parameterserver.org

blog.smola.org @smolix

Logistic Regression

Motivation: Computational Advertising

sponsored search picks position of ad using

$$p(\text{click}|\text{ad}) \cdot \text{bid}(\text{ad})$$

estimate it

4 million/minute

Motivation: Computational Advertising

sponsored search picks position of ad using

$$p(\text{click}|\text{ad}) \cdot \text{bid}(\text{ad})$$

estimate it

4 million/minute

Estimating Probabilities

- **Logistic model (exponential family)**

$$p(y|t) \propto \exp\left(\frac{1}{2}yt\right) \quad \text{where } y \in \{\pm 1\}$$

  y will tend to agree with the sign of t (find t)

- **Normalizing terms**

$$p(y|t) = \frac{\exp\left(\frac{1}{2}yt\right)}{\exp\left(\frac{1}{2}t\right) + \exp\left(-\frac{1}{2}t\right)} = \frac{\exp\left(\frac{1}{2}yt\right)}{\exp\left(\frac{1}{2}yt\right) + \exp\left(-\frac{1}{2}yt\right)}$$

$$= \boxed{\frac{1}{1 + \exp(-yt)}}$$

(Penalized) Maximum Likelihood

- **Goal**
  Find t that correlates with y
- **Strategy**
  Use covariates x and function f(x)

$$p(y|x) = \frac{1}{1 + \exp(-yf(x))}$$

- **Penalty against overfitting / Bayes rule**

$$p(f|X,Y) \propto p(f) \prod_{i=1}^{m} \frac{1}{1 + \exp(-y_i f(x_i))}$$

Penalized Maximum Likelihood

- Picking a function class

$$f(x) = \langle w, x \rangle$$

- Picking a prior

$$\log p(f) = \lambda \|w\|_1 + \text{const.}$$

we want sparse models for advertising

- Picking an inference strategy

$$\underset{w}{\text{minimize}} - \log p(f|X, Y)$$

$$\underset{w}{\text{minimize}} \sum_{i=1}^{m} \log(1 + \exp(-y_i \langle w, x_i \rangle)) + \lambda \|w\|_1$$

Proximal Algorithm

- Problem - l_1 norm is non-smooth
- Proximal operator

$$\operatorname*{argmin}_{w} \|w\|_1 + \frac{\gamma}{2} \|w - (w_t - \eta g_t)\|^2$$

(more generally use penalty on w)

- Updates for l_1 are

$$w_i \leftarrow \operatorname{sgn}(w_i) \max(0, |w_i| - \epsilon)$$

All steps decompose by coordinates

Generic Parallel Template

- Compute gradient on (subset of data) on each client
- Send gradient from client to server asynchronously
  `push(key_list,value_list)`
- Proximal gradient update on server per coordinate
- Server returns parameters
  `pull(key_list,value_list)`

Background

Data per minute 2012

http://www.domo.com/learn/infographic-data-never-sleeps

Google

Data per minute 2014

> 100 TB data

> 1000 machines

> 100B parameters

http://www.domo.com/learn/data-never-sleeps-2

Google

Stuff fails a lot Deal with it

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures

Jeff Dean's Stanford slides

slow disks, bad memory, misconfigured machines, flaky machines, etc.

Machine Learning Problems

- Many models have O(1) blocks of O(n) terms (LDA, logistic regression, recommender systems)
- More features than what fits into RAM (personalized CTR, large inventory, action space)
- Local model typically fits into RAM
- Data needs many disks for distribution
- Decouple data processing from aggregation (similar idea as MapReduce)
- Sweet spot - optimize for 80% of ML

Communication pattern

put(keys,values,clock), get(keys,values,clock)

General parallel algorithm template

- Clients have local view of parameters
- P2P is infeasible since $O(n^2)$ connections
- Synchronize with parameter server
  - Reconciliation protocol
    average parameters, lock variables
  - Synchronization schedule
    asynchronous, synchronous, episodic
  - Load distribution algorithm
    uniform distribution, fault tolerance, recovery

Smola & Narayanamurthy, 2010, VLDB
Gonzalez et al., 2012, WSDM
Shervashidze et al., 2013, WWW

also at Google, Baidu, Facebook, Amazon, Yahoo, Microsoft, ...

