Big Data – Data Processing and Machine Learning in the Cloud

Author – Jakob Brown

Abstract – The following report examines parallelisation of data and its effects in terms of processing efficiency. This is a key challenge of big data analytics, as the choice between cloud configurations dictates not only the speed at which data is processed, but also the cost of said processing tasks, which can prove significant, especially for 40% of analytics jobs which are perpetual tasks [1]. Using an example of throughput and/or machine learning using image files, different factors that influence task running time and quality are identified and discussed in context of the domain literature hitherto.

Introduction

The experiment conducted which forms the basis for the present report was structure as following: Using code from the "Fast and Lean Data Science" course [4], a TensorFlow image dataset was read to the cloud using google cloud's DataProc platform, where various pre-processing tasks were carried out. Multiple Parameters were varied to test for differences in parallelisation speed. The throughput was documented for each combination of these parameters in images per second (IPS).

Parallelisation was used again to pre-process and then create the TFRecord files themselves and write them to a designated bucket. Different cloud configurations were tested, specified as clusters on the DataProc platform.

Finally, image classification neural networks were built using Tensorflow's Keras, which were applied to the image files in google cloud's ai-platform, varying on batch size, as well as different distributed learning strategies which could inform the way the virtual machines delegated the task between itself and/or its' workers.

Monitoring of batch size as a variable in processing speed and learning curve was motivated by research documenting the potential benefits of increased batch size mimicking that of decayed learning rate, such as significantly reduced number of require parameter updates to train a model, while maintaining near-identical test set accuracy [5]. This is a highly desirable change which inevitably leads to reduced training time, and consequently reduced cost, while avoiding the apparent trade-off between training time and accuracy [2].

The tasks were conducted using Pyspark. Resilient distributed datasets (RDDs) were made of the image file names, before being mapped through the pre-processing functions and subsequently some analytical functions. Initial cloud run time was ~5241 seconds. Operation logs show a swift spike in CPU utilization at the beginning the of the process, denoting the formation of the cluster (*figure 1).* The latter portion of the job, which took far longer, was the collection of the final RDDs to be saved to the designated bucket. The CPU utilization corroborates this, with lower, more steady activity around 10% for a considerable period. The Network can be seen to read in the data while the disk wrote out the processed data in tandem.

