Parameter-free Online Optimization
Part 4

Francesco Orabona   Ashok Cutkosky

ICML 2020

Outline of the Tutorial

- Part 1: Stochastic and Online Convex Optimization
- Part 2: Parameter-free Convex Optimization
- Part 3: More Adaptivity and Applications
- **Part 4: Implementation, Experiments, Open Problems**

Somebody Won a Kaggle Competition Using Parameter-Free Algorithms!

Arturus / **kaggle-web-traffic**

Watch 52 | Star 738 | Fork 321

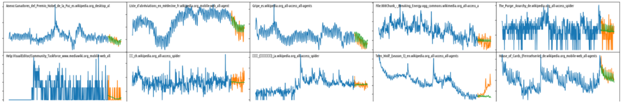
Code | Issues 12 | Pull requests 2 | Projects 0 | Wiki | Insights

1st place solution

kaggle-web-traffic · kaggle · time-series · timeseries · rnn-encoder-decoder · rnn · tensorflow · cudnn · cocob · seq2seq

Kaggle Web Traffic Time Series Forecasting

1st place solution



Training and validation

I used COCOB optimizer (see paper Training Deep Networks without Learning Rates Through Coin Betting) for training, in combination with gradient clipping. COCOB tries to predict optimal learning rate for every training step, so I don't have to tune learning rate at all. It also converges considerably faster than traditional momentum-based optimizers, especially on first epochs, allowing me to stop unsuccessful experiments early.

Convex Regression Tasks

Convex Regression Tasks

- We look at datasets from the OpenML repository, filtering for datasets that have a large number of examples and features.

Convex Regression Tasks

- We look at datasets from the OpenML repository, filtering for datasets that have a large number of examples and features.
- Each dataset specifies a linear regression problem, for which we minimize the absolute loss.

Convex Regression Tasks

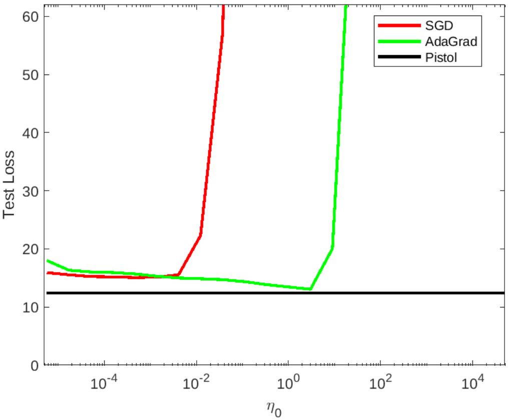
- We look at datasets from the OpenML repository, filtering for datasets that have a large number of examples and features.
- Each dataset specifies a linear regression problem, for which we minimize the absolute loss.
- We compare the parameter-free `Pistol` algorithm (which is available in the Vowpal Wabbit library) to `AdaGrad` and `SGD`.
  - `Pistol` uses per-coordinate updates, and guarantees the regret bound $O(\sum_{i=1}^{d} |x_i^\star| \sqrt{\sum_{t=1}^{T} |g_{t,i}| \log(|x_i^\star| T)})$.

Convex Regression Tasks

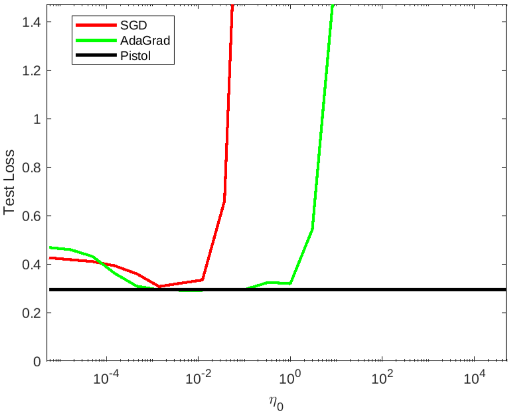
- We look at datasets from the OpenML repository, filtering for datasets that have a large number of examples and features.
- Each dataset specifies a linear regression problem, for which we minimize the absolute loss.
- We compare the parameter-free `Pistol` algorithm (which is available in the Vowpal Wabbit library) to `AdaGrad` and `SGD`.
  - `Pistol` uses per-coordinate updates, and guarantees the regret bound $O(\sum_{i=1}^{d} |x_i^\star| \sqrt{\sum_{t=1}^{T} |g_{t,i}| \log(|x_i^\star| T)})$.
- For `AdaGrad` and `SGD`, we sweep the learning rate η_0.

[Orabona, NeurIPS'14] [`github.com/VowpalWabbit/vowpal_wabbit`]

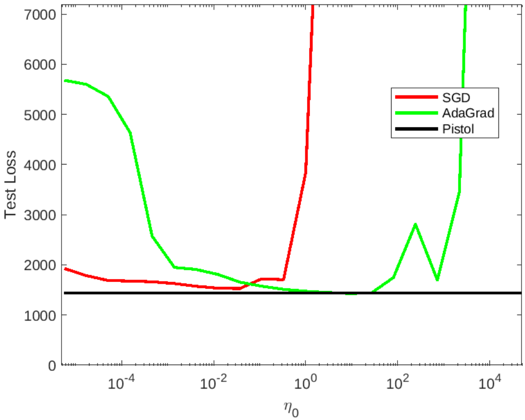
wisconsin (1187)



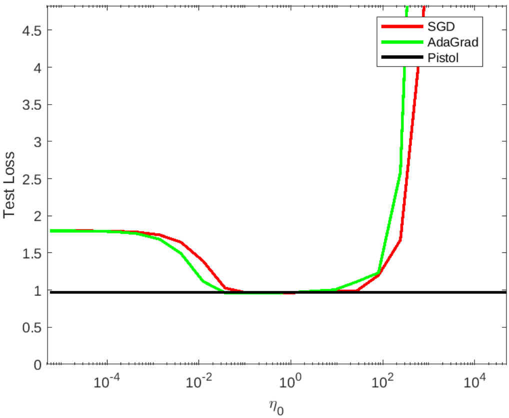
cleveland (1188)



