Vowpal Wabbit 5.1

http://hunch.net/~vw/

John Langford

Yahoo! Research

{With help from Nikos Karampatziakis & Daniel Hsu}

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 6338 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. Baseline online linear algorithm
2. Common Questions.
3. Importance Aware Updates
4. Adaptive updates.
5. Conjugate Gradient.
6. Active Learning.

Missing: Online LDA: See Matt's slides

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 $w_i \leftarrow w_i + \eta 2(y - \hat{y}) I x_i$ 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)
–ignore <a>: Remove a namespace and all features in it.
–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> [= 10]

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

Basic observation: there exists no one learning rate satisfying all uses.
Example: state tracking vs. online optimization.
–loss_function {squared,logistic,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
(There are other experimental cluster parallel options.)

The Tutorial Plan

1. Baseline online linear algorithm
2. Common Questions.
3. Importance Aware Updates
4. Adaptive updates.
5. Conjugate Gradient.
6. Active Learning.

Missing: Online LDA: See Matt's slides

Ask Questions!

How do I choose good features?

Think like a physicist: Everything has units.
Let x_i be the base unit. Output $\langle w \cdot x \rangle$ has unit "probability", "median", etc...
So predictor is a unit transformation machine.
The ideal w_i has units of $\frac{1}{x_i}$ since doubling feature value halves weight.
Update $\propto \frac{\partial L_w(x)}{\partial w} \simeq \frac{\Delta L_w(x)}{\Delta w}$ has units of x_i.
Thus update $= \frac{1}{x_i} + x_i$ unitwise, which doesn't make sense.

Implications

1. Choose x_i near 1, so units are less of an issue.
2. Choose x_i on a similar scale to x_j so unit mismatch across features doesn't kill you.
3. Use other updates which fix the units problem (later).

General advice:

1. Many people are happy with TFIDF = weighting sparse features inverse to their occurrence rate.
2. Choose features for which a weight vector is easy to reach as a combination of feature vectors.

How do I choose a Loss function?

Understand loss function semantics.

1. Minimizer of squared loss = conditional expectation. $f(x) = E[y|x]$ (default).
2. Minimizer of quantile = conditional quantile. $\Pr(y > f(x)|x) = \tau$
3. Hinge loss = tight upper bound on 0/1 loss.
4. Minimizer of logistic = conditional probability: $\Pr(y = 1|x) = f(x)$. Particularly useful when probabilities are small.

Hinge and logistic require labels in $\{-1, 1\}$.

How do I choose a learning rate?

1. Are you trying to *track* a changing system? –power_t 0 (forget past quickly).
2. If the world is adversarial: –power_t 0.5 (default)
3. If the world is iid: –power_t 1 (very aggressive)
4. If the error rate is small: -l <large>
5. If the error rate is large: -l <small> (for integration)
6. If –power_t is too aggressive, setting –initial_t softens initial decay.
7. For multiple passes –decay_learning_rate in $[0.5, 1]$ is sensible. values < 1 protect against overfitting.

How do I order examples?

There are two choices:

1. Time order, if the world is nonstationary.
2. Permuted order, if not.

A bad choice: all label 0 examples before all label 1 examples.

How do I debug?

1. Is your progressive validation loss going down as you train? (no => malordered examples or bad choice of learning rate)
2. If you test on the train set, does it work? (no => something crazy)
3. Are the predictions sensible?
4. Do you see the right number of features coming up?

How do I figure out which features are important?

1. Save state
2. Create a super-example with all features
3. Start with –audit option
4. Save printout.

(Seems whacky: but this works with hashing.)

How do I efficiently move/store data?

1. Use –noop and –cache to create cache files.
2. Use –cache multiple times to use multiple caches and/or create a supercache.
3. Use –port and –sendto to ship data over the network.
4. –compress generally saves space at the cost of time.

How do I avoid recreating cachefiles as I experiment?

1. Create cache with -b <large>, then experiment with -b <small>.
2. Partition features intelligently across namespaces and use –ignore <f>.