Architecture

Parameter Server Agenda

- Chord style key layout
- Messages and synchronization
- Filters for message compression
- Directed acyclic graph for consistency
- Range based vector clocks & synchronization
- Applications
  Logistic regression, Topic models, Sketches

Key layout & recovery

Consistent Hashing

- Caching
  - Store many (key,value) pairs
  - Linear scaling in clients & servers
  - Automatic key distribution
- memcached
  - (key,value) servers
  - client access library distributes access patterns
  - randomized O(n) bandwidth
  - aggregate O(n) bandwidth
  - load balancing via hashing
  - no versioned writes / vector clocks
  - very expensive to iterate over all keys for a given server

$$m(\text{key}, \mathcal{M}) = \underset{m' \in \mathcal{M}}{\text{argmin}}\, h(\text{key}, m')$$

Keys arranged in a DHT

- Virtual servers
  - loadbalancing
  - multithreading
- DHT
  - contiguous key range for clients
  - easy bulk sync
  - easy insertion of servers
- Replication
  - Machines hold replicas
  - Easy fallback
  - Easy insertion / repair

Keys arranged in a DHT

- Virtual servers
  - loadbalancing
  - multithreading
- DHT
  - contiguous key range for clients
  - easy bulk sync
  - easy insertion of servers
- Replication
  - Machines hold replicas
  - Easy fallback
  - Easy insertion / repair

Yes, we screwed up before!
And everyone copied us!

Key Replication

- Each segment is owned by one virtual server
- Subsequent machines hold replicas
- Easy fallback
- Easy insertion / repair
- Dynamic load balancing

Key layout

Key layout

Key layout

Key layout

Key layout

Recovery / server insertion

- Precopy server content to new candidate (3)
- After precopy ended, send log
- For k virtual servers this causes $O(k^{-2})$ delay
- Consistency using vector clocks

Communication

Simple API

- Clients and Servers share much code
- Send data to server
  asynchronously in an interval
  `push(key_list,value_list,flag)`
- Receive data from server in an interval
  `pull(key_list,value_list,flag)`
- Avoid sending single items
  - Serialization overhead - protobuf message
  - Consistency overhead - O(c) vector clocks

Simple API

- Datatypes are `eigen3` native
  - Dense vectors
  - Sparse vectors
- `Flag` may specify
  - Value or delta update
  - Recipient (all machines, particular node)
- Asynchrony means that push only queues. Send may happen later (more up to date values)
- Shared pointer. No copy on queue (by default) User needs to copy if desired.

Batched Communication

- Overhead of sending individual K/V is large
  - 10^{10}—10^{14} packages
  - Package header (e.g. TCP/IP) matters
  - Horrible examples: memcached, YahooLDA
- Communicate only when
  - Finish one local "iteration" (processed a group of samples or features)
  - Reached the end of a specific time window (prevent stale data)

Message Aggregation on Server

Send as little as possible

- Only send data the receiver needs
  - A server node maintain segments of keys
  - Client nodes may have different working sets

Filters

Message Compression

- Convergence speed depends on communication efficiency
  - Sending (key,value) pairs is inefficient
    Send only values (cache key list) instead
  - Sending small gradients is inefficient
    Send only sufficiently large ones instead
  - Updating near-optimal values is inefficient
    Send only large violators of KKT conditions
- Filter data before sending

Key compression

- Data Compression
  - Google Protobuf
  - Zippy
- Ignore keys if possible
  - client 0 sends to server 0
    - time 1: (2,2.3), (4,6.1), (8,9.9)

