Winnowing with Gradient Descent

Manfred K. Warmuth
Google Brain & UCSC

Foundations Reading Group
Deep Mind
11-23-2020

Joint work w. Ehsan Amid
Big picture overview

Connections to learning with kernels

Motivations of updates

Linear regression for GD ($\tau = 0$)

Reparameterization

Summary of updates and open problems
Winnow: to remove chaff from grain

Learning disjunctions when irrelevant attributes abound

$k$ out of $n$ literal disjunctions with $O(k \log n)$ mistakes

\[ \text{wheat} \quad \text{soy beans} \]
Notation of the Winnow algorithm

Learns disjunctions as linear threshold functions

▶ 2 out of 5 literal monotone disjunction $v_1 \lor v_3$

▶ Represented as $d = (1, 0, 1, 0, 0)^\top$

▶ Label for instance $x = (0, 1, 1, 0, 0)^\top$

$$
\begin{cases}
+1 & \text{if } d \cdot x \geq \frac{1}{2} \\
-1 & \text{otherwise}
\end{cases}
$$

▶ Alg. receives sequence of examples online

$$(x_1, y_1) (x_2, y_2), \ldots, (x_T, y_T)$$

$${\hat{y}}_1 {\hat{y}}_2 \ldots {\hat{y}}_T$$

instances $[0, 1]^n$, labels and predictions are $\pm 1$
Winnow algorithm

Initialize $w_1 = w_0 (1, 1, \ldots 1)^\top$

for $t = 1$ to $T$ do
  Receive instance $x_t \in [0, 1]^n$
  Predict with linear threshold
  $$\hat{y}_t = \begin{cases} +1 & \text{if } w_t \cdot x_t \geq \theta \\ -1 & \text{otherwise} \end{cases}$$
  Receive label $y_t \in \{+1, -1\}$
  Multiplicative update: $w_{t+1,i} = w_{t,i} \exp(-\eta(\hat{y}_t - y_i)x_{t,i})$
end for

$\leq k \log n$ mistakes

Perceptron alg., additive: $w_{t+1,i} = w_{t,i} - \eta (\hat{y}_t - y_i)x_{t,i}$

$\geq k n$ mistakes
Winnow algorithm

Initialize $w_1 = w_0 (1, 1, \ldots 1)^\top$

for $t = 1$ to $T$ do

Receive instance $x_t \in [0, 1]^n$

Predict with linear threshold

$$\hat{y}_t = \begin{cases} +1 & \text{if } w_t \cdot x_t \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

Receive label $y_t \in \{+1, -1\}$

Multiplicative update: $w_{t+1,i} = w_{t,i} \exp(-\eta(\hat{y}_t - y_i)x_{t,i})$

end for

$\leq k \log n$ mistakes

Perceptron alg., additive: $w_{t+1,i} = w_{t,i} - \eta (\hat{y}_t - y_i)x_{t,i}$

$\geq k n$ mistakes
Mirror descent [NY83]

\[ f(w_{s+1}) - f(w_s) = -\nabla L(w_s) \quad \text{(where } f \text{ is strictly increasing)} \]
\[ w_{s+1} = f^{-1}(f(w_s) - \nabla L(w_s)) \]

Gradient Descent (GD): \( f = \text{id} \)
\[ w_{s+1} - w_s = -\nabla L(w_s) \]
\[ w_{s+1} = w_s - \nabla L(w_s) \]

Unnormalized Exponentiated Gradient Descent (EGU): \( f = \log \)
\[ \log(w_{s+1}) - \log(w_s) = -\nabla L(w_s) \]
\[ w_{s+1,i} = w_{s,i} \exp(-\eta(\nabla L(w_s))_i) \quad \text{(now } w_i \geq 0) \quad [KW97] \]

Normalized version called Exponentiated Gradient (EG)
\[ w_{s+1,i} = \frac{w_{s,i} \exp(-\eta(\nabla L(w_s))_i)}{\sum_j w_{s,j} \exp(-\eta(\nabla L(w_s))_j)} \quad \text{(now } w \text{ prob.vect.)} \]
Mirror descent \[NY83\]

\[f(w_{s+1}) - f(w_s) = -\nabla L(w_s) \text{ (where } f \text{ is strictly increasing)}\]

\[w_{s+1} = f^{-1}(f(w_s) - \nabla L(w_s))\]

Gradient Descent (GD): \(f=\text{id}\)

\[w_{s+1} - w_s = -\nabla L(w_s)\]

\[w_{s+1} = w_s - \nabla L(w_s)\]

Unnormalized Exponentiated Gradient Descent (EGU): \(f = \log\)

\[\log(w_{s+1}) - \log(w_s) = -\nabla L(w_s)\]

\[w_{s+1,i} = w_{s,i} \exp(-\eta(\nabla L(w_s))_i) \text{ (now } w_i \geq 0) \quad [KW97]\]

Normalized version called Exponentiated Gradient (EG)

\[w_{s+1,i} = \frac{w_{s,i} \exp(-\eta(\nabla L(w_s))_i)}{\sum_j w_{s,j} \exp(-\eta(\nabla L(w_s))_j)} \text{ (now } w \text{ prob.vect.)}\]
Mirror descent \[ \text{[NY83]} \]

\[
f(w_{s+1}) - f(w_s) = -\nabla L(w_s) \quad \text{(where } f \text{ is strictly increasing)}
\]

\[
w_{s+1} = f^{-1}(f(w_s) - \nabla L(w_s))
\]

