Quantmetry
Data Science Consulting

Online learning, Vowpal Wabbit and Hadoop

Héloïse Nonne

June 2nd, 2015

hnonne@quantmetry.com

Quantmetry

A long time ago,

in a country far far away

after one of the most terrible conflict

the world has ever known

a few inspired men invented

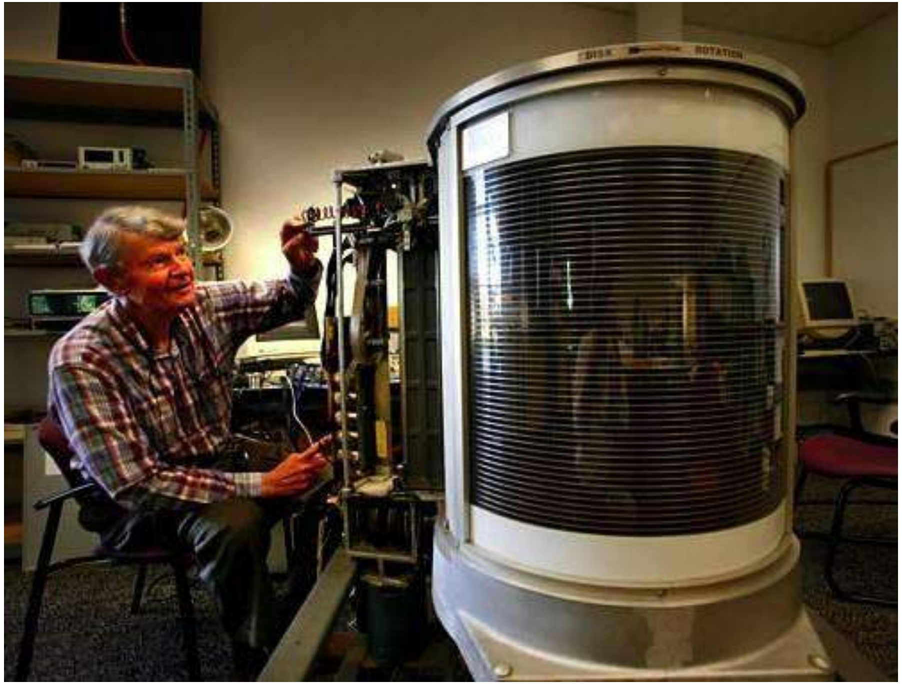
one of the greatest inventions of all

times

Episode IV: A New Hope

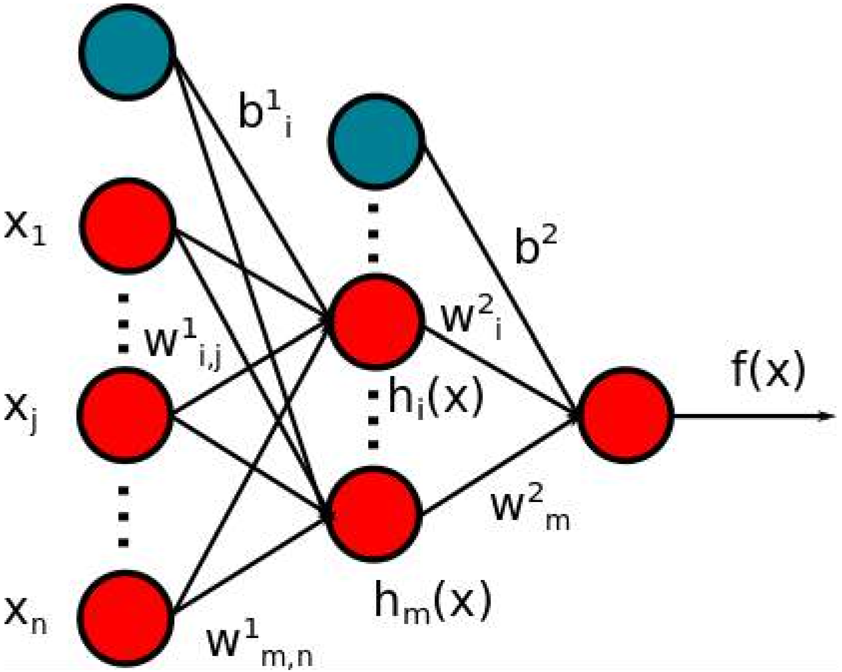
1936: Turing machine: Online processing

The first computers



1956
IBM 350 RAMAC
Capacity: 3.75 MB

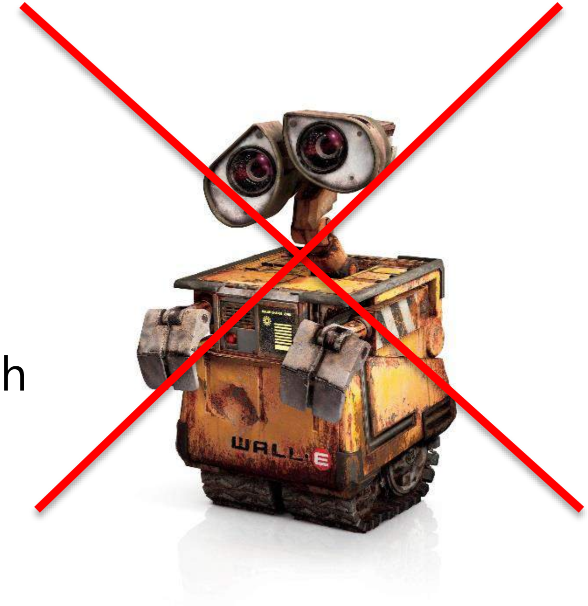
Artificial intelligence: almost done!