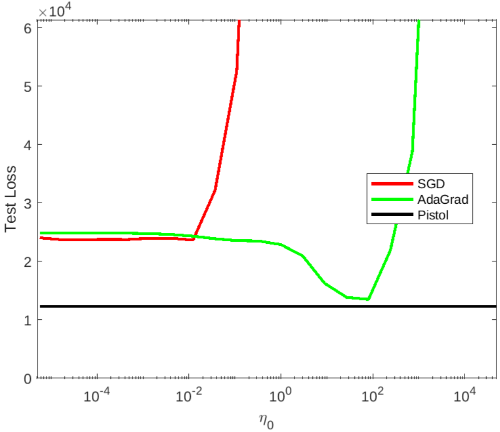
auto price (1189)



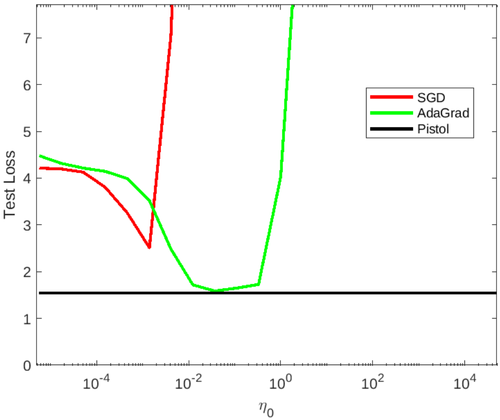
2dplanes (215)



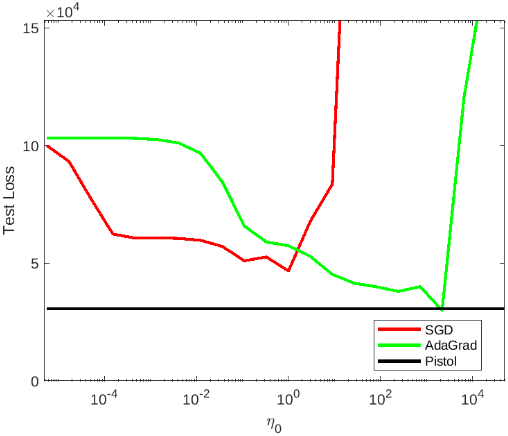
2dplanes (218)



house 8L (344)

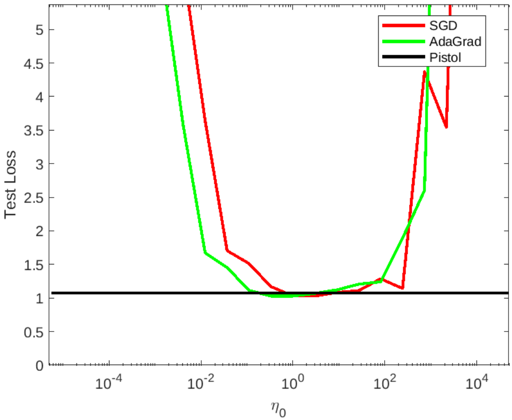

Test Loss
0
1
2
3
4
5
6
7
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
η_0
SGD
AdaGrad
Pistol

houses (573)



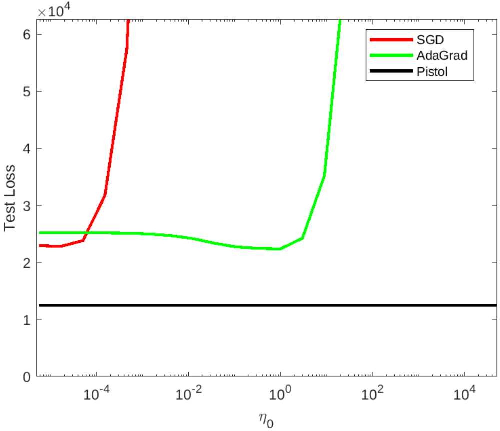
$\times 10^4$
15
10
5
0
Test Loss
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
η_0
SGD
AdaGrad
Pistol

fried (564)



SGD
AdaGrad
Pistol
Test Loss
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
η_0

house 16H (574)



Takeaways from Convex Experiments

Takeaways from Convex Experiments

1. Not only is there good theory, this actually works!

Takeaways from Convex Experiments

1. Not only is there good theory, this actually works!
2. There is no single “common default” learning rate that can work for all datasets.

Takeaways from Convex Experiments

1. Not only is there good theory, this actually works!
2. There is no single “common default” learning rate that can work for all datasets.
3. Often, the parameter-free algorithm will outperform even a tuned gradient descent!

- Our theoretical analysis relies strongly on global convexity properties.

Deep Learning

- Our theoretical analysis relies strongly on global convexity properties.But let’s try to train some neural networks anyway.

Deep Learning

- Our theoretical analysis relies strongly on global convexity properties.But let's try to train some neural networks anyway.
- Some tweaks are necessary to get the best performance.

- Our theoretical analysis relies strongly on global convexity properties.But let's try to train some neural networks anyway.
- Some tweaks are necessary to get the best performance.
- Providing a solid theoretical foundation here is a great open problem!

Dynamics of Coin Betting (Recall from Section 2)

- Remember the optimal gradient descent tuning:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \frac{\|\boldsymbol{x}^\star\|}{G\sqrt{T}}\boldsymbol{g}_t$$

Dynamics of Coin Betting (Recall from Section 2)

- Remember the optimal gradient descent tuning:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \frac{\|\boldsymbol{x}^\star\|}{G\sqrt{T}}\boldsymbol{g}_t$$

- The typical betting fraction satisfies $\beta \approx \Theta\left(\frac{1}{G\sqrt{T}}\right)$.

