Apr 20th, 12:00 AM - 12:00 AM

GPU Utilization: Predictive SARIMAX Time Series Analysis

Dorothy Dorie Parry
Old Dominion University

GPU UTILIZATION: PREDICTIVE SARIMAX TIME SERIES ANALYSIS

Dorothy (Dorie) Parry

Department of Electrical
and Computer Engineering
(Modeling and Simulation)
Old Dominion University
1318 Engr. & Comp. Sci. Bldg.
NORFOLK, VA, USA
dparr005@odu.edu

ABSTRACT

This work explores collecting performance metrics and leveraging the output for prediction on a memory-intensive parallel image classification algorithm - *Inception v3* (or *"Inception3"*). Experimental results were collected by nvidia-smi on a computational node DGX-1, equipped with eight Tesla V100 Graphic Processing Units (GPUs). Time series analysis was performed on the GPU utilization data taken, for multiple runs, of *Inception3's* image classification algorithm (see Figure 1). The time series model applied was Seasonal Autoregressive Integrated Moving Average Exogenous (*SARIMAX*).

Keywords:time series, machine learning, gpu utilization, performance measures

1 INTRODUCTION

Performance measures Collecting performance measures on an algorithm enables a better understand and leads investigations into implementing further optimizations. After understanding how an algorithm executes, metrics can be collected to track the improvements, or lack thereof of the proposed optimizations implemented. Nvidia-smi is a profiler that collects performance measures on each individual parallel GPU, such as memory, GPU utilization, power consumption, and GPU temperature. When picking an appropriate performance measuring tool, it is important to consider the limitations of data collected and any implications on the performance.

Nvidia-smi provides the best basic overview of the performance statistics for each individual parallel GPU while minimizing the effects on the data collected. Other performance profilers such as DLProf, PyProf, and NVIDIA Nsight Systems profiler provide more detailed statistics, such as wall GPU idle percentage and tensor core kernel efficiency that are not necessary to collect, unnecessarily increasing memory usage (Ethem Can and Chitale 2020). With an already memory-intensive algorithm, using a profiler that utilized memory would increase and introduce the possibility of a bottle neck that would skew the true GPU utilization values. Hence, collecting GPU utilization data herein employed nvidia-smi.

Forecasting models Forecasting models attempt to predict future values based on learned historical data trends. Forecasting models that take into account knowledge about the data provides more accurate results. One such category of forecasting models are those associated with time series. These models are applied to data that is taken at regular intervals and exhibit data trends.

SARIMAX model The *SARIMAX* model is defined as *(p, d, q) (P, D, Q, s)*. The variable *p* is the order for the non-seasonal Autoregressive (*AR*) component, *d* is for the non-seasonal differencing, *q* is the Moving Average (*MA*) order. The seasonal components are defined by the later variables: *P* is for the seasonal AR component, *D* is for the integration order, *Q* is for the *MA* seasonal order, and *s* is for the number of periods in season.

Two classes of predictive time series algorithms were applied - Autoregressive (*AR(p)*) and moving average (*MA(q)*) through the *SARIMAX* function. The data GPU utilization data was collected on different days and was concatenated together and separated by GPU (see Figure 1). The number of data points to predict was 1,800. This number is approximately the size of a single call to the algorithm.

The remainder of the paper consists of section 2, which consists of research similar to that of this paper. Section 3, describing the algorithm. Section 4 explains the time series analysis performed. Section 5 includes the results of the *SARIMAX* predictive algorithm.

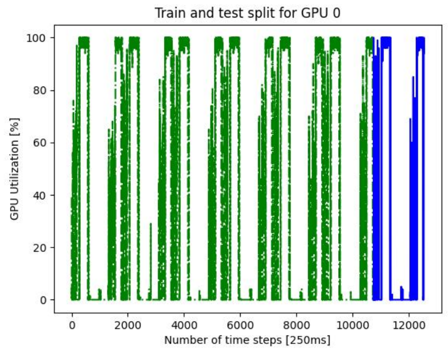

Figure 1: An example of the GPU utilization data (for GPU 0) split between training and test (number to predict is 1800).