1943 – 1960’s: Multiple layers neural networks

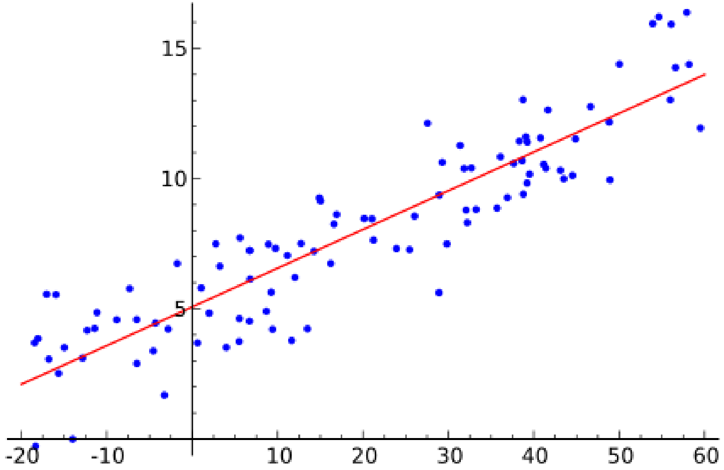
Episode V: Reality Strikes Back

- Not enough computing power
- Not enough storage capacity
- Not enough data
- Algorithms are not efficient enough



Episode VI: Return of the Regression

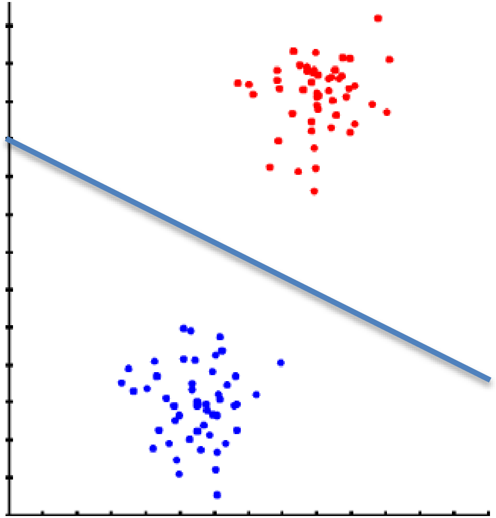
N samples with m features: $x^{p} \in R^{M}$
Result to predict: $y^{p} \in R$



Learn a weight vector $w \in R^{M}$ such that :

$$y_w(x) = \sum_i w_i x_i \sim y$$

N samples with m features: $x^{p} \in R^{M}$
Class to predict: $y^{p} = 0{,}1/$ blue,red



Learn a weight vector $w \in R^{M}$ such that :

$$y_w(x) = \frac{1}{1+\exp(-\sum_i w_i x_i)}$$ is close to y

Loss functions

	Loss function	Model: $f_w(x)$	Meaning of $y_w(x)$
Linear regression	$\frac{1}{2}(y_w - y)^2$	$\sum_i w_i x_i$	Conditional expectation $E(y\|x)$
Logistic regression	$\log(y_w)$	$\frac{1}{1+\exp(-\sum_i w_i x_i)}$	Probability $P(y\|x)$
Hinge regression	$\max(0,1 - y y_w)$	$\operatorname{sign}(x)$	Approximation -1 or 1

Accounts for your model error

Choose a loss function according to your usecase

Batch learning algorithm

Predictive model with parameters w

N samples
(x^p, y^p)

