Winnowing with Gradient Descent

Manfred K. Warmuth
Google Brain & UCSC

Foundations Reading Group
Deep Mind
11-23-2020

Joint work w. Ehsan Amid

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

Winnow: to remove chaff from grain

wheat

soy beans

Learning disjunctions when irrelevant attributes abound [L89]

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

Notation of the Winnow algorithm

Learns disjunctions as linear threshold functions

- 2 out of 5 literal monotone disjunction $v_1 \vee v_3$
- Represented as $\boldsymbol{d} = (1, 0, 1, 0, 0)^\top$
- Label for instance $\boldsymbol{x} = (0, 1, 1, 0, 0)^\top$

$$\begin{cases} +1 & \text{if } \boldsymbol{d} \cdot \boldsymbol{x} \geq {}^1\!/_2 \\ -1 & \text{otherwise} \end{cases}$$

- Alg. receives sequence of examples online

$$(\boldsymbol{x}_1 \underset{\hat{y}_1}{,} y_1)\ (\boldsymbol{x}_2 \underset{\hat{y}_2}{,} y_2),\ \ldots,\ (\boldsymbol{x}_T \underset{\hat{y}_T}{,} y_T)$$

instances $[0, 1]^n$, labels and predictions are ± 1

Winnow algorithm

```
Initialize $\boldsymbol{w}_1 = w_0\,(1,1,\ldots 1)^\top$
for $t = 1$ to $T$ do
    Receive instance $\boldsymbol{x}_t \in [0,1]^n$
    Predict with linear threshold
        $\hat{y}_t = \begin{cases} +1 & \text{if } \boldsymbol{w}_t \cdot \boldsymbol{x}_t \geq \theta \\ -1 & \text{otherwise} \end{cases}$
    Receive label $y_t \in \{+1,-1\}$
    Multiplicative update: $w_{t+1,i} = w_{t,i}\,\exp(-\eta(\hat{y}_t - y_i)\boldsymbol{x}_{t,i})$
end for
```

$\leq k \log n$ mistakes

Perceptron alg., additive: $w_{t+1,i} = w_{t,i} - \eta \underbrace{(\hat{y}_t - y_i)\boldsymbol{x}_{t,i}}_{\text{gradient of hingle loss}}$

$\geq k\ n$ mistakes

Winnow algorithm

```
Initialize $\boldsymbol{w}_1 = w_0\,(1, 1, \ldots 1)^\top$
for $t = 1$ to $T$ do
    Receive instance $\boldsymbol{x}_t \in [0, 1]^n$
    Predict with linear threshold
        $$\hat{y}_t = \begin{cases} +1 & \text{if } \boldsymbol{w}_t \cdot \boldsymbol{x}_t \geq \theta \\ -1 & \text{otherwise} \end{cases}$$
    Receive label $y_t \in \{+1, -1\}$
    Multiplicative update:  $w_{t+1,i} = w_{t,i} \exp(-\eta(\hat{y}_t - y_i)\boldsymbol{x}_{t,i})$
end for
```

$\leq k \log n$ mistakes

Perceptron alg., additive: $$w_{t+1,i} = w_{t,i} - \eta \underbrace{(\hat{y}_t - y_i)\boldsymbol{x}_{t,i}}_{\text{gradient of hingle loss}}$$

$\geq k\ n$ mistakes

Mirror descent [NY83]

$$f(\boldsymbol{w}_{s+1}) - f(\boldsymbol{w}_s) = -\nabla L(\boldsymbol{w}_s) \quad \text{(where } f \text{ is strictly increasing)}$$
$$\boldsymbol{w}_{s+1} = f^{-1}(f(\boldsymbol{w}_s) - \nabla L(\boldsymbol{w}_s))$$

Gradient Descent (GD): f=id

$$\boldsymbol{w}_{s+1} - \boldsymbol{w}_s = -\nabla L(\boldsymbol{w}_s)$$
$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \nabla L(\boldsymbol{w}_s)$$

Unnormalized Exponentiateed Gradient Descent (EGU): $f = \log$

$$\log(\boldsymbol{w}_{s+1}) - \log(\boldsymbol{w}_s) = -\nabla L(\boldsymbol{w}_s)$$
$$\boldsymbol{w}_{s+1,i} = \boldsymbol{w}_{s,i} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_i \quad \text{(now } w_i \geq 0) \quad [\text{KW97}]$$

Normalized version called Exponentiated Gradient (EG)

$$\boldsymbol{w}_{s+1,i} = \frac{\boldsymbol{w}_{s,i} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_i}{\sum_j \boldsymbol{w}_{s,j} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_j} \quad \text{(now } \boldsymbol{w} \text{ prob.vect.)}$$

Mirror descent [NY83]

$$f(\boldsymbol{w}_{s+1}) - f(\boldsymbol{w}_s) = -\nabla L(\boldsymbol{w}_s) \quad \text{(where } f \text{ is strictly increasing)}$$
$$\boldsymbol{w}_{s+1} = f^{-1}(f(\boldsymbol{w}_s) - \nabla L(\boldsymbol{w}_s))$$

Gradient Descent (GD): f=id

$$\boldsymbol{w}_{s+1} - \boldsymbol{w}_s = -\nabla L(\boldsymbol{w}_s)$$
$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \nabla L(\boldsymbol{w}_s)$$

Unnormalized Exponentiateed Gradient Descent (EGU): $f = \log$

$$\log(\boldsymbol{w}_{s+1}) - \log(\boldsymbol{w}_s) = -\nabla L(\boldsymbol{w}_s)$$
$$\boldsymbol{w}_{s+1,i} = \boldsymbol{w}_{s,i} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_i \quad \text{(now } w_i \geq 0) \quad \text{[KW97]}$$

Normalized version called Exponentiated Gradient (EG)

$$\boldsymbol{w}_{s+1,i} = \frac{\boldsymbol{w}_{s,i} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_i}{\sum_j \boldsymbol{w}_{s,j} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_j} \quad \text{(now } \boldsymbol{w} \text{ prob.vect.)}$$