Dynamics of Coin Betting (Recall from Section 2)

- Remember the optimal gradient descent tuning:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \frac{\|\boldsymbol{x}^\star\|}{G\sqrt{T}}\boldsymbol{g}_t$$

- The typical betting fraction satisfies $\beta \approx \Theta\left(\frac{1}{G\sqrt{T}}\right)$.
- Supposing a 1-d problem with $x_t \geq 0$ always, we have:

$$\begin{aligned} x_{t+1} &= \beta \text{Wealth}_t = \beta \text{Wealth}_{t-1} - \beta g_t x_t \\ &= x_t - \frac{g_t |x_t|}{\sqrt{T}} \end{aligned}$$

Dynamics of Coin Betting (Recall from Section 2)

- Remember the optimal gradient descent tuning:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \frac{\|\boldsymbol{x}^\star\|}{G\sqrt{T}}\boldsymbol{g}_t$$

- The typical betting fraction satisfies $\beta \approx \Theta\left(\frac{1}{G\sqrt{T}}\right)$.
- Supposing a 1-d problem with $x_t \geq 0$ always, we have:

$$\begin{aligned} x_{t+1} &= \beta \text{Wealth}_t = \beta \text{Wealth}_{t-1} - \beta g_t x_t \\ &= x_t - \frac{g_t |x_t|}{\sqrt{T}} \end{aligned}$$

- Conclusion: parameter-free algorithms behave similarly to gradient descent with learning rate $\eta_t = \frac{\|\boldsymbol{x}_t\|}{G\sqrt{T}}$.

Dynamics of Coin Betting (Recall from Section 2)

- Remember the optimal gradient descent tuning:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \frac{\|\boldsymbol{x}^\star\|}{G\sqrt{T}}\boldsymbol{g}_t$$

- The typical betting fraction satisfies $\beta \approx \Theta\left(\frac{1}{G\sqrt{T}}\right)$.
- Supposing a 1-d problem with $x_t \geq 0$ always, we have:

$$\begin{aligned} x_{t+1} &= \beta \text{Wealth}_t = \beta \text{Wealth}_{t-1} - \beta g_t x_t \\ &= x_t - \frac{g_t |x_t|}{\sqrt{T}} \end{aligned}$$

- Conclusion: parameter-free algorithms behave similarly to gradient descent with learning rate $\eta_t = \frac{\|\boldsymbol{x}_t\|}{G\sqrt{T}}$.
- Coincidentally, scaling the "learning rate" by the magnitude of the weights has been recently suggested as an empirically useful heuristic in deep learning [You et al., 2017, 2019; Bernstein et al., arXiv'20].

Stability and Initialization

- In convex problems, it is always possible to “recover” from even arbitrarily crazy behavior because the gradient provides global information.

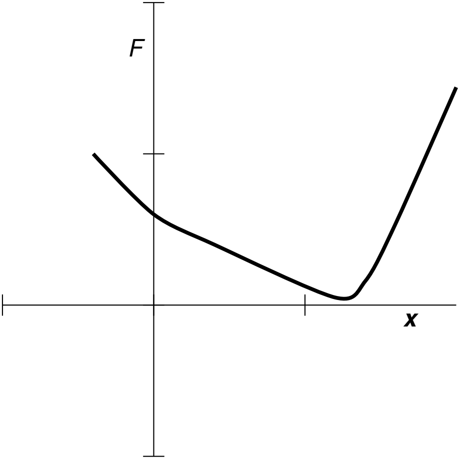
Stability and Initialization

- In convex problems, it is always possible to “recover” from even arbitrarily crazy behavior because the gradient provides global information.
- In non-convex problems, this may not be true.

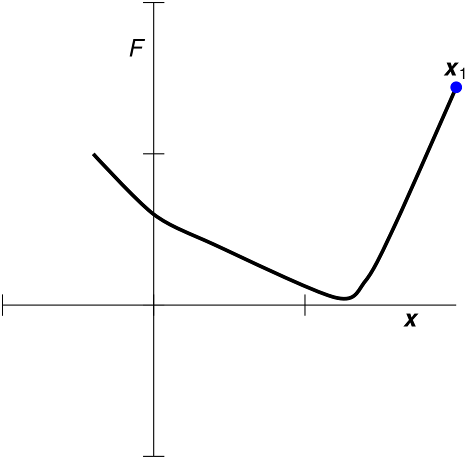
Stability and Initialization

- In convex problems, it is always possible to “recover” from even arbitrarily crazy behavior because the gradient provides global information.
- In non-convex problems, this may not be true.
- Neural network initialization schemes seem to be important in training - we don’t want to destroy this initialization with some bad early updates.

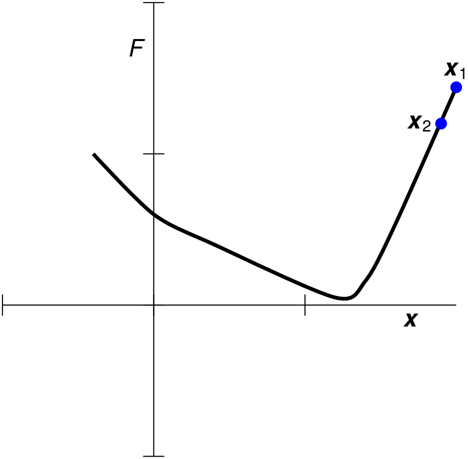
Parameter-Free Algorithms as Binary Search



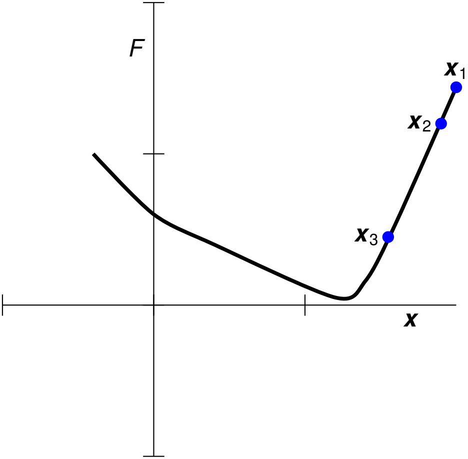
Parameter-Free Algorithms as Binary Search



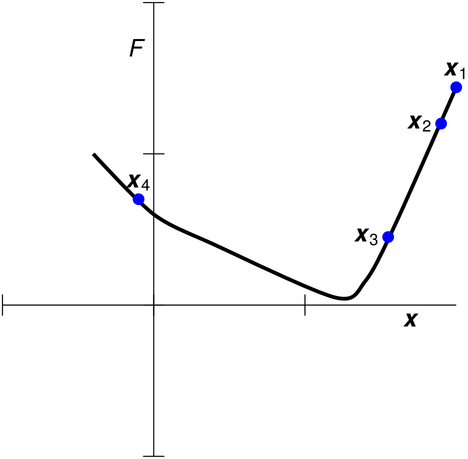
Parameter-Free Algorithms as Binary Search



