Vowpal Wabbit 5.0 

http://hunch.net/~vw/

John Langford, Nikos Karampatziakis, Daniel Hsu,
Matt Hoffman

Yahoo! Research

git clone git://github.com/JohnLangford/vowpal_wabbit.git

Why VW?

1. There should exist an open source online learning system.

2. Online learning $\Rightarrow$ online optimization, which is or competes with best practice for many learning algorithms.

3. VW is a multitrick pony, all useful, many orthogonally composable. [hashing, caching, parallelizing, feature crossing, features splitting, feature combining, etc...]

4. It's simple. No strange dependencies, currently only 6255 lines of code.

On RCV1, training time = ~3s [caching, pipelining]

On "large scale learning challenge" datasets $\leq$ 10 minutes [caching]

[ICML 2009] 10^5-way personalize spam filter. [-q, hashing]

[UAI 2009] 10^6-way conditional probability estimation. [library, hashing]

[Rutgers grad] Gexample/day data feed. [–daemon]

[Matt Hoffman] LDA-100 on 2.5M Wikipedia in 1 hour.

[Paul Mineiro] True Love @ eHarmony

[Stock Investors] Unknown

The Tutorial Plan

1. John: Baseline & Conjugate Gradient.
2. Nikos: Importance Aware & Adaptive updates.
3. Daniel: Absurdly fast agnostic active learning.
4. Matt: Efficient Online LDA.
5. 15+ minute break before the real workshop.

Ask Questions!

The basic learning algorithm (classic)

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

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

Input Format

Label [Importance] [Tag]|Namespace Feature ... |Namespace Feature ... ... \n

Namespace = String[:Float]

Feature = String[:Float]

Feature and Label are what you expect.

Importance is multiplier on learning rate.

Tag is an identifier for an example, echoed on example output.

Namespace is a mechanism for feature manipulation and grouping.

Valid input examples

1 | 13:3.96e-02 24:3.47e-02 69:4.62e-02

example_39|excuses the dog ate my homework

1 0.500000 example_39|excuses:0.1 the:0.01 dog ate my homework |teacher male white Bagnell AI ate breakfast

Example Input Options

[-d] [–data] <f> : Read examples from f. Multiple ⇒ use all

cat <f> | vw : read from stdin

–daemon : read from port 39524

–port <p> : read from port p

–passes <n> : Number of passes over examples. Can't multipass a noncached stream.

-c [–cache] : Use a cache (or create one if it doesn't exist).

–cache_file <fc> : Use the fc cache file. Multiple ⇒ use all. Missing ⇒ create. Multiple+missing ⇒ concatenate

–compressed <f>: Read a gzip compressed file.

Example Output Options

Default diagnostic information:

Progressive Validation, Example Count, Label, Prediction, Feature Count

-p [–predictions] <po>: File to dump predictions into.

-r [–raw_predictions] <ro> : File to output unnormalized prediction into.

–sendto <host[:port]> : Send examples to host:port.

–audit : Detailed information about feature_name: feature_index: feature_value: weight_value

–quiet : No default diagnostics

Example Manipulation Options

-t [–testonly] : Don't train, even if the label is there.

-q [–quadratic] <ab>: Cross every feature in namespace a* with every feature in namespace b*.

Example: -q et (= extra feature for every excuse feature and teacher feature)

–sort_features: Sort features for small cache files.

–ngram <N>: Generate N-grams on features. Incompatible with sort_features

–skips <S>: ...with S skips.

–hash all: hash even integer features.

Update Rule Options

–decay_learning_rate <d> [= 1]

–initial_t <i> [= 1]

–power_t <p> [= 0.5]

-l [–learning_rate] <l> [= 0.1]

$$\eta_e = \frac{l d^{n-1} i^p}{(i + \sum_{e' < e} i_{e'})^p}$$

Basic observation: there exists no one learning rate satisfying all uses.

Example: state tracking vs. online optimization.

–loss_function {squared,log,hinge,quantile} Switch loss function

Weight Options

-b [–bit_precision] <b> [=18] : Number of weights. Too many features in example set⇒ collisions occur.

-i [–initial_regressor] <ri> : Initial weight values. Multiple ⇒ average.

-f [–final_regressor] <rf> : File to store final weight values in.

–random_weights <r>: make initial weights random. Particularly useful with LDA.

–initial_weight <iw>: Initial weight value

Useful Parallelization Options

–thread-bits <b> : Use 2^b threads for multicore. Introduces some nondeterminism (floating point add order). Only useful with -q

–multisource : Assemble examples piecemeal from multiple sources. For cluster parallelism.

–predictto <host[:port]> : Send prediction to host:port. Use with –multisource

Experimental Parallelization Options

–unique_id <i>: Identify nodes in a parallel environment.

–corrective: correct local update when global information arrives.

–backprop: use backprop when global information arrives.

–global_multiplier <m>: multiplier on backprop updates.

–delayed_global: use delayed global updates.

Conjugate Gradient Options

–conjugate_gradient: Use batch mode preconditioned conjugate gradient learning. 2 passes/update. Output predictor compatible with base algorithm. Requires more RAM. Uses cool trick:

$$d^T H d = \frac{\partial^2 l(z)}{\partial^2 z} \langle x, d \rangle^2$$

–regularization <r>: Add r time the weight magnitude to the optimization. Reasonable choice = 0.001.

"I have a better loss function"

1. Implement in loss_functions.cc.

2. Send a patch / github pull request.

"My online learning algorithm is better."

1. Copy {gd.cc, cg.cc, lda.cc, sender.cc, noop.cc} to a new file and tweak.

2. Add flag to parse_args.cc

3. Implement flag in vw.cc

4. Send a patch / github pull request.

Goals for Future Development

1. Finish scaling up. I want a kilonode program.
2. Native learning reductions. Just like more complicated losses.
3. Other learning algorithms, as interest dictates.
4. Persistent Daemonization.