A Method for Parallel Online Learning

John Langford, Yahoo! Research

(on joint work with Daniel Hsu & Alex Smola & Martin Zinkevich & others)

MMDS 2010

My Personal Supercomputer

A RCV1 derived binary classification task:

1. 424MB Gzip compressed
2. 781K examples
3. 60M (nonunique) features

How long does it take to learn a good predictor?

My Personal Supercomputer

A RCV1 derived binary classification task:

1. 424MB Gzip compressed
2. 781K examples
3. 60M (nonunique) features

How long does it take to learn a good predictor?
20 seconds (1.2 seconds on desktop) = 3 M features/second

My Personal Supercomputer

A RCV1 derived binary classification task:

1. 424MB Gzip compressed
2. 781K examples
3. 60M (nonunique) features

How long does it take to learn a good predictor?
20 seconds (1.2 seconds on desktop) = 3 M features/second
Other systems:

1. AD-LDA (2000 topics) (KDD 2008): 205K features/second using 1000 nodes.

My Personal Supercomputer

A RCV1 derived binary classification task:

1. 424MB Gzip compressed
2. 781K examples
3. 60M (nonunique) features

How long does it take to learn a good predictor?
20 seconds (1.2 seconds on desktop) = 3 M features/second
Other systems:

1. AD-LDA (2000 topics) (KDD 2008): 205K features/second using 1000 nodes.
2. PSVM (2007): 23K features/second using 500 nodes (on RCV1)

My Personal Supercomputer

A RCV1 derived binary classification task:

1. 424MB Gzip compressed
2. 781K examples
3. 60M (nonunique) features

How long does it take to learn a good predictor?
20 seconds (1.2 seconds on desktop) = 3 M features/second
Other systems:

1. AD-LDA (2000 topics) (KDD 2008): 205K features/second using 1000 nodes.
2. PSVM (2007): 23K features/second using 500 nodes (on RCV1)
3. PLANET (depth 10 tree) (VLDB2009): 3M features/second using 200 nodes

How does Vowpal Wabbit work?

Start with $\forall i: \quad w_i = 0$, Repeatedly:

1. Get example $x \in R^*$.
2. Make prediction $\hat{y} = \frac{\sum_i w_i x_i}{\sqrt{|\{i: x_i \neq 0\}|}}$ clipped to interval $[0, 1]$.
3. Learn truth $y \in [0, 1]$ with importance I or goto (1).
4. Update $w_i \leftarrow w_i + \frac{\eta 2 (y - \hat{y}) I x_i}{\sqrt{|\{i: x_i \neq 0\}|}}$ and go to (1).

How does Vowpal Wabbit work?

Start with $\forall i: \quad w_i = 0$, Repeatedly:

1. Get example $x \in R^*$.
2. Make prediction $\hat{y} = \frac{\sum_i w_i x_i}{\sqrt{|\{i: x_i \neq 0\}|}}$ clipped to interval $[0, 1]$.
3. Learn truth $y \in [0, 1]$ with importance I or goto (1).
4. Update $w_i \leftarrow w_i + \frac{\eta 2(y - \hat{y}) I x_i}{\sqrt{|\{i: x_i \neq 0\}|}}$ and go to (1).

This is routine, but with old and new optimization tricks like hashing.
This is open source @ `http://hunch.net/~vw`
Also reimplemented in Torch, Streams, and Mahout projects.

Why I'm dissatisfied, and What I've learned so far.

Why I'm dissatisfied, and What I've learned so far.

1 Ghz processor should imply 1B features/second. And it's easy to imagine datasets with 1P features. How can we deal with such large datasets?

Why I'm dissatisfied, and What I've learned so far.

1 Ghz processor should imply 1B features/second. And it's easy to imagine datasets with 1P features. How can we deal with such large datasets?

Core Problem for Learning on much data = Bandwidth limits
1 Gb/s ethernet = 450GB/hour $\Rightarrow$ 1T features is reasonable.

Why I'm dissatisfied, and What I've learned so far.