Prediction on (x, y)
$y_w(x) = f_w(x)$
Compute the loss

Accumulate for all samples
→ global loss

$$E_w = \frac{1}{N} \sum_p L(y^p, f_w\,(x^p))$$

Minimize the global loss to find the best parameters

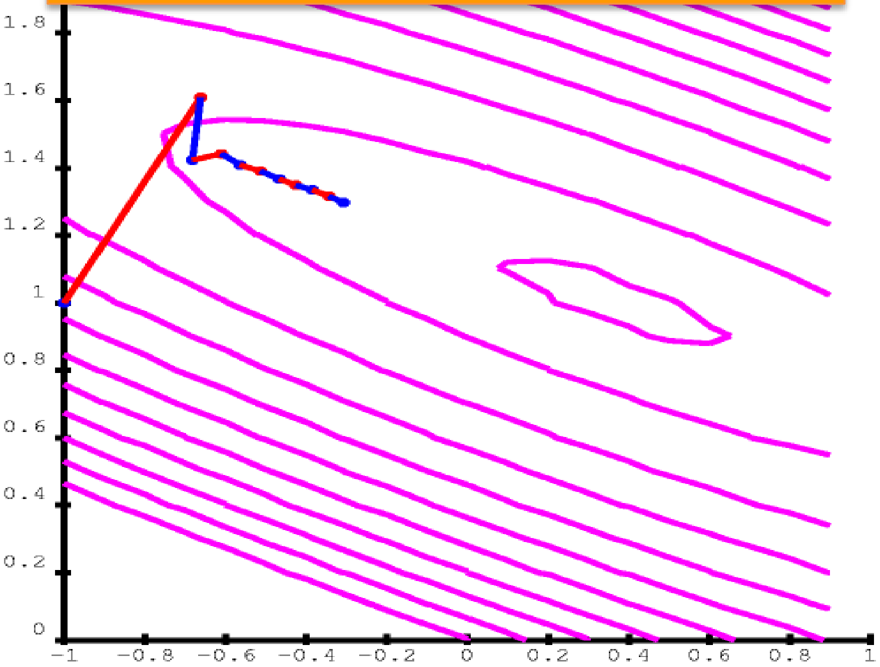
$$w_{t+1} = w_t - \frac{\eta}{N} \sum_p \nabla_w L(y^p, f_w(x^p))$$

Complexity:
$\vartheta(N)$ for each iteration

Many iterations
Each on the entire dataset

Batch learning algorithm

Direction is orthogonal to the isocontours



Global loss function in weights space
Lines = isocontours

Complexity:
$\vartheta(N)$ for each iteration

Minimize the global loss to find the best parameters

$$w_{t+1} = w_t - \frac{\eta}{N} \sum_p \nabla_w L(y^p, f_w(x^p))$$

Many iterations
Each on the entire dataset

Source: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Episode I: The Big Data Menace

What if

- data does not fit in memory?
- we want to combine features together (polynomials, n-grams)?

→ dataset size inflation

- new samples come with new features?
- the phenomenon we try to model drift with time?

Online learning algorithm

N samples
(x^p, y^p)

Predictive model with parameters w

Prediction on (x, y)
$y_w(x) = f_w(x)$

Compute the individual loss

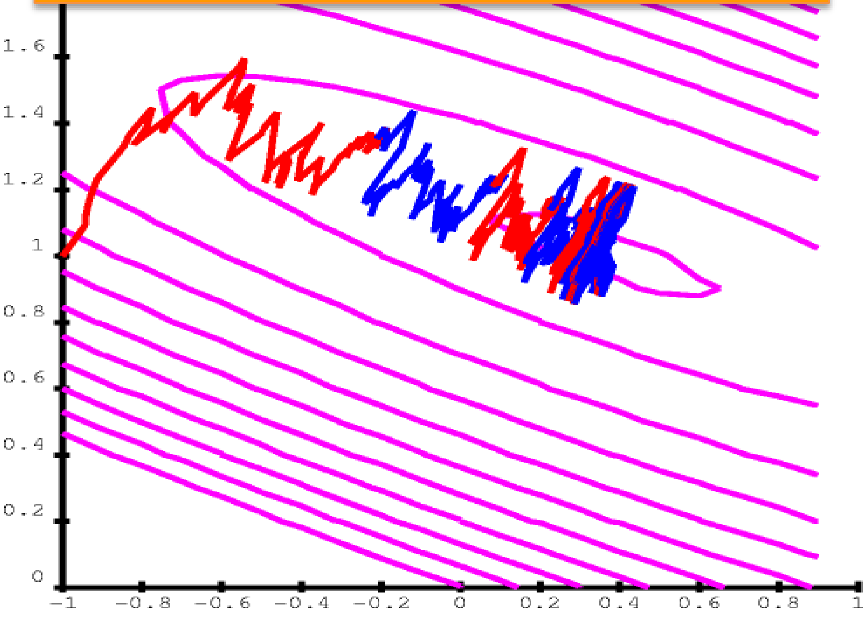
Update the parameters to minimize individual loss
$$w_{t+1} = w_t - \eta \nabla_w L(y, f_w(x))$$