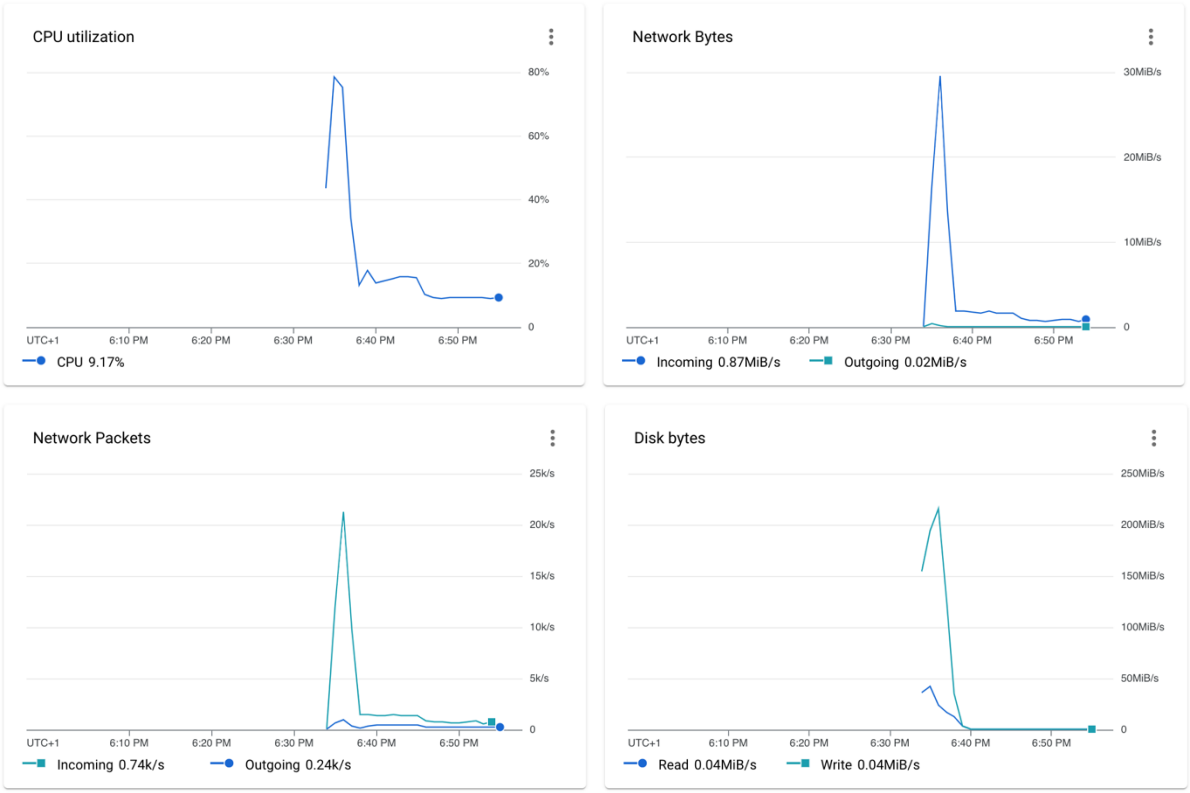

Figure 1 – Google Cloud metrics for Spark processing job (without caching).

These metrics above were compared with those of the same job, conducted with caching of existing results added to the process. This allows for less time spent re-reading in data if it is being used multiple times for a job, instead storing close-to-hand in google cloud storage network which acts as a content delivery network (CDN). This is magnitudes faster due to memory being faster than disk. An L1 cache reference takes ~0.5 nanoseconds. A disk-seek, conversely, takes approximately 10,000,000 nanoseconds. Furthermore, it reduces bandwidth consumption, thus decreasing network traffic and congestion for the cloud, reducing risk of straggling machines, thus making job performance more reliable [3].

With caching implemented, run time was ~740 seconds; just over 7 x quicker than the same job without caching. This cached data can be seen to be brought in by the network after the initial spike at the start of the task in network bandwidth in *figure 2.* It also can be seen to cost some more CPU utilization, but saves network, a tradeoff to be weary of when more data is being cached than in the present task.

Other inefficiencies which were found to improve run time were avoiding nested structures, as well as using enumerating objects in lieu of lists of strings for keys. This keeps the working set for any given operation small enough to keep in the Spark execution memory, which significantly shares a unified region with storage memory. Increasing the level of parallelism also induced the usage of a much higher amount of the cluster's workers as well, aiding processing. This is due to Spark setting the number of "map" tasks to run according to the size of the file, for which it uses the number of partitions of the largest parent RDD.

The results of conducting a linear regression on the returned RDDs which contained each parameter with a list of its corresponding throughputs in IPS. Stark improvement in throughput speed is observable between processing each image, and by writing multiple

samples to a single file (*Figure 3)*. The figure shows an example of the two compared, in this instance by batch size. *Table 1* notes the correlation coefficients for the other parameters.