1 Ghz processor should imply 1B features/second. And it's easy to imagine datasets with 1P features. How can we deal with such large datasets?

Core Problem for Learning on much data = Bandwidth limits
1 Gb/s ethernet = 450GB/hour $\Rightarrow$ 1T features is reasonable.

Outline

1. Multicore parallelization
2. Multinode parallelization

How do we paralellize across multiple cores?

How do we paralellize across multiple cores?

Answer 1: It's no use because it doesn't address the bandwidth problem.

How do we paralellize across multiple cores?

Answer 1: It's no use because it doesn't address the bandwidth problem.
But there's a trick. Sometimes you care about the interaction of two sets of features—queries with results for example. Tweak the algorithm so as to specify (query features, result features), then use a fast hash to compute the outer product in the core.

How do we paralellize across multiple cores?

Answer 1: It's no use because it doesn't address the bandwidth problem.
But there's a trick. Sometimes you care about the interaction of two sets of features—queries with results for example. Tweak the algorithm so as to specify (query features, result features), then use a fast hash to compute the outer product in the core.
Possibilities:

1. Example Sharding: Each core handles an example subset.
2. Feature Sharding: Each core handles a feature subset.

How do we paralellize across multiple cores?

Answer 1: It's no use because it doesn't address the bandwidth problem.
But there's a trick. Sometimes you care about the interaction of two sets of features—queries with results for example. Tweak the algorithm so as to specify (query features, result features), then use a fast hash to compute the outer product in the core.
Possibilities:

1. Example Sharding: Each core handles an example subset.
2. Feature Sharding: Each core handles a feature subset.

Empirically: Feature Sharding $>$ Example Sharding. Both work on two cores, but Example Sharding doesn't scale. Feature sharding provides about x3 speedup on 4 cores.

How do we paralellize across multiple cores?

Answer 1: It's no use because it doesn't address the bandwidth problem.
But there's a trick. Sometimes you care about the interaction of two sets of features—queries with results for example. Tweak the algorithm so as to specify (query features, result features), then use a fast hash to compute the outer product in the core.
Possibilities:

1. Example Sharding: Each core handles an example subset.
2. Feature Sharding: Each core handles a feature subset.

Empirically: Feature Sharding $>$ Example Sharding. Both work on two cores, but Example Sharding doesn't scale. Feature sharding provides about x3 speedup on 4 cores.
But, again, this is just for a special case. Need multinode parallelization to address data scaling.

Algorithms for Speed

1. Multicore parallelization
2. Multinode parallelization

Multinode = inevitable delay

Ethernet latency = 0.1 milliseconds = 10^5 cycles = many examples.

1. Example Sharding $\Rightarrow$ weights out of sync by delay factor.
2. Feature Sharding $\Rightarrow$ global predictions delayed by delay factor.

How bad is delay?

Multinode = inevitable delay

Ethernet latency = 0.1 milliseconds = 10^5 cycles = many examples.

1. Example Sharding $\Rightarrow$ weights out of sync by delay factor.
2. Feature Sharding $\Rightarrow$ global predictions delayed by delay factor.

How bad is delay?
Theorem: (Mesterharm 2005) Delayed updates reduce convergence by delay factor in worst case for expert algorithms.
Theorem: (LSZ NIPS 2009) Same for linear predictors.
(Caveat: there are some special cases where you can do better.)
Empirically: Delay can hurt substantially when examples are structured.