Mirror descent [NY83]

$$f(\boldsymbol{w}_{s+1}) - f(\boldsymbol{w}_s) = -\nabla L(\boldsymbol{w}_s) \quad \text{(where } f \text{ is strictly increasing)}$$

$$\boldsymbol{w}_{s+1} = f^{-1}(f(\boldsymbol{w}_s) - \nabla L(\boldsymbol{w}_s))$$

Gradient Descent (GD): f=id

$$\boldsymbol{w}_{s+1} - \boldsymbol{w}_s = -\nabla L(\boldsymbol{w}_s)$$

$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \nabla L(\boldsymbol{w}_s)$$

Unnormalized Exponentiateed Gradient Descent (EGU): $f = \log$

$$\log(\boldsymbol{w}_{s+1}) - \log(\boldsymbol{w}_s) = -\nabla L(\boldsymbol{w}_s)$$

$$\boldsymbol{w}_{s+1,i} = \boldsymbol{w}_{s,i} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_i \quad \text{(now } w_i \geq 0) \quad \text{[KW97]}$$

Normalized version called Exponentiated Gradient (EG)

$$\boldsymbol{w}_{s+1,i} = \frac{\boldsymbol{w}_{s,i} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_i}{\sum_j \boldsymbol{w}_{s,j} \exp(-\eta(\nabla L(\boldsymbol{w}_s))_j} \quad \text{(now } \boldsymbol{w} \text{ prob.vect.)}$$

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

$$\begin{pmatrix} -1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$$

$\pm$ matrix random or Hadamard

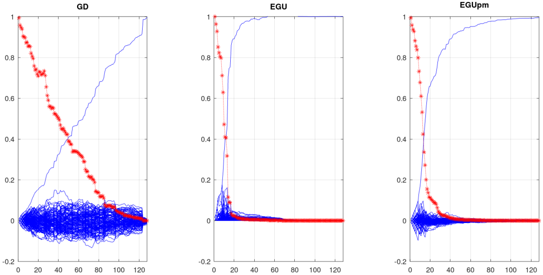
After receiving example $(\boldsymbol{x}_t, y_t)$
and incurring loss $(\boldsymbol{x}_t^\top \boldsymbol{w}_t - y_t)^2$ update:

multiplicative, EGU: $w_{t+1,i} = w_{t,i} \exp(-\eta \boldsymbol{x}_{t,i}(\boldsymbol{x}_t^\top \boldsymbol{w}_t - y_t))$

additive, GD: $w_{t+1,i} = w_{t,i} - \eta \underbrace{\boldsymbol{x}_{t,i}(\boldsymbol{x}_t^\top \boldsymbol{w}_t - y_t)}_{\text{gradient}}$

Linear regression with random ± instances

Major differences in following paradigmic setup:
128x128 random ± 1 matrix
Rows are instances, labels are the first column

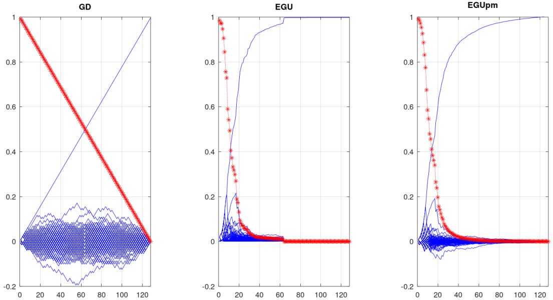


x-axis: $t = 1..128$
y-axis: all 128 weights Loss when trained on examples $1..t$

Upshot: After half examples, GD has average loss $\approx {}^{1}\!/_{2}$
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



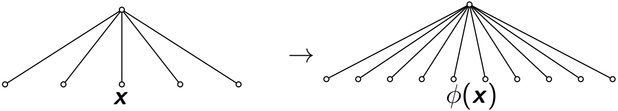
Loss when trained on examples 1..t is

$$1 - {}^{t}\!/_{n}$$

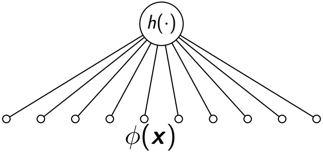
Upshot: After half examples, GD has average loss is $= {}^{1}\!/_{2}$
EG family converges in essentially $\log(n)$ many examples

Hardness for GD Hadamard

- Linear decay of loss remains for GD even if
  - linear neuron with kernel inputs [WV05]



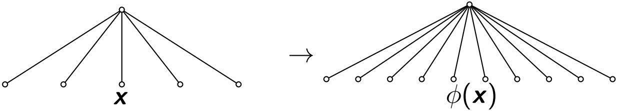
  - neuron with any transfer function h and kernel inputs [DW14]



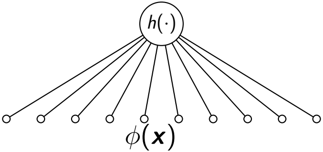
Conjecture: Hadamard problem remains hard for any neural net trained with GD [DW14]

Hardness for GD Hadamard

- Linear decay of loss remains for GD even if
  - linear neuron with kernel inputs [WV05]



  - neuron with any transfer function h and kernel inputs [DW14]



Conjecture: Hadamard problem remains hard
for any neural net trained with GD [DW14]

Crux: consider continuous time MD [NY83]

- Parameter vector $\boldsymbol{w}(t)$ continuous function of time
- Continuous update

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

- Examples are still discrete

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

Again two main updates:

$$\begin{aligned} &\text{GD} & \dot{\boldsymbol{w}}(t) &= -\nabla L(\boldsymbol{w}(t)) \\ &\text{EGU} & \dot{\log}(\boldsymbol{w}(t)) &= -\nabla L(\boldsymbol{w}(t)) \end{aligned}$$

Motivate updates in the continuous domain
and then "discretize" these updates

I) Three stunning surprises

I) - Continous 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) - Continous 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

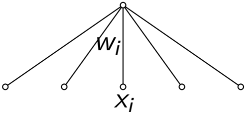
I) - Continous EGU can be simulated with continuous GD on a spindly 2-layer linear network
- Discretized versions of continuous GD simulation solves the Hadamard problem efficiently

