Big Data: Data pre-processing and ML training

adbr039 – Marlon Martins

https://colab.research.google.com/drive/16-x7zWrJmzeALBwv9J8UXNhmbzw1ajHd?usp=sharing

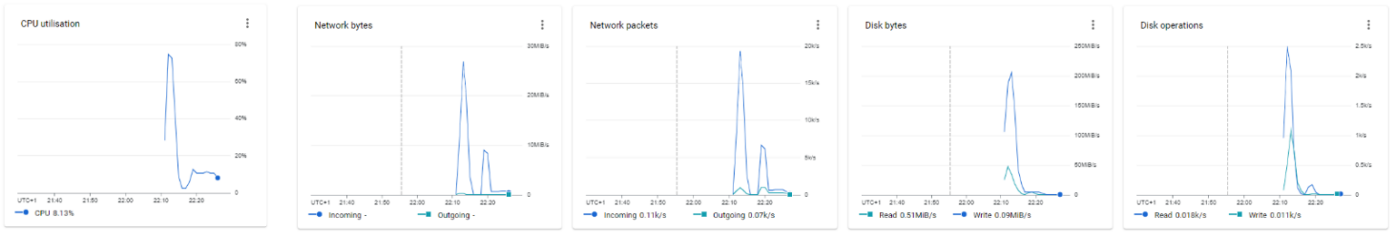
Task 2: Parallelising the speed test with Spark in the cloud

2c) Improve cluster efficiency.

Having created a script to use spark to run multiple tests in parallel on Google Cloud, we now want to look at ways in which we can improve our clusters efficiency. One of the ways we can do this after parallelization is to implement caching. The graph below shows the cluster performance when running our cluster without any caching. After the initial spike (due to the cluster starting) we can see another spike from the cluster processing our script. Our clusters consist of 1 Master with 1vCPU and 7 Workers with 1vCPU's each.

CPU utilisation during the second spike rose to about 10% and remained at this level until the task was complete. Network bytes rose to just below 10MB/s for incoming data and Network packets rose to 6-7k/s. Disk bytes had a marginal increase in read/write speeds. The whole task took ~8 minutes to run.

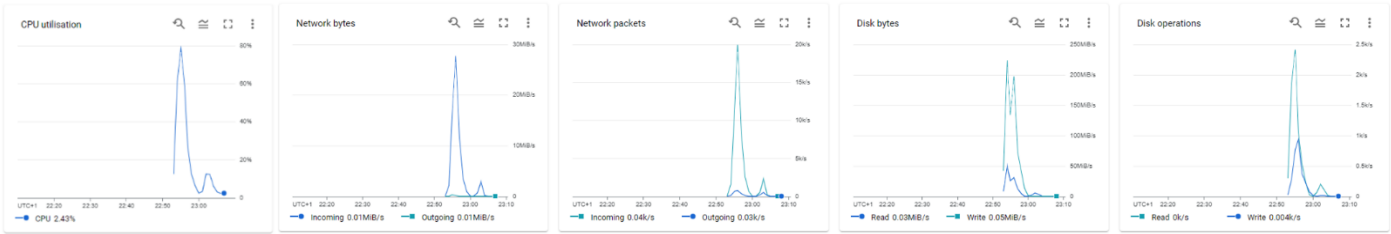
Master with 1vCPUs and 7 workers with 1vCPUs - No Caching



The graph below illustrates the cluster performance for the same script only this time caching was used.

CPU utilisation during the second spike (task processing) is very similar to the cluster performance seen with no caching reaching ~10% utilisation, however with caching this level of CPU utilisation was sustained for a much shorter period. There is a significant drop in Network bytes when using caching, with incoming only reaching ~4MB/s and Network packets incoming reaching ~3k/s. The whole task with caching also only took ~2 minutes to run.

Master with 1vCPUs and 7 workers with 1vCPUs - With Caching



Caching it in memory significantly increases reading speeds, even if there is not enough memory to store all the data, caching is still very useful as it will cache as much data as it can in memory and assigning the rest to disk.

