Analyzing the trade-off in speed and accuracy of PCA in FP16 vs FP32 precision

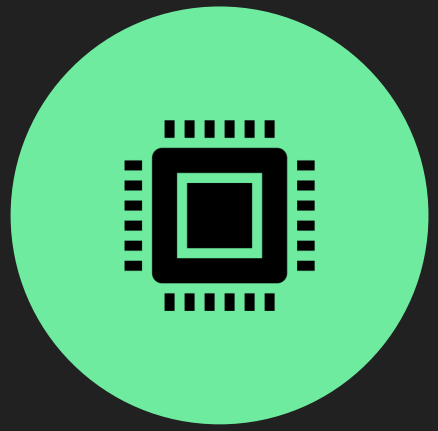
Akash Palrecha

2017B4PS0559P

Why is this problem Important?

- According to a study by OpenAI, the amount of compute needed for AI research has been doubling every 3.5 months since 2012.
- In industry, companies use terabytes of data every day to train ML models. The data pipeline uses PCA as a primary step to reduce the dimensions of the data before training the ML models.
- Server costs per hour of usage (AWS, GCP, etc.) are at an all time high and it's important to reduce the number of hours used for heavy compute tasks. (server costs just for this project are close to INR 7,000)
- Data with millions/billions of rows and thousands of columns is commonplace in industry ML applications and it is very common to use PCA to remove redundant features/columns.

What's the advantage of FP16?



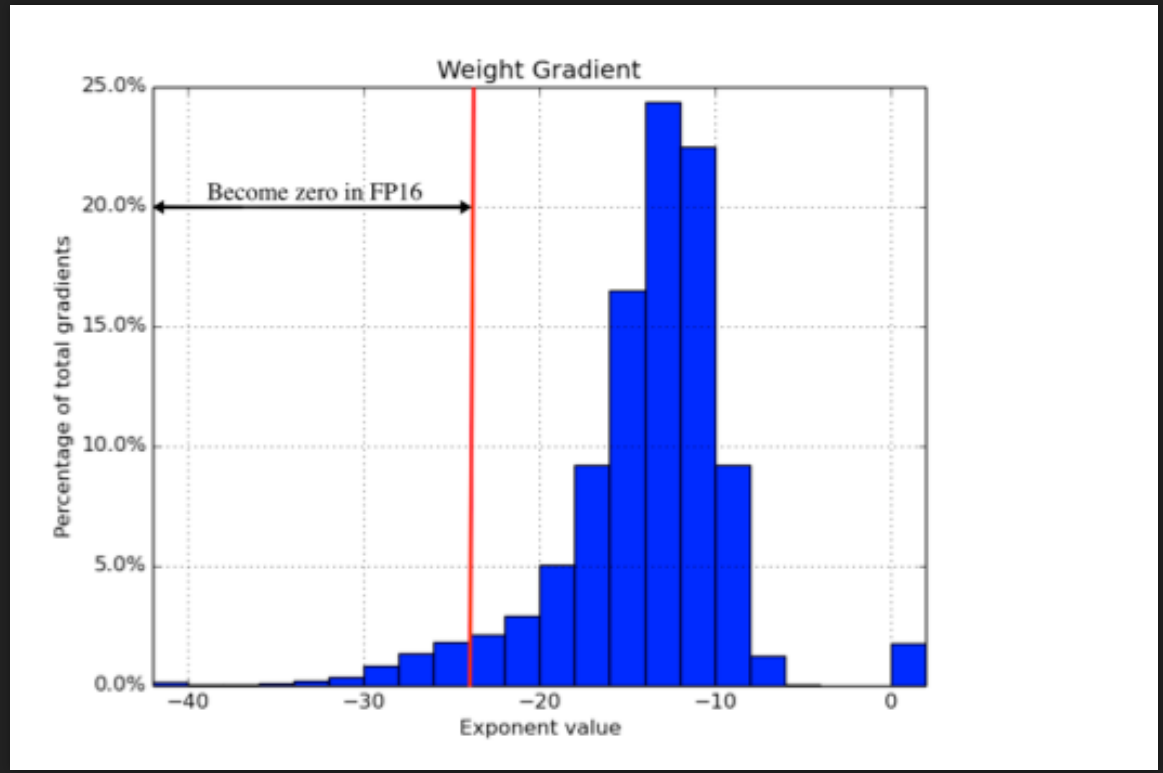
PERFORMING COMPUTATIONS ON GPUS IN FP16 PRECISION ALLOWS ANYWHERE BETWEEN 2X-6X SPEEDUPS OVER FP32 COMPUTATIONS DEPENDING ON THE GPU BEING USED.

A SPEED-UP OF UP TO ***4X WAS ACHIEVED*** IN THE EXPERIMENTS DONE FOR THIS PROJECT

Background on FP16 Computations

Mixed Precision Training: ***Micikevicius et al*** (NVIDIA)
https://arxiv.org/pdf/1710.03740.pdf

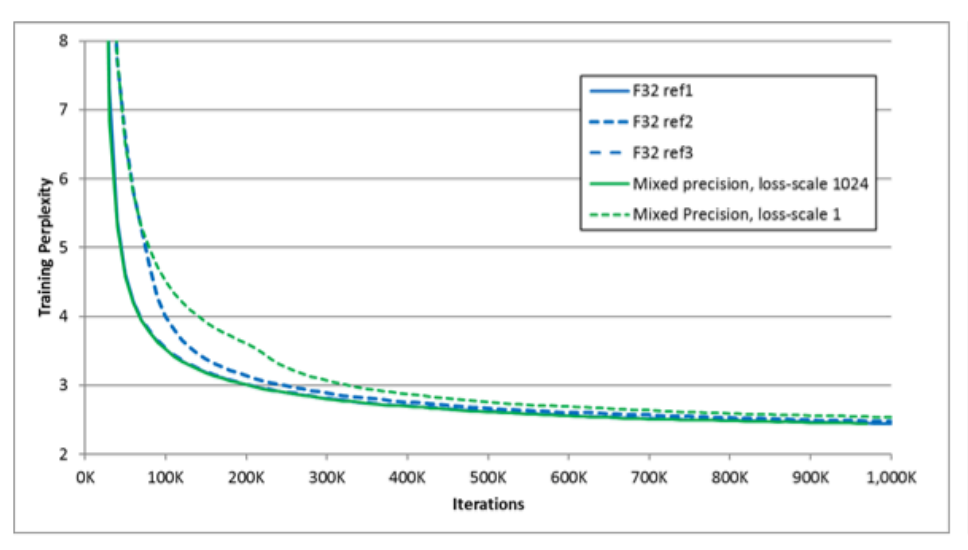
- The graph on the right shows that majority of the gradients during ML training lie in a range of very small magnitude.
- A small percentage of gradients lie in the "Zero" range of FP16 precision