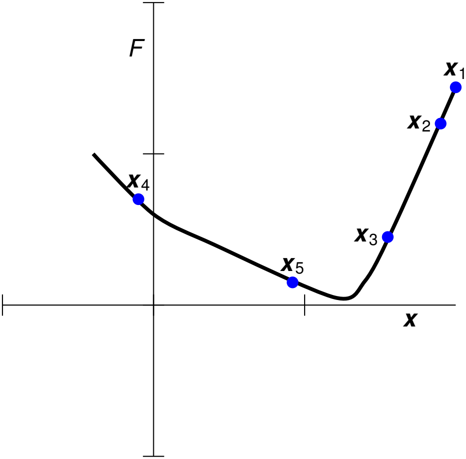
Parameter-Free Algorithms as Binary Search



Parameter-Free Algorithms as Binary Search

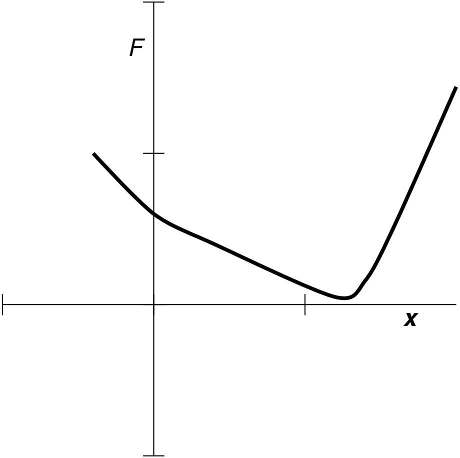


Parameter-Free Algorithms as Binary Search



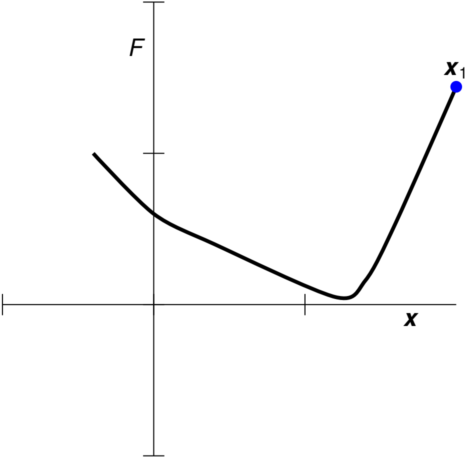
Parameter-Free Algorithms as Binary Search

What if we use a different search factor?

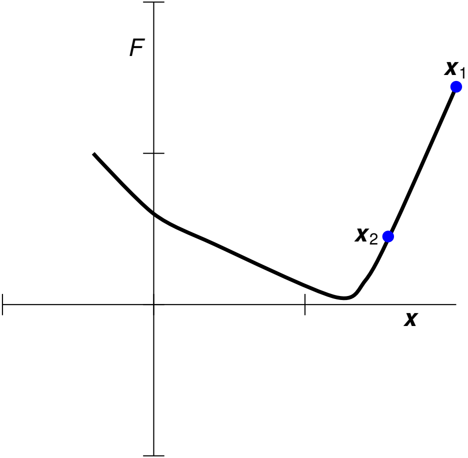


Parameter-Free Algorithms as Binary Search

What if we use a different search factor?

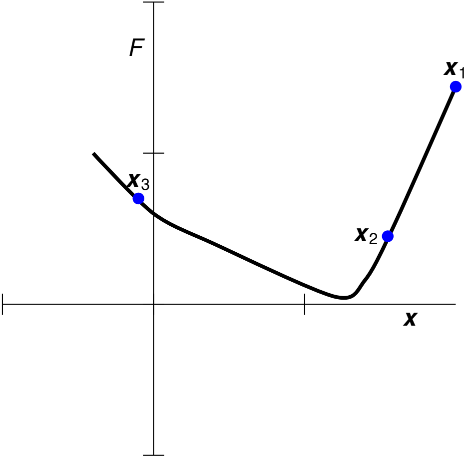


What if we use a different search factor?



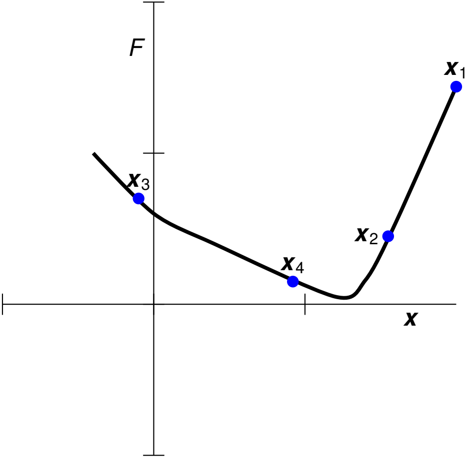
Parameter-Free Algorithms as Binary Search

What if we use a different search factor?



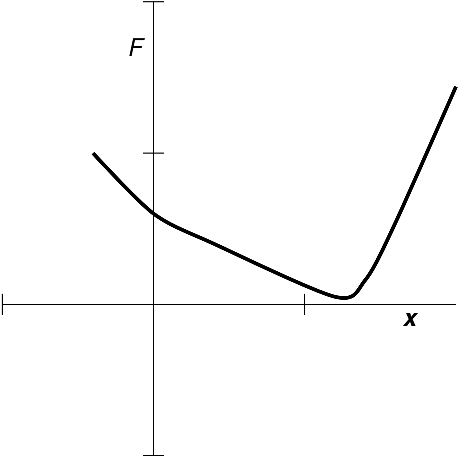
Parameter-Free Algorithms as Binary Search

What if we use a different search factor?

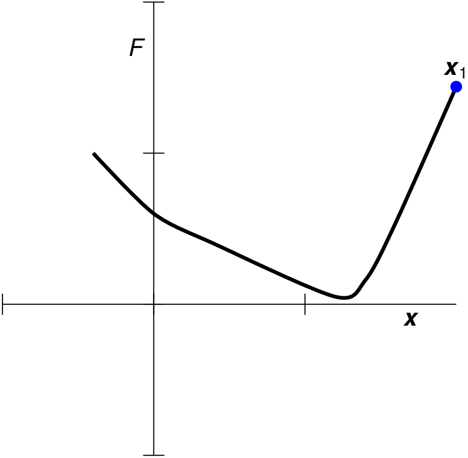


Parameter-Free Algorithms as Binary Search

What if we use a different search factor?

