How Walmart Improves Forecast Accuracy with NVIDIA GPUs

Smart Forecasting

March 19, 2019
Agenda

❖ Walmart’s forecasting problem

❖ Initial (non-GPU) approach
  ❖ Algorithms
  ❖ Pipeline

❖ Integrating GPUs into every aspect of the solution
  ❖ History cleansing
  ❖ Feature engineering
  ❖ Off-the-shelf algorithms
  ❖ In-house algorithms

❖ Benefits – translating speed into forecast accuracy
Walmart

- Over $500B annual sales (over $330B in the U.S.)
- Over 11,000 stores worldwide (over 4700 stores in the U.S.)
- Over 90% of the population in the U.S. lives within 10 miles of a Walmart store
- The largest grocer in the U.S.
- The largest commercial producer of solar power in the U.S.
Problem description

• Short-term: forecast weekly demand for all item x store combinations in the U. S.
  – Purpose:
    • Inventory control (short horizons, e.g., 0-3 weeks)
    • Purchase / vendor production planning (longer horizons)
  – Scope:
    • Size: 500M item x store combinations
    • Forecast horizon: 0 – 52 weeks
    • Frequency: every week

• Longer term: forecast daily demand for everything, everywhere.

• Pipeline constraints
  – Approximately 12 hour window to perform all forecasting (scoring) tasks
  – Approximately 3 days to perform all training tasks
Pre-existing system

- COTS (Commercial Off The Shelf) solution integrated with Walmart replenishment and other downstream systems

- Uses Lewandowski (Holt-Winters with “secret sauce” added) to forecast U.S.-wide sales on a weekly basis

- Forecasts are then allocated down to the store level

- Works quite well – it beat three out of four external vendor solutions in out-of-sample testing during our RFP for a new forecasting system

- ... still used for about 80% of store-item combinations, expect to be fully replaced by end of the year.
History cleansing

• Most machine learning algorithms are not robust in a formal sense, resulting in:

  Garbage in, garbage out

• Three approaches:
  – Build robust ML algorithms (best)
  – Clean the data before giving it to the non-robust ML algorithms that exist today
  – Hope that your data is better than everyone else’s data (worst)

• We’ve taken the second approach, but are thinking about the first.
Identifying outliers using robust time series – U. S. Romaine sales

**Romaine lettuce (3 outliers added)**

We show two years of weekly sales + a robust Holt-Winters time series model.

We’ve constructed an artificial three-week drop in sales for demonstration purposes.

Outlier identification occurs as part of the estimation process.

Imputation uses a separate algorithm.
Identifying store closures using repeated median estimators

Hurricane Harvey stands out clearly in this plot.

Our GPU-based implementation of the (computationally intensive) RM estimator offers runtime reductions of > 40-1 over parallelized CPU-based implementations using 48 CPU cores.
Feature Engineering

Initial architecture

1. **Tumbleweed**
   - Pull
2. **MainFrame**
   - Push
   - NFS
3. **Terradata**
4. **Airflow**
   - Orchestrated and scheduling using Airflow

Data Sources:
- Tumbleweed
- MainFrame
- Terradata

Spark Cluster:
- Compute
  - Staging area
  - Processed and Transformed data

- Feature Engineering
  - Apply DQ rules and store back data on S3
  - Perform transformations and aggregations
  - Store featured engineered data on S3

Spark Cluster:
- ML Pipeline
- GPU Cluster

Model Training & Forecasting
Feature engineering - Roadblock

• Initial FE strategy:
  – Extract raw data from databases
  – FE execute on Spark / Scala (giving us scalability)
  – Push features to GPU machines for consumption by algorithms

• As the volume of data grew, the Spark processes began to fail erratically
  – Appeared to be a memory issue internal to Spark – nondeterministic feature outputs and crashes
  – Six+ weeks of debugging / restructuring code had essentially no effect

• Eventually, we were unable to complete any FE processes at all
Revised Feature Engineering Pipeline

- Spark code ported to R / C++ / CUDA

- Port took 2 weeks + 1 week code cleanup
- Performance was essentially the same as the Spark cluster
- CUDA code runtime reduction of \(~50-100x\) relative to C++ parallelized on 48 CPU cores
- With a full port to CUDA, we’d expect \(~4x\) reduction in FE computation runtime over today
- Reliability has been essentially 100%!

GPU Cluster: 14 SuperMicro servers with 4x P100 NVIDIA GPU cards
Future Revised Feature Engineering Pipeline

- R / C++ / CUDA code ported to Python / RAPIDS

- Walmart is working with NVIDIA to ensure RAPIDS functionality encompasses our use cases

- Our in-house testing indicates very significant runtime reductions are almost assured – exceeding what we could do on our own

- Implementation expected in June – August timeframe
Better Features - detection of spatial anomalies

Spatial anomaly detection using:
- k-NN estimation of store unit lift
- $G^*$ z-score estimate of spatial autocorrelation
- False Discovery Rate

Takes about 2 minutes to run on a single CPU – obviously infeasible to use this for our problem.

k-NN is part of RAPIDS; early tests indicate a runtime reduction of $>100x$ by switching to the RAPIDS implementation.