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

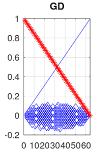
II) The structure of the network determines regularization when training with GD

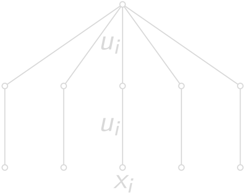
III) Next talk: The linear lower bound for the Hadamard problem remains for any GD trained neural net
with a fully connected input layer

I) Pictorially



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





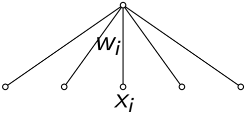
Reparameterize weights w_i by u_i^2 (if u_i initialized equal ⇒ stay equal)

Continuous GD on u_i exactly simulates EGU on w_i

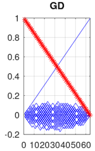
$\dot{\boldsymbol{u}} = -2\left(\boldsymbol{u}\odot\boldsymbol{u}\cdot\boldsymbol{x} - y\right)\,\boldsymbol{u}\odot\boldsymbol{x}$ exactly simulates

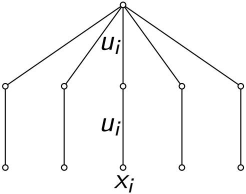
$$\dot{\log}(\boldsymbol{w}) = -2\eta\left(\boldsymbol{w}\cdot\boldsymbol{x} - y\right)\,\boldsymbol{x}$$

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 ⇒ stay equal)

Continuous GD on u_i exactly simulates EGU on w_i

$$\dot{\boldsymbol{u}} = -2\left(\boldsymbol{u}\odot\boldsymbol{u}\cdot\boldsymbol{x} - y\right)\,\boldsymbol{u}\odot\boldsymbol{x}$$ exactly simulates

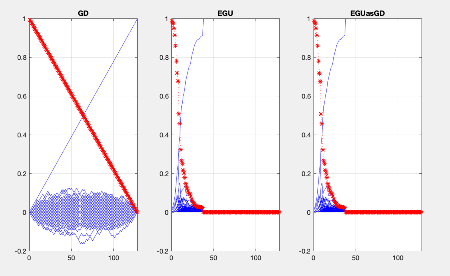
$$\dot{\log}(\boldsymbol{w}) = -2\eta\left(\boldsymbol{w}\cdot\boldsymbol{x} - y\right)\,\boldsymbol{x}$$

I) Simulations

Discretization

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t - 2\eta\left(\boldsymbol{u}_t \odot \boldsymbol{u}_t \cdot \boldsymbol{x}_t - y_t\right)\, \boldsymbol{u}_t \odot \boldsymbol{x}_t \qquad \text{(EGasGD tracks)}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t \odot \exp(-2\eta\left(\boldsymbol{w}_t \cdot \boldsymbol{x}_t - y_t\right) \boldsymbol{x}_t) \qquad \text{(EGU)}$$



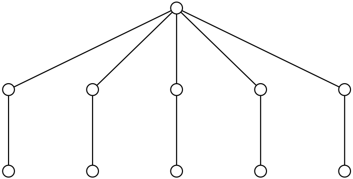
Simulation visually identical but slightly different numerically

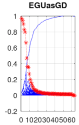
Same regret bounds

Upshot: 2-layer neural net trained w. GD cracks Hadamard

Not just a matter of initialization

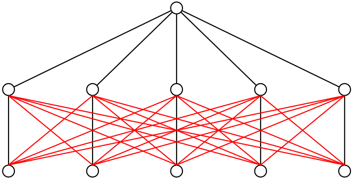
Case A

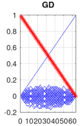




When trained with GD: approximates EGU and cracks Hadamard

Case B





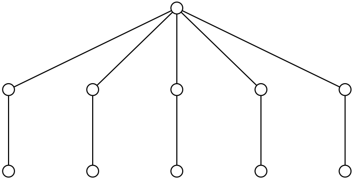
Red weights initialized to zero

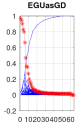
Linear loss on Hadamard when trained with GD

Also true if all bottom weights initialized to zero

Not just a matter of initialization

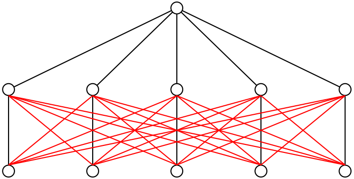
Case A

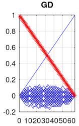




When trained with GD: approximates EGU and cracks Hadamard

Case B



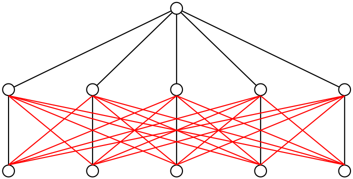


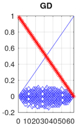
Red weights initialized to zero

Linear loss on Hadamard when trained with GD

Also true if all bottom weights initialized to zero

Clamping

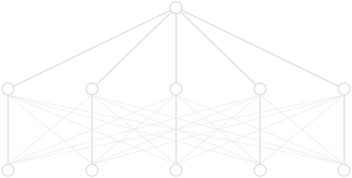




Case B

GD on all weights

Linear loss for Hadamard



Case A

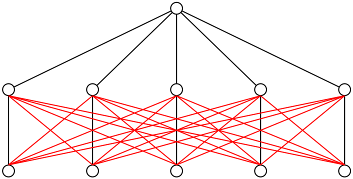
GD on all weights and then

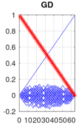
Red weights clamped to zero

i.e. $W = \mathrm{diag}(\mathrm{diag}(W))$

Cracks Hadamard

Clamping

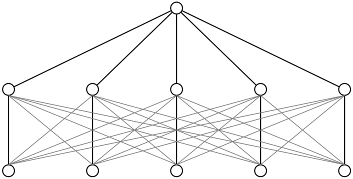


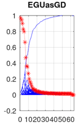


Case B

GD on all weights

Linear loss for Hadamard





Case A

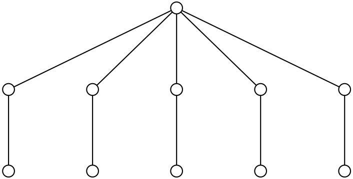
GD on all weights and then

Red weights clamped to zero

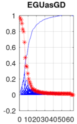
i.e. $W = \mathrm{diag}(\mathrm{diag}(W))$