The Tutorial Plan

1. Baseline online linear algorithm
2. Common Questions.
3. Importance Aware Updates
4. Adaptive updates.
5. Conjugate Gradient.
6. Active Learning.

Missing: Online LDA: See Matt's slides

Ask Questions!

Examples with importance weights

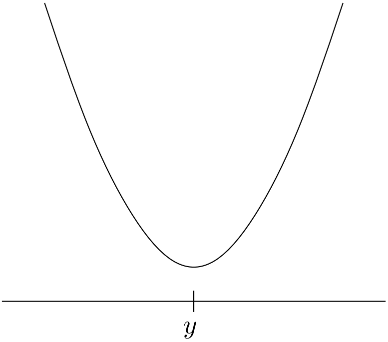
The preceeding is not correct (use "–loss_function classic" if you want it).
The update rule is actually importance invariant, which helps substantially.

Principle

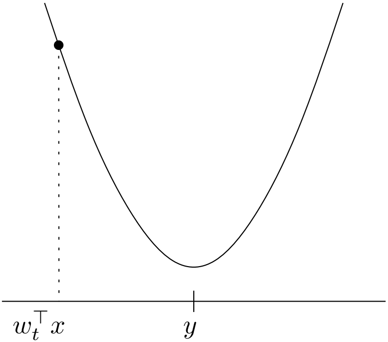
Having an example with importance weight h should be equivalent to having the example h times in the dataset.

(Karampatziakis & Langford, http://arxiv.org/abs/1011.1576 for details.)