Gradient Descent (GD): \( f = \text{id} \)

\[
w_{s+1} = w_s - \nabla L(w_s)
\]

Unnormalized Exponentiated Gradient Descent (EGU): \( f = \log \)

\[
\log(w_{s+1}) - \log(w_s) = -\nabla L(w_s)
\]

\[
w_{s+1,i} = w_{s,i} \exp(-\eta(\nabla L(w_s))_i) \quad \text{(now } w_i \geq 0) \quad \text{[KW97]}
\]

Normalized version called Exponentiated Gradient (EG)

\[
w_{s+1,i} = \frac{w_{s,i} \exp(-\eta(\nabla L(w_s))_i)}{\sum_j w_{s,j} \exp(-\eta(\nabla L(w_s))_j)} \quad \text{(now } w \text{ prob. vect.})
\]
Major differences between the two families

GD: stochastic gradient descent, backprop, kernel methods
EG: Winnow, expert algorithms, Boosting, Bayes

Performance of GD linear in $n$ for sparse targets

Performance of EG linear in $\log n$ for sparse targets

Here we will reparameterize EG as GD:
Reparameterized forms act like original EG

Winnowing with GD!
Major differences between the two families

GD: stochastic gradient descent, backprop, kernel methods
EG: Winnow, expert algorithms, Boosting, Bayes

Performance of GD linear in $n$ for sparse targets

Performance of EG linear in $\log n$ for sparse targets

Here we will reparameterize EG as GD:
Reparameterized forms act like original EG

Winnowing with GD!
Paradigmic sparse linear problem

\[
\begin{pmatrix}
-1 & -1 & 1 & -1 & -1 \\
1 & -1 & -1 & -1 & -1 \\
1 & -1 & 1 & -1 & -1 \\
1 & 1 & -1 & 1 & 1 \\
1 & -1 & 1 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
0 \\
0 \\
0 \\
1 \\
0
\end{pmatrix}
= \begin{pmatrix}
-1 \\
-1 \\
1 \\
1 \\
1
\end{pmatrix}
\]

± matrix random or Hadamard

After receiving example \((x_t, y_t)\) and incurring loss \((x_t^\top w_t - y_t)^2\) update:

- multiplicative, EGU: 
  \[w_{t+1,i} = w_{t,i} \exp(-\eta x_{t,i}(x_t^\top w_t - y_t))\]

- additive, GD: 
  \[w_{t+1,i} = w_{t,i} - \eta x_{t,i}(x_t^\top w_t - y_t)\]
Linear regression with random ± instances

Major differences in following paradigmic setup:

**128x128 random ± 1 matrix**

Rows are instances, labels are the first column

x-axis:  \( t = 1..128 \)

y-axis:  all 128 weights  Loss when trained on examples 1..\( t \)

Upshot: After half examples, GD has average loss \( \approx \frac{1}{2} \)

EG family converges in essentially \( \log(n) \) many examples
Linear regression with Hadamard instances

Major differences in following paradigmic setup:

**128x128 Hadamard matrix**

**Permuted** rows are instances, labels are any fixed column

Loss when trained on examples $1..t$ is

$$1 - \frac{t}{n}$$

**Upshot:** After half examples, GD has average loss is $= \frac{1}{2}$

EG family converges in essentially $\log(n)$ many examples
Hardness for GD Hadamard

- Linear decay of loss remains for GD even if
  - linear neuron with kernel inputs
    \[ \phi(x) \]
  - neuron with any transfer function \( h \) and kernel inputs
    \[ h(\cdot) \]

Conjecture: Hadamard problem remains hard for any neural net trained with GD

[DW14]
Hardness for GD Hadamard

- Linear decay of loss remains for GD even if
  - linear neuron with kernel inputs  
    \[ x \rightarrow \phi(x) \]
  - neuron with any transfer function \( h \) and kernel inputs  
    \[ h(\cdot) \rightarrow \phi(x) \]

Conjecture: Hadamard problem remains hard for any neural net trained with GD  

[DW14]
Crux: consider continuous time MD

- Parameter vector $w(t)$ continuous function of time
- Continuous update

$$\dot{f}(w(t)) = -\nabla L(w(t))$$

- Examples are still discrete

$$(x_s, y_s) \text{ for time } t \in [s, s + 1)$$

Again two main updates:

GD

$$\dot{w}(t) = -\nabla L(w(t))$$

EGU

$$\log(w(t)) = -\nabla L(w(t))$$

Motivate updates in the continuous domain and then “discretize” these updates
I) Three stunning surprises

- Continuous EGU can be simulated with continuous GD on a spindly 2-layer linear network
- Discretized versions of continuous GD simulation solves the Hadamard problem efficiently

Conjecture about GD training of neural nets is false
Neural nets trained w. GD more powerful than kernel methods

II) The structure of the network determines regularization when training with GD

III) Next talk: The linear lower bound for the Hadamard problem remains for any GD trained neural net with a fully connected input layer
I) Three stunning surprises

I) - Continuous EGU can be simulated with continuous GD on a spindly 2-layer linear network
- Discretized versions of continuous GD simulation solves the Hadamard problem efficiently

Conjecture about GD training of neural nets is false
Neural nets trained w. GD more powerful than kernel methods