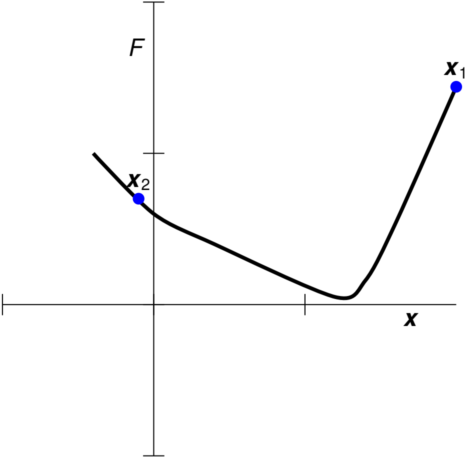


What if we use a different search factor?

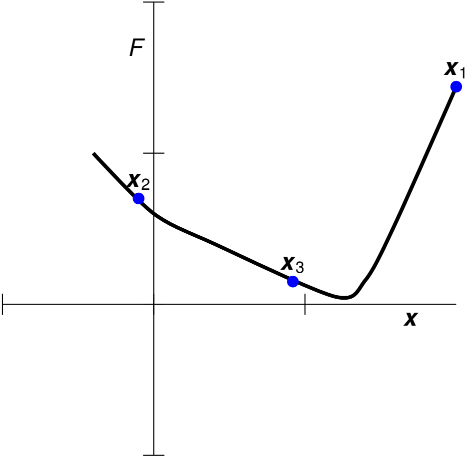


Parameter-Free Algorithms as Binary Search

What if we use a different search factor?

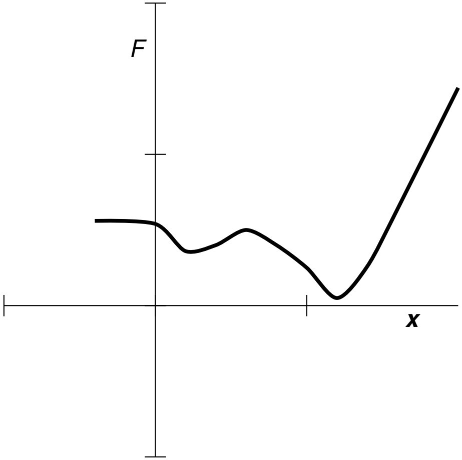


What if we use a different search factor?

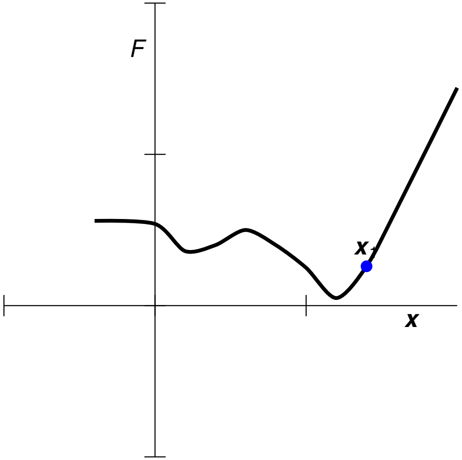


Binary Search on Non-Convex Objectives

First-order methods on non-convex objectives can be sensitive to large jumps:

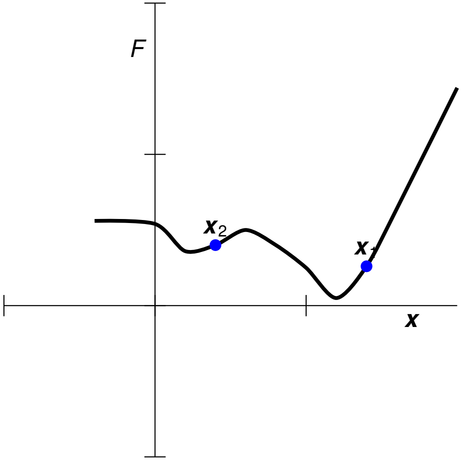


First-order methods on non-convex objectives can be sensitive to large jumps:



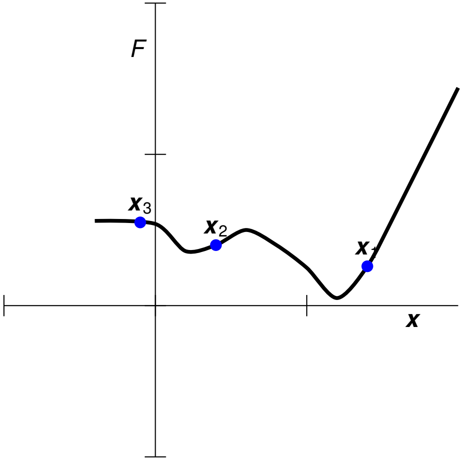
Binary Search on Non-Convex Objectives

First-order methods on non-convex objectives can be sensitive to large jumps:



Binary Search on Non-Convex Objectives

First-order methods on non-convex objectives can be sensitive to large jumps:



Early-stage Dynamics of Coin-Betting Algorithms

Early-stage Dynamics of Coin-Betting Algorithms

- The “binary search factor” of a coin-betting algorithm is determined by the initial wealth.

Early-stage Dynamics of Coin-Betting Algorithms

- The “binary search factor” of a coin-betting algorithm is determined by the initial wealth.

Early-stage Dynamics of Coin-Betting Algorithms

- The “binary search factor” of a coin-betting algorithm is determined by the initial wealth.
- Small initial wealth means we will make very small changes at first, while large initial wealth allows for large changes.

Early-stage Dynamics of Coin-Betting Algorithms

- The "binary search factor" of a coin-betting algorithm is determined by the initial wealth.
- Small initial wealth means we will make very small changes at first, while large initial wealth allows for large changes.
- The wealth changes exponentially fast, so one expects the algorithm to be very insensitive to the initial value.