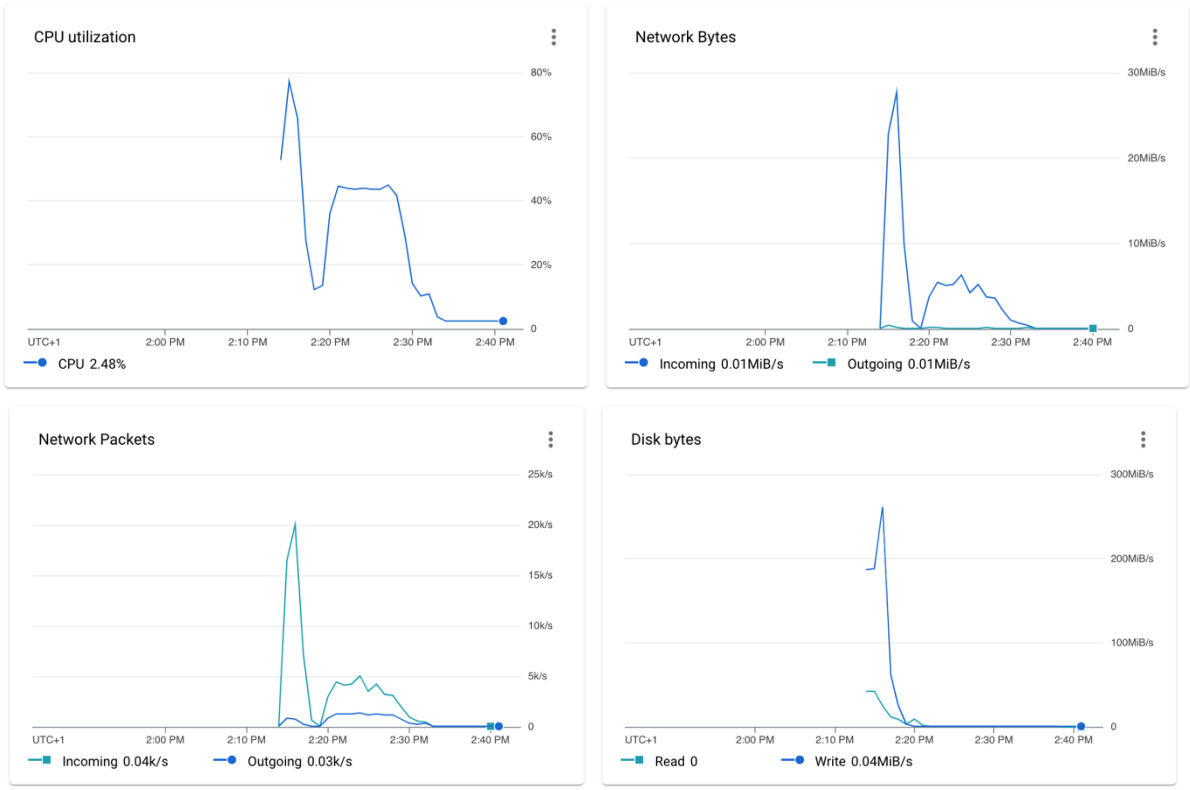

Figure 2 - Google Cloud metrics for Spark processing job (with caching).

Weak positive correlations can be observed for all parameters save for repetition. Looking at *Figure* 3, the peak for throughput it would seem would be somewhere around batch size 20 (of the batch sizes measured), before a gradual decline at greater amounts. This trend is also seen in the data size metric, peaking early at around 200-250, but still going strong up to 1200. An important factor to consider when conducting a speed test on the parallelisation methods is the potential for the receiving entity, in this case the google cloud bucket, to limit the speed of proceedings from its end (buckets have an initial I/O capacity of approximately 5000 object read requests per second).

In general, however, these results would suggest that the given parameters most likely affect the processing efficiency of jobs in a *non*-linear fashion. This highlights how the nature of a specific task can warrant different optimal values for parameters such as batch size [1]. Having cloud configurations to optimize as well thus creates a wealth of choices for implementing a task, the optimums of which are influenced by each other. For large-scale tasks including machine learning, the cost incurred for failing to optimize these parameters can be seen to be extremely large when scaled up [1]. Alipourfard et al (2012) highlight the two ‘strawman solutions’ to this issue as either base a configuration on the one that gives the best performance for a particular model, or to tediously iterate through each job that needs doing and determine the best configuration through trial and error. The former lacks adaptivity, which is essential to address the aforementioned non-linear, capricious nature by which optimal configurations and parameters vary from task to task. The latter is too costly, both fiscally and in terms of time. It therefore demonstrates great value in any proposed system that provides a kind of solution to the problem of automating correct configurations of cloud machines as well as the methods by which their inputs are fed to them. *CherryPick*

poses a potential solution to the cloud configuration side of things, allowing great adaptivity through the non-parametric Bayesian Optimization framework [1].

Filetype	Parameter	Correlation Coefficient
TFR	Batch size	0.28
IMG	Batch size	0.39
TFR	Batch number	0.31
IMG	Batch number	0.32
TFR	Data size	0.23
IMG	Data size	0.33
TFR	Repetition	0.06
IMG	Repetition	0.01

Table 1 – Correlation Coefficients from Linear Regression Between Parameters and throughput in Images Per Second, filtered by Filetype.

