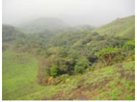
Normalized Online Learning Tutorial

Paul Mineiro
joint work with Stephane Ross & John Langford

December 9th, 2013

Motivation: Covertype Data Set



54 total features

Name	Units
Elevation, Distance to X	meters
Aspect, Slope	degrees
Hillshade at time t	"hillshade index" (0-255)
Wilderness Area	$\{0, 1\}^4$
Soil Type	$\{0, 1\}^{40}$

The Geometry of Real Data

- In practice, features often have different scales.

The Geometry of Real Data

- In practice, features often have different scales.
- This is a problem for first-order online learning methods.

The Geometry of Real Data

- In practice, features often have different scales.
- This is a problem for first-order online learning methods.
- Example: "vanilla" online GD regret:

$$R \leq \sqrt{T} ||w^*||_2 \max_{t \in 1:T} ||g_t||_2$$

The Geometry of Real Data

- In practice, features often have different scales.
- This is a problem for first-order online learning methods.
- Example: "vanilla" online GD regret:

$$R \leq \sqrt{T} ||w^*||_2 \max_{t \in 1:T} ||g_t||_2$$

- This can be made arbitrarily bad in only two dimensions by scaling one of the dimensions while leaving the other fixed. <u>Not an artifact of the analysis.</u>

Example

- Generate data like this

$$
\begin{aligned}
x_1 &\sim N(0, 1) \\
x_2 &\sim N(0, \sqrt{s}) \\
z &\sim N(x_1 + \frac{1}{s} x_2, 1)
\end{aligned}
$$

- Do squared-loss prediction of z.
- NB: x_2 is statistically identical to x_1 scaled by s.

Example

Demo

Summary of Demo

- Un-normalized learning
  - Lots of fiddling with learning rate.
  - Slow convergence at extreme scales.
- Normalized learning
  - No fiddling with learning rate.
  - Same convergence across different scales.

On "Non-Demo" Datasets

- Un-normalized learning
  - Lots of fiddling with learning rate.
  - Slow convergence at extreme scales.
- Normalized learning
  - ~~No~~ Less fiddling with learning rate.
  - ~~Same~~ Similar convergence across different scales.

How it Works (Mechanically)

- Intuition: if feature i scaled by s, then j^{th} coordinate of w^* should be scaled by $1/s$.

How it Works (Mechanically)

- Intuition: if feature i scaled by s, then j^{th} coordinate of w^* should be scaled by $1/s$.
- Ergo:

How it Works (Mechanically)

- Intuition: if feature i scaled by s, then j^{th} coordinate of w^* should be scaled by $1/s$.
- Ergo:
  - Algorithm keeps track of $\max_{s<t} |x_i^{(s)}|$ for each j.

How it Works (Mechanically)

- Intuition: if feature i scaled by s, then j^{th} coordinate of w^* should be scaled by $1/s$.
- Ergo:
  - Algorithm keeps track of $\max_{s<t} |x_i^{(s)}|$ for each j.
  - When $|x_i^{(t)}| > \max_{s<t} |x_i^{(s)}|$, rescale w_i via

$$w_i \leftarrow w_i \frac{\max_{s<t} |x_i^{(s)}|}{x_i^{(t)}}.$$

How it Works (Mechanically) II

- Intuition: learning rate parameter should control average change in the prediction.

How it Works (Mechanically) II

- Intuition: learning rate parameter should control average change in the prediction.
- But: gradient is proportional to input size.

How it Works (Mechanically) II

- Intuition: learning rate parameter should control average change in the prediction.
- But: gradient is proportional to input size.
- Ergo:

How it Works (Mechanically) II

- Intuition: learning rate parameter should control average change in the prediction.
- But: gradient is proportional to input size.
- Ergo:
  - Divide each ∂/∂_i by $\max_{s \leq t} \left|x_i^{(s)}\right|$, and ...

How it Works (Mechanically) II

- Intuition: learning rate parameter should control average change in the prediction.
- But: gradient is proportional to input size.
- Ergo:
  - Divide each ∂ / ∂_i by $\max_{s \leq t} |x_i^{(s)}|$, and ...
  - Normalize the entire update by the average change in prediction N_t / t, where

$$N_t = N_{t-1} + \sum_i \frac{(x_i^{(t)})^2}{(\max_{s \leq t} |x_i^{(s)}|)^2}$$

How it Works (Mechanically) II

- Intuition: learning rate parameter should control average change in the prediction.
- But: gradient is proportional to input size.
- Ergo:
  - Divide each ∂ / ∂_i by $\max_{s \leq t} |x_i^{(s)}|$, and ...
  - Normalize the entire update by the average change in prediction N_t / t, where

$$N_t = N_{t-1} + \sum_i \frac{(x_i^{(t)})^2}{(\max_{s \leq t} |x_i^{(s)}|)^2}$$

  - Intuition behind N_t: if this is an example with small x_i, prediction is not changing very fast because gradient is normalized by scale.

When it fails

- Algorithm normalizes by scale estimate derived from history.

When it fails

- Algorithm normalizes by scale estimate derived from history.
- If the scale suddenly gets very large near the end of the input sequence, the scale estimates have been poor for most of the updates.

When it fails

- Algorithm normalizes by scale estimate derived from history.
- If the scale suddenly gets very large near the end of the input sequence, the scale estimates have been poor for most of the updates.
- Theorems are driven by $\Delta_i = \frac{\max_{t \in 1:T} |x_{ti}|}{|x_{t_0^i i}|}$.

How to use

- It is enabled by default in `vw`.

How to use

- It is enabled by default in `vw`.
- To not use:

How to use

- It is enabled by default in `vw`.
- To <u>not use</u>:

  `--adaptive --invariant`

How to use

- It is enabled by default in `vw`.
- To not use:

  `--adaptive --invariant`

  ...will you give vanilla AdaGrad without normalization.