II) The structure of the network determines regularization when training with GD

III) Next talk: The linear lower bound for the Hadamard problem remains for any GD trained neural net with a fully connected input layer
I) Three stunning surprises

- Continuous EGU can be simulated with continuous GD on a spindly 2-layer linear network
- Discretized versions of continuous GD simulation solves the Hadamard problem efficiently

Conjecture about GD training of neural nets is false
Neural nets trained w. GD more powerful than kernel methods

II) The structure of the network determines regularization when training with GD

III) Next talk: The linear lower bound for the Hadamard problem remains for any GD trained neural net

*with a fully connected input layer*
I) Pictorially

When linear neuron is trained with GD, then linear decrease of loss

Reparameterize weights $w_i$ by $u_i^2$ (if $u_i$ initialized equal $\Rightarrow$ stay equal)

Continuous GD on $u_i$ exactly simulates EGU on $w_i$

\[
\dot{u} = -2 (u \odot u \cdot x - y) u \odot x \quad \text{exactly simulates}
\]

\[
\dot{\log(w)} = -2\eta (w \cdot x - y) x
\]
1) Pictorially

When linear neuron is trained with GD, then linear decrease of loss

Reparameterize weights $w_i$ by $u_i^2$
(if $u_i$ initialized equal $\Rightarrow$ stay equal)

Continuous GD on $u_i$ exactly simulates EGU on $w_i$

\[
\dot{u} = -2 (u \odot u \cdot x - y) \ u \odot x \text{ exactly simulates}
\]

\[
\dot{\log(w)} = -2\eta (w \cdot x - y) \ x
\]
I) Simulations

Discretization

$$u_{t+1} = u_t - 2\eta (u_t \odot u_t \cdot x_t - y_t) \ u_t \odot x_t \quad \text{(EGasGD tracks)}$$

$$w_{t+1} = w_t \odot \exp(-2\eta (w_t \cdot x_t - y_t) \ x_t) \quad \text{(EGU)}$$

Simulation visually identical but slightly different numerically

Same regret bounds

Upshot: 2-layer neural net trained w. GD cracks Hadamard
Not just a matter of initialization

Case A
When trained with GD: approximates EGU and cracks Hadamard

Case B
Red weights initialized to zero
Linear loss on Hadamard when trained with GD
Also true if all bottom weights initialized to zero
Not just a matter of initialization

Case A
When trained with GD: approximates EGU and cracks Hadamard

Case B
Red weights initialized to zero
Linear loss on Hadamard when trained with GD
Also true if all bottom weights initialized to zero
Clamping

Case B
GD on all weights
Linear loss for Hadamard

Case A
GD on all weights and then
Red weights clamped to zero
i.e. $W = \text{diag} (\text{diag} (W))$
Cracks Hadamard
Clamping

Case B
GD on all weights
Linear loss for Hadamard

Case A
GD on all weights and then
Red weights clamped to zero
i.e. $W = \text{diag}(\text{diag}(W))$
Cracks Hadamard
II) Structure determines regularization

Case A
In continuous case, converges to smallest $L_1$ norm solution
In discrete case, same regret bounds as for EGU

Case B
→ smallest $L_2$ norm solution when bottom weights initialized to 0
More complicated for other initializations, but experimentally satisfies linear lower bound
Implications for neural net training?

- Take your favorite neural net trained w. GD
  Replace each weight $w_i$ by $(u_i^+)^2 - (u_i^-)^2$
  Train $\{u_i^+, u_i^-\}$ with GD

- Acts like EGU$^\pm$ on the $\{w_i\}$
  which is close to 1-norm regularization
MD with different link functions can simulate each other

Equal in continuous case
Same regret bounds for last 2 cases
Table of Contents

Big picture overview

Connections to learning with kernels

Motivations of updates

Linear regression for GD ($\tau = 0$)

Reparameterization

Summary of updates and open problems
2-layer linear neural net GD can beat any kernel

For Hadamard problem

Any kernel has linear decaying loss on average

EGUasGD has exponentially decaying loss
ψ maps a log n bit pattern b into all $2^{\log n}$ target products

- Products hard to learn from log n bits by any alg.
- Easy to learn by EGU after expansion with ψ
- $\psi(b) \cdot \tilde{\psi}(\tilde{b}) = \sum_{I \subseteq \text{log } n} \prod_{i \in I} b_i \tilde{b}_i = \prod_{i=1}^{\log n} (1+b_i \tilde{b}_i)$ is $O(\log n)$
- Hard to learn with any kernel (i.e. any feature map $\phi$)
Learning a single feature / conjunction

<table>
<thead>
<tr>
<th></th>
<th>update time</th>
<th>regret</th>
</tr>
</thead>
<tbody>
<tr>
<td>additive</td>
<td>$O(\log n)$</td>
<td>linear in $n$</td>
</tr>
<tr>
<td>multiplicative</td>
<td>$O(n)$</td>
<td>$O(\sqrt{L^* \log n})$</td>
</tr>
</tbody>
</table>

Loss is square loss or # of mistakes

The miracle of Winnow
- learns $k$-term DNF sample efficiently but not time efficiently

<table>
<thead>
<tr>
<th></th>
<th>update time</th>
<th>regret</th>
</tr>
</thead>
<tbody>
<tr>
<td>additive</td>
<td>$O(\log n)$</td>
<td>linear in $n$</td>
</tr>
<tr>
<td>Winnow</td>
<td>$O(n)$</td>
<td>$O(\sqrt{A^* k \log n})$</td>
</tr>
</tbody>
</table>

Loss is attribute loss which can be $k$ times # of mistakes
What’s next?