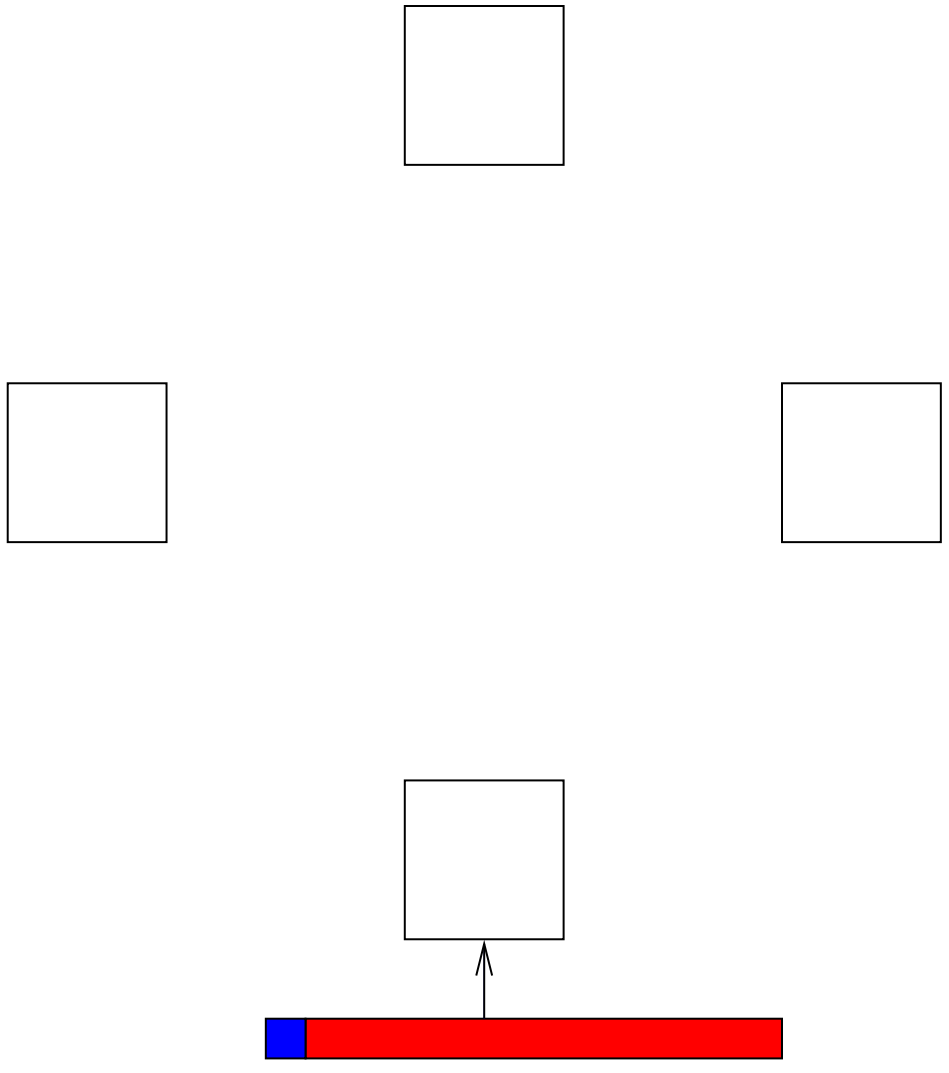
Multinode = inevitable delay

Ethernet latency = 0.1 milliseconds = 10^5 cycles = many examples.