2 RELATED WORKS

Predicting GPU utilization was previously studied by (Yeung) but for Deep Learning frameworks in the cloud. The research utilized a predictive engine that extracted metrics from its model computation graph. There was an RMSLE of 0.154 and achieved up to a 61.5% improvement to GPU cluster utilization. In (Sundriyal and Sosonkina 2022), an algorithm was proposed that would predict and then change the energy/power at runtime for an algorithm. The paper boasted a highest average energy savings of 17.7% with the lowest average performance loss of 5.1%. In (Sosonkina and Galvez Vallejo 2022), the continuance of (Sundriyal and Sosonkina 2022), a similar approach was taken - GPU utilization was modeled as time series and an Autoregressive Integrated Moving Average (ARIMA) process was used to dynamically predict the GPU utilization in the next timeslice to be executed. This paper utilized the predictions in each GPU to allocated power accordance to the predicted utilization.

3 INCEPTION V3 ALGORITHM

The "application," or "algorithm," that the performance measures are collected about is *Inception v3* (Labs 2021). This is an image classification algorithm. The call to the application includes a smaller batched run as well as a larger batched run with a larger number of epochs, hence why there are two similar data trends on the GPU utilization graphs (see Figure 2, which shows a singular call to the application). To best match the benchmarks provided from (Labs 2022) for the Tesla V100 SXM2-16GB, the following image settings were mirrored. The code used TensorFlow 1.15's Inception3 model on imagenet/synthetic data in training mode. Note that synthetic data is created using random pixel colors generated on the GPU memory (Labs 2021). Since the benchmark code results from (Labs 2022) utilized 16GB GPUs instead of the 32GB ones, it was postulated that the number of images per second should be larger for the DGX-1 node, and it was. Also, note that there is a warm up portion before the high utilization portion.

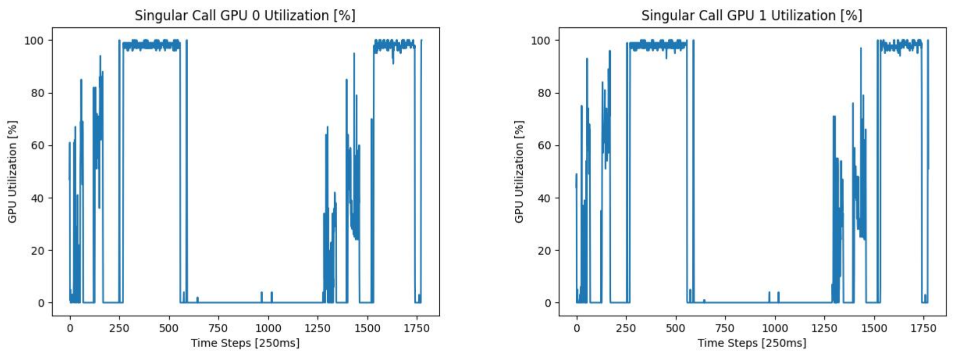

Figure 2: Left: An example of the GPU utilization (for GPU 0) for a singular call to the Inception3 algorithm. Right: An example of the GPU utilization (for GPU 1) for a singular call to the Inception3 algorithm.

Inception3 already employs an optimization - XLA (Accelerated Linear Algebra), which applies auto-jitter, enabling the TensorFlow's (TensorFlow is a python machine learning platform) algorithms to run faster. Using XLA is very useful for linear algebra intensive applications. XLA combines and computes linear algebra related tasks on a single GPU kernel. The biggest improvement that XLA provides relates to the memory bandwidth. Instead of writing intermediate computations to memory, it "streams" the information (Tensorflow 2022). In the second portion of the application execution, the batch size increases by a factor of 4 (batch size equating to 3,072) and the number of epochs by a factor of 4 as well, but the throughput, measured in images per second, only increases by a factor of 2.88 (2170.56/6258.27). Another possible explanation is because, as noted in (Tensorflow 2022), XLA's autoclustering is experimental for multiple GPUs.