The next task was to implement the previous code for pre-processing images before writing the files from the cloud straight to the bucket. This further restricts the speed at which things can potentially be done for larger jobs, as google cloud's bucket I/O capacity for write-requests is approximately 1000 a second. This job was performed on a variety of configurations, the cluster details and runtime speeds for which are documented in *Table 2.*

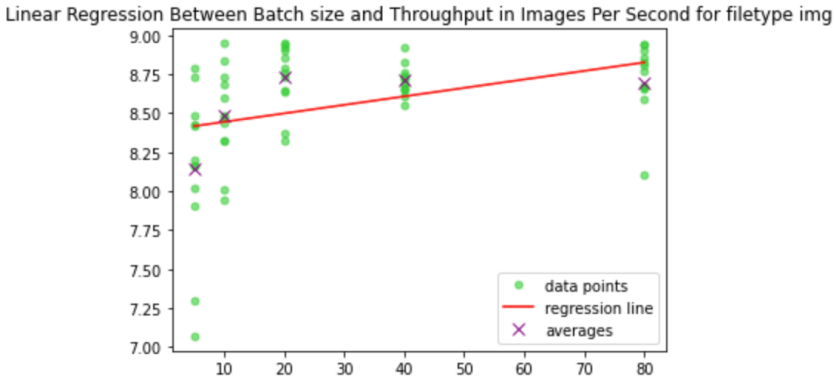

Figure 3 – Linear Regression Between Batch Size and throughput In images per second for TFRecords (top), and image files (bottom).

The disk I/O and network bandwidth allocation of each of the tested clusters in the cloud are shown in *Figure 4.* Examining the scale of the axes of the figures shows a clear diminution in the network bandwidth and disk I/O as more resources are utilised. The peak disk I/O for the 1 machine, 7 workers with 1 CPU each was ~2.8 kb/s, compared to 1.5 kb/s for 2 machines with double the resources, and just 0.4 kb/s for the single machine with 8 CPUs. A similar pattern is observed with the network bandwidth, with 1 machine and 7 workers maxing out incoming bandwidth at ~37 mebibytes per second (MiB/s), 2 machines with 3 workers with 2 CPUs each maxed at 21 MiB/s, and 1 machine with 8 CPUs topped reached at most 4 MiB/s. The time taken for each cluster to complete the task at hand seen in *Table 2* also compounds the significance of the effect that bulking up resources to parallelise over can have on processing efficiency.

Cluster	**Time taken (seconds)**
1 Machine 7 worker 1 CPU each	137.9
2 Machines 3 workers 2 CPUs each	98.45
1 Machine 8 CPUs	79.8

Table 2 – Different cluster configurations, and the time taken for them to complete image file writing task.

For the image classification section, batch size once again was manipulated, in order to test the hypothesis posed by Smith et al (2018) regarding increased batch size emulating the same role of decaying learning rate in reaching model convergence while also reducing training time. As well as this, two different distributed learning strategies were implemented across all batch sizes: a mirrored strategy, using on one machine with multiple GPUs (in this case, 8). This distributes replica copies of the data across the GPUs, replicating the variables as well, which in turn are kept in sync with each other by applying the identical updates. Secondly, a multi-worker-mirroring strategy was applied, which implements synchronous training of replicas distributed across multiple workers, potentially each with multiple GPUs. For this, the master machine was a standard GPU, with 4 workers, each with

a standard GPU. Training the model without any strategy was also done with both a standard GPU and a complex model GPU with 8 k80 GPUs. Each model was trained for 25 epochs, such that near-total convergence was allowed, while also minimising running time and thus cloud usage which was limited by using free account credits.