One feature per target expansion:
- Good for EGU
- Bad for GD
- And yet provably best expansion of LLS
Previous work

- Our work was triggered by \[GWBNNS17\]
  They show that quadratic reparameterization converges to minimum $L_1$-norm solution in underconstrained case
  Generalized to the matrix setting ...

- Reparameterization of EG as GD was known to game theorists \[AGin79\]
  (cont. EG = Replicator Dynamics of Evolutionary Game Th.)
  All previous work in the continuous case

- Here: Same regret bounds hold for reparameterized discrete updates

- Regret bounds first proven using XMAPLE
Two ways for obtaining discrete updates

1. Regularizing with Bregman divergences
2. As discretizations of continuous updates

For any convex function $F(w)$, the Bregman divergence is

$$
\Delta_F(w, w_s) = F(w) - F(w_s) - f(w_s)^\top (w - w_s)
$$

$$
= \Delta_{F^*}(f(w_s), f(w)) \tag{duality}
$$

Since $F(w)$ convex, $\nabla F(w) =: f(w) = w^*$ is increasing

$f(w)$ called the link function
Two ways for obtaining discrete updates

1. Regularizing with Bregman divergences
2. As discretizations of continuous updates

For any convex function $F(\mathbf{w})$, the Bregman divergence is

$$
\Delta_F(\mathbf{w}, \mathbf{w}_s) = F(\mathbf{w}) - F(\mathbf{w}_s) - f(\mathbf{w}_s)^\top(\mathbf{w} - \mathbf{w}_s)
$$

$$
= \Delta_{F^*}(f(\mathbf{w}_s), f(\mathbf{w}))
$$

(duality)

Since $F(\mathbf{w})$ convex, $\nabla F(\mathbf{w}) =: f(\mathbf{w}) = \mathbf{w}^*$ is increasing

$f(\mathbf{w})$ called the link function
1. Via Bregman divergences \[\text{[NY83,KW97]}\]

\[
\mathbf{w}_{s+1} = \arg\min_{\mathbf{\tilde{w}}} \Delta_F(\mathbf{\tilde{w}}, \mathbf{w}_s) + \eta L(\mathbf{\tilde{w}})
\]

Setting derivative at \(\mathbf{w}_{s+1}\) to zero

\[
f(\mathbf{w}_{s+1}) - f(\mathbf{w}_s) + \eta \nabla L(\mathbf{w}_{s+1}) = 0
\]

Implicit/Prox MD update \[\text{[R76,NY83]}\]

\[
\mathbf{w}_{s+1} = f^{-1}(f(\mathbf{w}_s) - \eta \nabla L(\mathbf{w}_{s+1}))
\]

Explicit MD update

\[
\mathbf{w}_{s+1} \approx f^{-1}(f(\mathbf{w}_s) - \eta \nabla L(\mathbf{w}_s))
\]
Continuous MD

\[ \dot{f}(w) = -\nabla L(w) \]

(Later: explicit and implicit MD as discretizations)

Main examples:
GD \((f(w) = w)\) and EGU \((f(w) = \log(w))\)

\[ \log_{\tau}(w) := \frac{1}{1-\tau}(w^{1-\tau} - 1) \]
\(\tau\) is temperature
(we use \(\tau \in [0, 1]\))

Tempered Logarithm

\[ \text{Log}_\tau x \]

\[ x \]

\[ \text{Log}_\tau x \]

\[ x \]

\[ \text{Log}_\tau x \]

\[ x \]
Second focus: updates derived from $\log_{\tau}$-divergence

Start with convex function for all $\tau$ (Tsallis entropy):

$$F_\tau(w) = \sum_i (w_i \log_\tau w_i - \frac{1}{2 - \tau} w_i^{2-\tau})$$

$$= \sum_i \left( \frac{1}{(1-\tau)(2-\tau)} w_i^{2-\tau} - \frac{1}{1-\tau} w_i \right)$$

with $f_\tau(w) = \nabla F_\tau(w) = \log_\tau(w) = \frac{1}{1-\tau}(w^{1-\tau} - 1)$

Generalized KL-divergence ($\beta$ divergence):

$$\Delta_{F_\tau}(\tilde{w}, w) = \sum_i (\tilde{w}_i \log_\tau \tilde{w}_i - \tilde{w}_i \log_\tau w_i - \frac{1}{2 - \tau} \tilde{w}_i^{2-\tau} + \frac{1}{2 - \tau} w_i^{2-\tau})$$

$$= \frac{1}{1-\tau} \sum_i \left( \frac{1}{2-\tau}(\tilde{w}_i^{2-\tau} - w_i^{2-\tau}) - (\tilde{w}_i - w_i)w_i^{\tau-1} \right)$$