Background on FP16 Computations

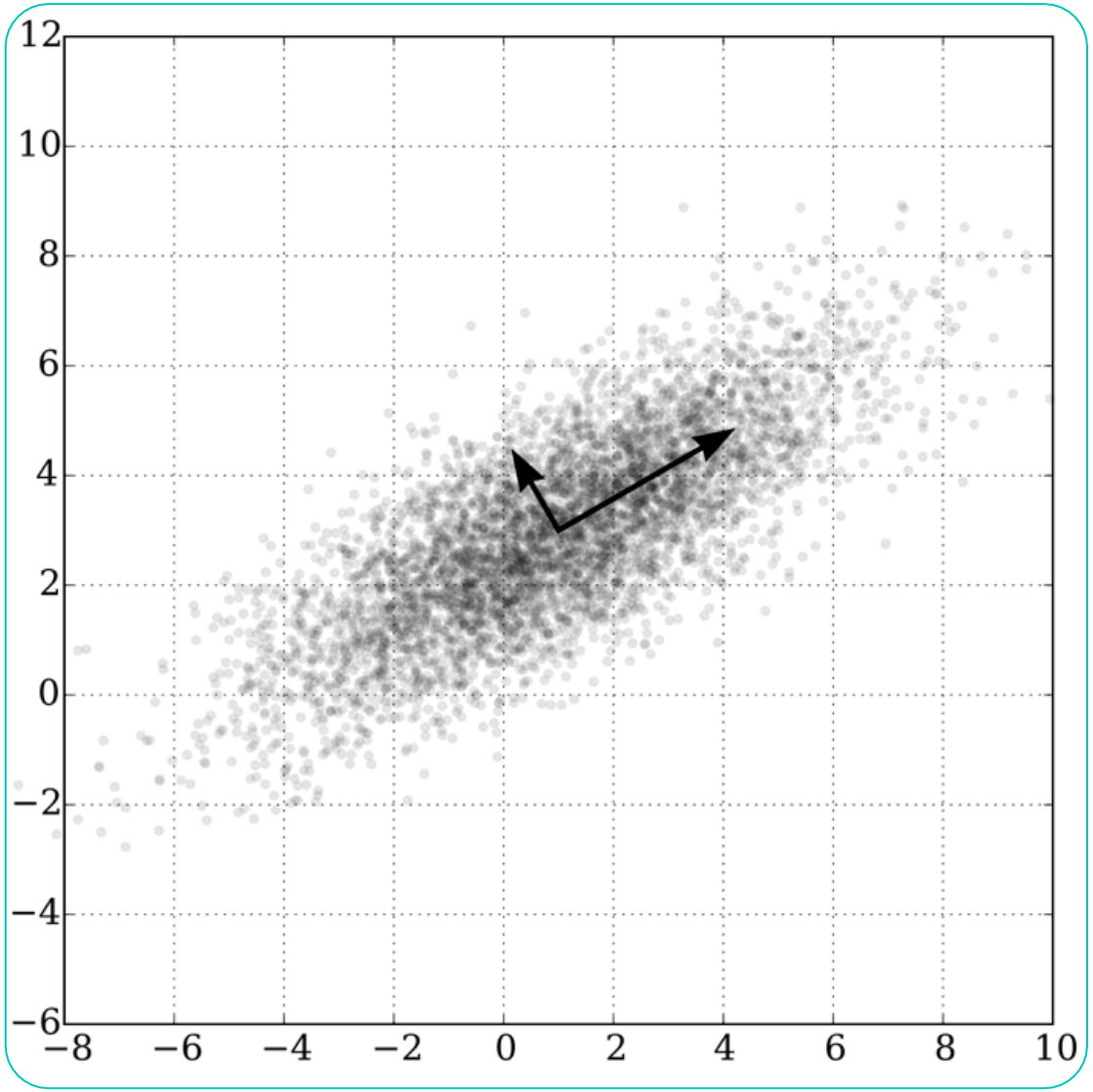
Mixed Precision Training: ***Micikevicius et al*** (NVIDIA)
https://arxiv.org/pdf/1710.03740.pdf

- This graph shows that training models with mixed precision tends to give pretty much the same results



What is PCA?

- Also known as Principal Component Analysis, PCA is a popular dimensionality reduction technique for matrices while maintaining as much variance as possible in the reduced data.



How does PCA Work?

- Reducing a matrix M from N to K dimensions using PCA involves three broad steps:
  1. Calculating the covariance matrix C of the columns of M
  2. Getting the top K eigenvectors (sorted by eigen values) of C
  3. Transforming M into the space defined by the above eigenvectors by a simple matrix multiplication.

Step 1: Calculating the covariance matrix C

- Let M be our data matrix of size $L \times N$
- $X = M - M^{\circ}$
  - where M° is a column vector of row-wise means of M
  - This step is necessary to center the matrix around 0
- $C = \frac{(X^T \times X)}{L - 1}$
  - We divide by $L - 1$ to get an unbiased estimate.
- C is our required symmetric covariance matrix of M of size $N \times N$