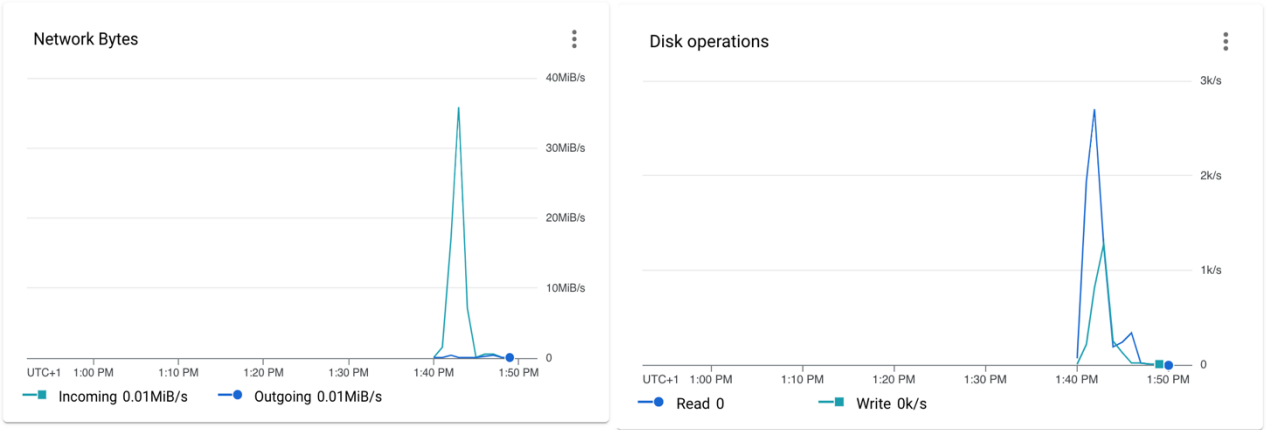

Figure 4(a) – Disk I/O and Network bandwidth allocation for cluster with 1 machine and 7 workers, each with 1 CPU.

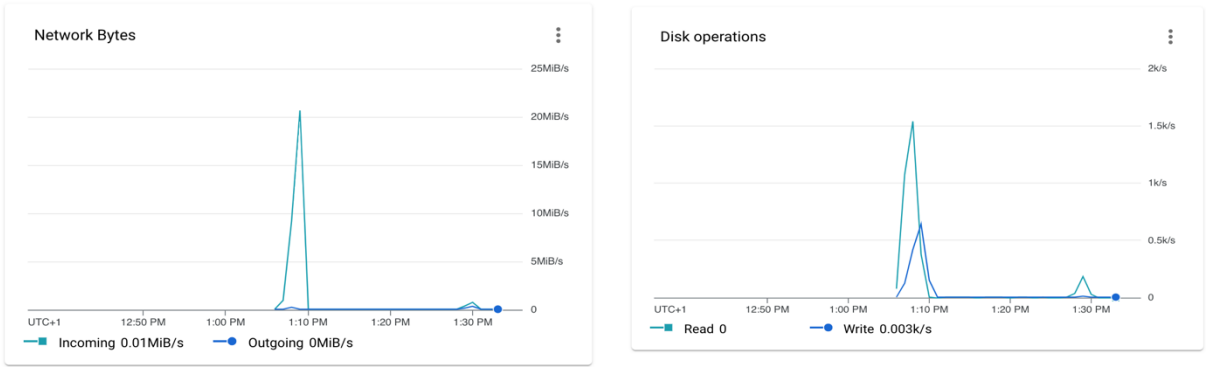

Figure 4(b) - Disk I/O and Network bandwidth allocation for cluster with 2 machines and 3 workers, each with 2 CPUs.

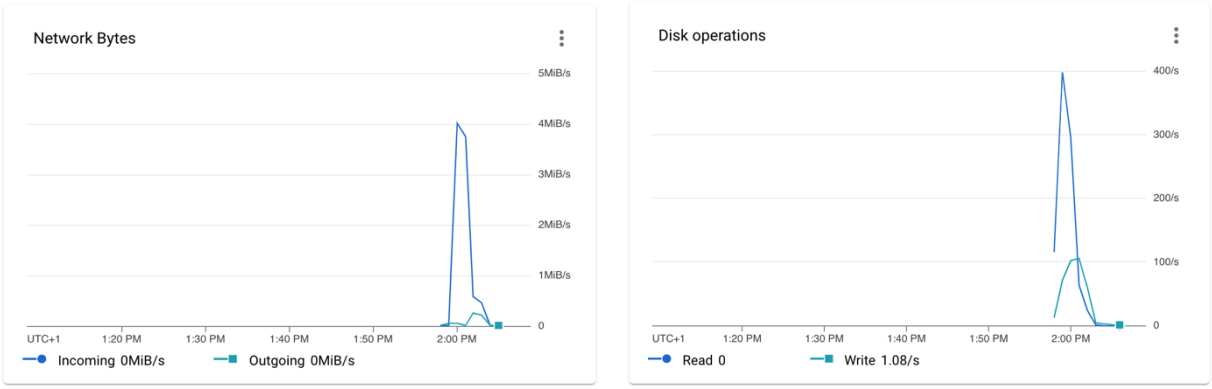

Figure 4(c) - Disk I/O and Network bandwidth allocation for cluster with 1 machines and 8 CPUs.