2d) Retrieve, analyse and discuss the output

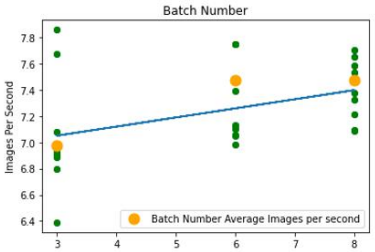
Within this section we extracted the results from the data which was parallelized without caching and ran through the Google Cloud platform. In the graphs below we are implementing a linear regression over each parameter for both datasets; Image files and TFRecord files. The visualisations clearly show that the throughput of images per second for each dataset vary greatly, although both show an increase in throughput as the size and number of batches increase, images read from the TFRecord files have a much faster level of throughput in terms of images per second. The reason behind why TFRecord files can be read at a faster rate than regular image files is since TFRecords are essentially dictionaries storing the mapping between a key and the actual data, this enables faster reading due to the low access times. When running tests in parallel on the cloud one bottleneck which arises when using different configurations of devices, depending on the configuration some may be faster and must wait for the slower devices to catch up. We must also consider our partitions, using too few partitions will mean that we are not fully utilising all the cores available in our cluster, use too many and we are incurring cost.

This has significant implications for large-scale machine learning, improving training time and allow for better model tuning, facilitating tasks such as grid searches.

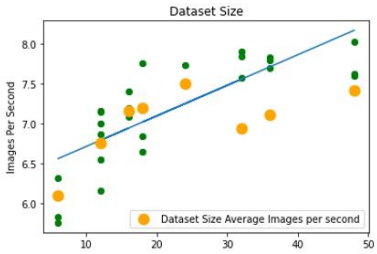
Linear regression can be useful to reflect observed effects, in our specific use case it can be useful to demonstrate throughput vs other parameters, however in practice it does have some shortfalls, most significant of which is how it does not show variances.

```
--------------------------------------------------------------------------------
Individual Image Files
--------------------------------------------------------------------------------
```

	Batch Size	Average Images per second
0	2	6.918224
1	4	7.261011
2	6	7.657867



	Batch Number	Average Images per second
0	6	7.471554
1	8	7.477173
2	3	6.976056



	Dataset Size	Average Images per second
0	6	6.099600
1	12	6.758606
2	16	7.155102
3	24	7.505533
4	32	6.936509
5	18	7.202462
6	36	7.114588
7	48	7.412767

	Batch Size	Images per second
0	2	6.805383
1	2	6.385636
2	2	6.549568
3	2	7.078980
4	2	6.922069
5	2	6.558484
6	2	7.279193
7	2	7.441220
8	2	7.413148
9	4	6.440596
10	4	6.943207
11	4	6.382044
12	4	7.628477
13	4	7.245739
14	4	7.624262
15	4	7.106559
16	4	7.657027
17	4	7.540611
18	6	6.643679
19	6	6.982617
20	6	7.028400
21	6	7.957898
22	6	7.728667
23	6	7.707259
24	6	7.531015
25	6	7.353190
26	6	7.679247

	Batch Number	Images per second
0	3	6.937491
1	3	6.388893
2	3	7.076888
3	6	7.102450
4	6	6.985064
5	6	7.059020
6	8	7.098342
7	8	7.085617
8	8	7.326665
9	3	6.907408
10	3	6.919506
11	3	7.859556
12	6	7.393489
13	6	7.126520
14	6	7.135595
15	8	7.377495
16	8	7.584099
17	8	7.702948
18	3	6.797726
19	3	6.889260
20	3	7.678941
21	6	7.047163
22	6	7.748101
23	6	7.749443
24	8	7.212165
25	8	7.536342
26	8	7.654753

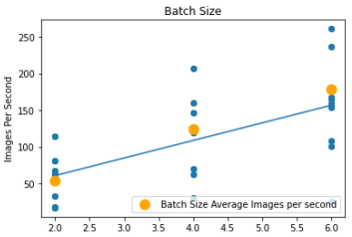
	Dataset Size	Images per second
0	6	5.820794
1	6	6.317808
2	6	5.754269
3	12	6.161216
4	12	7.160976
5	12	6.553559
6	16	7.406052
7	16	7.195872
8	16	7.088756
9	12	6.868984
10	12	7.150084
11	12	6.999275
12	24	7.504857
13	24	7.517675
14	24	7.741489
15	32	7.572517
16	32	7.849745
17	32	7.908557
18	18	7.755525
19	18	6.843496
20	18	6.645915
21	36	7.800189
22	36	7.703315
23	36	7.837994
24	48	7.626469
25	48	7.605769
26	48	8.034742