Step 2: Getting the top K eigenvectors of C

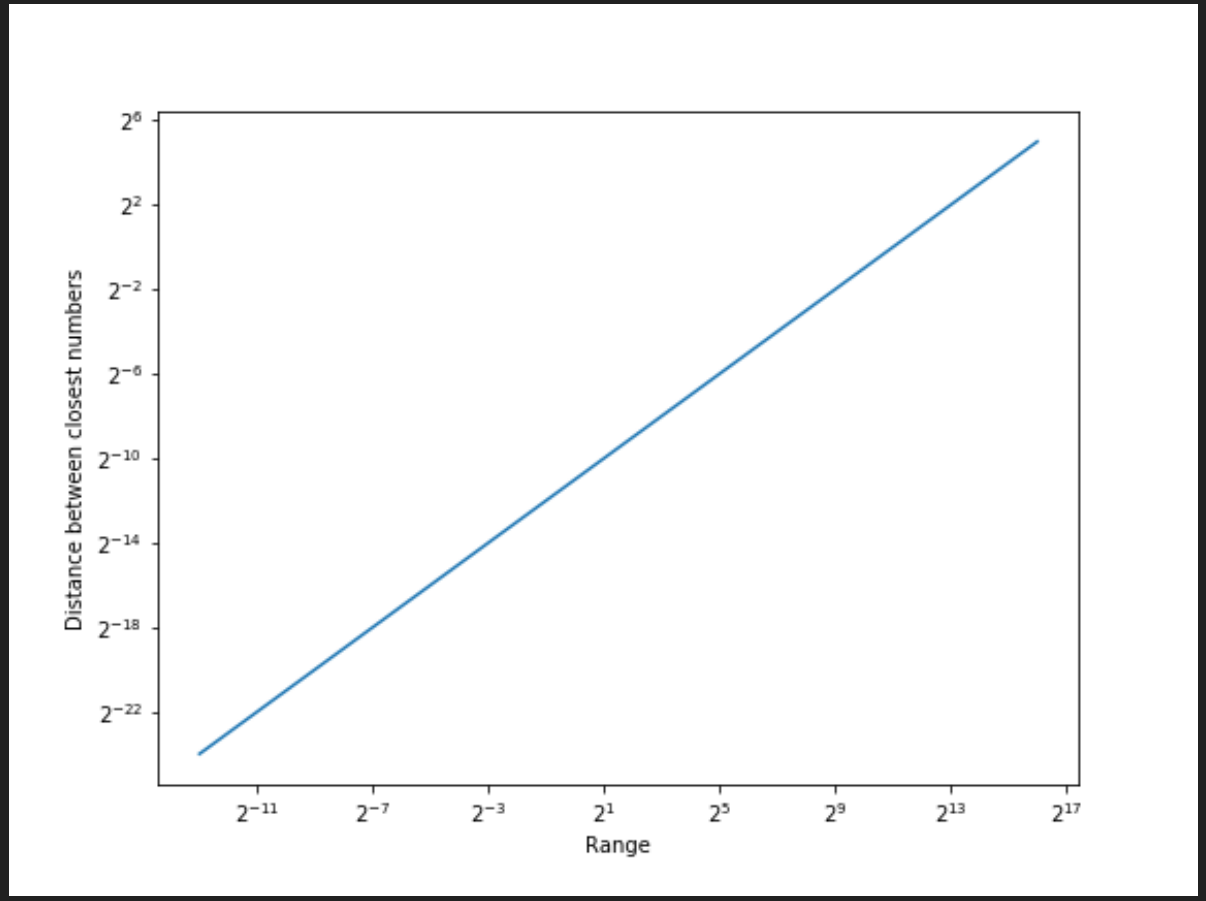
- First, we obtain eigen values by solving the below equation to obtain λ:
- $(C - \lambda I) = 0$; where λ represents the column vector of eigen values.
- Solve the below to obtain eigen vectors associated with the top K eigen values by magnitude:
- $(C - \lambda_i)V_i = 0 \; \forall \; i \leq K$
- Form a matrix V of size $N \times K$ where V_i is the eigen vector corresponding to the i^{th} largest eigen value.

Step 3: Reducing the matrix M

- This is the simplest step. We simply perform:
- $M = M \times V$
- M is now a reduced matrix of size $L \times K$ which was previously $L \times N$

But PCA isn't the problem, FP16 is!

- After a certain range, numbers tend to become extremely inaccurate when working in floating point 16 precision.
- The closer to 0, the better.
- The distance between any 2 representable numbers in FP16 can go up to 32 in the range from 2^{15} to 2^{16}
- This graph shows the distance between the 2 representable numbers in FP16 format as we go up the range.
- Everything above 65519 is rounded to infinity



Fortunately…

COMPUTATIONS ARE STILL VERY ACCURATE AS LONG AS WE ARE IN THE 0 TO 1 RANGE.

WE CAN SIMPLY SCALE DOWN DATA THAT HAS TOO MUCH VARIANCE AND PERFORM ACCURATE CALCULATIONS IN THE 0-1RANGE.

Method

PCA

- Language: Python
- Computation Framework: PyTorch
- Both the FP16 and FP32 versions of PCA were written from scratch to work on GPUs. No off-the-shelf GPU versions of PCA were available.
- Extensive experiments were performed with over 1600 combinations of different gaussian data distributions each repeated over 50 times for accurate readings.
- For each experiment, relative and absolute errors were recorded along with the time of execution in seconds.

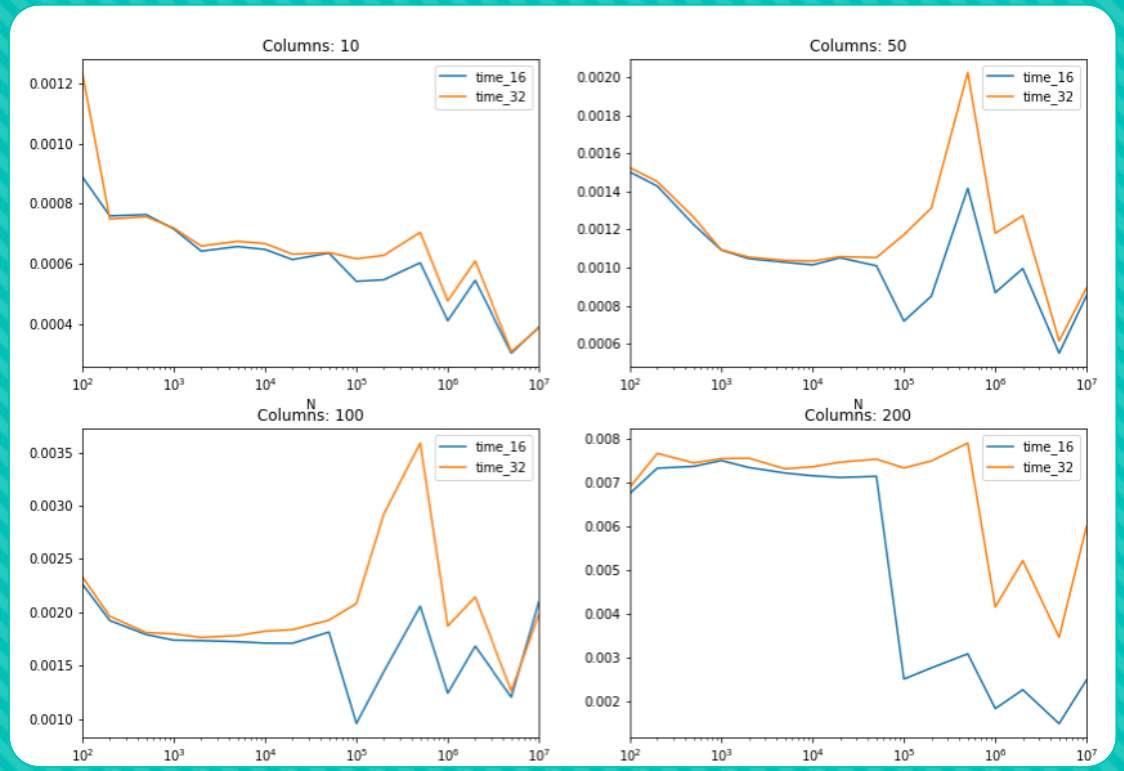
Data permutations:

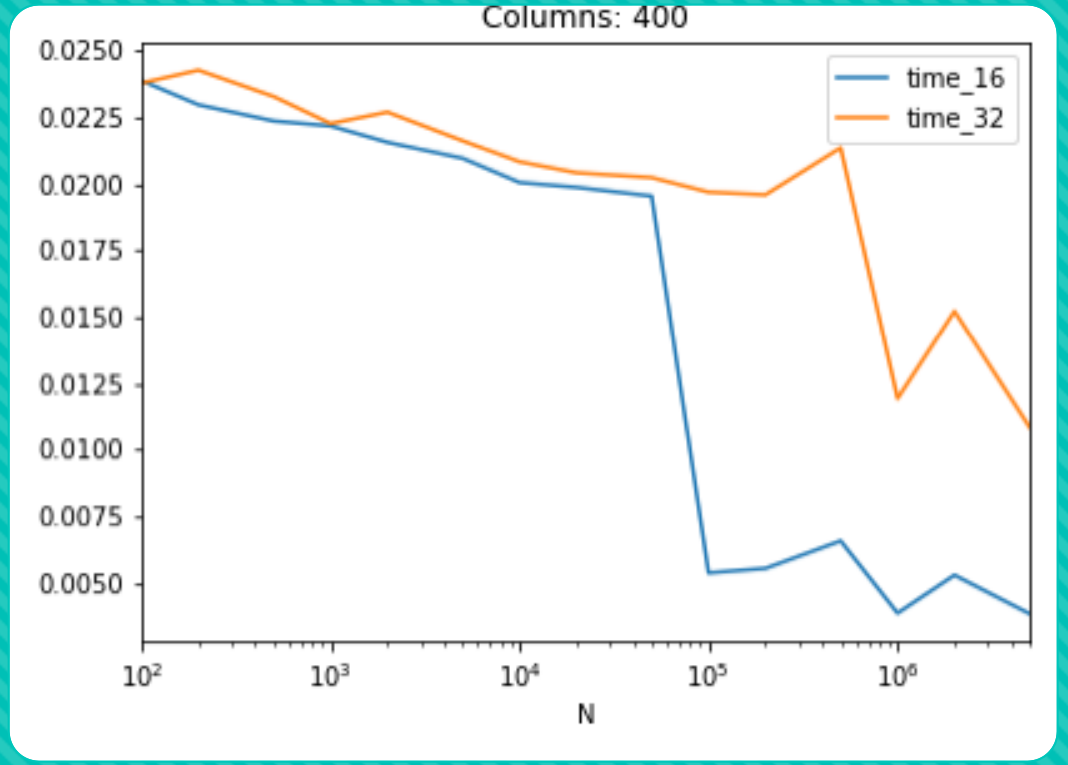
- Number of rows: [100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000, 500000, 1000000, 2000000 ,5000000, 10000000]
- Number of columns: [10, 50, 100, 200, 400]
- Mean and variance pairs: [[0, 1], [0, 4], [0, 16], [0, 64], [0, 128], [0, 512], [0, 2048], [0, 8192], [0, 32768], [0, 65519]]
- Scaled and non-scaled versions for each experiment both in FP16 and FP32 precision.
- Repeated all experiments 50 times to get accurate time recordings
- Total experiments: 16 x 5 x 10 x 2 x 2 x 50 = 160,000 experiments

Hardware

- EC2 Instance on AWS
- GPU: NVIDIA V100 16 GB Server Version
- RAM: 64 GB
- CPUs: 8 i7 Skylake
- Cost: $3/hour (over 30 hours used)