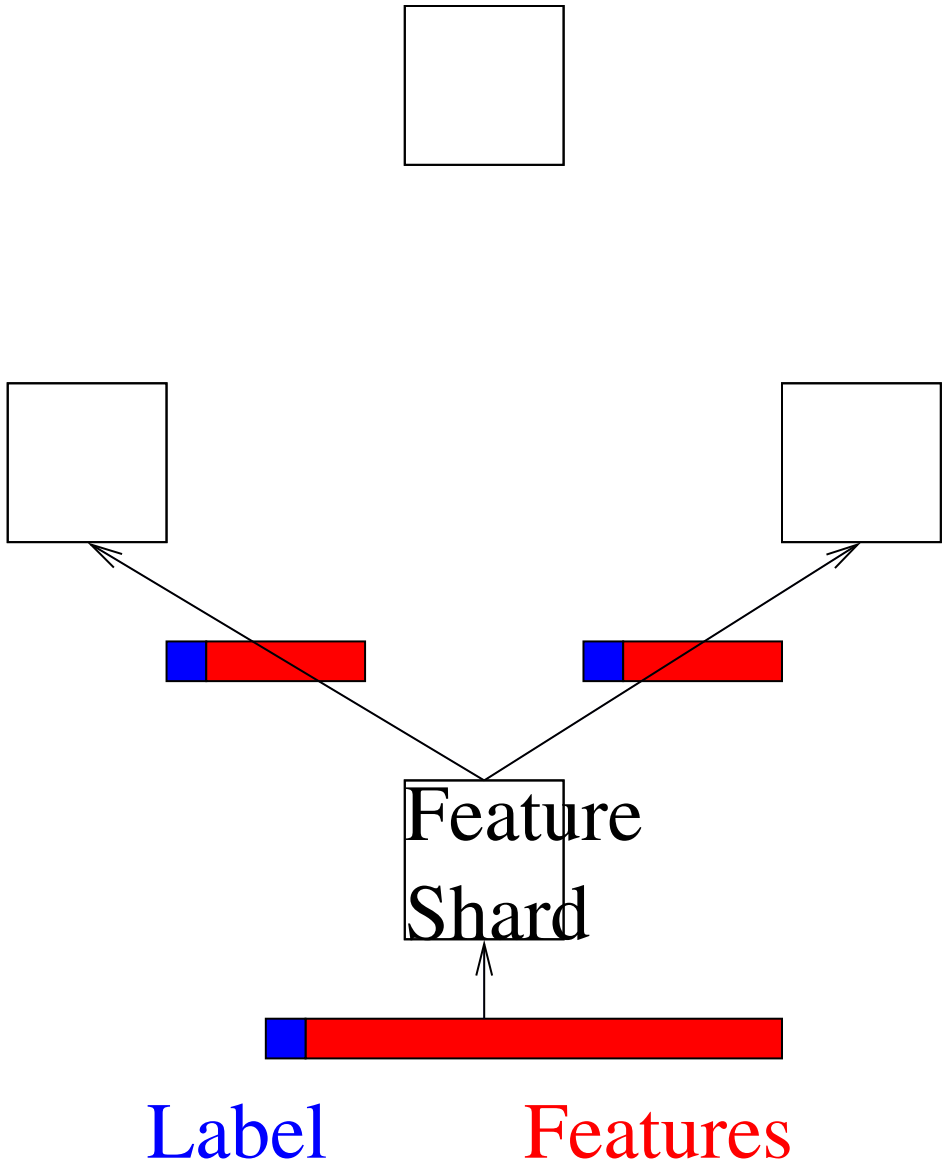
1. Example Sharding $\Rightarrow$ weights out of sync by delay factor.
2. Feature Sharding $\Rightarrow$ global predictions delayed by delay factor.

How bad is delay?
Theorem: (Mesterharm 2005) Delayed updates reduce convergence by delay factor in worst case for expert algorithms.
Theorem: (LSZ NIPS 2009) Same for linear predictors.
(Caveat: there are some special cases where you can do better.)
Empirically: Delay can hurt substantially when examples are structured.
What do we do?