Cracks Hadamard

II) Structure determines regularization

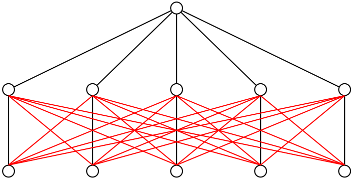


Case A

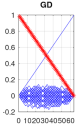


In continuous case, converges to smallest L_1 norm solution

In discrete case, same regret bounds as for EGU



Case B

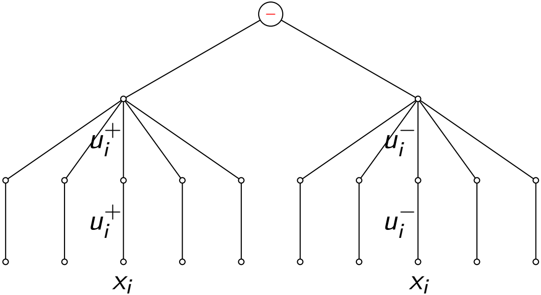


→ smallest L_2 norm solution when bottom weights initialized to 0

More complicated for other initializations, but experimentally satisfies linear lower bound

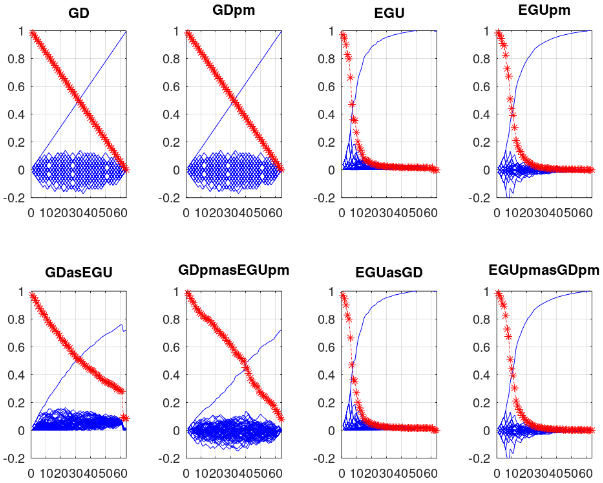
Implications for neural net training?

- 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

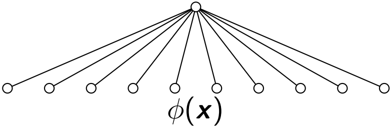
Linear regression for GD ($\tau = 0$)

Reparameterization

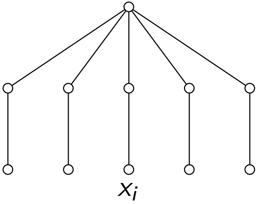
Summary of updates and open problems

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

From XOR to Hadamard

$$
n\left\{\overbrace{\begin{array}{ccc}
\begin{pmatrix}
+ & + & + \\
- & + & + \\
+ & - & + \\
- & - & + \\
+ & + & - \\
- & + & - \\
+ & - & - \\
- & - & -
\end{pmatrix} \\
b_1 \quad b_2 \quad b_3
\end{array}}^{\log n}\right.
\xrightarrow{\psi}
\overbrace{\begin{array}{c}
\begin{pmatrix}
+ & + & + & + & + & + & + & + \\
+ & - & + & - & + & - & + & - \\
+ & + & - & - & + & + & - & - \\
+ & - & - & + & + & - & - & + \\
+ & + & + & + & - & - & - & - \\
+ & - & + & - & - & + & - & + \\
+ & + & - & - & - & - & + & + \\
+ & - & - & + & - & + & + & -
\end{pmatrix} \\
1 \quad b_1 \quad b_2 \quad b_1 b_2 \quad b_3 \quad b_1 b_3 \quad b_2 b_3 \quad b_1 b_2 b_3
\end{array}}^{n}
$$

ψ maps a $\log n$ bit pattern $\boldsymbol{b}$ 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
- $\psi(\boldsymbol{b})\cdot\psi(\tilde{\mathbf{b}}) = \sum_{I \subseteq 1..\log n} \prod_{i \in I} b_i \tilde{b}_i = \prod_{i=1}^{\log n}(1+b_i \tilde{b}_i)$ is $O(\log n)$
- Hard to learn with any kernel (i.e. any feature map ϕ)

Learning a single feature / conjunction

	update time	regret
additive	$O(\log n)$	linear in n
multiplicative	$O(n)$	$O(\sqrt{L^* \log n})$

Loss is square loss or # of mistakes

The miracle of Winnow

- learns k-term DNF sample efficiently but not time efficiently

	update time	regret
additive	$O(\log n)$	linear in n
Winnow	$O(n)$	$O(\sqrt{A^* k \log n})$

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 [GWBNS17]
  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 [Akin79]
  (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

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

Two ways for obtaining discrete updates

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

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

$$\begin{aligned}\Delta_F(\boldsymbol{w}, \boldsymbol{w}_s) &= F(\boldsymbol{w}) - F(\boldsymbol{w}_s) - f(\boldsymbol{w}_s)^\top(\boldsymbol{w} - \boldsymbol{w}_s) \\ &= \Delta_{F^*}(\underbrace{f(\boldsymbol{w}_s)}_{\boldsymbol{w}_s^*}, \underbrace{f(\boldsymbol{w})}_{\boldsymbol{w}^*}) \qquad \text{(duality)}\end{aligned}$$