Experimental investigation Initially, the memory clock rate for our GPUs were set to 0.24GHz with the throughput equating to 418.48 images per second, after changing the configuration files to reflect the same batch size as those provided by Lambda Labs (Labs 2022). These results were approximately five times slower than what was reported as benchmarks, 2130.15 images per second. This decrease in memory clock rate also led to less jitter. Though jitter amounts seem to affect the image per second rate, further research is required to draw conclusions about said relationship.

After the memory clock rate was increased to 1.53GHz, the images per second, matching the benchmark amounts, increased the throughput to 2170.56 images per second. This is a great improvement in throughput amount from when the memory clock rate was 0.24GHz compared to when it was increased to 1.53GHz, 418.48 compared to 2170.56 images per second. To further investigate the throughput, the algorithm was run again with a batch size of 1536 (192 per GPU), with the same memory clock rate, but a doubled number of epochs. The throughput was 2254.21 images per second. Further investigation will need to be made, but preliminary analysis leads to the conclusion that the amount of memory (when the batch sizes are less than 1,536) does not affect the throughput speed a significant amount, but decreasing the clock memory rate does.

4 TIME SERIES FORECASTING

The GPU utilization data, collected using nvidia-smi, is a strong candidate for being time series because the data is taken at regular, equally spaced time intervals. The GPU utilization data was collected every 250ms. However, since there are a couple moments in time where the data trends change, it was appropriate to concatenate multiple runs of the algorithm. This was done so that there was more data to learn on. To deduce an appropriate forecasting method to apply to the time series data, use the decision sequence explained in Section 4.1, as suggested in (Peixeiro). The GPU utilization data was very similar, but for simplicity sake, only graphs for GPU 0 and GPU 1 will be included.

4.1 Decision Sequence

First, determine whether the data is stationary by using the augmented Dickey-Fuller (ADF) test. The results of this test - finding the presence of a unit root - determines whether a differencing transformation should be applied to make the data stationary. In other words, stabilizing the mean and variance. Making the data stationary makes predictions easier, since the mean, variance, and autocorrelation does not change over time. Without these assumptions, forecasting will be unreliable (Peixeiro). Once the data is stationary, the augmented Dickey-Fuller (ADF) test results in a p-value that is below 0.05 (for each GPU), therefore rejecting the null hypothesis that the series is non-stationary (Peixeiro).

Next, check if there is any linear temporal correlation (autocorrelation). Check if values with a certain amount of lag linearly correlate to future values. This analysis is done by studying the autocorrelated graphs. Significant autocorrelation coefficients are above or below the significance level of 95% on the plot delimited by the blue area (Figure 3). So the correlated lag values above or below the significance area are significantly different than zero.

If there is autocorrelation, there is a lag value after which there is an abrupt change to non-significance (Peixeiro). Autocorrelation plots where the plot slowly decays as the lag amount increases, as seen in GPU utilization graphs (Figure 3) is not indicative of a random walk, but rather tend to be auto-regressive. It is not indicative of a random walk because of the continuation of significant values after a lag value of zero. To find the order, p of an auto-regressive ($AR(p)$) series, plot a partial autocorrelation function.

Partial Autocorrelation measures the autocorrelation between the value at t and $t-1$, so it measures the true impact between the values when removing the influence of other correlated lagged values (so values past $t-1$).

$$y_t = \phi y_{t-1} + \phi y_{t-2} + \phi y_{t-3} + \dots + \varepsilon_t$$

Where ε_t is white noise (Hyndman and Athanasopoulos 2018). The order is defined as the lag amount at which there is an abrupt change from significantly to non-significantly different from zero.

If there was an abrupt change to non-significance after lag q, then use a moving average predictive process ($MA(q)$). The order of a moving average model corresponds to the number of values in the past that linearly affect the current value.

4.2 Autoregressive and Moving Average Time Series Algorithms

For the p-values resulting from the augmented Dickey-Fuller (ADF) test that were below 0.05 for each GPU (see Table 1), the data was determined to be stationary.