Complexity:
$\vartheta(1)$ for each iteration

Many iterations
One sample at a time

Online learning algorithm

Direction is not perpendicular,
but is updated much more often



Having more updates allows to stabilize
and approach the minimum very quickly

Complexity:
$\vartheta(1)$ for each iteration

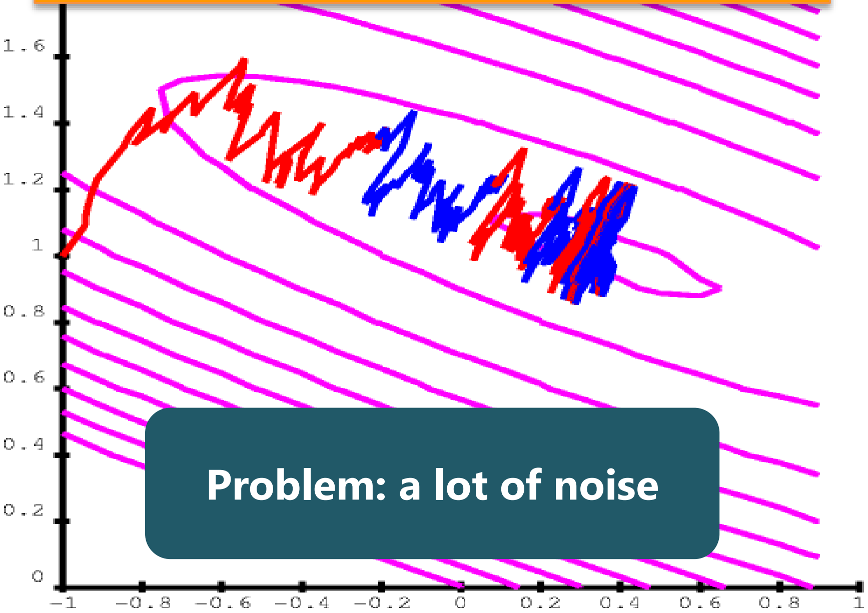
Update the parameters to minimize individual loss

$$w_{t+1} = w_t - \eta \nabla_w L(y, f_w(x))$$

Many iterations
One sample at a time

Online learning algorithm

Direction is not perpendicular,
but is updated much more often



Problem: a lot of noise

Having more updates allows to stabilize
and approach the minimum very quickly

Complexity:
$\vartheta(1)$ for each iteration

Update the parameters to minimize individual loss

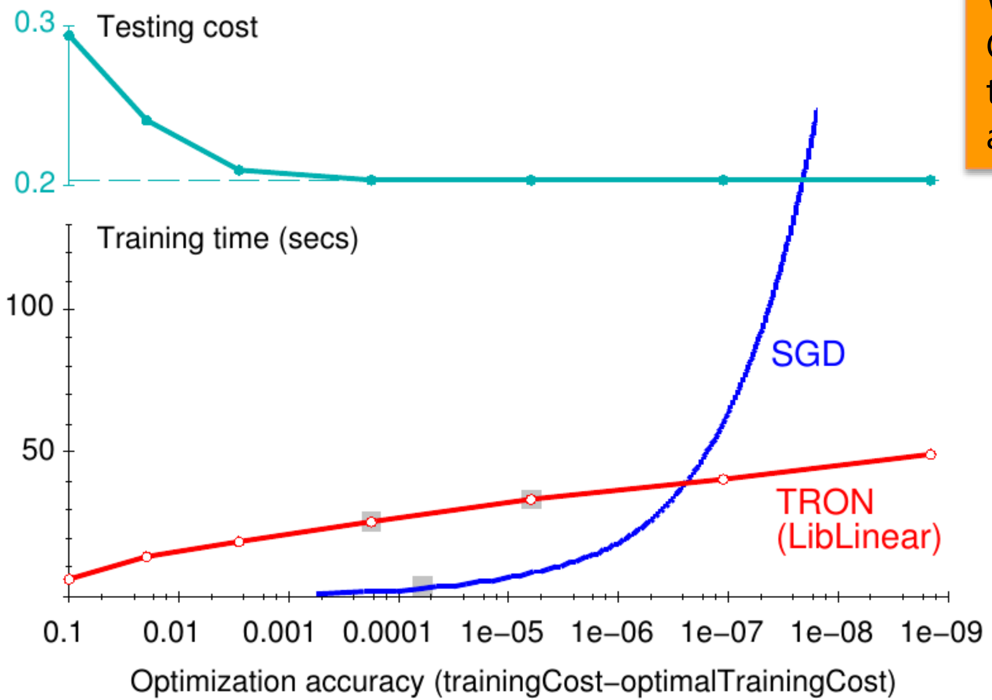
$$w_{t+1} = w_t - \eta \nabla_w L(y, f_w(x))$$

Many iterations
One sample at a time

The time required for convergence

Optimization accuracy against training time for online (SGD) and batch (TRON)