      ...

      time 6: (2,5,4), (4,2.5), (8,2.9)

Key compression

- Data Compression
  - Google Protobuf
  - Zippy
- Ignore keys if possible
  - client 0 sends to server 0
    - time 1: (2,2.3), (4,6.1), (8,9.9)

      ...

      time 6: (2,5,4), (4,2.5), (8,2.9)

Both sender and receiver cache the key list. If hit cache, then send checksum only

Filters

- Scheduling
  have controller decide when to send
  (this requires very smart controller - difficult)
- Filtering
  have algorithm decide when to shut up (easier)
  - Gradient (only send large gradients)
  - KKT (only send variables violating KKT)
  - Randomized (sparse random vectors)
  - Quantization (reduce accuracy)

Filters in practice

- Sparse Logistic Regression
  - Only send large updates
  - Compress sparse value list

20x compression

Clocks and Consistency

Consistency Zoo

- Samplers only need loose synchronization (large delay, eventual consistency)
- Hogwild (fully asynchronous, unclear how messy)
- Distributed proximal gradient (needs bounded delay, but delay differs)
- Brittle ML algorithms (off the shelf) (fully synchronous, no delay)

Consistency Zoo

- Samplers only need loose synchronization (large delay, eventual consistency)
- Hogwild (fully asynchronous, unclear how messy)
- Distributed proximal gradient (needs bounded delay, but delay differs)
- Brittle ML algorithms (off the shelf) (fully synchronous, no delay)

Which side do you pick?

Consistency models

Sequential

Eventual

Bounded delay

Consistency models

Sequential

Eventual

Bounded delay

via task processing engine on client/controller

Consistency models

- Change dependency on the fly
- Task granularity programmatically defined (small or large tasks)
- Subtree controlled by worker

Vector Clocks for Ranges

- Keep track of when we received an update from a client / server.
- For c clients this means O(c) metadata
  This is impossible to store per key (Dynamo)
- Very cheap and feasible for ranges
- When inconsistent ranges, split segments [A,D] splits into [A,B], [B,C] and [C,D] when receiving message for [B,C]
- This is infrequent + defragmentation

Vector Clocks for Ranges

- For each (key,value) pair we know all timestamps from all clients
- If client dies and restarts, we know whether we already received the message
- Use with dependency DAG

Experiments

Logistic Regression

Guinea pig - logistic regression

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1$$

- Implementation on Parameter Server

	Method	Consistency	LOC
System-A	L-BFGS	Sequential	10,000
System-B	Block PG	Sequential	30,000
Parameter Server	Block PG	Bounded Delay KKT Filter	300

Recall: Parallel Template

- Compute gradient on (subset of data) on each client
- Send gradient from client to server asynchronously
  `push(key_list,value_list)`
- Proximal gradient update on server per coordinate
- Server returns parameters
  `pull(key_list,value_list)`

Recall: Parallel Template

- Compute gradient on (subset of data) on each client
- Send gra
  to serve
  push(ke
- Proxima
  on server per coordinate
- Server returns parameters
  pull(key_list,value_list)

with theorem for convergence

Convergence speed

500TB CTR data
100B variables
1000 machines

- System A and B are production systems at a very large internet company …

Scheduling Efficiency

Parameter Server as Stream Processor

Communication Pattern

- Client - Ingest data/query from network (users, CTR, event logger)
- Server - Aggregate sketch (CountMin, SpaceSaver, CounterBraid)

Guinea pig - CountMin Sketch

- Intuition - Bloom Filter with integers (see Muthukrishnan and Cormode, 2005)
- Insert

$$M[h(k,j),j] \leftarrow M[h(k,j),j] + v \text{ for all } j \in \{1,\dots d\}$$

Each counter is an upper bound on counts

- Query

$$m(k) \leq \min_j M[h(k,j),j]$$

- Extensions to time series (see Matyusevych, Ahmed, Smola, 2012)