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

$f(\boldsymbol{w})$ 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 $F(\boldsymbol{w})$, the Bregman divergence is

$$\begin{aligned}\Delta_F(\boldsymbol{w}, \boldsymbol{w}_s) &= F(\boldsymbol{w}) - F(\boldsymbol{w}_s) - f(\boldsymbol{w}_s)^\top(\boldsymbol{w} - \boldsymbol{w}_s) \\ &= \Delta_{F^*}(\underbrace{f(\boldsymbol{w}_s)}_{\boldsymbol{w}_s^*}, \underbrace{f(\boldsymbol{w})}_{\boldsymbol{w}^*}) \qquad \text{(duality)}\end{aligned}$$

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

$f(\boldsymbol{w})$ called the link function

1. Via Bregman divergences [NY83,KW97]

$$\boldsymbol{w}_{s+1} = \operatorname*{argmin}_{\tilde{\boldsymbol{w}}} \; \Delta_F(\tilde{\boldsymbol{w}}, \boldsymbol{w}_s) + \eta L(\tilde{\boldsymbol{w}})$$

Setting derivative at $\boldsymbol{w}_{s+1}$ to zero

$$f(\boldsymbol{w}_{s+1}) - f(\boldsymbol{w}_s) + \eta \nabla L(\boldsymbol{w}_{s+1}) = \boldsymbol{0}$$

Implicit/Prox MD update [R76,NY83]

$$\boldsymbol{w}_{s+1} = f^{-1}(f(\boldsymbol{w}_s) - \eta \nabla L(\boldsymbol{w}_{\color{red}{s+1}}))$$

Explicit MD update

$$\boldsymbol{w}_{s+1} \approx f^{-1}(f(\boldsymbol{w}_s) - \eta \nabla L(\boldsymbol{w}_{\color{red}{s}}))$$

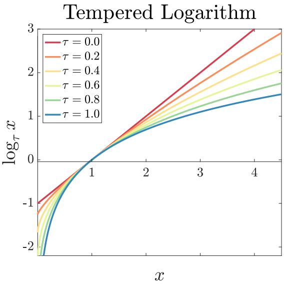
Continuous MD [NY83]

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

(Later: explicit and implicit MD as discretizations)

Main examples:
GD ($f(\boldsymbol{w}) = \boldsymbol{w}$) and EGU ($f(\boldsymbol{w}) = \log(\boldsymbol{w})$)

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



[N02]

Second focus: updates derived from $\log_\tau$-divergence

Start with convex function for all τ (Tsallis entropy):

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

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

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

Generalized KL-divergence (β divergence):

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

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

2-sided gives the $\operatorname{arcsinh}$ divergence for $\tau = 1$

Large family of divergences

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

$$\Delta_{F_0}(\tilde{\boldsymbol{w}}, \boldsymbol{w}) = \frac{1}{2}\sum_i(\tilde{w}_i - w_i)^2 \quad \text{(squared Euclidean, Domain} = \mathbb{R})$$

$$\Delta_{F_{\frac{1}{2}}}(\tilde{\boldsymbol{w}}, \boldsymbol{w}) = \sum_i(\frac{4}{3}\tilde{w}_i^{\frac{3}{2}} - 2\tilde{w}_i\sqrt{w_i} + \frac{3}{2}w_i^{\frac{3}{2}})$$

$$\Delta_{F_1}(\tilde{\boldsymbol{w}}, \boldsymbol{w}) = \sum_i(\tilde{w}_i \log\frac{\tilde{w}_i}{w_i} - \tilde{w}_i + w_i) \quad \text{(KL-divergence)}$$

$$\Delta_{F_{\frac{3}{2}}}(\tilde{\boldsymbol{w}}, \boldsymbol{w}) = 2\sum_i\frac{(\sqrt{\tilde{w}_i} - \sqrt{w_i})^2}{\sqrt{w_i}} \quad \text{(squared Xi on roots)}$$

$$\Delta_{F_2}(\tilde{\boldsymbol{w}}, \boldsymbol{w}) = \sum_i(\log\frac{w_i}{\tilde{w}_i} - \frac{\tilde{w}_i}{w_i} - 1) \quad \text{(Itakura-Saito)}$$

$$\Delta_{F_3}(\tilde{\boldsymbol{w}}, \boldsymbol{w}) = \frac{1}{2}\sum_i(\frac{1}{\tilde{w}_i} - \frac{2}{w_i} + \frac{\tilde{w}_i}{w_i^2}) \quad \text{(inverse)}$$

1. Motivation with Bregman momentum

$$\boldsymbol{w}(t) = \operatorname*{argmin}_{\tilde{\boldsymbol{w}}(t)} \; \underbrace{\dot{\Delta}_F(\tilde{\boldsymbol{w}}(t), \boldsymbol{w}_s)}_{\text{Bregman momentum}} + L(\tilde{\boldsymbol{w}}(t))$$

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

$$\frac{\partial}{\partial \tilde{\boldsymbol{w}}(t)} \Big(\frac{\partial}{\partial t} \Big(F(\tilde{\boldsymbol{w}}(t)) - f(\boldsymbol{w}_s)^\top \tilde{\boldsymbol{w}}(t) \Big) + L(\tilde{\boldsymbol{w}}(t)) \Big) \quad \text{(differentiate)}$$

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

$$= (\boldsymbol{J}f(\tilde{\boldsymbol{w}})\,\dot{\tilde{\boldsymbol{w}}}(t) + \underbrace{\Big(\frac{\partial \dot{\tilde{\boldsymbol{w}}}(t)}{\partial \tilde{\boldsymbol{w}}(t)}\Big)^\top}_{\mathbf{0}} (f(\tilde{\boldsymbol{w}}(t) - f(\boldsymbol{w}_s)) + \nabla L(\tilde{\boldsymbol{w}}(t))$$

(By calculus of variations, $\tilde{\boldsymbol{w}}(t)$ and $\dot{\tilde{\boldsymbol{w}}}(t)$ are independent variables)

$$= \dot{f}(\tilde{\boldsymbol{w}}(t)) + \nabla L(\tilde{\boldsymbol{w}}(t)) \overset{\tilde{\boldsymbol{w}}(t)=\boldsymbol{w}(t)}{=} \mathbf{0}$$