The Mirrored and multi-worker mirrored both performed similarly in terms of training time, but the mirrored strategy appeared to perform best across both metrics. Analysis of master GPU utilization and memory shows a strain on resources when no strategy is harnessed and a standard GPU is used, with both remaining high near 100%. This is reduced greatly when using the complex 8x k80 GPUs down to <20% for both. Using strategies has both oscillating more than no strategy, suggesting the strategies perform well in distributing the processing across parallelisations, but also some extra workload perhaps to perform the task of keeping all replicas in sync with each other using the conceptual 'mirror variable'. The multi-worker strategy had roughly similar activity across all workers, suggesting an increase in workers may create a more productive distribution of workload.

batch size	strategy	cluster	training time	25 epoch accuracy	max accuracy
32	None	complex_model_l_gpu	157.434	0.362	0.362
64	None	complex_model_l_gpu	157.434	0.308	0.320
128	None	complex_model_l_gpu	153.631	0.519	0.519
32	None	standard_gpu	172.481	0.243	0.256
64	None	standard_gpu	157.205	0.449	0.449
128	None	standard_gpu	153.911	0.487	0.487
32	Mirrored	complex_model_l_gpu	144.300	0.242	0.258
64	Mirrored	complex_model_l_gpu	101.342	0.399	0.410
128	Mirrored	complex_model_l_gpu	83.517	0.502	0.502
256	Mirrored	complex_model_l_gpu	74.614	0.446	0.446
512	Mirrored	complex_model_l_gpu	67.593	0.348	0.355
32	MultiWorkerMirrored	standard_gpu - 4 workers	143.234	0.353	0.353
64	MultiWorkerMirrored	standard_gpu - 4 workers	96.113	0.401	0.419
128	MultiWorkerMirrored	standard_gpu - 4 workers	76.851	0.363	0.369
256	MultiWorkerMirrored	standard_gpu - 4 workers	70.305	0.326	0.326
512	MultiWorkerMirrored	standard_gpu - 4 workers	64.515	0.390	0.390

Table 3 – Neural Network performance on image classification, varied by batch size, strategy, and cloud configuration.

Model training time decreased with increased batch size, without losing much validation accuracy at all, giving credence to Smith et al's (2018) findings. Just some of the many potential configurations were tested, on only one of the available platforms, contextualising the vastness of this domain, and the need for a means of discerning efficient cloud configurations for a given job with greater ease than is currently available [1].

Considering real-world applications, it can be seen many different bottlenecks and problems can arise depending on the kind of processing required. For batch-processing, many errors can arise, which may require constant maintenance. For stream process, output speed needs to be as fast as input speed. Real-time processing of data through these commercial cloud companies is at the mercy of the software system. This level of

abstraction has been shown to generate a consistent coefficient of variation in almost every run of a cloud configuration [3]. In stream processing machine learning pipelines where processing speed matters most, increased batch size appears to be an excellent way to minimise processing speed, and so a middle ground of mini-batch processing could be implemented, bringing in enough data to benefit from this phenomenon while also keeping reaction to data as swiftly as necessary. Batch processing can of course maximise this, but may need to spend to save, by utilising the most heavy-load configurations available in order to harness enough memory to process the biggest batches, which will save the most in processing training time [5]. A system like *CherryPick [1],* which could take in to account these tailored needs in terms of cost/speed would be highly instructive in this process.

References

[1] Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M. and Zhang, M., 2017. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)* (pp. 469-482).

[2] Byrd, R.H., Chin, G.M., Nocedal, J. and Wu, Y., 2012. Sample size selection in optimization methods for machine learning. *Mathematical programming*, *134*(1), pp.127-155.

[3] Ferguson, A.D., Bodik, P., Kandula, S., Boutin, E. and Fonseca, R., 2012, April. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM european conference on Computer Systems* (pp. 99-112).

[4] Gorner, M. 2020. Fast and Lean Data Science Course. Github Repository, https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/fast-and-lean-data-science

[5] Smith, S.L., Kindermans, P.J., Ying, C. and Le, Q.V., 2017. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.

Word count: 1998

*Note: My conclusion may look small for a 20% section. this is because a good portion of my answers to question 5 - particularly a lot of contextualization in relation to the 2 concepts proposed by the given papers (5a) - are mixed in earlier in discussion of tasks earlier in the report. I hope this is ok and clear to see.

Link for the online accompanying notebook: https://colab.research.google.com/drive/1Qo7bIWyn39okzb2V2OBs9rSMi59pk_ET?usp=sharing