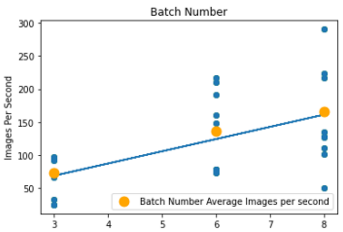
Batch Size: `LinregressResult(slope=0.11606362285065033, intercept=6.706640860437396, rvalue=0.4123728683093956, pvalue=0.03255581140910788, stderr=0.051281608367255684)`

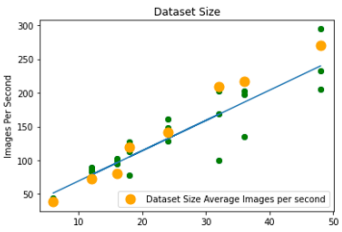
Batch number: `LinregressResult(slope=0.06944584837509178, intercept=6.842804600853794, rvalue=0.40613532295070653, pvalue=0.035548971886432496, stderr=0.0312509213917589)`

Dataset Size: `LinregressResult(slope=0.038421330809266 74, intercept=6.330075579990327, rvalue=0.7778406140553842, pvalue=1.7977769802518677e-06, stderr=0.006208554937793 08)`

```
--------------------------------------------------------------------------------
TFRecord Image Files
--------------------------------------------------------------------------------
```







	Batch Size	Average Images per second
0	2	53.314818
1	4	123.800682
2	6	178.751329

	Batch Number	Average Images per second
0	6	136.886578
1	8	165.343534
2	3	73.651984

	Dataset Size	Average Images per second
0	6	39.228888
1	12	72.442340
2	16	80.133973
3	24	141.833407
4	32	209.139351
5	18	119.782475
6	36	217.312020
7	48	270.517109

	Batch Size	Images per second
0	2	16.919077
1	2	32.802254
2	2	18.332817
3	2	60.055398
4	2	61.985399
5	2	67.830673
6	2	81.320421
7	2	61.221275
8	2	114.580078
9	4	29.919600
10	4	62.242266
11	4	69.892932
12	4	146.398258
13	4	122.105216
14	4	118.810676
15	4	126.108213
16	4	159.761137
17	4	206.664636
18	6	108.249260
19	6	100.822326
20	6	25.784127
21	6	154.054151
22	6	159.363605
23	6	167.202371
24	6	236.100681
25	6	260.911803
26	6	164.707772

	Batch Number	Images per second
0	3	25.356530
1	3	25.267323
2	3	32.641683
3	6	79.151128
4	6	74.942225
5	6	79.326774
6	8	110.956387
7	8	50.334077
8	8	102.052899
9	3	74.705519
10	3	69.581513
11	3	66.204249
12	6	73.471925
13	6	148.413586
14	6	160.258953
15	8	127.874635
16	8	135.242869
17	8	126.392396
18	3	93.117661
19	3	97.575487
20	3	91.933645
21	6	209.479536
22	6	216.537018
23	6	191.246378
24	8	290.238145
25	8	223.403059
26	8	216.489983

	Dataset Size	Images per second
0	6	44.048638
1	6	41.963462
2	6	37.610442
3	12	83.546464
4	12	89.603053
5	12	72.143416
6	16	102.033194
7	16	100.283572
8	16	94.336965
9	12	85.244306
10	12	86.240001
11	12	76.164751
12	24	160.981427
13	24	148.115069
14	24	128.164763
15	32	202.717334
16	32	168.689621
17	32	99.530022
18	18	77.642342
19	18	112.916717
20	18	126.731235
21	36	135.228395
22	36	202.233407
23	36	198.041999
24	48	205.623506
25	48	294.704642
26	48	232.258822

Batch Size: `LinregressResult(slope=23.948575183825252, intercept=12.877789022611623, rvalue=0.6039570644304189, pvalue=0.0008503628273712321, stderr=6.320786412270673)`

Batch number: `LinregressResult(slope=18.428800543352832, intercept=13.799596298987112, rvalue=0.5635616894762623, pvalue=0.0022050637936269716, stderr=5.402621546125731)`

Dataset Size: `LinregressResult(slope=4.482104546748962, intercept=24.58331789134617, rvalue=0.9199422256953084, pvalue=1.1631508597105517e-11, stderr=0.38202981716512024)`

Task 3: Write TFRecord files to the cloud with Spark

3ci) Explain the difference between this use of Spark and most standard applications