Online learning requires a lot less time to approximately converge



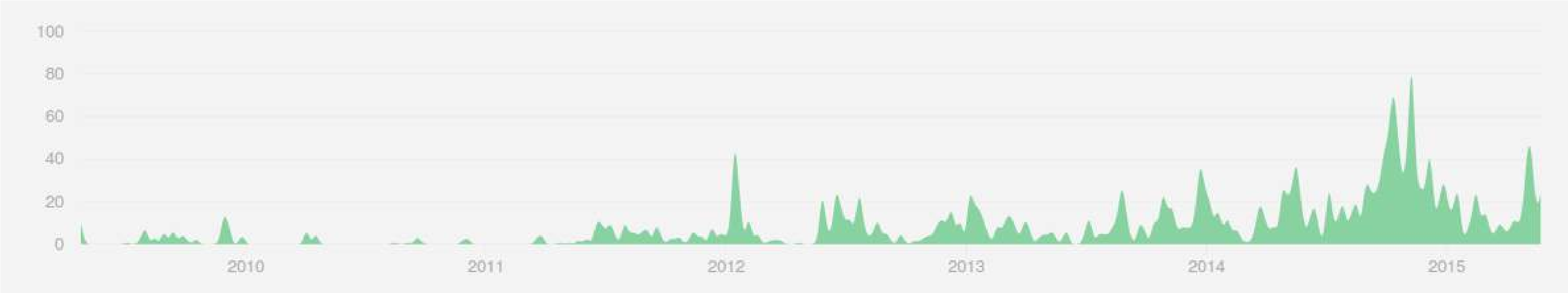
WARNING
Once batch becomes better, the validation error has already converged anyway.

Once close to the minimum, batch is much faster because it is noiseless

Bottou, *Stochastic gradient descent tricks, Neural Networks: Tricks of the Trade* (2012).

An implementation of online learning: Vowpal Wabbit

- Originally developed at Yahoo!, currently at Microsoft
- Led by John Langford
- C++
- efficient scalable implementation of online learning
- First public version 2007
- 2015: 4400 commits, 81 contributors, 18 releases



Nice features of VW

- **MANY algorithms** are implemented
- **Optimization algorithms** (BFGS, Conjugate gradient, etc.)
- **Combinations** of features, N-grams (NLP)
- **Automatic tuning** (learning rate, adaptive learning, on the fly normalization features)
- And more (boostraping, multi-core CPUs, etc.)

Vowpal Wabbit

Input agnostic

- Binary
- Numerical
- Categorical (hashing trick)
- Can deal with missing values/sparse-features

Very little data preparation

```
1 1.0 |height:1.5 length:2.0 |has stripes
1 1.0 |length:3.0 |has four legs
-1 1.0 |height:0.3 |has wings
-1 1.0 |height: 0.9 length: 0.6 |has a shell and a nice color
```

Vowpal Wabbit

- Fast learning for scoring on large datasets
- Can handle quite raw (unprepared data)
- Great for exploring a new dataset with simple and fast models
  - Uncover phenomena
  - Figure out what your should do for feature engineering

Yes but …

Episode II: Attack of the Clones

Why parallelizing?

- Speed-up
- Data does not fit on a single machine

  (Subsampling is not always good if you have many features)
- Take advantage of distributed storage and avoid the bottleneck of data transfer
- Take advantage of distributed memory to explore combination (multiplication to billions of distinct features)

The solution: online + batch learning

1. Each node k makes **an online pass** over its data (adaptive gradient update rule)
2. **Average the weights**

$$\bar{w} = \left(\sum_{k=1}^{K} G_k\right)^{-1} \left(\sum_{k=1}^{K} G_k w_k\right)$$

3. $\bar{w}$ is broadcasted down to all nodes and continue learning.
4. Iterate then with batch learning to make the last few steps to the minimum.

How is it implemented in Vowpal Wabbit project?

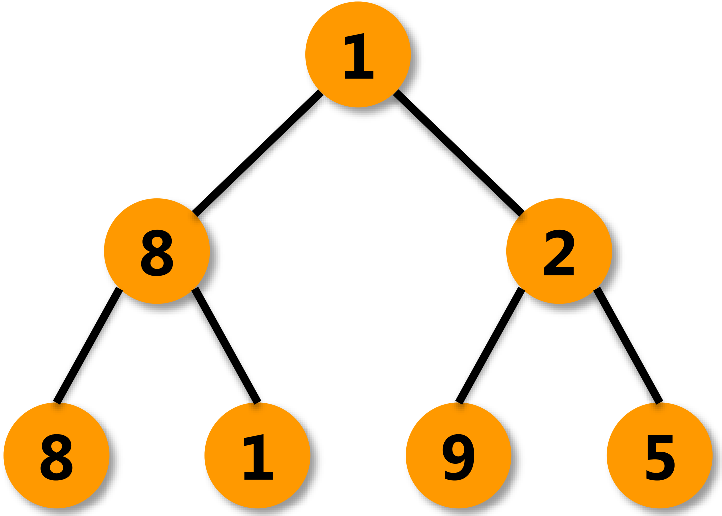
Implementation by VW: AllReduce + MapReduce