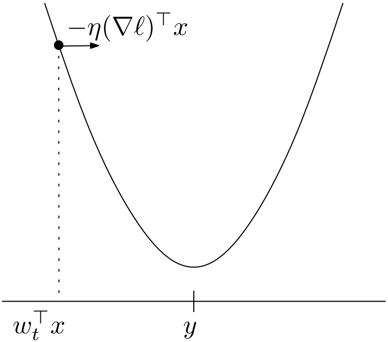
Learning with importance weights



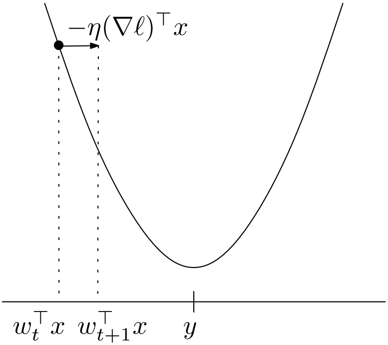
Learning with importance weights



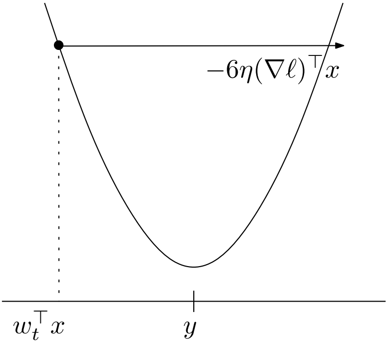
Learning with importance weights



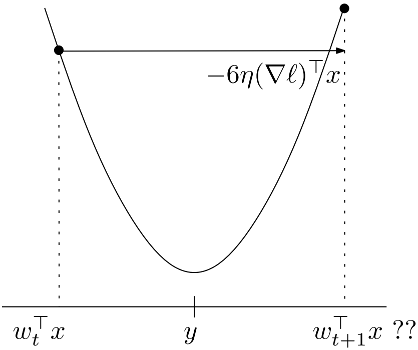
Learning with importance weights



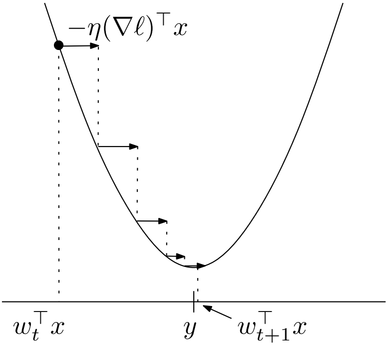
Learning with importance weights



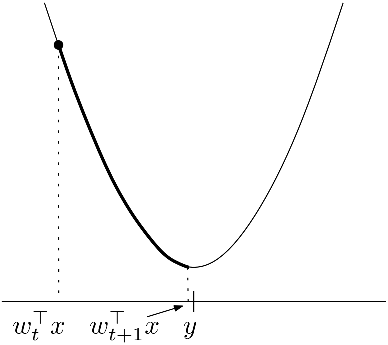
Learning with importance weights



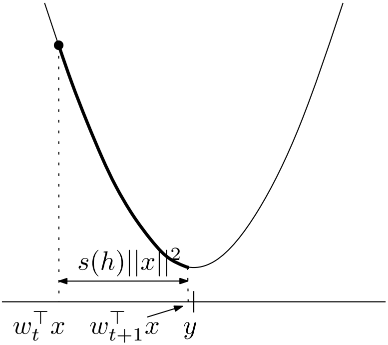
Learning with importance weights



Learning with importance weights



Learning with importance weights



What is $s(\cdot)$?

Take limit as update size goes to 0 but number of updates goes to ∞.

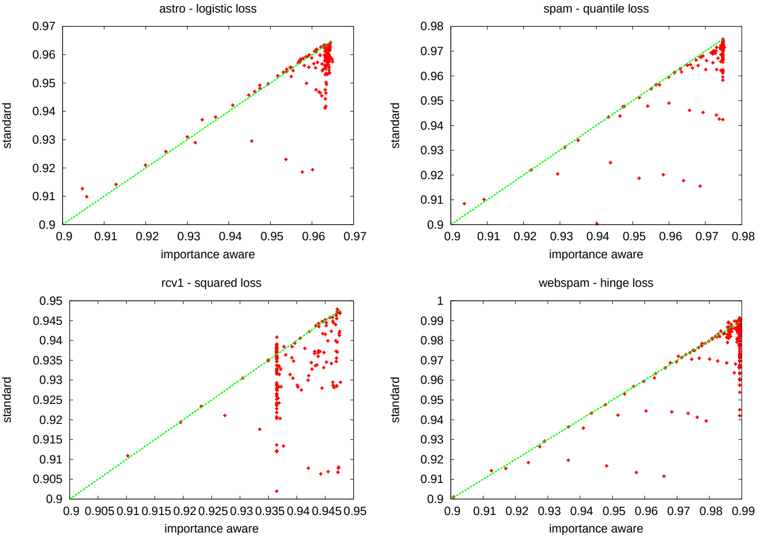
What is $s(\cdot)$?

Take limit as update size goes to 0 but number of updates goes to ∞.
Surprise: simplifies to closed form.

Loss	$\ell(p, y)$	Update $s(h)$
Squared	$(y - p)^2$	$\frac{p-y}{x^\top x}\left(1 - e^{-h\eta x^\top x}\right)$
Logistic	$\log(1 + e^{-yp})$	$\frac{W(e^{h\eta x^\top x + yp + e^{yp}}) - h\eta x^\top x - e^{yp}}{yx^\top x}$ for $y \in \{-1, 1\}$
Hinge	$\max(0, 1 - yp)$	$-y \min\left(h\eta, \frac{1-yp}{x^\top x}\right)$ for $y \in \{-1, 1\}$
τ-Quantile	if $y > p$ $\quad \tau(y - p)$ if $y \leq p$ $\quad (1 - \tau)(p - y)$	if $y > p$ $\quad -\tau \min(h\eta, \frac{y-p}{\tau x^\top x})$ if $y \leq p$ $\quad (1 - \tau)\min(h\eta, \frac{p-y}{(1-\tau)x^\top x})$

+ many others worked out. Similar in effect to "implicit gradient", but closed form.

Robust results for unweighted problems



The Tutorial Plan

1. Baseline online linear algorithm
2. Common Questions.
3. Importance Aware Updates
4. Adaptive updates.
5. Conjugate Gradient.
6. Active Learning.

Missing: Online LDA: See Matt's slides

Ask Questions!

