta) = \frac{\sinh(\theta)}{\cosh(\theta)} = \frac{\exp(\theta) - \exp(-\theta)}{\exp(\theta) + \exp(-\theta)} \qquad (14)$$

5. Discussion

The empirical results confirmed our hypothesis, i.e. neural networks learn better from randomly sampled and shuffled data. This is due to the fact that the randomisation process that alleviates overfitting in training. For the Susy dataset, we observe similar accuracy between the two competing algorithms. However, the iterations and total time required for convergence for the network trained with



(a) Iterations until convergence (b) Time taken per iteration (c) Total time to converge

Figure 5: Results for the Susy dataset for different s.

TABLE 2: Results for Susy dataset

Scheme	s	Iterations	Time/Iteration(s)	Total Time(m)	Accuracy
Sample-and-Learn	10	278150	0.0216	102.48	80.23(0.97)
	50	81350	0.0540	73.45	81.98(0.75)
	100	51200	0.0673	57.42	82.19(0.32)
	300	38350	0.0901	57.12	81.82(0.42)
	500	28150	0.1201	56.34	81.03(0.61)
Mini-Batching	10	364070	0.0188	117.21	80.15(1.57)
	50	159950	0.0427	113.79	80.23(0.43)
	100	79950	0.0536	71.42	80.14(0.12)
	300	69700	0.0599	69.96	80.01(0.21)
	500	57800	0.0721	69.45	79.88(0.38)

TABLE 3: Results for Higgs dataset

Scheme	s	Iterations	Time/Iteration(s)	Total Time(m)	Accuracy
Sample-and-Learn	10	99550	0.0876	144.34	72.61(1.03)
	50	62100	0.1347	138.69	74.13(0.74)
	100	36050	0.2104	126.37	74.37(0.78)
	300	27400	0.2931	134.75	73.80(0.71)
	500	23750	0.3685	131.13	71.02(0.69)
Mini-Batching	10	147100	0.0681	166.13	70.10(3.05)
	50	83300	0.1022	139.61	72.98(0.68)
	100	52750	0.1720	151.21	73.21(0.71)
	300	45150	0.2122	159.68	70.78(0.46)
	500	35150	0.2658	165.24	69.79(0.2)

Sample-and-Learn were significantly less.

For the Higgs dataset, similar behaviour is observed with regards to training time and iterations to convergence. Additionally, the network trained with Sample-and-Learn on the Higgs data had a slight accuracy advantage over mini-batching. Results for the Susy and Higgs datasets can be found in Table 2 and Table 3 respectively. The averaged results from 10 trial runs for each simulation are presented including the number of iterations needed to converge, the time taken per iteration, the total training time and model accuracy with deviation in parenthesis. Additionally, we provide the statistical signifigance between the competing techniques for different sample sizes in Table 4.

Remark #5 As our analysis in Section 3.2 suggested, Sample-and-Learn is indeed more computationally intensive for each iteration than mini-batching.

Remark #6 Our empirical results indicate that randomly sampled shuffled data created using Sample-and-Learn enabled our neural networks to learn faster. This reduced the number of iterations needed for a network to converge, see Fig. 5a. As a result, we observed a consistent speedup in the total training time of the network, as seen in Fig.5c. The maximum speedup for the Susy dataset was 34% when $s = 50$. For the Higgs dataset the maximum speedup was 16% and was observed at $s = 100$.

Remark #7 For both mini-batching and Sample-and-Learn, the optimal sample size to achieve the highest accuracy is within the region of $s \in [50, 100]$; this result is consistent

TABLE 4: Statistical signifigance (p-value) for classification accuracy between the competing algorithms for different sample sizes

Sample size s	10	50	100	300	500
Susy	0.0217	0.0097	0.0015	0.0007	0.0001
Higgs	0.0266	0.0237	0.0018	0.0011	0.0012

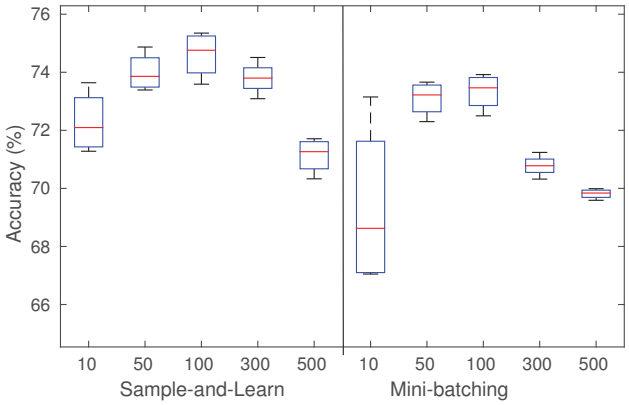


Figure 6: Relative accuracy for Sample-and-Learn (left) and mini-batching (right) for different sample sizes (s) for the Higgs dataset.

with mini-batching as discussed in [14].