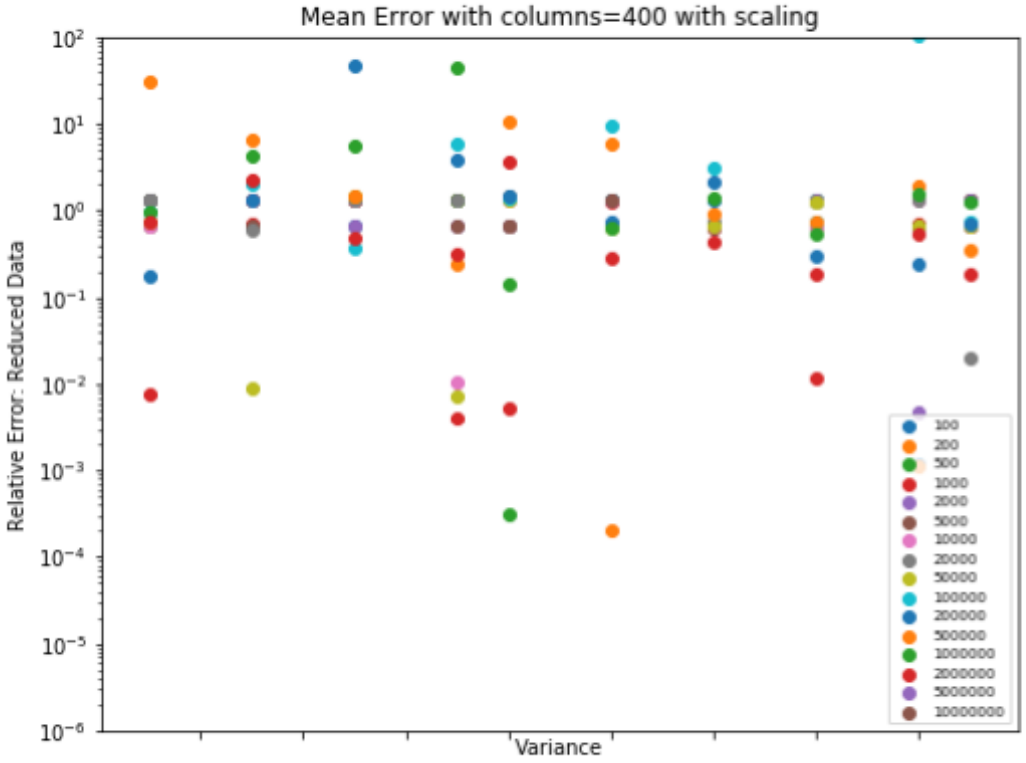
Results

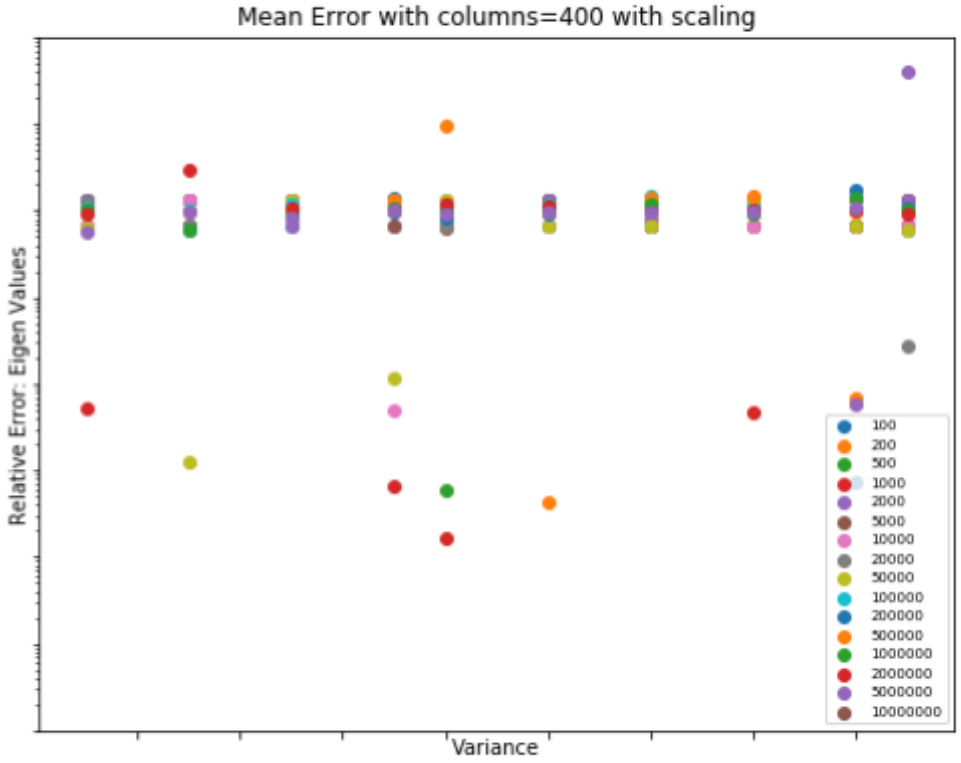




Gains in Speed over FP32:
visible consistently after 200 columns or more

(Reported in seconds)

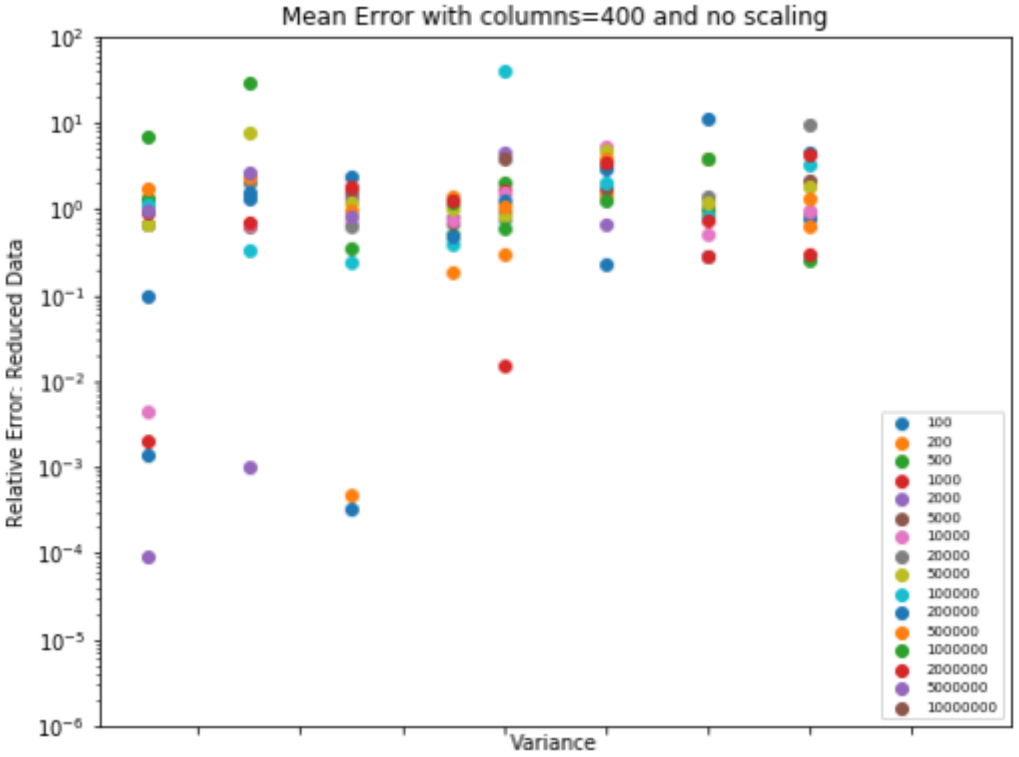


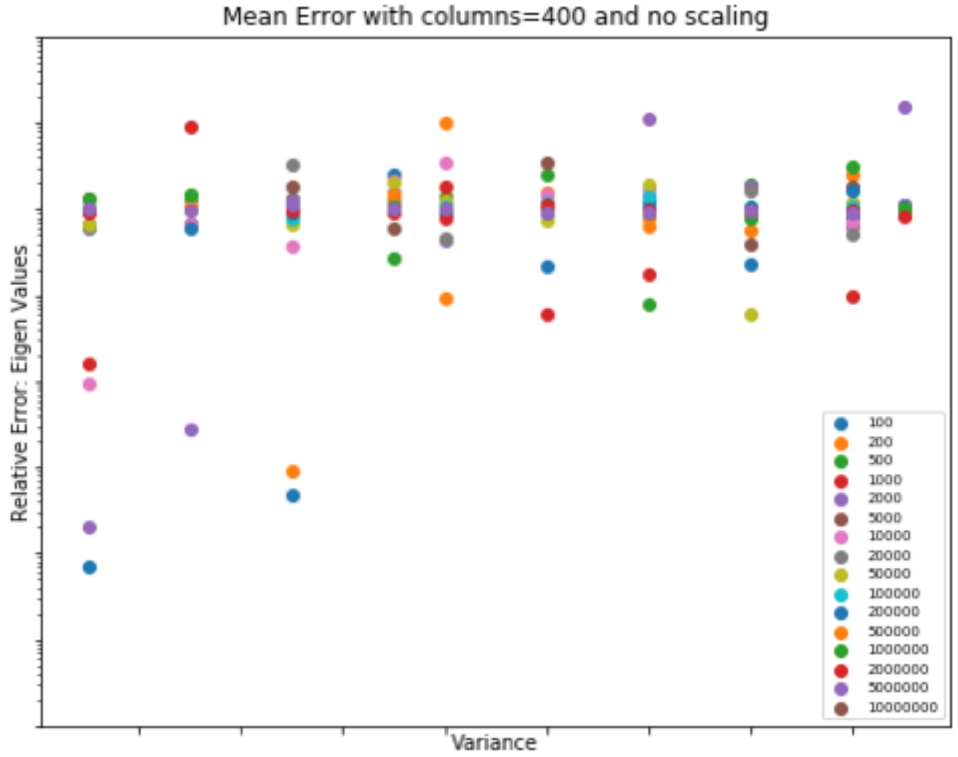


Relative errors for scaled data

(as compared to PCA with FP32 precision)