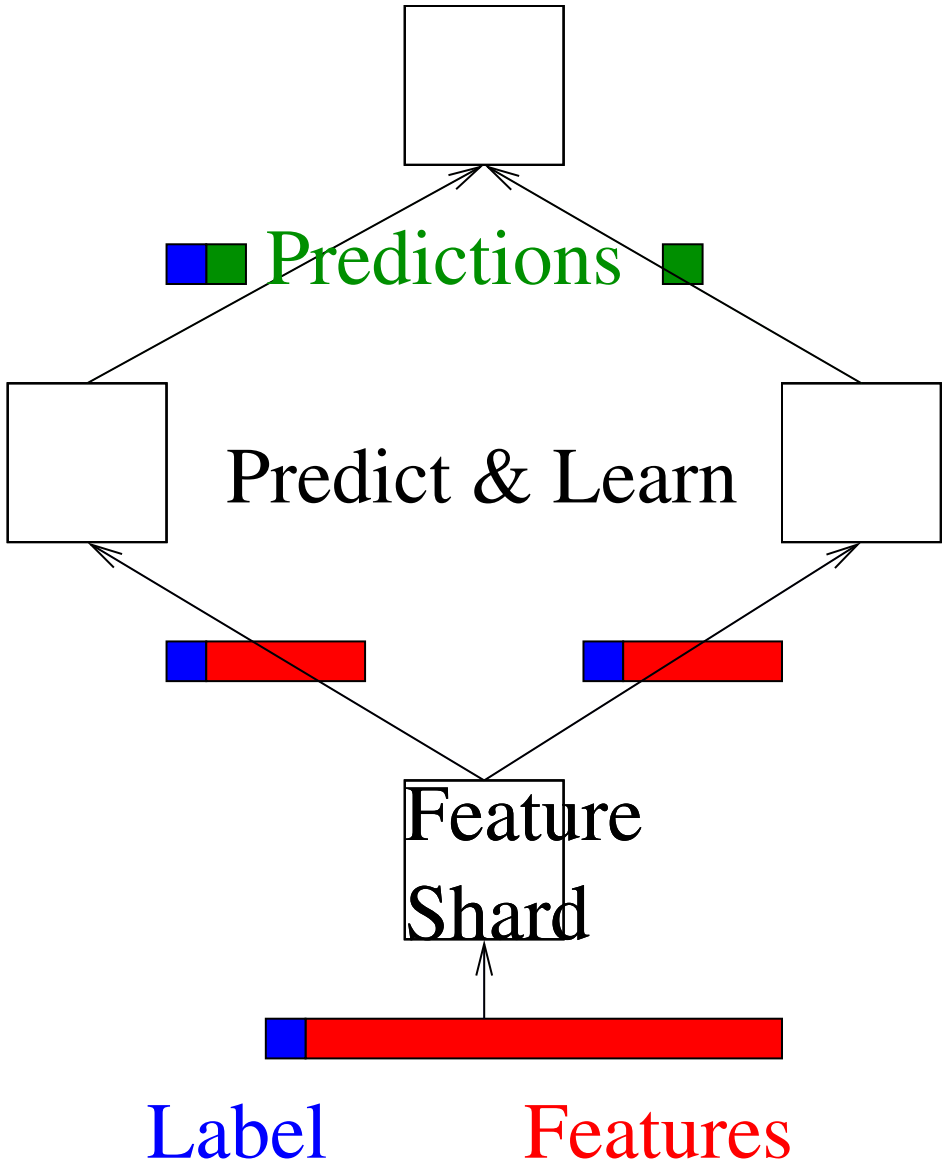
How can we avoid delay?