Adding constraint $c(\boldsymbol{w}(t)) = 0$

Projected MD update:

$$\boldsymbol{w}(t) = \operatorname*{argmin}_{\tilde{\boldsymbol{w}}(t)} \ \dot{\Delta}_F(\tilde{\boldsymbol{w}}(t), \boldsymbol{w}_s) + L(\tilde{\boldsymbol{w}}(t)) + \lambda\, c(\tilde{\boldsymbol{w}}(t))$$

$$\dot{f}(\boldsymbol{w}(t)) = -\underbrace{\left(\boldsymbol{I} - \frac{\boldsymbol{c}(t)\boldsymbol{c}(t)^\top \left(\boldsymbol{J}f(\boldsymbol{w}(t))\right)^{-1}}{\boldsymbol{c}^\top(t)(\boldsymbol{J}f(\boldsymbol{w}(t)))^{-1}\,\boldsymbol{c}(t)}\right)}_{:=\boldsymbol{P}(t)} \ \nabla L(\boldsymbol{w}(t))$$

(where $c(t) := \nabla c(\boldsymbol{w}(t)))$

Initial weight vector has to satisfy constraint

2. Explicit and implicit updates from cont. MD

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

Explicit discretization (Euler)

$$\frac{f(\boldsymbol{w}_{s+h}) - f(\boldsymbol{w}_s)}{h} = -\nabla L(\boldsymbol{w}_s)$$

$$\iff \boldsymbol{w}_{s+h} = f^{-1}(f(\boldsymbol{w}_s) - h\nabla L(\boldsymbol{w}_s))$$

Implicit discretization (forward Euler)

$$\frac{f(\boldsymbol{w}_{s+h}) - f(\boldsymbol{w}_s)}{h} = -\nabla L(\boldsymbol{w}_{s+h})$$

$$\boldsymbol{w}_{s+h} = f^{-1}(f(\boldsymbol{w}_s) - h\nabla L(\boldsymbol{w}_{s+h}))$$

Right way to discretize

Continuous Mirror Descent update

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

Integral continuous MD update

$$\begin{aligned} f(\boldsymbol{w}_{s+h}) - f(\boldsymbol{w}_s) &= -h \int_s^{s+h} \nabla L(\boldsymbol{w}(t))\, d\, t \\ \boldsymbol{w}_{s+h} &= f^{-1}\Big(f(\boldsymbol{w}_s) - h \int_s^{s+h} \nabla L(\boldsymbol{w}(t))\, d\, t\Big) \end{aligned}$$

Right discretization of continuous MD

w.o. constraints

$$f(\boldsymbol{w}_{s+h}) - f(\boldsymbol{w}_s) = -h \int_s^{t+h} \nabla L(\boldsymbol{w}(t))\, d\, t$$

w. constraints

$$f(\boldsymbol{w}_{s+h}) - f(\boldsymbol{w}_s) = -h \int_s^{t+h} \boldsymbol{P}(t)\, \nabla L(\boldsymbol{w}(t))\, d\, t$$

Updates motivation from integrated continuous MD

Integrated update

$$f(\boldsymbol{w}_{s+h}) - f(\boldsymbol{w}_s) = -h \int_s^{s+h} \nabla L(\boldsymbol{w}(t))\, d\, t$$

Explicit approximation

$$= -h\, \nabla L(\boldsymbol{w}_s)$$

Implicit approximation

$$= -h\, \nabla L(\boldsymbol{w}_{s+h})$$

Natural gradient view of continuous MD

Legendre transform

$$\begin{aligned} \boldsymbol{w}^* &= f(\boldsymbol{w}) \\ \boldsymbol{w} &= f^*(\boldsymbol{w}^*) \end{aligned}$$

Dual updates [WJ98]

$$\begin{aligned} \dot{f}(\boldsymbol{w}) &= -\nabla L(\boldsymbol{w}) \\ \dot{f}^*(\boldsymbol{w}^*) &= -\nabla L \circ f^*\,(\boldsymbol{w}^*) \end{aligned}$$

As natural gradient updates

$$\begin{aligned} \dot{\boldsymbol{w}} &= -(\nabla^2 F(\boldsymbol{w}))^{-1}\ \ \nabla L(\boldsymbol{w}) \\ \dot{\boldsymbol{w}}^* &= -(\nabla^2 F^*(\boldsymbol{w}^*))^{-1}\, \nabla L \circ f^*(\boldsymbol{w}^*) \end{aligned}$$

Pairs of updates are same, but not when discretized

Ditto with constraint

Recall $\boldsymbol{c} = \nabla c(\boldsymbol{w})$ and $\boldsymbol{P} = \boldsymbol{I} - \frac{\boldsymbol{c}\boldsymbol{c}^\top (\boldsymbol{J}f(\boldsymbol{w}))^{-1}}{\boldsymbol{c}^\top (\boldsymbol{J}f(\boldsymbol{w}))^{-1}\,\boldsymbol{c}}$

Dual updates

$$\dot{f}(\boldsymbol{w}) = -\boldsymbol{P}\,\nabla L(\boldsymbol{w})$$
$$\dot{f}^*(\boldsymbol{w}^*) = -\boldsymbol{P}^\top\,\nabla L \circ f^*\,(\boldsymbol{w}^*)$$

As natural gradient updates

$$\dot{\boldsymbol{w}} = -\boldsymbol{P}^\top\,(\nabla^2 F(\boldsymbol{w}))^{-1}\;\;\nabla L(\boldsymbol{w})$$
$$\dot{\boldsymbol{w}}^* = -\boldsymbol{P}\,(\nabla^2 F^*(\boldsymbol{w}^*))^{-1}\,\nabla L \circ f^*(\boldsymbol{w}^*)$$