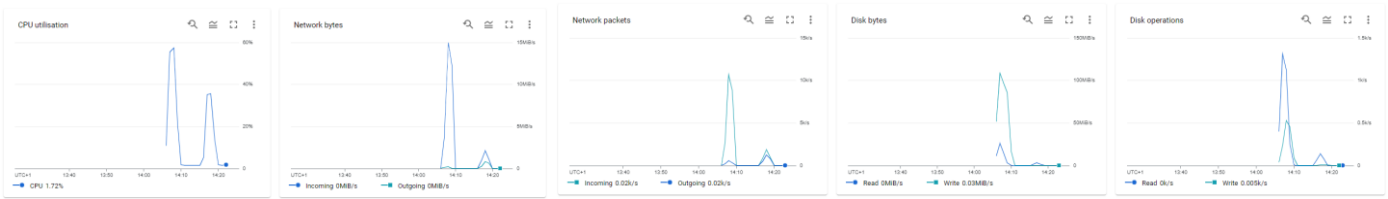
The main difference in this tasks implementation of Spark compared to previous tasks is that previously we were using the map method to convert each element in the RDD through a function and returns a new RDD representing the results. In this task we are using *mapPartitionsWithIndex* which is like *map* but converts each partition/shard of the RDD through a function and returns a new RDD representing the results with an integer value representing the index of each partition. Therefore, it is faster to use the method *mapPartitionsWithIndex* as unlike map, it does not process each element at a time but processes elements in blocks or partitions and only needs to be called once.

3cii) Test your program with 4 machines with double the resources each (2 vCPUs, memory, disk) and 1 machine with quadruple resources. Discuss the results in terms of disk I/O and network bandwidth allocation in the cloud.

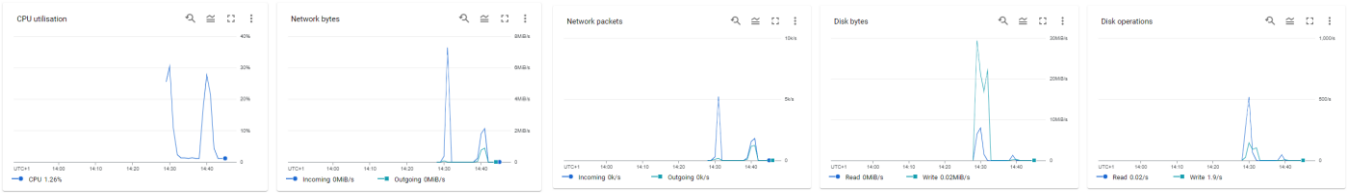
In this task we are running the script we created in task 3a on different cluster VM configurations to determine how it impacts the running time and resource utilisation. In the graphs below we are looking at the following configurations: Master with 1vCPU and 7 workers with 1 vCPU's, Master with 2vCPU's and 3 workers with 2vCPU's and one machine with 8vCPU's. When looking at the graphs below, we are interested in the second spike as the first spike is the cluster initialising.

Master with 1vCPUs and 7 workers with 1vCPUs - Runtime: ~2 minutes 20 seconds

Master with 2vCPUs and 3 workers with 2vCPUs - Runtime: ~2 minutes 68 seconds



Master with 8vCPUs - Runtime: ~2 minutes 32 seconds



When running our program with the above configurations it is easy to see a clear distinction in resource utilisation.

The configuration of a Master and 7 Workers with a single vCPU each reach; CPU utilisation of ~41%, Network Bytes: ~4MB/s, Network Packets: ~4k/s, Disk Bytes Read: ~10MB/s and Disk Operation Read of 0.5k/s.

The configuration of a Master and three workers with two vCPU's each reach; CPU utilisation of 38%, Network Bytes: ~2.5MB/s, Network Packets: ~2.5k/s, Disk Bytes Read: ~20MB/s and Disk Operation Read of 0.2k/s.

The final configuration of a Master with 8vCPU's reaches; CPU utilisation of ~29%, Network Bytes: ~2MB/s, Network Packets: ~2k/s, Disk Bytes Read: ~2MB/s and Disk Operation Read of 50/s.

The run times for all the configurations were the same, however the experiments conducted above clearly indicate that the more workers there are, the higher the resource utilisation for CPU, Network and Disk I/O in these cases.

Task 4: Machine learning in the cloud

4c) Distributed Learning.