Needs	MPI (Message Passing Interface)	MapReduce
Effective communication infrastructure	Y **Allreduce** is simple N Data transfers across network	N Large overhead (job scheduling, some data transfer, data parsing)
Data-centric platform (avoid data transfer)	N Lack of internal knowledge of data location	Y Full knowledge of data location
Fault tolerant system	N Little fault tolerant by default	Y Robust and fault tolerant
Easy to code / good programming language	Y Standard and portable (Fortran, C/C++, Java)	Y Automatic cleanup of temp files by default N Rethink and rewrite learning code into map and reduce operations N Java eats up a lot of RAM
Good optimization approach	Y No need to rewrite learning code	N Map/reduce operations does not easily allow iterative algorithms
Overall time must be minimal	N Data transfers across network	N Increased number of read/write operations N Increased time while waiting for free nodes

All Reduce

Allreduce is based on a communication structure in trees

(binary is easier to implement)



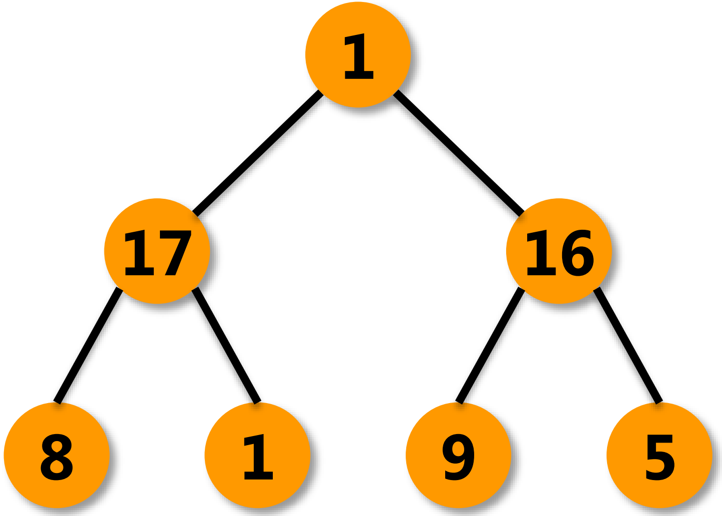
Every node starts with a number

1. Reduce = sum up the tree

All Reduce

Allreduce is based on a communication structure in trees

(binary is easier to implement)



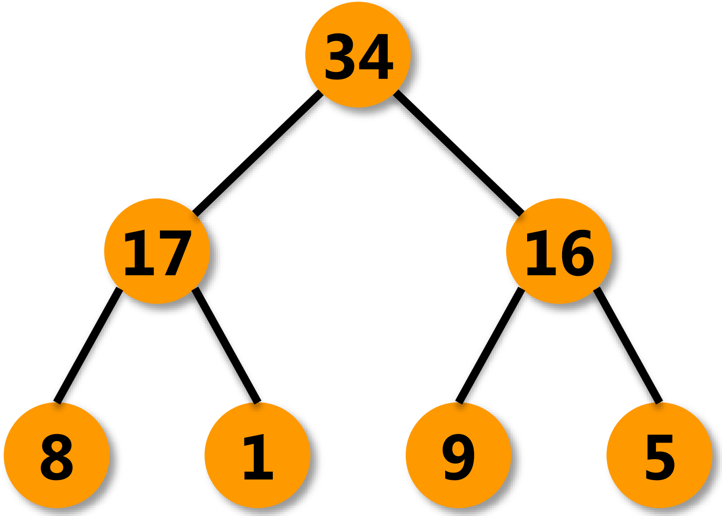
Every node starts with a number

1. Reduce = sum up the tree

All Reduce

Allreduce is based on a communication structure in trees

(binary is easier to implement)



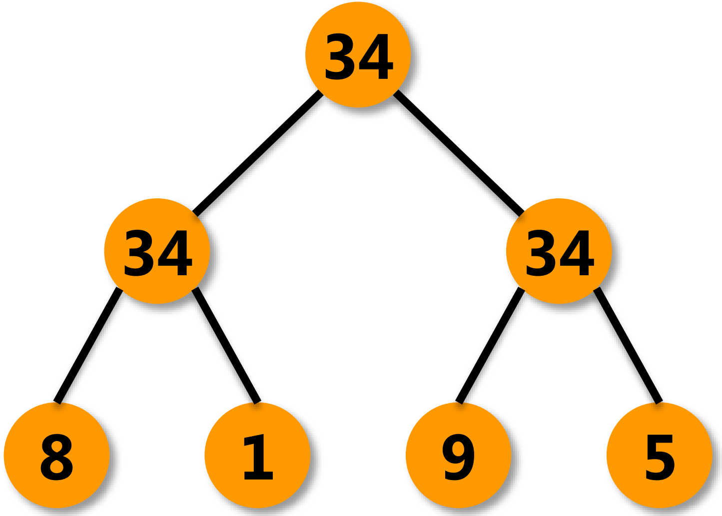
Every node starts with a number

1. Reduce = sum up the tree

All Reduce

Allreduce is based on a communication structure in trees

(binary is easier to implement)