Pairs of updates are same, but not when discretized

Projected MD in the dual

Recall $c = \nabla c(\boldsymbol{w})$ and $\boldsymbol{P} = \boldsymbol{I} - \frac{\boldsymbol{c}\boldsymbol{c}^\top (\boldsymbol{J}f(\boldsymbol{w}))^{-1}}{\boldsymbol{c}^\top (\boldsymbol{J}f(\boldsymbol{w}))^{-1}\boldsymbol{c}}$
(Here $\boldsymbol{c}$ is shorthand for $\boldsymbol{c}(t)$, $\boldsymbol{P}$ shorthand for $\boldsymbol{P}(t)$, ...)

$$\begin{aligned}
\dot{\boldsymbol{w}}^* &= \dot{f}(\boldsymbol{w}) \\
&= \boldsymbol{J}f(\boldsymbol{w})\,\dot{\boldsymbol{w}} \\
&= -\boldsymbol{P}\,\nabla L(\boldsymbol{w}) \\
&= -\boldsymbol{P}\boldsymbol{J}f(\boldsymbol{w})\,\nabla L\circ f^*(\boldsymbol{w}^*) \\
&= -\boldsymbol{P}(\nabla^2 F^*(\boldsymbol{w}^*))^{-1}\nabla L\circ f^*(\boldsymbol{w}^*)
\end{aligned}$$

$$\begin{aligned}
\iff \dot{\boldsymbol{w}} &= -(\boldsymbol{J}f(\boldsymbol{w}))^{-1}\,\boldsymbol{P}\,\nabla L(\boldsymbol{w}) \\
&= -\boldsymbol{P}^\top(\boldsymbol{J}f(\boldsymbol{w}))^{-1}\,\nabla L(\boldsymbol{w}) \\
&= -\boldsymbol{P}^\top(\nabla^2 F(\boldsymbol{w}))^{-1}\,\nabla L(\boldsymbol{w})
\end{aligned}$$

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 $\|\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}\||_2^2$, where $\boldsymbol{X}$ does not have full rank

Continuous GD: $\tau = 0$

$$\begin{aligned}\dot{\boldsymbol{w}}(t) &= -\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w}(t)-\boldsymbol{y})\\ \boldsymbol{w}(t) &= \mathbf{exp}(-\boldsymbol{X}^\top\boldsymbol{X}\,t)(\boldsymbol{w}(0)-\boldsymbol{X}^\dagger\boldsymbol{y})+\boldsymbol{X}^\dagger\boldsymbol{y}\end{aligned}$$

Continuous EGU case: $\tau = 1$

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

$$w_i = \exp\left(-\left(\sum_t x_{t,i}(\boldsymbol{x}_t\cdot\boldsymbol{w}-y_t)w_i - {}^1\!/_2\sum_t x_{t,i}^2 w_i^2\right)\right)$$

$$\boldsymbol{w} = \exp\left(-\left(\left(\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})\right)\odot\boldsymbol{w} - {}^1\!/_2\sum_t \boldsymbol{x}_t^{\odot 2}\odot\boldsymbol{w}^{\odot 2}\right)\right)$$

No closed-form solution for $0 < \tau < q \leq 1$

II) $(2-\tau)$-norm updates for linear regression

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

$$\boldsymbol{w}_\alpha(t) = \operatorname*{argmin}_{\tilde{\boldsymbol{w}}(t)} \Delta_\tau(\dot{\tilde{\boldsymbol{w}}}(t), \alpha\mathbf{1}) + \|\boldsymbol{X}\tilde{\boldsymbol{w}}(t) - \boldsymbol{y}\|_2^2\,, \text{for } \alpha > 0.$$

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

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

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

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

Reparameterizing cont. MD w. link f i.t.o. link g

Theorem For the reparameterization function $\boldsymbol{w} = q(\boldsymbol{u})$ with the property that range$(q) =$ dom(f),
$\dot{g}(\boldsymbol{u}) = -\nabla L \circ q\,(\boldsymbol{u})$ simulates $\dot{f}(\boldsymbol{w}) = -\nabla L(\boldsymbol{w})$ if

$$(\boldsymbol{J}f(\boldsymbol{w}))^{-1} = \boldsymbol{J}q(\boldsymbol{u})\,(\boldsymbol{J}g(\boldsymbol{u}))^{-1}\,(\boldsymbol{J}q(\boldsymbol{u}))^{\top}$$

and $q(\boldsymbol{u}(0)) = \boldsymbol{w}(0)$

For reparameterization as GD use $g = id$

Our main example: EGU as GD

Link

$$f(\boldsymbol{w}) = \log(\boldsymbol{w})$$

Reparameterization

$$\begin{aligned}\boldsymbol{w} &= q(\boldsymbol{u}) := {}^1\!/_4\, \boldsymbol{u} \odot \boldsymbol{u} \\ \boldsymbol{u} &= 2\sqrt{\boldsymbol{w}}\end{aligned}$$

$$(\boldsymbol{J}f(\boldsymbol{w}))^{-1} = (\operatorname{diag}(\boldsymbol{w})^{-1})^{-1} = \operatorname{diag}(\boldsymbol{w})$$

$$\boldsymbol{J}q(\boldsymbol{u})(\boldsymbol{J}q(\boldsymbol{u}))^\top = {}^1\!/_2 \operatorname{diag}(\boldsymbol{u})\, ({}^1\!/_2 \operatorname{diag}(\boldsymbol{u}))^\top = \operatorname{diag}(\boldsymbol{w})$$

Conclusion

$$\dot{\log}(\boldsymbol{w}) = -\nabla L(\boldsymbol{w}) \text{ equals } \dot{\boldsymbol{u}} = -\underbrace{\nabla L \circ q\,(\boldsymbol{u})}_{\nabla_{\boldsymbol{u}} L\,({}^1\!/_4\, \boldsymbol{u} \odot \boldsymbol{u})} = -{}^1\!/_2\, \boldsymbol{u} \odot \nabla L(\boldsymbol{w})$$