Distributed CountMin Sketch

- Clients only act as data preprocessors
- Shard keys over servers for balancing
- Replication between machines on DHT
- Servers perform simple updates

$$M[h(k,j),j] \leftarrow M[h(k,j),j] + v \text{ for all } j \in \{1,\ldots d\}$$

- 15 servers, 40GBit network (dedicated)

Peak inserts per second	1.3 billion
Average inserts per second	1.1 billion
Peak network bandwidth per machine	4.37 GBit/s
Time to recover a failed node	0.8 second

Limited by DRAM Latency

Models

Clustering & Topic Models

clustering

Latent Dirichlet Allocation

Topics in text

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Latent Dirichlet Allocation; Blei, Ng, Jordan, JMLR 2003

Collapsed Gibbs Sampler

Griffiths & Steyvers, 2005

$$p(z, x|\alpha, \beta)$$

$$= \prod_{i=1}^{m} p(z_i|\alpha) \prod_{k=1}^{k} p(\{x_{ij}|z_{ij} = k\}|\beta)$$

$$\frac{n^{-ij}(t,d) + \alpha_t}{n^{-i}(d) + \sum_t \alpha_t} \qquad \frac{n^{-ij}(t,w) + \beta_t}{n^{-i}(t) + \sum_t \beta_t}$$

Collapsed Gibbs Sampler

Griffiths & Steyvers, 2005

$$p(z, x|\alpha, \beta)$$

$$= \prod_{i=1}^{m} p(z_i|\alpha) \prod_{k=1}^{k} p(\{x_{ij}|z_{ij} = k\}|\beta)$$

$$\frac{n^{-ij}(t,d) + \alpha_t}{n^{-i}(d) + \sum_t \alpha_t}$$

$$\frac{n^{-ij}(t,w) + \beta_t}{n^{-i}(t) + \sum_t \beta_t}$$

Gibbs Sampler

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Lock (word,topic) table
      - Update local (document, topic) table
      - Update (word,topic) table
      - Unlock (word,topic) table

this kills parallelism

Gibbs Sampler

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Lock local (word,topic) table
      - Update local (document, topic) table
      - Update local (word,topic) table
      - Unlock local (word,topic) table
    - Synchronize local and global tables

this kills multithreading

Gibbs Sampler for LDA

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Generate local update message
    - Update local table
      - Lock local (word,topic) table
      - Update local (word,topic) table
      - Unlock local (word,topic) table
    - Synchronize local and global tables

4B documents, 1M tokens, 60k cores, 2k topics

4B documents, 1M tokens, 60k cores, 2k topics

Log-Likelihood distribution as a function of iteration count for workers

-6.1e+08
-6.2e+08
-6.3e+08
-6.4e+08
-6.5e+08
0
10
20
30
40
50
60
70
1000 machines
6000 machines

Palo Verde, AZ
3 Gigawatt (4 million people)
Largest nuclear reactor in the USA
Courtesy of Palo Verde Nuclear Generati

Palo Verde, AZ
3 Gigawatt (4 million people)
Largest nuclear reactor in the USA
1 machine = 10 cores
1 core = 50 watt
consumption of 3 Megawatt
Courtesy of Palo Verde Nuclear Generat

Advance preview

- Come to our KDD 2014 prize talk in NYC (Aaron Li, Amr Ahmed, Sujith Ravi, Alex Smola)
- Problem
  - Language model gets very dense when you have many documents.
  - Destroys sparsity of Mimno/McCallum trick
- Solution
  - Process stale distribution with Alias trick
  - Accept with Metropolis Hastings
- Order of magnitude speedup (fastest sampler)

Saving Nuclear Power Plants

Google

Carnegie Mellon University

Saving Nuclear Power Plants

Google

Carnegie Mellon University

Convenience

Performance

See our paper at OSDI 2014

parameterserver.org

blog.smola.org @smolix