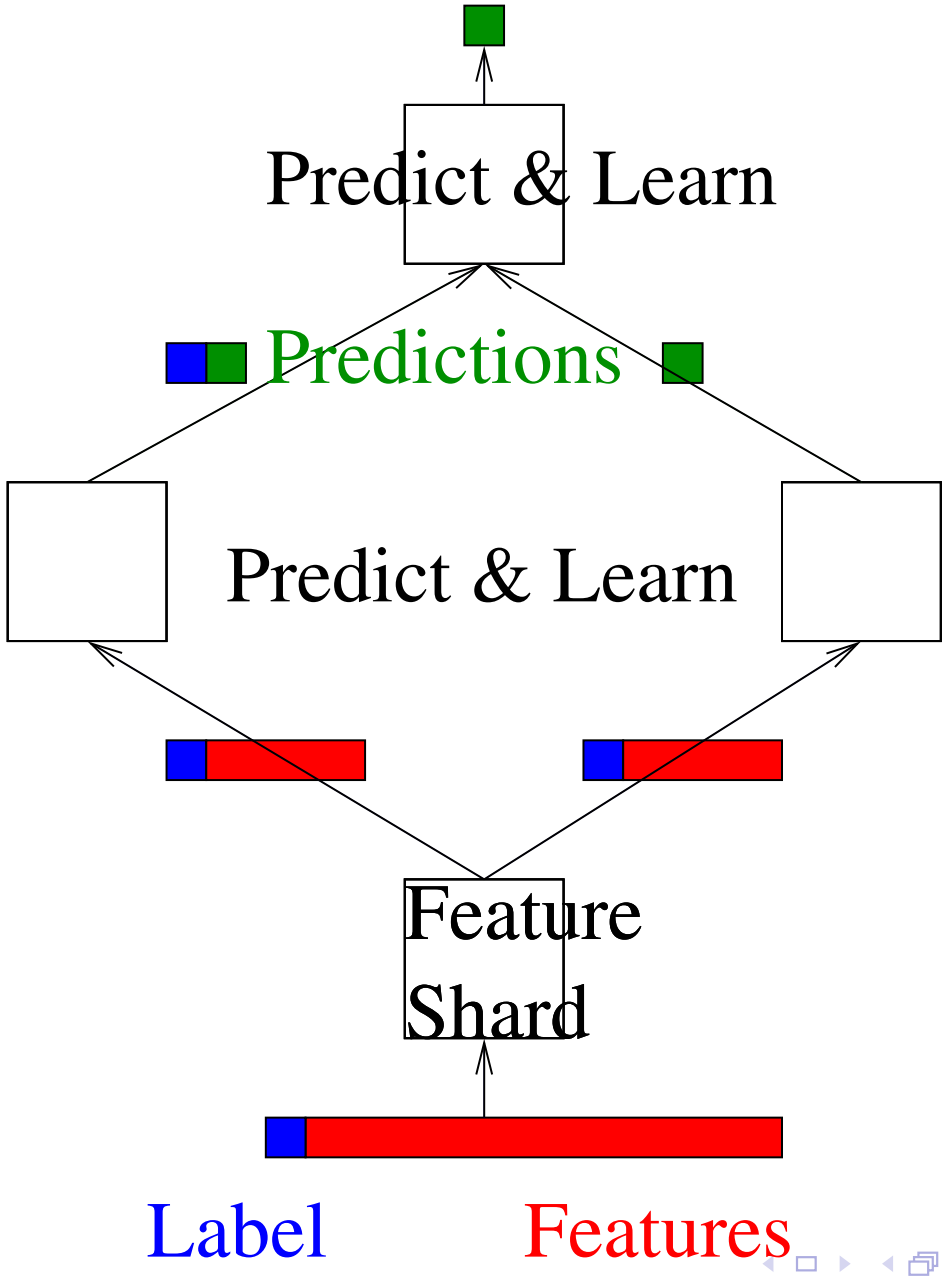
How can we avoid delay?



How can we avoid delay?



How can we avoid delay?



Observations about Feed Forward

1. No longer the same algorithm—it's designed for parallel environments.
2. Bandwidth = few bytes per example, per node $\Rightarrow$ Tera-example feasible with single master, arbitrarily more with hierarchical structure.
3. No delay.
4. Feature Shard is stateless $\Rightarrow$ parallelizable & cachable.

Bad News: Feed Forward can't compete with general linear predictors

Probability	y	x_1	x_2	x_3
0.25	1	1	1	0
0.125	1	1	0	1
0.125	1	0	1	1
0.25	0	0	0	1
0.125	0	1	0	0
0.125	0	0	1	0

Features 1&2 are imperfect predictors. Feature 3 is uncorrelated with truth. Optimal predictor = majority vote on all 3 features.