The rest of the calculations will have to be ported to CUDA by us.
Algorithm Technology

- Gradient Boosting Machine
- State Space model
- Random Forests
- ... others ...
- Ensembling
Production configuration

Our training and scoring are run on a cluster of 14 SuperMicro servers each with 4x P100 NVIDIA GPU cards

• Kubernetes manages Dockerized production processes.

• Each server can run four groups of store-item combinations in parallel, one on each GPU card.

• For CPU-only models, our parallelization limits us to one group per server at a time.
## Forecasting Algorithms – the two mainstays

### Gradient Boosting Machine

- **Gradient boosting** is a machine learning technique for regression and classification problems.
- GBM prediction models are an ensemble of hundreds of weak decision tree prediction models.
- Each weak model tries to predict the errors from the cumulation of all the previous prediction models.
- Features (such as Events, Promotions, SNAP calendar, etc.) are directly added as regressors.
- Interactions between the regressors are also detected by the boosting machine and automatically incorporated in the model.
- Mostly works by reducing the bias of the forecasts for small subsets of the data.

**Pros**
- Ability to easily incorporate external factors (features) influencing demand.
- The algorithm infers the relationships between demand and features automatically.

### State Space Model

- Defines a set of equations to describe hidden states (e.g. demand level, trend, and seasonality) and observations.
- The **Kalman Filter** is an algorithm for estimating parameters in a linear state-space system. It sequentially updates our best estimates for the states after having the "observations" (sales) and other features (such as price), and is very fast.
- "Linearizes" features before incorporating them.

**Pros**
- Can forecast for any horizon using a single model.
- Can work seamlessly even if some of the data is missing – it just iterates over the gap.
- Very fast.
Gradient Boosting Machine

- Underlying engine: NVIDIA’s XGBoost / GPU code
  - Both R package and Python library
  - Can be called from C/C++ as well
  - Performance comparison:
    - Pascal P100 (16GB memory) vs 48 CPU cores (out of 56) on a Supermicro box
    - Typical category size (700K rows, 400 features)
    - GPU speedup of $\sim 25x$.

- Features
  - Lagged demands -> level, trends, seasonal factors
  - Holidays, other U. S. – wide events (e.g., Super Bowl weekend)
  - (lots of) other features
State space model

• State space (DLM) model adapted from one developed for e-Commerce

• Generates forecasts for a cluster of items at all stores at once

• Multiple control parameterizations of model treated as an ensemble of models and a weighted average is returned as the forecast

• Used for all long-horizon forecasts and about 30% of short-horizon forecasts

• Implemented in TensorFlow (port from C++)
  – GPU version of TensorFlow did not offer much speed improvement over CPU version (< 2x)

• Uses Kalman Filtering for updating state parameters
  – Preliminary tests indicate RAPIDS Kalman Filter routine is far faster than what we are using today
Forecasting Algorithms – the next wave

Random Forests

- **Random Forests** is a machine learning technique for regression and classification problems.
- RF prediction models are an ensemble of hundreds of strong (deep) decision tree prediction models averaged together.
- Each strong model tries to predict the errors from a random sample of the data.
- More randomization is added by selecting a subset of features to be evaluated at each node.
- Features (such as Events, Promotions, SNAP calendar, etc.) are directly added as regressors.
- Interactions between the regressors are also detected by the deep tree and automatically incorporated in the model.
- Mostly works by averaging out model- and dataset- specific overfitting of the forecasts.

**Pros**
- Ability to easily incorporate external factors (features) influencing demand.
- The algorithm infers the relationships between demand and features automatically.

Ensembles

- Uses forecasts generated by different models, possibly along with other features, as predictors in a “final stage” model.
- Can be as simple as a weighted average of different predictors or much more complex.
- Mostly works by averaging out model-specific overfitting of the data.

**Pros**
- Typically superior to “pick the best model” approaches.
- Almost always offers at least some improvement over any individual forecast.
Random Forests

• Underlying engine: scikit-learn’s random forest algorithm
  – Many other random forest implementations exist; this one works well for us...
  – However, scikit-learn’s implementation is too slow for scoring given our time window
  – Developed custom CUDA code to score using the model object returned by Python:
    • GPU speedup of > 300x relative to scikit-learn (> 50x including file i/o)
    • Makes Random Forests a practical alternative to our GBM and State Space models

• NVIDIA is developing a GPU-based random forest implementation using some of our data for test purposes

• Features - uses the same feature set as the GBM; this will likely change
Ensembling

- Ensembling implemented as weighted averages of different models’ predictions
  - Analysis to determine weights done separately
  - Reduces scoring time, increases forecast stability

- Weighted averages of a small number of large vectors is an ideal task for CUDA

- However, in this case, the CPU performs well too:
  - Large numbers of fast operations on localizable memory
  - No transfer of data to and from the GPU (about half the time in our tests)

- The advantage of the GPU is < 2x.

- With NVLINK and a newer generation of GPU card, it would be perhaps 4x.
Benefits

- Runtime improvements enable:
  - Better history cleansing algorithms
  - Better and more comprehensive feature engineering
  - A broader suite of forecasting algorithms

- However, incremental improvements do little. We need large improvements to be able to revolutionize the forecasting pipeline.

- With RAPIDS and CUDA, we’ve been able to implement a forecasting pipeline that:

  reduces overall forecast error by ~ 1.7 percentage points

relative to the reduction in forecast error that would have been possible with CPU-only code.

And... we’re not done yet!
Thank you, NVIDIA!