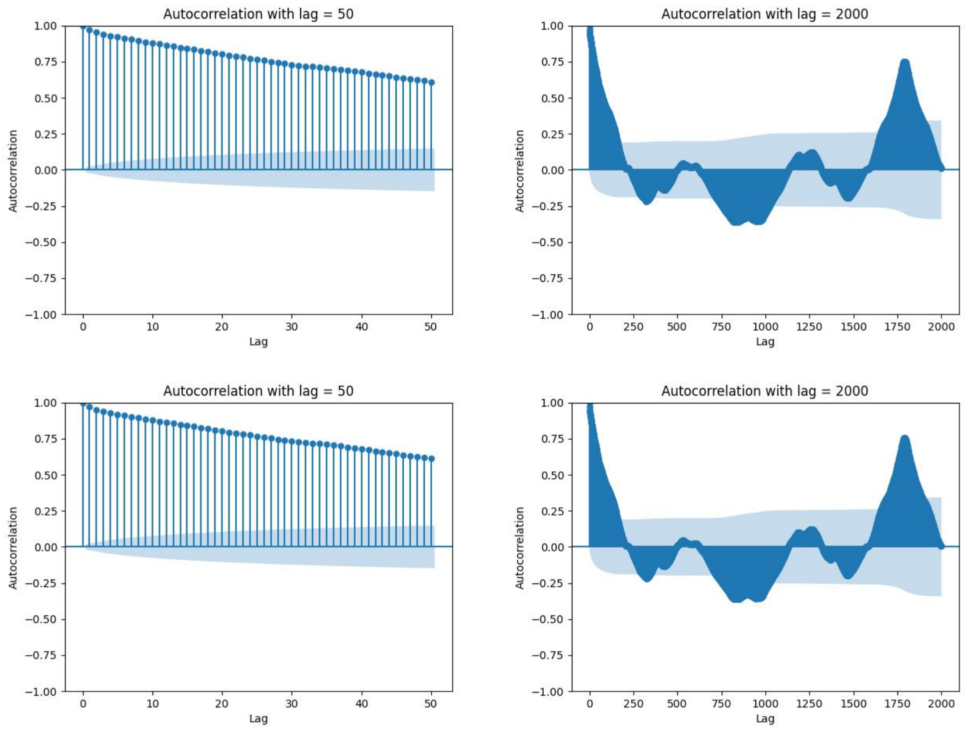

Figure 3: Top left: Autocorrelation graph for GPU 0 with lag up to 50. Top right: Autocorrelation graph for GPU 0 with lags up to 2000. Bottom left: Autocorrelation graph for GPU 1 with lag up to 50. Bottom right: Autocorrelation graph for GPU 1 with lag up to 2000.

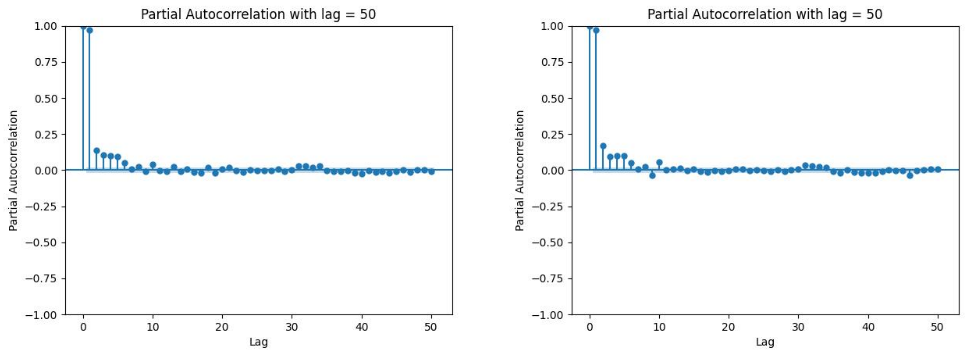

Figure 4: Left: Partial Autocorrelation graph for GPU 0 with lag up to 50. Right: Partial Autocorrelation graph for GPU 1 with lags up to 50.

Table 1: P-values from ADF Test with the second row equating to the number of lags used.