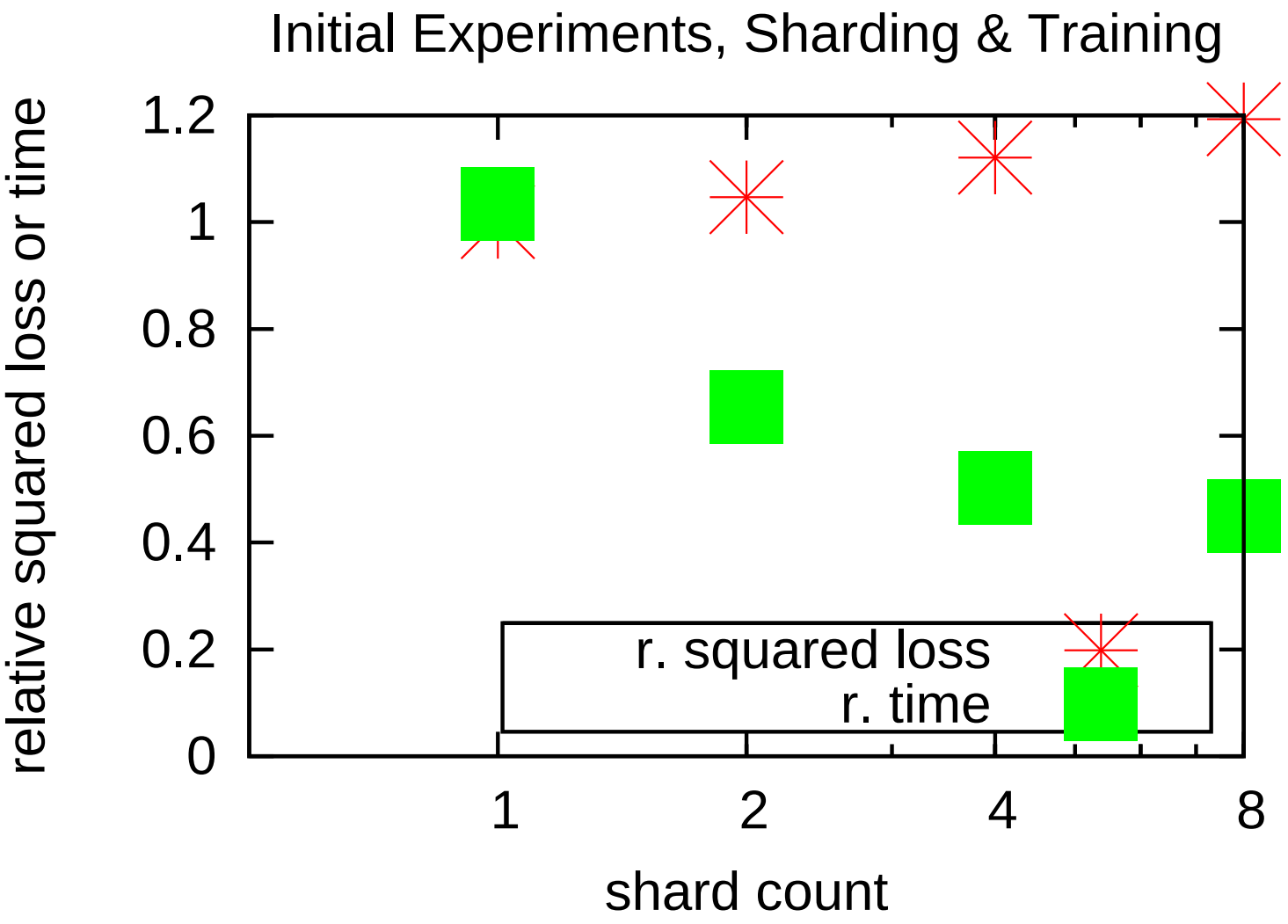
Good news

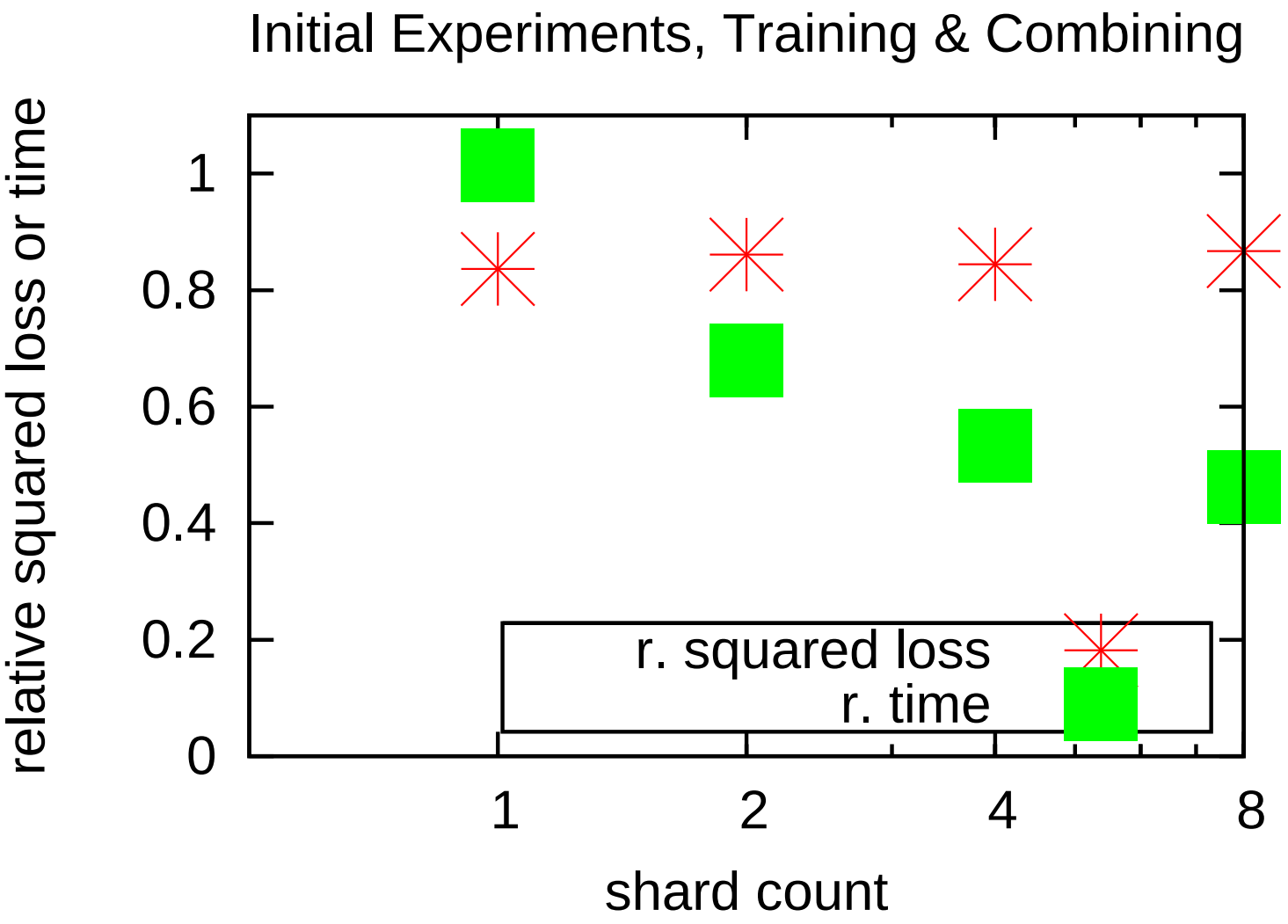
If Naive Bayes holds $P(x_1|y)P(x_2|y) = P(x_1, x_2|y)$, you win.
Better news: x_1 = first shard, x_2 = second shard
Even better: There are more complex problem classes for which this also works.

Initial experiments on a medium size text Ad dataset @ Yahoo!

1. $\sim$100GB when gzip compressed.
2. $\sim$10M examples.
3. $\sim$125G nonzero features
4. Uses outerproduct features

Relative progressive validation (BKL COLT 1999) squared loss & relative wall-clock time reported.


Initial Experiments, Sharding & Training
relative squared loss or time
1.2
1
0.8
0.6
0.4
0.2
0
1
2
4
8
shard count
r. squared loss
r. time


Initial Experiments, Training & Combining
relative squared loss or time
1
0.8
0.6
0.4
0.2
0
1
2
4
8
shard count
r. squared loss
r. time

Final thoughts

About x6 speedup achieved over sequential system so far.
This general approach, unlike averaging approaches, is fully applicable to nonlinear systems.
Code at: http://github.com/JohnLangford/vowpal_wabbit
Patches welcome. Much more work needs to be done.
Some further discussion @ http://hunch.net