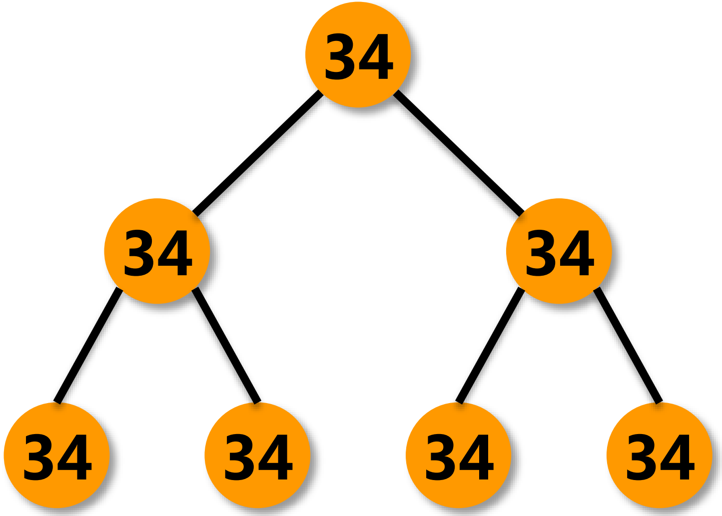
Every node starts with a number

1. Reduce = sum up the tree
2. Broadcast down the tree

All Reduce

Allreduce is based on a communication structure in trees

(binary is easier to implement)



Every node starts with a number

1. Reduce = sum up the tree
2. Broadcast down up the tree

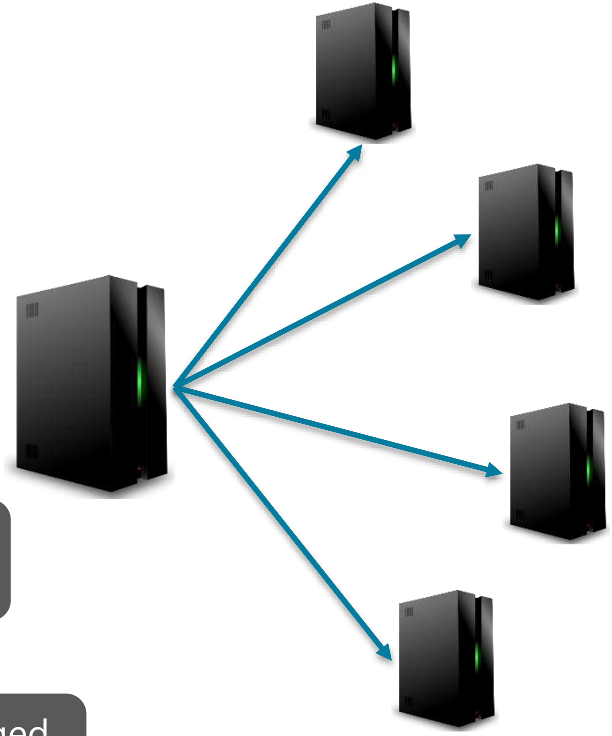
Every node ends up with the sum of the numbers across all the nodes

MapReduce (streaming)/AllReduce-online/batch

1. Start the daemon (communication system)

2. Each node makes an online pass over its data

3. Initialize a tree on the masternode



4. Use allreduce to average the weights over all nodes

5. Broadcast the averaged weights down to all nodes

6. Use it to initialize a batch learning step

7. Send back the weights and average with allreduce

8. Iterate other batch steps

Advantages of VW implementation

- **Minimal additional programming effort**
- **Data location knowledge**: use mapreduce infrastructure with only one mapper
- Vowpal wabbit (C/C++) is **not RAM greedy**
- **Small synchronisation overhead**
  - time spent in AllReduce operation $\ll$ computation time
- **Reduced stalling time** while waiting for other nodes
  - delayed initialization of AllReduce's tree to capitalize on Hadoop speculative execution
- **Rapid convergence** with online then **accuracy** with batch

Episode III: Revenge of Hadoop

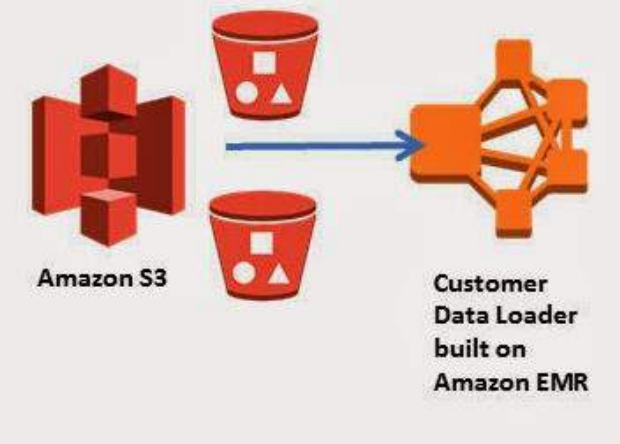
1. Start the daemon (./spanning_tree.cc)
2. Launch the MapReduce job
3. Kill the spanning tree

```
hadoop jar /home/hadoop/contrib/streaming/hadoop-streaming.jar \
    -D mapreduce.map.speculative=true \
    -D mapreduce.job.reduces=0 \
    -input $in_directory \
    -output $out_directory \
    -files [“/usr/local/bin/vw,
/usr/lib64/libboost_program_options.so, /lib64/libz.so.1”]
    -file runvw.sh \
    -mapper runvw.sh \
    -reducer NONE
```