One of the distributed learning strategies implemented is MultiWorkerMirroredStrategy, which is a type of synchronous training. As per TensorFlow's documentation, this strategy works by creating copies of all variables in the models' layers on each device across all workers, we have found that removing the *'relu'* function improves learning time and accuracy significantly for all strategies, however for a fair comparison all activation functions will remain the same as per the pretrained model provided. The second distributed learning strategy used is called One Device Strategy which places variables and computations on a single specified device, it is very similar to utilising no strategy, except, this strategy will ensure that all variables created within the *strategy.scope* are explicitly placed on the specified device.

Learning Strategy: No Strategy
Batch Size: 64
Time Taken: 36.919182538986206

	Accuracy	Validation Accuracy	Loss	Validation Loss
0	0.247768	0.295759	4.580953	1.581033
1	0.286458	0.339286	1.572894	1.556272
2	0.301339	0.287946	1.551295	1.572056
3	0.297991	0.292411	1.553896	1.563832
4	0.308780	0.318080	1.530839	1.532782

Learning Strategy: No Strategy
Batch Size: 128
Time Taken: 35.93513083457947

	Accuracy	Validation Accuracy	Loss	Validation Loss
0	0.216518	0.389509	4.064557	1.544672
1	0.325521	0.398438	1.541319	1.512172
2	0.341890	0.406250	1.515101	1.510826
3	0.343378	0.371652	1.518589	1.502479
4	0.366815	0.396205	1.474814	1.488355

Learning Strategy: Multi-Worker Mirrored Strategy
Batch Size: 64
Time Taken: 42.667688608169556

	Accuracy	Validation Accuracy	Loss	Validation Loss
0	0.272321	0.331368	3.267859	1.492625
1	0.321429	0.380896	1.486807	1.427352
2	0.319568	0.412736	1.459903	1.424055
3	0.309524	0.358491	1.480568	1.489054
4	0.295015	0.349057	1.486048	1.482223

Learning Strategy: Multi-Worker Mirrored Strategy
Batch Size: 128
Time Taken: 28.878906726837158

	Accuracy	Validation Accuracy	Loss	Validation Loss
0	0.229539	0.236650	4.355719	1.567727
1	0.283482	0.316748	1.589471	1.557127
2	0.293527	0.343447	1.553979	1.505046
3	0.302083	0.337379	1.526516	1.493428
4	0.319196	0.361650	1.491370	1.421552

Learning Strategy: One-Device Strategy
Batch Size: 64
Time Taken: 36.97475004196167

	Accuracy	Validation Accuracy	Loss	Validation Loss
0	0.225446	0.237723	3.691486	1.607056
1	0.254464	0.234375	1.602063	1.608294
2	0.250744	0.235491	1.608757	1.605337
3	0.254464	0.236607	1.597935	1.602103
4	0.255580	0.236607	1.598367	1.602958

Learning Strategy: One-Device Strategy
Batch Size: 128
Time Taken: 36.4362211227417

	Accuracy	Validation Accuracy	Loss	Validation Loss
0	0.230655	0.301339	5.019625	1.580752
1	0.263393	0.313616	1.574900	1.547816
2	0.293899	0.353795	1.549908	1.537496
3	0.318080	0.379464	1.493103	1.442321
4	0.316964	0.381696	1.461083	1.430415

As shown in the results above when utilising a batch size of 128, training time is much shorter which is expected, however, where we see a significant drop-in training time is with the larger batch size and using the multi-worker mirrored strategy which uses distributed learning.

Task 5: Discussion in context.

5a) Contextualise

The focus of this task is to outline and consider applications for two concepts. The first concept was introduced by *Alipourfard et al (2017)* in the paper *'Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics',* the paper proposes an alternative method for selecting the most near-optimal cloud configuration for a given task while maintaining costs to a minimum. This concept is especially appropriate when considering our previous work in Task 3, writing TFRecord files to the cloud with Spark, where selecting the best cloud configuration can be an expensive process. One method of finding this optimal cloud configuration is referred in this paper as *Brute-force search*, where through a process of elimination the optimal configuration is found, however, due to the large variety of virtual machine configurations available, this would be a very expensive process. Utilising the concept of *CherryPick*, we would be able to build a performance model utilising Bayesian Optimization that is accurate enough to allow us to distinguish near-optimal configurations from the rest and therefore not have to run all configurations like in the *Brute-force search*, saving costs and running time which is especially important when running recurring jobs such as Spark in the cloud.