GPU 0	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7
1.417e-07	2.105e-08	1.0627e-08	1.374e-08	1.929e-08	5.8108e-9	5.408e-10	3.7498e-08
33	39	39	39	38	38	5	37

5 EXPERIMENTAL RESULTS

All of the eight GPUs were predicted using a version of both Autoregressive (AR) and Moving Average (MA) predictive algorithms. Both AR and MA model components can be combined in a *SARIMAX* model, which was from *statsmodels.tsa.statespace.sarimax* (Josef Perktold 2023). To find the hyperparameters, or orders for *SARIMAX*, both *arselectorder* function statsmodels.tsa.armodel and *autoarima* from pmdarima.arima were used (Josef Perktold 2023) (Smith and Others 2022).

5.1 Autoregressive and Moving Average Predictive Algorithms

For GPU 0, the *arselectorder* function found the best model to be *AR(2).* and for GPU 1, *AR(10).*

For GPU 0, the best model according to *auto-arima* searched from *p = [1, 15], q = [1, 15], d = None*. The model with the lowest Akaike Information Criterion (AIC) without seasonality was *SARIMAX(9,0,3)* with an AIC score of 79945. This very high AIC score is also evidence that these models will not be optimal for prediction, since the best models will have a low AIC score.

For GPU 0, the best model according to *auto-arima* searched from *p = [1, 15], q = [1, 15], d = None, seasonal=True, m=[2,10].* The prediction errors are very similar for the best models when the period for seasonal differencing *m* was between [2, 10]. This is due to the AIC scores being close to 80,000 (plus or minus less than 1,000) for each model. This very high AIC score is also evidence that these models will not be optimal for prediction, since the best models will have a low AIC score. One of the best model was found to be *SARIMAX(5,0,1)(0,0,1)[7]* with an AIC score of 79966.636.

For GPU 1, the best model according to *auto-arima* searched from *p = [1, 15], q = [1, 15], d = None*. The model with the lowest Akaike Information Criterion (AIC) without seasonality was *SARIMAX(10,0,1)* with an AIC score of 80470.413. This very high AIC score is also evidence that these models will not be optimal for prediction, since the best models will have a low AIC score.

For GPU 1, the best model according to *auto-arima* searched from *p = [1, 15], q = [1, 15], d = None, seasonal=True, m=[2,10].* The prediction errors are very similar for the best models when the period for seasonal differencing *m* was between [2, 10]. This is due to the AIC scores being close to 80,000 (plus or minus less than 1,000) for each model. This very high AIC score is also evidence that these models will not be optimal for prediction, since the best models will have a low AIC score. One of the best model was found to be *SARIMAX(5,0,1)(0,0,1)[7]* with an AIC score of 79966.636.

For the *arselectorder*, the maxLag was set to 100, the seasonal flag was set to both true and false, and the period was varied from *[2, 30].* The models with either a lower AIC score, or lower maximum error were saved were determined to be more optimal. For GPU 0, *AR(2)* was the best model for when there was no seasonality, and when there was seasonality, the periods were best when they were 7, 9, 17, and 21. For GPU 1, the best model was *AR(10)* with no seasonality, and with seasonal periods of 2, 7, 15, 21, and 35. The prediction error was graphed, see Figure 5.

For both algorithms, the most accurate segment of the forecast prediction was the two portions of the graphs where the GPU utilization was above 90%. Also, prediction was less accurate for the secondary portion, when the number of epochs and batch size increases. This could be due to the predictive algorithms not