2-sided gives the arcsinh divergence for $\tau = 1$
Large family of divergences

\[ \Delta_{F_{-1}}(\tilde{w}, w) = \frac{1}{6}(\tilde{w}_i + 2w_i)(\tilde{w}_i - w_i)^2 \]

\[ \Delta_{F_0}(\tilde{w}, w) = \frac{1}{2} \sum \limits_i (\tilde{w}_i - w_i)^2 \quad \text{(squared Euclidean, Domain = \mathbb{R})} \]

\[ \Delta_{F_1}(\tilde{w}, w) = \sum \limits_i \left( \frac{4}{3} \tilde{w}_i^\frac{3}{2} - 2\tilde{w}_i\sqrt{w_i} + \frac{3}{2}w_i^\frac{3}{2} \right) \]

\[ \Delta_{F_1}(\tilde{w}, w) = \sum \limits_i (\tilde{w}_i \log \frac{\tilde{w}_i}{w_i} - \tilde{w}_i + w_i) \quad \text{(KL-divergence)} \]

\[ \Delta_{F_2}(\tilde{w}, w) = \sum \limits_i \left( \log \frac{w_i}{\tilde{w}_i} - \frac{\tilde{w}_i}{w_i} - 1 \right) \quad \text{(Itakura-Saito)} \]

\[ \Delta_{F_3}(\tilde{w}, w) = \frac{1}{2} \sum \limits_i \left( \frac{1}{\tilde{w}_i} - \frac{2}{w_i} + \frac{\tilde{w}_i}{w_i^2} \right) \quad \text{(inverse)} \]
Motivation with Bregman momentum

\[ \mathbf{w}(t) = \arg\min_{\tilde{\mathbf{w}}(t)} \mathbf{\Delta}_F(\tilde{\mathbf{w}}(t), \mathbf{w}_s) + L(\tilde{\mathbf{w}}(t)) \]

Derivation of the optimum curve \( \mathbf{w}(t) \):

\[
\frac{\partial}{\partial \tilde{\mathbf{w}}(t)} \left( \frac{\partial}{\partial t} \left( F(\tilde{\mathbf{w}}(t)) - f(\mathbf{w}_s)^\top \tilde{\mathbf{w}}(t) \right) + L(\tilde{\mathbf{w}}(t)) \right) \quad \text{(differentiate)}
\]

\[
= \frac{\partial}{\partial \tilde{\mathbf{w}}(t)} ((f(\tilde{\mathbf{w}}(t))) - f(\mathbf{w}_s)) ^\top \dot{\tilde{\mathbf{w}}}(t) + \nabla L(\tilde{\mathbf{w}}(t))
\]

\[
= (J_f(\tilde{\mathbf{w}}) \dot{\tilde{\mathbf{w}}}(t) + \left( \frac{\partial \dot{\tilde{\mathbf{w}}}(t)}{\partial \tilde{\mathbf{w}}(t)} \right) ^\top (f(\tilde{\mathbf{w}}(t) - f(\mathbf{w}_s)) + \nabla L(\tilde{\mathbf{w}}(t)))
\]

\[
= f(\tilde{\mathbf{w}}(t)) + \nabla L(\tilde{\mathbf{w}}(t)) \quad \text{(By calculus of variations, } \tilde{\mathbf{w}}(t) \text{ and } \dot{\mathbf{w}}(t) \text{ are independent variables)}
\]

\[
\dot{\mathbf{w}}(t) = \mathbf{w}(t) = 0
\]
Adding constraint $c(w(t)) = 0$

Projected MD update:

$$w(t) = \arg\min_{\tilde{w}(t)} \Delta F(\tilde{w}(t), w_s) + L(\tilde{w}(t)) + \lambda c(\tilde{w}(t))$$

$$\dot{f}(w(t)) = -\left(I - \frac{c(t)c(t)^\top (Jf(w(t)))^{-1}}{c^\top(t)(Jf(w(t)))^{-1} c(t)}\right) \nabla L(w(t))$$

$\left(\text{where } c(t) := \nabla c(w(t))\right)$

Initial weight vector has to satisfy constraint
2. Explicit and implicit updates from cont. MD

\[ f(w) = -\nabla L(w) \]

Explicit discretization (Euler)

\[ \frac{f(w_{s+h}) - f(w_s)}{h} = -\nabla L(w_s) \]

\( \iff \)

\[ w_{s+h} = f^{-1}(f(w_s) - h \nabla L(w_s)) \]

Implicit discretization (forward Euler)

\[ \frac{f(w_{s+h}) - f(w_s)}{h} = -\nabla L(w_{s+h}) \]

\[ w_{s+h} = f^{-1}(f(w_s) - h \nabla L(w_{s+h})) \]
Right way to discretize

Continuous Mirror Descent update

\[
\dot{f}(w(t)) = -\nabla L(w(t))
\]

Integral continuous MD update

\[
f(w_{s+h}) - f(w_s) = -h \int_{s}^{s+h} \nabla L(w(t)) \, dt
\]

\[
w_{s+h} = f^{-1}\left(f(w_s) - h \int_{s}^{s+h} \nabla L(w(t)) \, dt\right)
\]
Right discretization of continuous MD

w.o. constraints

\[ f(w_{s+h}) - f(w_s) = -h \int_s^{t+h} \nabla L(w(t)) \, dt \]

w. constraints

\[ f(w_{s+h}) - f(w_s) = -h \int_s^{t+h} P(t) \nabla L(w(t)) \, dt \]
Updates motivation from integrated continuous MD

Integrated update

\[ f(w_{s+h}) - f(w_s) = -h \int_s^{s+h} \nabla L(w(t)) \, dt \]