The second concept to consider was introduced by *Smith et al (2018)* in the paper *'Don't Decay the Learning Rate, increase the Batch Size',* where it proposes an alternative to learning rate decay, stating that by increasing batch sizes during training one can achieve the same test accuracies but with much shorter training times. This concept is especially relevant to Task 4, where we are applying machine learning in the cloud, more specifically this concept can be applied when applying distributed learning. Although we did not use momentum or learning rate decay, we did implement differing batch sizes (see task 4 above). This paper proposes that increasing batch sizes is favourable compared to momentum and learning rate decay as it can achieve similar performance with similar accuracy and shorter training times (as shown in the experiments on CIFAR-10 and ImageNet illustrated in this paper), which coupled with distributed learning would allow for a higher rate of models being run.

5b) Strategize

The three main types of data processing are: Batch Processing, Real-time Processing and Stream Processing.

Batch processing occurs when we are processing large volumes of data such as a large part of a dataset or the entire dataset at once and unlike the other two types of data processing the data does not change, these types of data can include any kind of historical data such as financial transactions or Images as we have done. Increasing batch sizes *(as per Smith et al. (2018))* and processing the data through distributed learning strategies as done in Task 4 can significantly reduce processing time.

Real-time processing on the other hand refers to cases where the data grows as and when a new datapoint is received, examples of this would include stock markets, as the data grows as and when new stock quotes come in. When dealing with recurring data it is important to consider costs of utilising this data on the cloud and therefore would benefit most from the most cost-effective cluster configuration using *CherryPick*, this system finds the most near-optimal configuration with low search cost and time. *CherryPick* also relies on previous workloads to learn and suggest good cloud configurations for similar workloads, as this type of processing is always growing, it allows for the system to learn and continuously suggest good cloud configurations as the data changes over time.

Stream processing can be seen as the middle ground between batch processing and real-time as it groups new incoming data at regular intervals, most frequently these intervals are determined by time, such as it groups new incoming data and every minute it will process all the aggregated data within that time. Utilising the methods mentioned in the paper by *Smith et al. (2018)* such as increasing batch sizes during training combined with increased learning rate and momentum, we would be able to significantly minimise the number of parameter updates and be able to process data at a high rate whilst maintaining validation accuracy. This would be significant in this situation where the data would be processed at a regular interval that matches the training time of our models. Utilising these concepts, we would be able to minimise training time and the interval of which data is received, creating a process by which as soon as data is received the model will have finished training on the previous mini batch of data.

Word Count:	1995

References

Janetzky, P. (2021) *A practical guide to TFRecords*, *Medium*. Available at: https://towardsdatascience.com/a-practical-guide-to-tfrecords-584536bc786c (Accessed: 16 April 2021).https://towardsdatascience.com/apache-spark-caching-603154173c48

RDD Programming Guide - Spark 3.1.1 Documentation (2020) *Spark.apache.org*. Available at: https://spark.apache.org/docs/latest/rdd-programming-guide.html#passing-functions-to-spark (Accessed: 15 April 2021).https://stackoverflow.com/questions/21185092/apache-spark-map-vs-mappartitions

Multi-worker training with Keras | TensorFlow Core (2020) *TensorFlow*. Available at: https://www.tensorflow.org/tutorials/distribute/multi_worker_with_keras?hl=nb#choose_the_right_strategy (Accessed: 14 April 2021).https://blog.tensorflow.org/2020/12/getting-started-with-distributed-tensorflow-on-gcp.html

Distributed training with TensorFlow | TensorFlow Core (2020) *TensorFlow*. Available at: https://www.tensorflow.org/guide/distributed_training#multiworkermirroredstrategy (Accessed: 13 April 2021).

What is distributed training? - Azure Machine Learning (2020) *Docs.microsoft.com*. Available at: https://docs.microsoft.com/en-us/azure/machine-learning/concept-distributed-training#:~:text=In%20distributed%20training%20the%20workload,mini%20processors%2C%20called%20worker%20nodes.&text=Distributed%20training%20can%20be%20used,for%20training%20deep%20neural%20networks. (Accessed: 15 April 2021).

Smith, S., Kindermans, P., Ying, C. and Le, Q. (2018) "Don't Decay the Learning Rate, Increase the Batch Size", *ICLR*.

Alipourfard, O., Harry Liu, H., Chen, J., Venkataraman, S., Yu, M. and Zhang, M. (2017) *CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics*. 14th ed. Boston, MA, USA: Usenix: The advanced computing systems association, pp. 469 - 482. Available at: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard (Accessed: 17 April 2021).