Reported for both reduced data and eigen values of covariance matrix

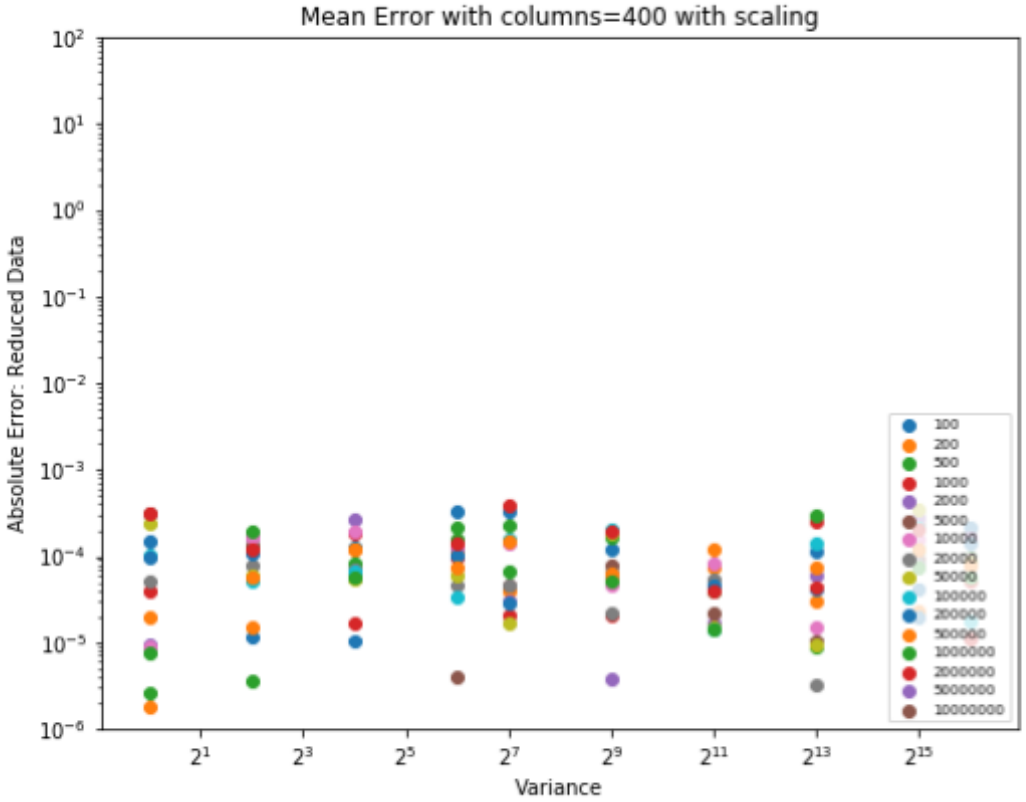


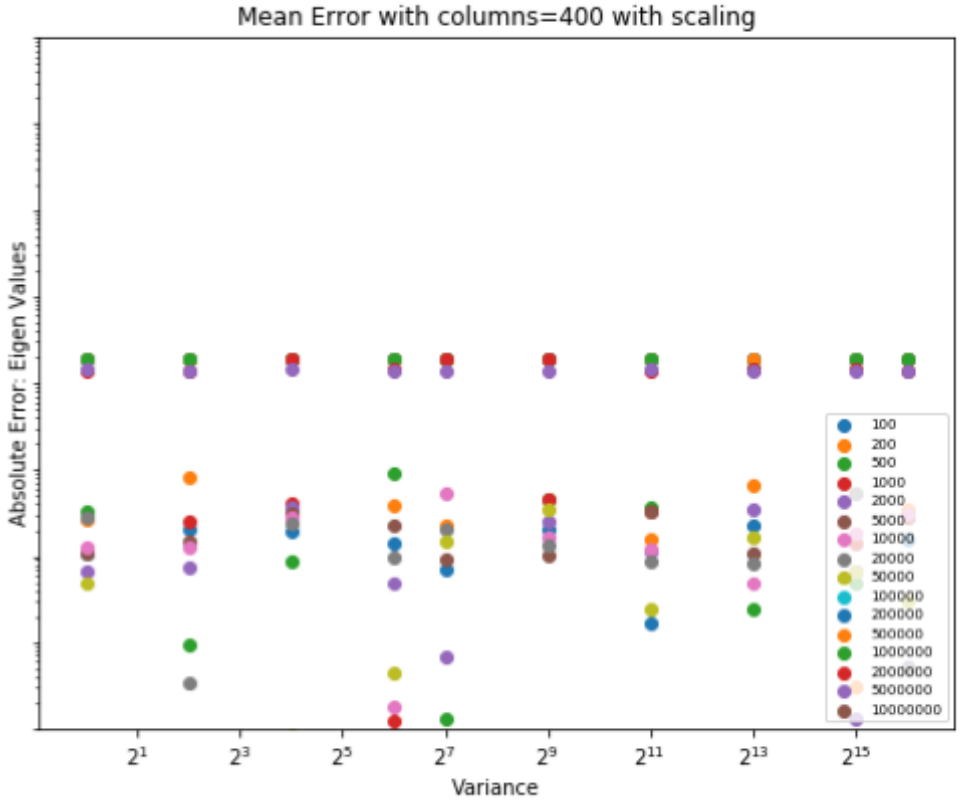


Relative errors for non-scaled data

(as compared to PCA with FP32 precision)