Early-stage Dynamics of Coin-Betting Algorithms

- The “binary search factor” of a coin-betting algorithm is determined by the initial wealth.
- Small initial wealth means we will make very small changes at first, while large initial wealth allows for large changes.
- The wealth changes exponentially fast, so one expects the algorithm to be very insensitive to the initial value.
- Unfortunately, since we cannot recover from bad behavior in the non-convex setting, we may need to be more conservative and start with smaller initial wealth.

Stability and Initialization

- For some large language models, it is necessary to decrease the initial wealth smaller than 1.
- Keep initial betting fraction from being too large initially:

```
beta = clip(true_beta, -0.1 * sum_grad, 0.1 * sum_grad)
```

- This formula is motivated by the fact that most algorithms set:

```
true_beta =  sum_grad * some_multiplier
```

Tweaks

- Neural networks are NOT initialized to zero. Record the initial value of the weights and translate the optimizer outputs by these values
- Normalize gradients by the maximum gradient.

Tweaks

- Neural networks are NOT initialized to zero. Record the initial value of the weights and translate the optimizer outputs by these values
- Normalize gradients by the maximum gradient.

```
def apply_gradient(grad, var):
    max_grad = max(grad, max_grad)
    grad = grad/max_grad
    offset = get_param_free_output(grad, var)
    initial_value = get_initial_value(var)
    var.assign(initial_value + offset)
```

More Tweaks

- Add average of previous outputs (similar to the reduction for strongly-convex adaptivity)

```python
def apply_gradient(grad, var):
    max_grad = max(grad, max_grad)
    grad = grad/max_grad
    offset = get_param_free_output(grad, var)
    grad_norm_sq = squared_norm(grad)
    sum_squared_grad += grad_norm_sq
    weighted_sum_offset += grad_norm_sq * offset
    average_offset = weighted_sum_offset/sum_squared_grad
    initial_value = get_initial_value(var)
    var.assign(initial_value + average_offset + offset)
```

One More Algorithm: Parameter-Free Preconditioning

- A preconditioned regret bound looks like:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \leq \sqrt{d \sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}^\star \rangle^2}$$

One More Algorithm: Parameter-Free Preconditioning

- A <u>preconditioned</u> regret bound looks like:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \le \sqrt{d \sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}^\star \rangle^2}$$

- This might be much better than $\|\boldsymbol{x}^\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2}$, except for the d dependence.

One More Algorithm: Parameter-Free Preconditioning

- A preconditioned regret bound looks like:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \leq \sqrt{d \sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}^\star \rangle^2}$$

- This might be much better than $\|\boldsymbol{x}^\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2}$, except for the d dependence.
- Algorithms that achieve this typically require expensive matrix calculations.

One More Algorithm: Parameter-Free Preconditioning

- A preconditioned regret bound looks like:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \leq \sqrt{d \sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}^\star \rangle^2}$$

- This might be much better than $\|\boldsymbol{x}^\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2}$, except for the d dependence.
- Algorithms that achieve this typically require expensive matrix calculations.
- Recent work has suggested that such algorithms are helpful in optimizing neural networks [Gupta et al., ICML'18; Agarwal et al., ICML'19]

[Koren&Livni, NeurIPS'17; Kotlowski, ALT'17]

Faster Preconditioning (Sometimes) Using Coin-Betting

Using a coin-betting approach, we can obtain an algorithm such that:

Faster Preconditioning (Sometimes) Using Coin-Betting

Using a coin-betting approach, we can obtain an algorithm such that:

- In all cases, it is guaranteed:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \leq \tilde{O}\left(\|\boldsymbol{x}^\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2} \right)$$

Faster Preconditioning (Sometimes) Using Coin-Betting

Using a coin-betting approach, we can obtain an algorithm such that:

- In all cases, it is guaranteed:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \le \tilde{O}\left(\|\boldsymbol{x}^\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2} \right)$$

- If $\left|\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}^\star \rangle\right| \ge \|\boldsymbol{x}_\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2}$, then:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \le \tilde{O}\left(\sqrt{\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_\star \rangle^2} \right) \qquad \text{(note lack of } \sqrt{d}\text{)}$$

Faster Preconditioning (Sometimes) Using Coin-Betting

Using a coin-betting approach, we can obtain an algorithm such that:

- In all cases, it is guaranteed:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \le \tilde{O}\left(\|\boldsymbol{x}^\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2} \right)$$

- If $\left| \sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}^\star \rangle \right| \ge \|\boldsymbol{x}_\star\|_2 \sqrt{\sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2}$, then:

$$\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^\star \rangle \le \tilde{O}\left(\sqrt{\sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_\star \rangle^2} \right) \qquad \text{(note lack of } \sqrt{d}\text{)}$$

- The algorithm runs in linear time (same as SGD).

[Cutkosky&Sarlos, ICML'19]

Vector Betting Fractions

Vector Betting Fractions

- In the previous sections, we focused on 1-dimensional coin-betting algorithms, but this is not necessary.

Vector Betting Fractions

- In the previous sections, we focused on 1-dimensional coin-betting algorithms, but this is not necessary.
- All the coin-betting framework seamlessly generalizes to vector betting:

$$\text{Wealth}_T = 1 - \sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{x}_t \rangle$$

$$\boldsymbol{x}_t = \boldsymbol{\beta}_t \text{Wealth}_{t-1}, \ \|\boldsymbol{\beta}_t\| \leq 1$$

Parameter-Free Inside Parameter-Free

Parameter-Free Inside Parameter-Free

- In the previous section, we saw how to write the problem of choosing betting fractions β_t as itself an exp-concave online learning problem.

Parameter-Free Inside Parameter-Free

- In the previous section, we saw how to write the problem of choosing betting fractions β_t as itself an exp-concave online learning problem.
- Unfortunately, taking advantage of exp-concavity is slow for learning high-dimensional quantities.

Parameter-Free Inside Parameter-Free

- In the previous section, we saw how to write the problem of choosing betting fractions β_t as itself an exp-concave online learning problem.
- Unfortunately, taking advantage of exp-concavity is slow for learning high-dimensional quantities.
- Instead of using exp-concavity, we can compute that the optimal $\|\boldsymbol{\beta}^\star\|$ is usually very small ($\approx 1/\sqrt{T}$).