Burg as GD

Link

$$f(\boldsymbol{w}) = -\frac{1}{\boldsymbol{w}}$$

Reparameterization

$$\boldsymbol{w} = q(\boldsymbol{u}) := \exp(\boldsymbol{u})$$
$$\boldsymbol{u} = \log(\boldsymbol{w})$$

$$(\boldsymbol{J}f(\boldsymbol{w}))^{-1} = \operatorname{diag}(\frac{1}{\boldsymbol{w} \odot \boldsymbol{w}})^{-1} = \operatorname{diag}(\boldsymbol{w})^2$$

$$\boldsymbol{J}q(\boldsymbol{u})(\boldsymbol{J}q(\boldsymbol{u}))^\top = \operatorname{diag}(\exp(\boldsymbol{u}))\operatorname{diag}(\exp(\boldsymbol{u}))^\top = \operatorname{diag}(\boldsymbol{w})^2$$

Conclusion

$$\dot{\left(-\frac{1}{\boldsymbol{w}}\right)} = -\nabla L(\boldsymbol{w}) \text{ equals } \dot{\boldsymbol{u}} = -\underbrace{\nabla L \circ q\,(\boldsymbol{u})}_{\nabla_{\boldsymbol{u}} L\,(\exp(\boldsymbol{u}))} = -\exp(\boldsymbol{u}) \odot \nabla L(\boldsymbol{w})$$

$\log_\tau \boldsymbol{w} = \frac{1}{1-\tau}(\boldsymbol{w}^{1-\tau} - 1)$ as GD

Link $\quad f(\boldsymbol{w}) = \log_\tau \boldsymbol{w}$

Reparameterization

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

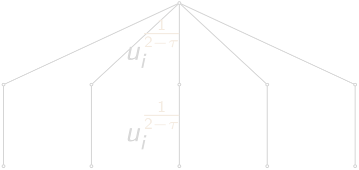
$$\boldsymbol{u} = \frac{2}{2-\tau}\boldsymbol{w}^{\frac{2-\tau}{2}}$$

$$(\boldsymbol{J}\log_\tau(\boldsymbol{w}))^{-1} = (\operatorname{diag}(\boldsymbol{w})^{-\tau})^{-1} = \operatorname{diag}(\boldsymbol{w})^{\tau}$$

$$\boldsymbol{J}q(\boldsymbol{u})(\boldsymbol{J}q(\boldsymbol{u}))^\top = \left(\left(\frac{2-\tau}{2}\right)^{\frac{\tau}{2-\tau}} \operatorname{diag}(\boldsymbol{u})^{\frac{\tau}{2-\tau}}\right)^2 = \operatorname{diag}(\boldsymbol{w})^{\tau}$$

Conclusion

$$\dot{\log}_\tau(\boldsymbol{w}) = -\nabla L(\boldsymbol{w}) \;\text{ equals }\; \dot{\boldsymbol{u}} = -\underbrace{\nabla L \circ q\,(\boldsymbol{u})}_{\nabla_{\boldsymbol{u}} L\left(\left(\frac{2-\tau}{2}\right)^{\frac{2}{2-\tau}} \boldsymbol{u}^{\frac{2}{2-\tau}}\right)} = -\frac{2-\tau}{2}\boldsymbol{u}^{\frac{\tau}{2-\tau}} \odot \nabla L(\boldsymbol{w})$$



$\tau = 1$: EGU $\qquad \tau = 0$: GD

$\log_\tau \boldsymbol{w} = \frac{1}{1-\tau}(\boldsymbol{w}^{1-\tau} - 1)$ as GD

Link $\quad f(\boldsymbol{w}) = \log_\tau \boldsymbol{w}$

Reparameterization

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

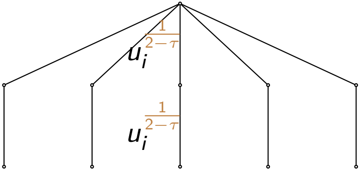
$$\boldsymbol{u} = \frac{2}{2-\tau}\boldsymbol{w}^{\frac{2-\tau}{2}}$$

$$(\boldsymbol{J}\log_\tau(\boldsymbol{w}))^{-1} = (\operatorname{diag}(\boldsymbol{w})^{-\tau})^{-1} = \operatorname{diag}(\boldsymbol{w})^{\tau}$$

$$\boldsymbol{J}q(\boldsymbol{u})(\boldsymbol{J}q(\boldsymbol{u}))^\top = \left(\left(\frac{2-\tau}{2}\right)^{\frac{\tau}{2-\tau}} \operatorname{diag}(\boldsymbol{u})^{\frac{\tau}{2-\tau}}\right)^2 = \operatorname{diag}(\boldsymbol{w})^{\tau}$$

Conclusion

$$\dot{\log}_\tau(\boldsymbol{w}) = -\nabla L(\boldsymbol{w}) \text{ equals } \dot{\boldsymbol{u}} = -\underbrace{\nabla L \circ q(\boldsymbol{u})}_{\nabla_{\boldsymbol{u}} L\left(\left(\frac{2-\tau}{2}\right)^{\frac{2}{2-\tau}} \boldsymbol{u}^{\frac{2}{2-\tau}}\right)} = -\frac{2-\tau}{2}\boldsymbol{u}^{\frac{\tau}{2-\tau}} \odot \nabla L(\boldsymbol{w})$$



$\tau = 1$: EGU $\qquad \tau = 0$: 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 $$\tilde{u}_i = \frac{u_i(1 - \eta \ell_i)}{\| \sum_j u_j^2(1 - \eta \ell_j)^2 \|_2^2}$$

$$\left(\tilde{u}_i^2 = \frac{u_i^2(1 - \eta \ell_i)^2}{\sum_j u_j^2(1 - \eta \ell_j)^2}\right)$$

Exponential alternates w. $\eta/2$

EGUasGD becomes EGU

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

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

EGasGD becomes EG

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

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

Regret bounds

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

update	regret bound
EGUasGD, hinge loss	as Winnow
EGUasGD, linear regression	as EGU but only one-sided case
EGasGD, linear regression	as EG
EGasGD, dot loss	as Hedge

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 $\mathrm{EG}^{\pm}$
- 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 $\mathrm{EG}^{\pm}$
- 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 $\mathrm{EG}^{\pm}$
- 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/