Running vowpal wabbit on AWS

AWS Best practice: Transient clusters

- Get your data on S3 buckets
- Start an EMR (Elastic Map Reduce) cluster
- Bootstrap actions (install, config, etc.)
- Run your job(s) (steps)
- Shut down your cluster



Pros and cons

- Easy setup / works well
- Minimum maintenance
- Low cost
- **Logs ???????**
- Debugging ~~can be~~ is difficult

→ use an experimental cluster or a VM

Beware of the environment variables

VW needs MapReduce environment variables

- total number of mapper tasks
- number ID of the map task for each node
- ID of the MapReduce job
- private dns of the master node within the cluster

```
vw --total $nmappers --node $mapper \
        --unique_id $mapred_job_id -d /dev/stdin \
        --span_server $submit_host  \
        --loss_function=logistic -f sgd.vwmodel
```

Update the names in VW-cluster code
Hack the environment variables with python

Number of splits

You need to brute force the number of splits to the map reduce job

- Advice from John Langford / approach in the code
  - compute the size of your minimal data size (total / nb of nodes)
  - use option `-D mapreduce.min.split.size`
    → didn't work
- Dirty workaround
  - split the data into as many file as your nodes
  - store it in a .gz file

Running vowpal wabbit on AWS

```
ip-10-38-138-36.eu-west-1.compute.internal
Starting training
SGD ...
creating quadratic features for pairs: ft tt ff fi ti
final_regressor = sgd.vwmodel
Num weight bits = 18
learning rate = 0.5
initial_t = 0
power_t = 0.5

average  since          example        example  current  current  current
loss     last           counter         weight    label  predict features
0.693147 0.693147             2            1.0  -1.0000   0.0000      325
0.400206 0.107265             3            2.0  -1.0000  -2.1783      325
[...]
0.414361 0.404726        131073       131072.0  -1.0000  -4.5625      325
0.406345 0.398329        262145       262144.0   1.0000  -1.8379      325
0.388375 0.370405        524289       524288.0  -1.0000  -1.3313      325
```

Running vowpal wabbit on AWS

```
0.388375 0.370405       524289       524288.0  -1.0000  -1.3313      325
connecting to 10.38.138.36 = ip-10-38-138-36.eu-west-1.compute.internal:26543
wrote unique_id=2
wrote total=1
wrote node=0
read ok=1
read kid_count=0
read parent_ip=255.255.255.255
read parent_port=65535
Net time taken by process = 8.767000 seconds
finished run
number of examples per pass = 1000001
passes used = 1
weighted example sum = 1000000.000000
weighted label sum = -679560.000000
average loss = 0.380041
total feature number = 325000000
```

Running vowpal wabbit on AWS

```
BFGS ...
[...]
num sources = 1
connecting to 10.38.138.36 = ip-10-38-138-36.eu-west-1.compute.internal:26543
wrote unique_id=4
wrote total=1
wrote node=0
[...]
read parent_ip=255.255.255.255
read parent_port=65535
Maximum number of passes reached.
Net time taken by process = 10.55 seconds
finished run
number of examples = 1800002
weighted example sum = 1.8e+06
weighted label sum = -1.22307e+06
average loss = 0.350998 h
total feature number = 585000000
```

Running vowpal wabbit on AWS

6.4 GB
50 000 000 samples
52 974 510 080 features

On Hadoop with 5 nodes

6 minutes

On a single machine

26 minutes and 30 seconds

Concluding remarks

If possible, **use online learning** when time of computation is the bottleneck

Less computational time allows to explore more data
- Work on more data
- Include more features to the analysis
- Useful as a platform for research and experimentation

Optimization algorithms
- Lots of interesting papers (Langford, Bottou, Agarwal, LeCun, Duchi, Zinkevich, ...)

VW on Hadoop
Learn a lot by doing and debugging :-)

Episode VII: The Force Awakens

Coming soon

- Pushes on github

Benchmarks

- How is the training time affected by the size of the data set (measure overhead)
- Benchmark available approaches on **various large datasets and usecases**
- **Benchmark against Graphlab, MLlib**

More VW

- Exhaustive comparison and association with complex models (exploit vw for feature engineering and feature selection)
- Nonlinear online learning (neural networks, SVM, ...)

QUESTIONS?

Worker

Worker

Master node

Worker

A few references

- John Langford – Hunchnet – github
- Bottou, *Stochastic gradient descent tricks, Neural Networks: Tricks of the Trade* (2012)
- Yann LeCun's lectures
- **http://quantmetry-blog.com**

@HeloiseNonne