Explicit approximation

\[ = -h \nabla L(w_s) \]

Implicit approximation

\[ = -h \nabla L(w_{s+h}) \]
Natural gradient view of continuous MD

Legendre transform

\[ w^* = f(w) \]
\[ w = f^*(w^*) \]

Dual updates \[ [WJ98] \]

\[ \dot{f}(w) = -\nabla L(w) \]
\[ \dot{f}^*(w^*) = -\nabla L \circ f^*(w^*) \]

As natural gradient updates

\[ \dot{w} = -(\nabla^2 F(w))^{-1} \nabla L(w) \]
\[ \dot{w}^* = -(\nabla^2 F^*(w^*))^{-1} \nabla L \circ f^*(w^*) \]

Pairs of updates are same, but not when discretized
Recall $c = \nabla c(w)$ and $P = I - \frac{cc^\top (Jf(w))^{-1}}{c^\top (Jf(w))^{-1} c}$

Dual updates

\[
\begin{align*}
\dot{f}(w) &= -P \nabla L(w) \\
\dot{f^*}(w^*) &= -P^\top \nabla L \circ f^*(w^*)
\end{align*}
\]

As natural gradient updates

\[
\begin{align*}
\dot{w} &= -P^\top (\nabla^2 F(w))^{-1} \nabla L(w) \\
\dot{w^*} &= -P (\nabla^2 F^*(w^*))^{-1} \nabla L \circ f^*(w^*)
\end{align*}
\]

Pairs of updates are same, but not when discretized
Projected MD in the dual

Recall $c = \nabla c(w)$ and $P = I - \frac{cc^\top (Jf(w))^{-1}}{c^\top (Jf(w))^{-1} c}$

(Here $c$ is shorthand for $c(t)$, $P$ shorthand for $P(t)$, ...)

\[ \dot{w}^* = f(w) \]

\[ = Jf(w) \dot{w} \]

\[ = -P \nabla L(w) \]

\[ = -PJf(w) \nabla L \circ f^*(w^*) \]

\[ = -P(\nabla^2 F^*(w^*))^{-1} \nabla L \circ f^*(w^*) \]

\[ \iff \dot{w} = -(Jf(w))^{-1} P \nabla L(w) \]

\[ = -P^\top (Jf(w))^{-1} \nabla L(w) \]

\[ = -P^\top (\nabla^2 F(w))^{-1} \nabla L(w) \]
Table of Contents

Big picture overview

Connections to learning with kernels

Motivations of updates

Linear regression for GD ($\tau = 0$)

Reparameterization

Summary of updates and open problems
Underconstrained linear regression

Loss $\| Xw - y \|_2^2$, where $X$ does not have full rank

Continuous GD: $\tau = 0$

$$w(t) = -X^\top(Xw(t) - y)$$

$$w(t) = \exp(-X^\top X t)(w(0) - X^\dagger y) + X^\dagger y$$

Continuous EGU case: $\tau = 1$

$$\dot{\log}(w(t)) = -X^\top(Xw(t) - y) \quad \text{or} \quad \dot{w}(t) = -w(t) \odot X^\top(Xw(t) - y)$$

$$w_i = \exp \left( - \left( \sum_t x_{t,i}(x_t \cdot w - y_t)w_i - \frac{1}{2} \sum_t x_{t,i}^2 w_i^2 \right) \right)$$

$$w = \exp \left( - \left( (X^\top(Xw - y)) \odot w - \frac{1}{2} \sum_t x_t^\odot 2 \odot w^\odot 2 \right) \right)$$

No closed-form solution for $0 < \tau < q \leq 1$
(2 − \tau) -norm updates for linear regression

**Theorem** Let $X \in \mathbb{R}^{N \times d}_{\geq 0}$ and $y \in \mathbb{R}^N_{\geq 0}$ with $N < d$. Let $E = \{w \in \mathbb{R}^d \mid Xw = y\}$ be the set of solutions with zero error. Let

$$w_{\alpha}(t) = \arg\min_{\tilde{w}(t)} \Delta_\tau(\tilde{w}(t), \alpha \mathbf{1}) + \|X\tilde{w}(t) - y\|_2^2, \text{for } \alpha > 0.$$ 

Then $w_{\alpha}(\infty) \in E$ and as $\alpha \to 0$, $w_{\alpha}(t)$ converges to the minimum $L_{2-\tau}$-norm solution in $E$.

(Can be extended to a two-sided version (i.e. \pm trick with two sets of weights $w_+$ and $w_-$) for general $X \in \mathbb{R}^{N \times d}$ and $y \in \mathbb{R}^N$)

Also $\Delta_{F_\tau}(\tilde{w}, w)$ strongly convex w.r.t. $L_{2-\tau}$-norm
Big picture overview

Connections to learning with kernels

Motivations of updates

Linear regression for GD ($\tau = 0$)

Reparameterization

Summary of updates and open problems
Theorem For the reparameterization function $w = q(u)$ with the property that $\text{range}(q) = \text{dom}(f)$, 
$\dot{g}(u) = -\nabla L \circ q(u)$ simulates $\dot{f}(w) = -\nabla L(w)$ if

$$(Jf(w))^{-1} = Jq(u) (Jg(u))^{-1} (Jq(u))^\top$$

and $q(u(0)) = w(0)$

For reparameterization as GD use $g = id$
Our main example: EGU as GD

Link

\[ f(w) = \log(w) \]