Parameter-Free Inside Parameter-Free

- In the previous section, we saw how to write the problem of choosing betting fractions β_t as itself an exp-concave online learning problem.
- Unfortunately, taking advantage of exp-concavity is slow for learning high-dimensional quantities.
- Instead of using exp-concavity, we can compute that the optimal $\|\boldsymbol{\beta}^\star\|$ is usually very small ($\approx 1/\sqrt{T}$).
- This means that a parameter-free algorithm can learn $\boldsymbol{\beta}^\star$ with error $\tilde{O}(\|\boldsymbol{\beta}^\star\|\sqrt{T}) = \tilde{O}(1)$.

Parameter-Free Inside Parameter-Free

- In the previous section, we saw how to write the problem of choosing betting fractions β_t as itself an exp-concave online learning problem.
- Unfortunately, taking advantage of exp-concavity is slow for learning high-dimensional quantities.
- Instead of using exp-concavity, we can compute that the optimal $\|\boldsymbol{\beta}^\star\|$ is usually very small ($\approx 1/\sqrt{T}$).
- This means that a parameter-free algorithm can learn $\boldsymbol{\beta}^\star$ with error $\tilde{O}(\|\boldsymbol{\beta}^\star\|\sqrt{T}) = \tilde{O}(1)$.
- When $\left|\sum_{t=1}^T \langle \boldsymbol{g}_t, \boldsymbol{x}^\star \rangle\right| \geq \|\boldsymbol{x}_\star\|_2 \sqrt{\sum_{t=1}^T \|\boldsymbol{g}_t\|_2^2}$, a more refined analysis using the vector betting fractions shows that we even get the preconditioned bound.

[Cutkosky&Sarlos, ICML'19]

Recursive Optimizer

Initialize "inner" parameter-free algorithm $\mathcal{A}$.
Initialize $\text{Wealth}_0 = 1$
for $t = 1$ **to** T **do**
    Get $\boldsymbol{\beta}_t$ from $\mathcal{A}$.
    Play $\boldsymbol{x}_t = \boldsymbol{\beta}_t \text{Wealth}_{t-1}$
    Get gradient $\boldsymbol{g}_t$, define $\ell_t(\boldsymbol{\beta}) = -\log(1 - \langle \boldsymbol{\beta}, \boldsymbol{g}_t \rangle)$
    Compute $\boldsymbol{z}_t = \ell_t'(\boldsymbol{\beta}_t) = \frac{\boldsymbol{g}_t}{1 - \langle \boldsymbol{\beta}_t, \boldsymbol{g}_t \rangle}$
    Send $\boldsymbol{z}_t$ to $\mathcal{A}$.
    Set $\text{Wealth}_t = \text{Wealth}_{t-1} - \langle \boldsymbol{g}_t, \boldsymbol{x}_t \rangle$
end for

Two Lessons

1. The property of having small regret for small comparators can be used in surprising and non-intuitive ways.
2. Parameter-free algorithms can obtain bounds which are better than any currently known gradient-descent-like method, <u>even with oracle tuning</u>.

Deep Learning experiments

Deep Learning experiments

- We train this preconditioned parameter-free optimizer (`Recursive`) on several image recognition and language modeling architectures.

Deep Learning experiments

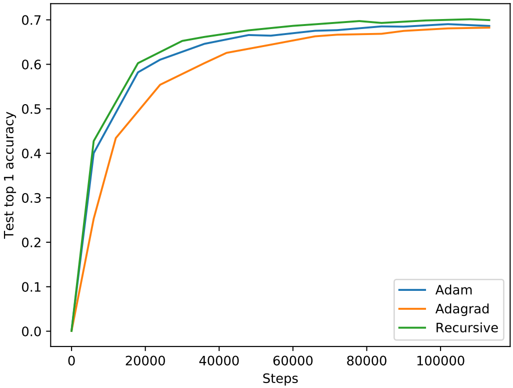
- We train this preconditioned parameter-free optimizer (`Recursive`) on several image recognition and language modeling architectures.
- We compare to `Adam` and `Adagrad` with fixed learning rates (no warm-up and decay or other complicated schedules).

Deep Learning experiments

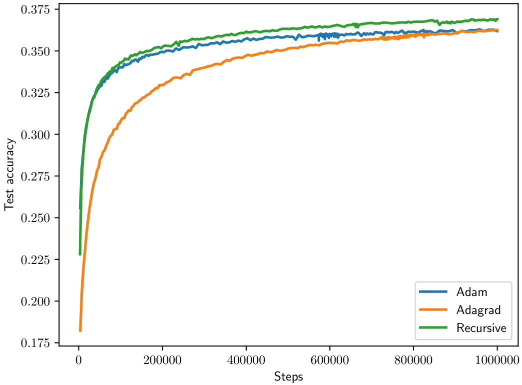
- We train this preconditioned parameter-free optimizer (`Recursive`) on several image recognition and language modeling architectures.
- We compare to `Adam` and `Adagrad` with fixed learning rates (no warm-up and decay or other complicated schedules).
- Caveat: if one does tune a more complicatedschedule it is possible to get better results than we'll show.