Remark #8 Our proposed scheme, Sample-and-Learn consistently matches or exceeds mini-batching in terms of the classification accuracy of the trained models.

Remark #9 It is clear from Fig.6, that both schemes have the most *consistent* performances in terms of accuracy and accuracy deviation for $50 <= s <= 100$. Additionally both schemes suffer from sample sizes that are too small, especially mini-batching for $s = 10$.

6. Conclusion

Indeed, small data helps big data learning. It is in this spirit that we have proposed a novel training scheme, i.e. Sample-and-Learn, to address the main issue in big data, that is the volume of the data. We have shown the significant reduction in computational cost when a neural network learns from smaller, randomly sampled data. For rigour, we have provided a statistical analysis of our Sample-and-Learn Scheme to demonstrate that it is possible to significantly reduce the size of the training data size per iteration without causing a significant increase in the number of iteration required to attain high accuracy. We have also compared Sample-and-Learn to mini-batching both from a statistical and a simulation perspective. As a real-world application, we considered the large scale classification of supersymmetric and Higgs boson particles. Mini-batching captures a localised statistical representation of the data whereas our Sample-and-Learn enhances the likelihood of capturing the complete statistics of the data. As such our proposed method allows for faster convergence of the trained models.

References

[1] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning", Transactions on Neural Networks vol. 16, no. 10, pp. 1429–1451, 2003.

[2] G. Casella and E.I. George, "Explaining the Gibbs sampler", the American Statistician vol. 46, no. 3, pp. 167–174, 1992.

[3] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization", in Proceedings of the 20th ACM SIGKDD 46, pp. 661–670, 2014.

[4] S. Hidetoshi, "Improving predictive inference under covariate shift by weighting the log-likelihood function", Journal of Statistical Planning and Inference, vol. 90, pp. 227–244, 2000.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in Proceedings of the 32^{nd} conference on Machine Learning, pp. 448–456 2015.

[6] Y. LeCun, L. Bottou, G.B. Orr, and K.-R. Mller. "Efficient backprop', Neural networks: Tricks of the trade", pp. 9–50. Springer, 1998.

[7] P. Balti, P. Sadowski and D. Whiteson. "Searching for exotic particles in high-energy physics with deep learning", Nature Communications vol. 5, 2014.

[8] P. S. Efraimidis and P. G. Spirakis, " Weighted random sampling with a reservoir", Information Processing Letters - IPL , vol. 97, no. 5, pp. 181–185, 2006.

[9] M. Al-Kateb, L. S. Byung and X. S. Wang, "Adaptive-size reservoir sampling over data streams", in Proceedings of the 19th International Conference in Scientific and Statistical Database Management, pp.22–31 2007.

[10] V. Malbasa and s. Vucetic, "Back to results A Reservoir sampling algorithm with adaptive estimation of conditional expectation", in the Proceedings International Joint Conference on Neural Networks, pp. 2200-2204, 2007.

[11] S. Vitter, "Random sampling with a reservoir", in ACM Transactions on Mathematical Software, pp. 33–57, 1985.

[12] K. H. Li *Reservoir-Sampling Algorithms of Time Complexity O(n(1 + log(N/n)))* , in ACM Transactions on Mathematical Software, vol. 20, pp. 481–493, 1994.

[13] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio, "Theano: A CPU and GPU math expression compiler", Proceedings of the Python for Scientific Computing Conference (SciPy) Jun 2010.

[14] Y. Bengio, K.-R. Muller and G. Montavon, "Practical recommendations for gradient-based training of deep architectures" in Neural Networks: Tricks of the Trade, Reloaded. Springer, pp. 437–478, 2013.