Reparameterization

\[ w = q(u) := \frac{1}{4} u \odot u \]
\[ u = 2\sqrt{w} \]

\[
(Jf(w))^{-1} = (\text{diag}(w)^{-1})^{-1} = \text{diag}(w) \\
Jq(u)(Jq(u))^\top = \frac{1}{2} \text{diag}(u)(\frac{1}{2} \text{diag}(u))^\top = \text{diag}(w)
\]

Conclusion

\[ \dot{\log}(w) = -\nabla L(w) \] equals \[ \dot{u} = -\nabla L \circ q(u) = -\frac{1}{2} u \odot \nabla L(w) \frac{\nabla u L(\frac{1}{4} u \odot u)}{\nabla L(\frac{1}{4} u \odot u)} \]
**Burg as GD**

**Link**

\[ f(w) = -\frac{1}{w} \]

**Reparameterization**

\[ w = q(u) := \exp(u) \]
\[ u = \log(w) \]

\[ (Jf(w))^{-1} = \text{diag}(\frac{1}{w \otimes w})^{-1} = \text{diag}(w)^2 \]

\[ Jq(u)(Jq(u))^\top = \text{diag}(\exp(u)) \text{diag}(\exp(u))^\top = \text{diag}(w)^2 \]

**Conclusion**

\[ -\frac{1}{w} = -\nabla L(w) \text{ equals } \dot{u} = -\nabla L \circ q(u) = -\exp(u) \odot \nabla L(w) \]

\[ \nabla_u L(\exp(u)) \]
\[ \log_{\tau} w = \frac{1}{1-\tau}(w^{1-\tau} - 1) \text{ as GD} \]

**Link**

\[ f(w) = \log_{\tau} w \]

**Reparameterization**

\[ w = q(u) := \left( \frac{2 - \tau}{2} \right)^{\frac{2}{2-\tau}} u^{\frac{2}{2-\tau}} \]

\[ u = \frac{2}{2 - \tau} w^{\frac{2-\tau}{2}} \]

\[ (J\log_{\tau}(w))^{-1} = (\text{diag}(w)^{-\tau})^{-1} = \text{diag}(w)^{\tau} \]

\[ Jq(u)(Jq(u))^\top = \left( \left( \frac{2 - \tau}{2} \right)^{\frac{\tau}{2-\tau}} \text{diag}(u)^{\frac{\tau}{2-\tau}} \right)^2 = \text{diag}(w)^{\tau} \]

**Conclusion**

\[ \log_{\tau}(w) = -\nabla L(w) \text{ equals } \dot{u} = -\nabla L \circ q(u) = -\frac{2 - \tau}{2} u^{\frac{\tau}{2-\tau}} \circ \nabla L(w) \]

\[ \nabla_u L\left( \left( \frac{2 - \tau}{2} \right)^{\frac{2}{2-\tau}} u^{\frac{2}{2-\tau}} \right) \]

\( \tau = 1: \text{EGU} \quad \tau = 0: \text{GD} \)
\[ \log_\tau \mathbf{w} = \frac{1}{1-\tau} (\mathbf{w}^{1-\tau} - 1) \] as GD

**Link** \[ f(\mathbf{w}) = \log_\tau \mathbf{w} \]

**Reparameterization**

\[ \mathbf{w} = q(\mathbf{u}) := \left( \frac{2 - \tau}{2} \right)^{\frac{2}{2-\tau}} \mathbf{u}^{\frac{2}{2-\tau}} \]

\[ \mathbf{u} = \frac{2}{2 - \tau} \mathbf{w}^{\frac{2-\tau}{2}} \]

\[ (J \log_\tau (\mathbf{w}))^{-1} = (\text{diag}(\mathbf{w})^{-\tau})^{-1} = \text{diag}(\mathbf{w})^\tau \]

\[ Jq(\mathbf{u})(Jq(\mathbf{u}))^\top = \left( \left( \frac{2 - \tau}{2} \right)^{\frac{\tau}{2-\tau}} \text{diag}(\mathbf{u})^{\frac{\tau}{2-\tau}} \right)^2 = \text{diag}(\mathbf{w})^\tau \]

**Conclusion**

\[ \log_\tau (\mathbf{w}) = -\nabla L(\mathbf{w}) \] equals \[ \dot{\mathbf{u}} = -\nabla L \circ q(\mathbf{u}) = -\frac{2 - \tau}{2} \mathbf{u}^{\frac{\tau}{2-\tau}} \circ \nabla L(\mathbf{w}) \]

\[ \nabla_u L \left( \left( \frac{2 - \tau}{2} \right)^{\frac{2}{2-\tau}} \mathbf{u}^{\frac{2}{2-\tau}} \right) \]

\[ \tau = 1: \text{ EGU} \quad \tau = 0: \text{ GD} \]
Table of Contents

Big picture overview

Connections to learning with kernels

Motivations of updates

Linear regression for GD ($\tau = 0$)

Reparameterization

Summary of updates and open problems
Discrete multiplicative updates for dot loss $\sum_i w_i \ell_i$

EGU
\[ \tilde{w}_i = w_i \exp(-\eta \ell_i) \]

Approx. EGU/PRODU
\[ \tilde{w}_i = w_i (1 - \eta \ell_i) \]