Reported for both reduced data and eigen values of covariance matrix

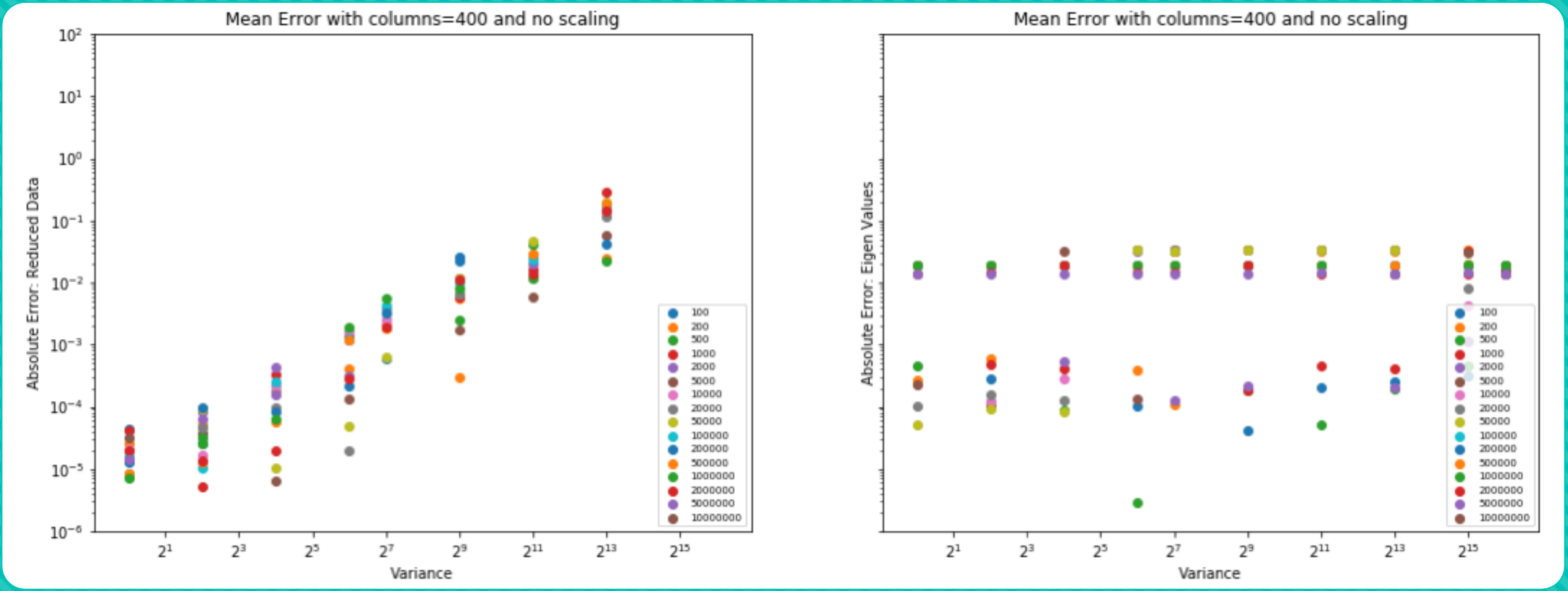




Absolute errors for scaled data

(as compared to PCA with FP32 precision)

Reported for both reduced data and eigen values of covariance matrix

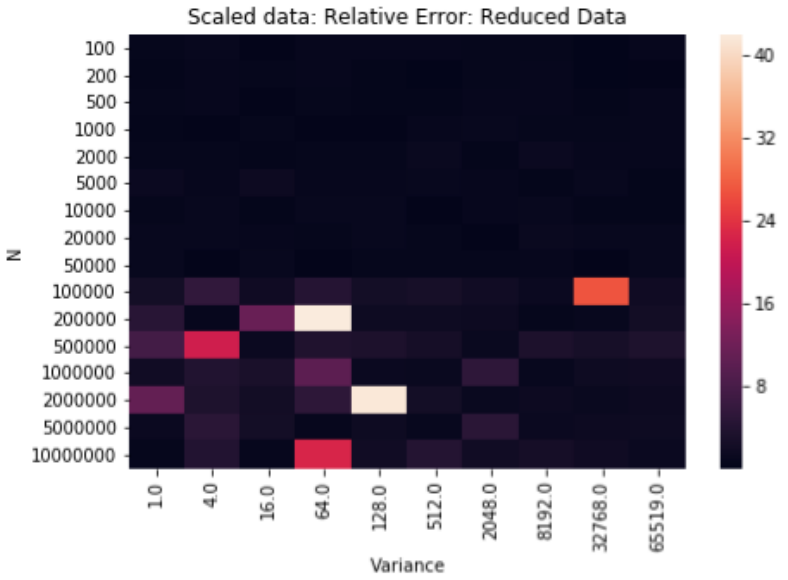


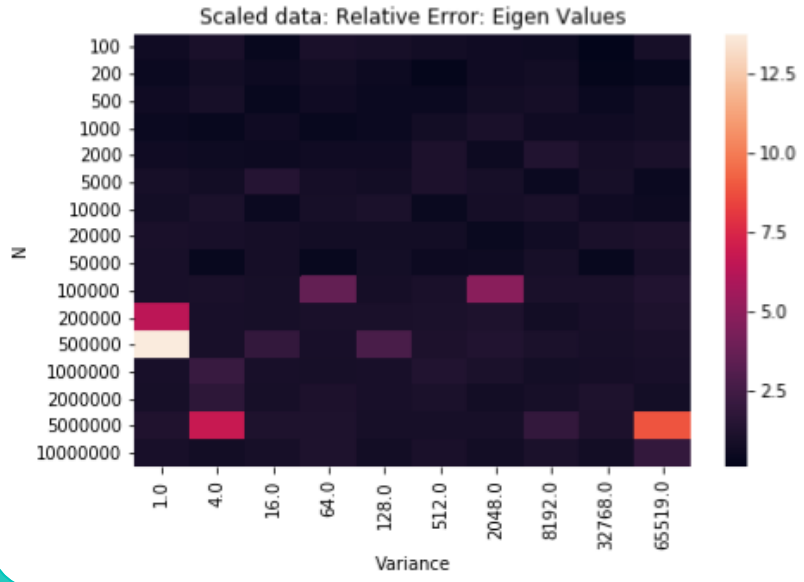
Absolute errors for non-scaled data

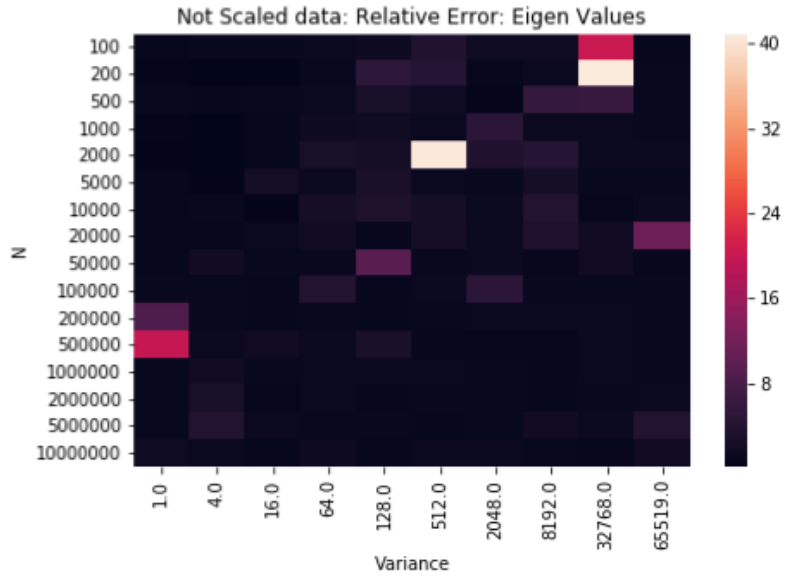
(as compared to PCA with FP32 precision)