`github.com/google-research/google-research/tree/master/recursive_optimizer`

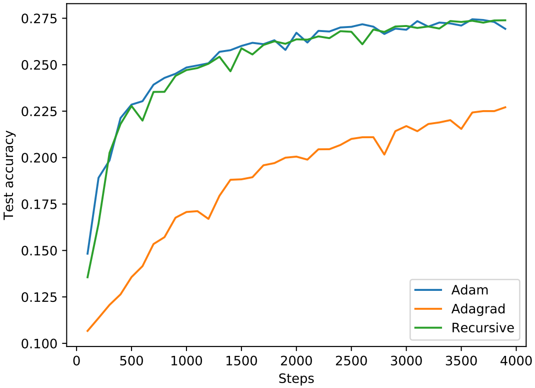
ResNet50 Imagenet



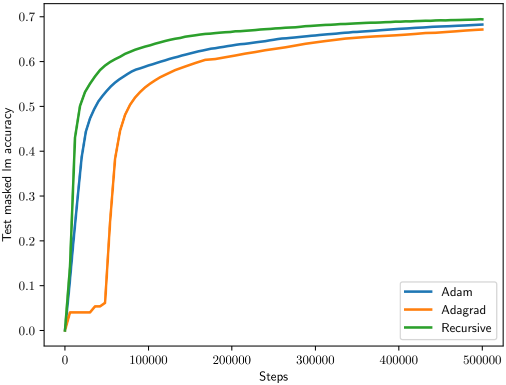
Transformer LM1B



Transformer Penn Tree Bank



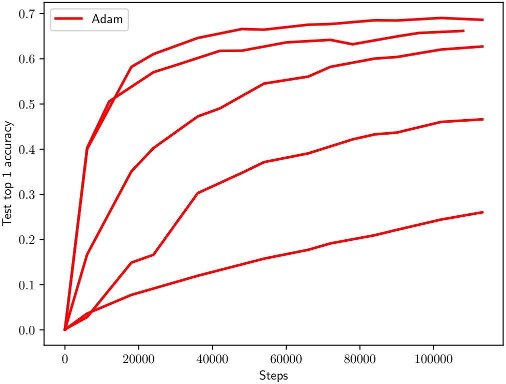
BERT-Base Pretraining



Robustness to Initial Wealth

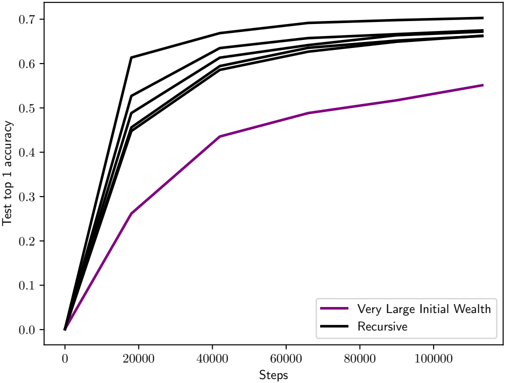
- Unfortunately, we now needed to make sure that initial wealth is not too big.
- This is a parameter, but since wealth changes exponentially fast, we might hope that the algorithm is very robust to making the initial wealth very small.

ResNet50 Imagenet Robustness



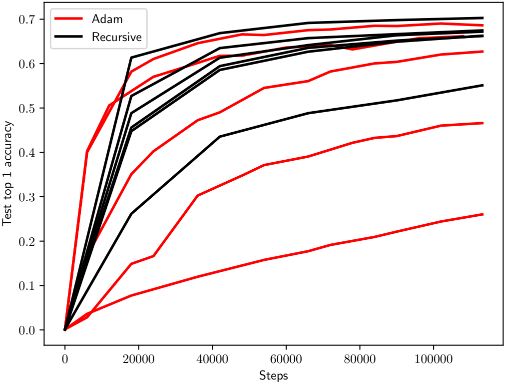
Robustness to learning rate.

ResNet50 Imagenet Robustness



Robustness to initial wealth.

ResNet50 Imagenet Robustness



Robustness to initial wealth or learning rate.

Other Topics

Other Topics

- Parameter-free learning with privacy [Jun&Orabona, COLT'19; van der Hoeven, NeurIPS'19]

Other Topics

- Parameter-free learning with privacy [Jun&Orabona, COLT'19; van der Hoeven, NeurIPS'19]
- Connections to variance-reduction [Cutkosky&Busa-Fekete, NeurIPS'18]

Other Topics

- Parameter-free learning with privacy [Jun&Orabona, COLT'19; van der Hoeven, NeurIPS'19]
- Connections to variance-reduction [Cutkosky&Busa-Fekete, NeurIPS'18]
- Scale Invariant learning [Kotlowski, ALT'17; Kempka et al., ICML'19; Mhammedi&Koolen, COLT'20]

Open Problems

Open Problems

- What can be said theoretically in the non-convex realm? Is there a more principled way to design parameter-free non-convex algorithms?

Open Problems

- What can be said theoretically in the non-convex realm? Is there a more principled way to design parameter-free non-convex algorithms?
- What if you are only allowed to query function values rather than gradients (i.e. bandit feedback). Can we build analogous bounds in this case?

Open Problems

- What can be said theoretically in the non-convex realm? Is there a more principled way to design parameter-free non-convex algorithms?
- What if you are only allowed to query function values rather than gradients (i.e. bandit feedback). Can we build analogous bounds in this case?
- Can we measure complexity with arbitrary non-norm function (i.e. some kind of Bregman divergence?).

Open Problems

- What can be said theoretically in the non-convex realm? Is there a more principled way to design parameter-free non-convex algorithms?
- What if you are only allowed to query function values rather than gradients (i.e. bandit feedback). Can we build analogous bounds in this case?
- Can we measure complexity with arbitrary non-norm function (i.e. some kind of Bregman divergence?).
- In stochastic settings, can we leverage some dynamics to get around lower bounds of $\tilde{\Omega}(\|\boldsymbol{x}^\star\|G\sqrt{T} + G\|\boldsymbol{x}^\star\|^3)$? Intuitively, the the problem is that the gradients can get "too big too fast" in the adversarial model. Empirically, this never actually happens.

Open Problems

- What can be said theoretically in the non-convex realm? Is there a more principled way to design parameter-free non-convex algorithms?
- What if you are only allowed to query function values rather than gradients (i.e. bandit feedback). Can we build analogous bounds in this case?
- Can we measure complexity with arbitrary non-norm function (i.e. some kind of Bregman divergence?).
- In stochastic settings, can we leverage some dynamics to get around lower bounds of $\tilde{\Omega}(\|\boldsymbol{x}^\star\|G\sqrt{T} + G\|\boldsymbol{x}^\star\|^3)$? Intuitively, the the problem is that the gradients can get "too big too fast" in the adversarial model. Empirically, this never actually happens.
- High probability bounds?

Section 4 Summary

1. Parameter-Free Algorithms work well on benchmark tasks in convex and even non-convex settings!
2. Sometimes Parameter-Free bounds can <u>exceed</u> tuned SGD bounds.
3. There are lots of good open problems to solve!