EGUasGD
\[ \tilde{u}_i = u_i (1 - \eta \ell_i) \]
\[ (\tilde{u}_i^2 = u_i^2 (1 - \eta \ell_i)^2) \]

EG/HEDGE
\[ \tilde{w}_i = \frac{w_i \exp(-\eta \ell_i)}{\sum_j w_j \exp(-\eta \ell_j)} \]

Approx. EG
\[ \tilde{w}_i = w_i (1 - \eta \ell_i + \eta \sum_j w_j \ell_j) \]

PROD
\[ \tilde{w}_i = \frac{w_i (1 - \eta \ell_i)}{\sum_j w_j (1 - \eta \ell_j)} \]

EGasGD
\[ \tilde{u}_i = \frac{u_i (1 - \eta \ell_i)}{\| \sum_j u_j^2 (1 - \eta \ell_j)^2 \|^2_2} \]
\[ (\tilde{u}_i^2 = \frac{u_i^2 (1 - \eta \ell_i)^2}{\sum_j u_j^2 (1 - \eta \ell_j)^2}) \]
Exponential alternates w. $\eta/2$

EGU as GD becomes EGU

$$\tilde{u}_i = u_i \exp(-\eta/2 \ell_i)$$

$$\tilde{u}_i^2 = u_i^2 \exp(-\eta \ell_i)$$

EG as GD becomes EG

$$\tilde{u}_i = \frac{u_i \exp(-\eta/2 \ell_i)}{\sqrt{\sum_j u_j^2 \exp(-\eta \ell_j)}}$$

$$\tilde{u}_i^2 = \frac{u_i^2 \exp(-\eta \ell_i)}{\sum_j u_j^2 \exp(-\eta \ell_j)}$$
Regret bounds

$$\text{total online loss of update} - \text{total online loss of best comparator} \leq \text{norms } \sqrt{\text{loss of best}}$$

<table>
<thead>
<tr>
<th>update</th>
<th>regret bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGUasGD, hinge loss</td>
<td>as Winnow</td>
</tr>
<tr>
<td>EGUasGD, linear regression</td>
<td>as EGU but only one-sided case</td>
</tr>
<tr>
<td>EGasGD, linear regression</td>
<td>as EG</td>
</tr>
<tr>
<td>EGasGD, dot loss</td>
<td>as Hedge</td>
</tr>
</tbody>
</table>

All proofs done with relative entropy as a measure of progress
Technical open problems

- Need regret bound linear regression EGU and EGUasGD when instances are in $[-1..1]^n$
- Ditto for the Approx. EGU and Approx. EG (PROD)
- Is the $\pm$ trick necessary (using $2d$ variables)? Can it be done with GD on $d$ variables?
- Is there any natural problem in which GD beats EGU$\pm$?
- Is the GD as EG$\pm$ simulation implementable in the brain?
- Relationship to $p$-norm perceptron
Far reaching open problems

- Solve the differential equation for linear regression EGU
- Regret bound for any log$_{\tau}$ update
- Revisit vanishing gradient issue, batch normalization, dropout, learning rate heuristics for EG$^\pm$
- Large scale simulations
  - Do multiplicative updates lead to sparse solutions
- New question: Does any GD trained neural net with complete input neurons satisfy the linear lower bound for the Hadamard problem?
- Next talk!
- What are the optimal kernels for GD and EGU?
  - In progress!

Thank you!
Far reaching open problems

- Solve the differential equation for linear regression EGU
- Regret bound for any $\log_\tau$ update
- Revisit vanishing gradient issue, batch normalization, dropout, learning rate heuristics for $\text{EG}^\pm$
- Large scale simulations
  - Do multiplicative updates lead to sparse solutions
- New question: Does any GD trained neural net with complete input neurons satisfy the linear lower bound for the Hadamard problem?

Next talk!

- What are the optimal kernels for GD and EGU?
  In progress!

Thank you!
Far reaching open problems

- Solve the differential equation for linear regression EGU
- Regret bound for any $\log_\tau$ update
- Revisit vanishing gradient issue, batch normalization, dropout, learning rate heuristics for EG$^\pm$
- Large scale simulations
  - Do multiplicative updates lead to sparse solutions
- New question: Does any GD trained neural net with complete input neurons satisfy the linear lower bound for the Hadamard problem?
  Next talk!
- What are the optimal kernels for GD and EGU?
  In progress!

Thank you!
Far reaching open problems

- Solve the differential equation for linear regression EGU
- Regret bound for any \( \log_\tau \) update
- Revisit vanishing gradient issue, batch normalization, dropout, learning rate heuristics for \( \text{EG}^\pm \)
- Large scale simulations
  - Do multiplicative updates lead to sparse solutions
- New question: Does any GD trained neural net with complete input neurons satisfy the linear lower bound for the Hadamard problem?
  
  Next talk!

- What are the optimal kernels for GD and EGU?
  
  In progress!

Thank you!
2020 papers

**COLT**  Winnowing with gradient descent  [with Ehsan Amid]

**NeurIPS**  Reparameterizing Mirror Descent as Gradient Descent  [with Ehsan Amid]

**ArXiv**  A case where a spindly two-layer linear network whips any neural network with a fully connected input layer  [with Ehsan Amid & Wojciech Kotłowski]

**All papers**  https://users.soe.ucsc.edu/~manfred/last/