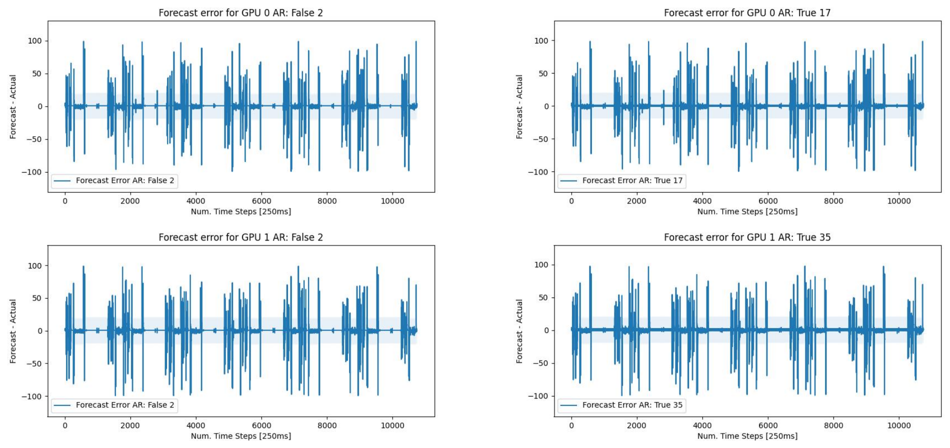

Figure 5: Top left: Forecasting error for GPU 0 *AR(2)* with no seasonality. Top right: Forecasting error for GPU 0 *AR(2)* with seasonal period of 17. Bottom left: Forecasting error for GPU 1 *AR(10)* with no seasonality. Bottom Right: Forecasting error for GPU 1 *AR(10)* with seasonal period of 35.

being able to capture the effects of bottle necking caused by the more memory-intensive portion of the algorithm run.

The predictions were not as optimal as hoped. This is due to several possible reasons. One is the irregular jumps in the time series patterns. As noted in Section 6, there might be ways of circumventing this problem. In addition, expanding the possible search space might produce better results.

6 FUTURE WORKS

Another interesting predictive algorithm to explore is the rolling versions of *SARIMAX*. The rolling versions of the algorithm utilize a rolling window. A window limits the amount the past values that the algorithm can learn on. The rolling window allows for the window to shift.

Another interesting predictive algorithm to explore is Transformer Neural Networks. Transformer Neural Networks (TNN) optimally solve problems relating to computer vision, natural language processing, and speech processing. TNN's are especially successful for learning sequential data, since it holds dependencies and interactions within memory. The success of TNNs for sequentially correlated data suggests that TNNs are a good candidate for predicting time series data.

Similar to the *SARIMAX*, the hyperparameters can really affect the results. These hyperparameters include the encoder and decoder length, number of epochs, proportion of test data, and the batch size. Other hyper-parameters exist that can be overwritten as well. Given the length of the data set, it will take a while to find the correct hyperparameters needed to produce optimal predictions.

Another possible improvement that will be explored is change point detection or piecewise time series analysis. These methods should help with separating the two different runs of the algorithm (one where the number of epochs is lower, and one where it is increased).

7 ACKNOWLEDGEMENT

This word was support by my advisor, Dr. Sosonkina. This work was also supported in part by the U.S. Department of Energy (DOE) Exascale Computing Project (ECP) through the Ames Laboratory, operated by Iowa State University under contract No. DE-AC00-07CH11358, in part by the Old Dominion University

Data Science Seed Funding 2022 Award and in part by the Old Dominion University Research and Cloud Computing.

REFERENCES

Ethem Can, Rajan Arora and Poonam Chitale 2020. “Profiling and Optimizing Deep Neural Networks with DLProf and PyProf”.

Hyndman, R. J., and G. Athanasopoulos. 2018. *Forcasting: Principles and Practice*. Melbourne, Australia, Texts.com/fpp2.

Josef Perktold, Skipper Seabold, Jonathan Taylor 2023. “statsmodels.tsa.statespace.sarimax.SARIMAX”.

Lambda Labs 2021. “lambda-tensorflow-benchmark”.

Lambda Labs 2022. “Deep Learning GPU Benchmarks”.

Peixeiro, M. *Time Series Forecasting in Python*. Manning.

Taylor G. Smith and Others 2017-2022. “pmdarima.arima.auto$_{a}rima''$.

Sosonkina, M., Sundriyal, V. and Galvez Vallejo, J.L. 2022. “Runtime Power Allocation Based on Multi-GPU Utilization in GAMESS”.

Sundriyal, V., and M. Sosonkina. 2022. “Runtime Energy Savings Based on Machine Learning Models for Multicore Applications”. *Journal of Computer and Communications* vol. 10, pp. 63–80.

Tensorflow 2022. “XLA: Optimizing Compiler for Machine Learning”.

Yeung, G. “Towards GPU Utilization Prediction for Cloud Deep Learning”.