Reported for both reduced data and eigen values of covariance matrix

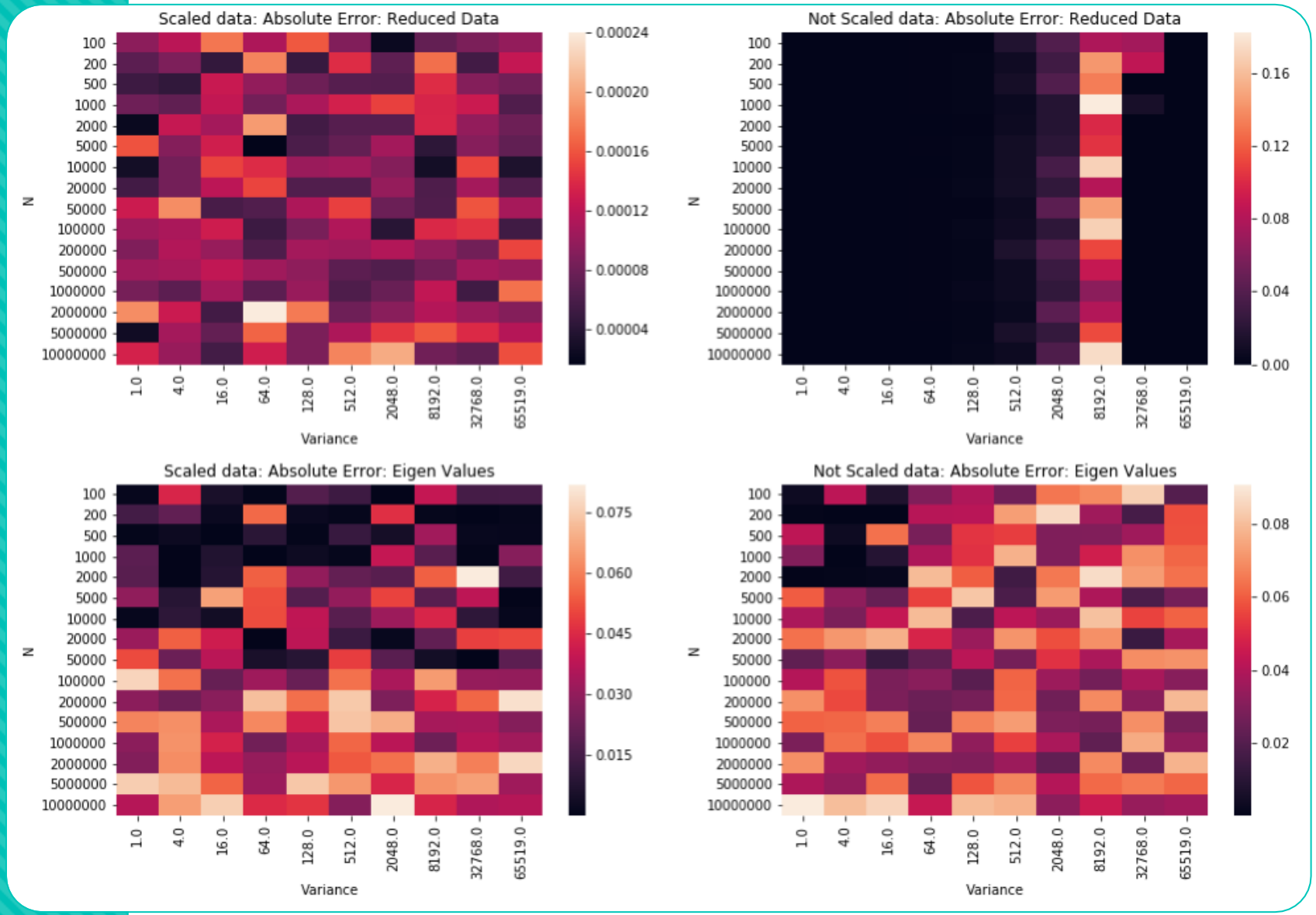
Heatmap of errors: relative







Heatmap of errors: absolute



Issues

- When calculating the covariance
- The lowest representable number in FP16 is 2^{-24} (~0.000000059604645). Any numbers below this becomes 0
- When N is too large ($> 1000{,}000$) we can have issues where a lot of numbers after division will round to 0.
- Also, in the matrix multiplication, numbers that go above 65519 will round to infinity.

Workarounds

- In their paper about Mixed Precision training, NVIDIA suggests shifting numbers by a few bits by multiplying them with a power of 2 such as 128 or 1024 before the division.
- We can scale down values by the same factor once we have our final result converted back to the FP32 type for storage/further processing.

Some observations

EXPERIMENTS SUGGEST THAT AS LONG AS THE DATA LIES IN THE 0-1 RANGE, IT SHOULD BE POSSIBLE TO SAFELY PERFORM PCA WITH MINIMAL ERROR RATES EVEN WITH A 10 MILLION ROWS OF DATA.

SPEEDUPS FROM FP16 CALCULATIONS ARE SIGNIFICANTLY NOTICEABLE WHEN WE HAVE MORE THAN 100,000 ROWS OF DATA.

FOR DATA THAT IS NOT IN THE CORRECT RANGE, SCALING UP/DOWN IS ALWAYS AN OPTION.

Conclusions & Remarks

- As hypothesized, performing PCA in FP16 precision give us not merely incremental, but manifold gains in computation time.
- As FP16 numbers take up half as memory as FP32 numbers, it can allow us process datasets that are double the size of what was previously possible.
- The errors that caused by low precision are generally less than 1% which suffices for fault tolerant applications.

Conclusions & Remarks

- The speed gains are only noticeable once we have more than 200 columns and more than 100,000 rows in our matrix.
- Scaling the matrix to the 0-1 range significantly reduces errors (avoids zero and infinity values)
- These advantages are not supported by all kinds of hardware as of now. Only some newer NVIDIA GPUs support such speed-ups.

Thank you!