Adaptive Updates

- Adaptive, individual learning rates in VW.
- It's really gradient descent separately on each coordinate i with

$$\eta_{t,i} = \frac{1}{\sqrt{\sum_{s=1}^{t} \left(\frac{\partial \ell(w_s^\top x_s, y_s)}{\partial w_{s,i}} \right)^2}}$$

- Coordinate-wise scaling of the data less of an issue (units issue addressed) see (Duchi, Hazan, and Singer / McMahan and Streeter, COLT 2010)
- Requires x2 RAM at learning time, but learned regressor is compatible.

Some tricks involved

- Store sum of squared gradients w.r.t w_i near w_i.
- ```
  float InvSqrt(float x){
     float xhalf = 0.5f * x;
     int i = *(int*)&x;
     i = 0x5f3759d5 - (i >> 1);
     x = *(float*)&i;
     x = x*(1.5f - xhalf*x*x);
     return x;
  }
  ```
  Special SSE rsqrt instruction is a little better
  ```

Experiments

- Raw Data

```
./vw --adaptive -b 24 --compressed -d tmp/spam_train.gz
average loss = 0.02878
./vw -b 24 --compressed -d tmp/spam_train.gz -l 100
average loss = 0.03267
```

- TFIDF scaled data

```
./vw --adaptive --compressed -d tmp/rcv1_train.gz -l 1
average loss = 0.04079
./vw --compressed -d tmp/rcv1_train.gz -l 256
average loss = 0.04465
```

The Tutorial Plan

1. Baseline online linear algorithm
2. Common Questions.
3. Importance Aware Updates
4. Adaptive updates.
5. Conjugate Gradient.
6. Active Learning.

Missing: Online LDA: See Matt's slides

Ask Questions!

Preconditioned Conjugate Gradient Options

–conjugate_gradient: Use batch mode preconditioned conjugate gradient learning. 2 passes/update. Output predictor compatible with base algorithm. Requires x5 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.

Works well with logistic or squared loss.

What is Conjugate Gradient?

1. Compute average gradient (one pass).
2. Mix gradient with previous step direction to get new step direction.
3. Compute step size using Newton's method. (one pass)
4. Update weights.

Step 2 is particular.
"Precondition" = reweight dimensions.

Why Conjugate Gradient?

Addresses the “units” problem.

A decent batch algorithm—requires 10s of passes sufficient.

Learned regressor is compatible.

See Jonathan Shewchuk’s tutorial for more details.

The Tutorial Plan

1. Baseline online linear algorithm
2. Common Questions.
3. Importance Aware Updates
4. Adaptive updates.
5. Conjugate Gradient.
6. Active Learning.

Missing: Online LDA: See Matt's slides

Ask Questions!

Importance Weighted Active Learning (IWAL) [BDL'09]

$S = \emptyset$

For $t = 1, 2, \ldots$ until no more unlabeled data

1. Receive unlabeled example x_t.
2. Choose a probability of labeling p_t.
3. With probability p get label y_t, and add $(x_t, y_t, \frac{1}{p_t})$ to S.
4. Let $h_t = \text{Learn}(S)$.

New instantiation of IWAL

[BHLZ'10]: strong consistency / label efficiency guarantees by using

$$p_t = \min\left\{1,\ C \cdot \left(\frac{1}{\Delta_t^2} \cdot \frac{\log t}{t-1}\right)\right\}$$

where Δ_t = increase in training error rate if learner is forced to change its prediction on the new unlabeled point x_t.

New instantiation of IWAL

[BHLZ'10]: strong consistency / label efficiency guarantees by using

$$p_t = \min\left\{1,\ C \cdot \left(\frac{1}{\Delta_t^2} \cdot \frac{\log t}{t-1}\right)\right\}$$

where Δ_t = increase in training error rate if learner is forced to change its prediction on the new unlabeled point x_t.

Using VW as base learner, estimate $t \cdot \Delta_t$ as the importance weight required for prediction to switch. For square-loss update:

$$\Delta_t := \frac{1}{t \cdot \eta_t} \cdot \log \frac{\max\{h(x_t),\ 1 - h(x_t)\}}{0.5}$$

Active learning in Vowpal Wabbit

Simulating active learning: (tuning paramter $C > 0$)

`vw -active_simulation -active_mellowness C`

(increasing $C \to \infty$ = supervised learning)

Active learning in Vowpal Wabbit

Simulating active learning: (tuning paramter $C > 0$)

`vw -active_simulation -active_mellowness C`

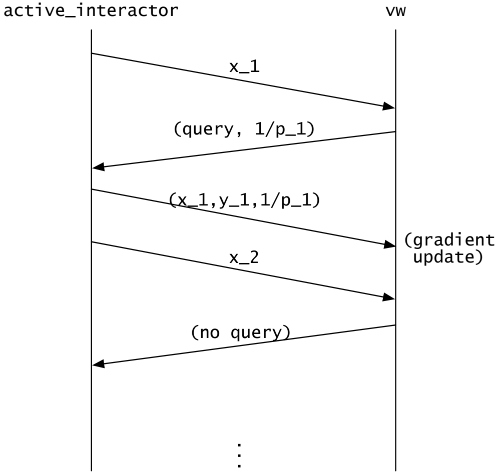
(increasing $C \to \infty$ = supervised learning)

Deploying active learning:

`vw -active_learning -active_mellowness C -daemon`

- `vw` interacts with an `active_interactor` (`ai`)
- for each unlabeled data point, `vw` sends back a query decision (+importance weight)
- `ai` sends labeled importance-weighted examples as requested
- `vw` trains using labeled weighted examples

Active learning in Vowpal Wabbit



`active_interactor.cc` (in git repository) demonstrates how to implement this protocol.

Active learning simulation results

RCV1 (text binary classification task):

training:

```
vw -active_simulation -active_mellowness 0.000001
   -d rcv1-train -f active.reg -l 10 -initial_t 10
```

number of examples = 781265

total queries = 98074 (*i.e.*, $<$ 13% of the examples)

(caveat: progressive validation loss not reflective of test loss)

Active learning simulation results

RCV1 (text binary classification task):

training:

```
vw -active_simulation -active_mellowness 0.000001
   -d rcv1-train -f active.reg -l 10 -initial_t 10
```

number of examples = 781265

total queries = 98074 (*i.e.*, $< 13\%$ of the examples)

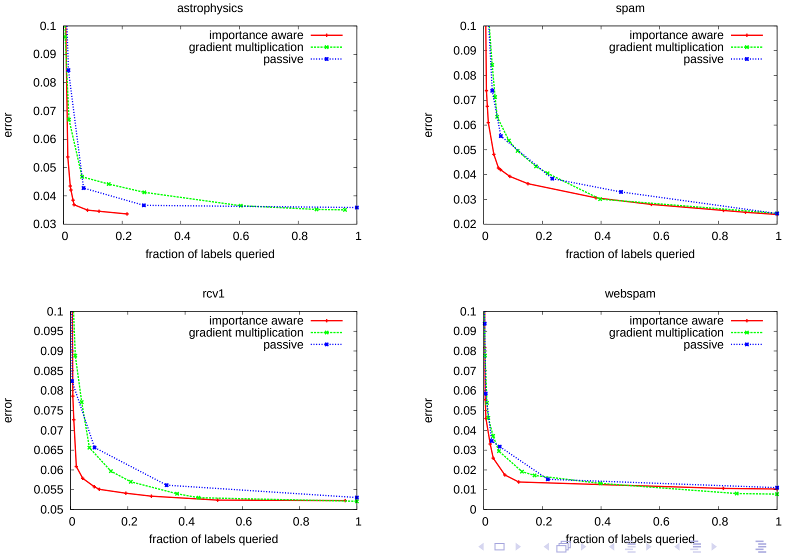
(caveat: progressive validation loss not reflective of test loss)

testing:

```
vw -t -d rcv1-test -i active.reg
```

average loss = 0.04872 (better than supervised)

Active learning simulation results



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.