UNIVERSITY OF
COPENHAGEN

Variations on Stochastic Gradient Descent

Computational Statistics

Johan Larsson

Department of Mathematical Sciences, University of Copenhagen

March 31, 2026

Last Times: Rcpp and SGD

Rcpp

We introduced Rcpp and how to use it to speed up R code.

SGD
Introduced stochastic gradient descent (SGD) and mini-batch version thereof.

Algorithm 1: Mini-Batch SGD

Data: $t_0 > 0$

for $k \leftarrow 1, 2, \ldots$ **do**

$A_k \leftarrow$ random mini-batch of m samples;

$x_k \leftarrow x_{k-1} - \frac{t_k}{|A_k|} \sum_{i \in A_k} \nabla f_i(x_{k-1})$;

Problems
We indicated that there were problems with vanilla SGD: poor convergence, erratic behavior.

Today

How can we improve stochastic gradient descent?

Momentum
Base update on combination of gradient step and previous point.

Two versions: Polyak and Nesterov momentum

Adaptive Gradients
Adapt learning rate to particular feature.

Basic Idea

Give the particle **momentum**: like a heavy ball

Not specific to SGD! To simplify, we will discuss standard gradient descent first.

Polyak Momentum

Classical version (Polyak 1964):

- $\mu \in [0, 1)$ decides strength of momentum; $\mu = 0$ gives standard gradient descent
- Guaranteed convergence for quadratic functions

Algorithm 2: GD with Polyak Momentum

Data: $t > 0$, $\mu \in [0, 1)$, $x_{-1} = x_0$

for $k \leftarrow 1, 2, \ldots$ **do**

$$x_k \leftarrow x_{k-1} - t\nabla f(x_{k-1}) + \mu(x_{k-1} - x_{k-2});$$

Polyak Momentum in Practice

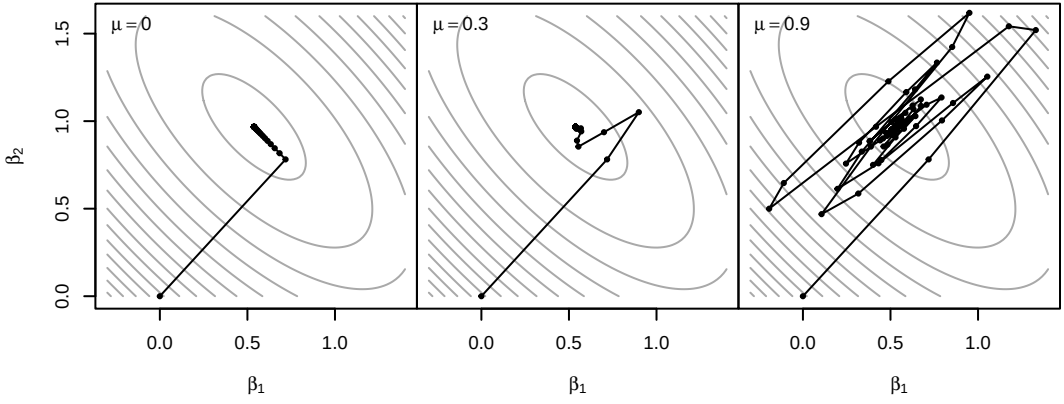


Figure 1: Trajectories of GD for different momentum values for a least-squares problem

Local Minima

For nonconvex f, gradient descent may get stuck at a local minimum.

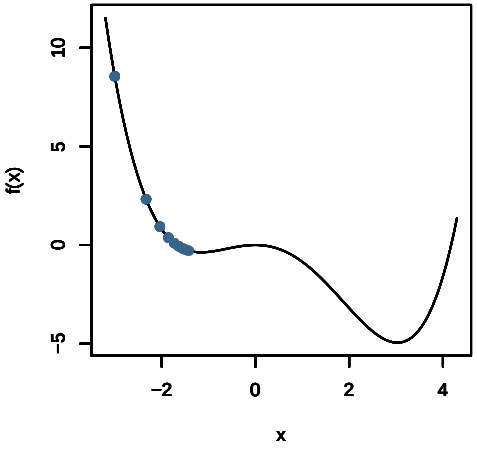


Figure 2: No momentum ($\mu = 0$).

Escaping Local Minima

With momentum, we can (sometimes) remedy this problem.

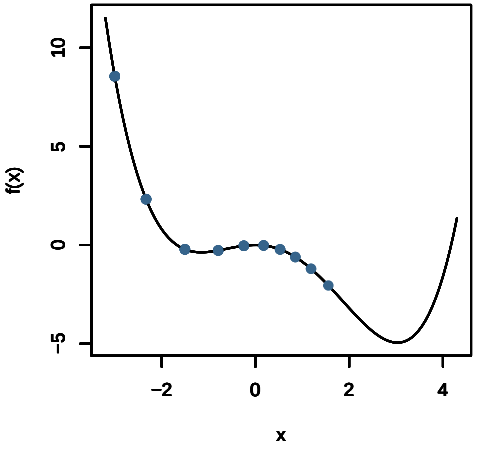


Figure 3: With momentum ($\mu = 0.8$).

Convergence Failure

For some problems, the momentum method may fail to converge (Lessard, Recht, and Packard 2016).

Example

Consider

$$f(x) = \begin{cases} \frac{25x^2}{2} & \text{if } x < 1, \\ \frac{x^2}{2} + 24x - 12 & \text{if } 1 \leq x < 2, \\ \frac{25x^2}{2} - 24x + 36 & \text{if } x \geq 2. \end{cases}$$

For an "optimal" step size $t = 1/L$ with $L = 25$, GD with momentum converges to three limit points.

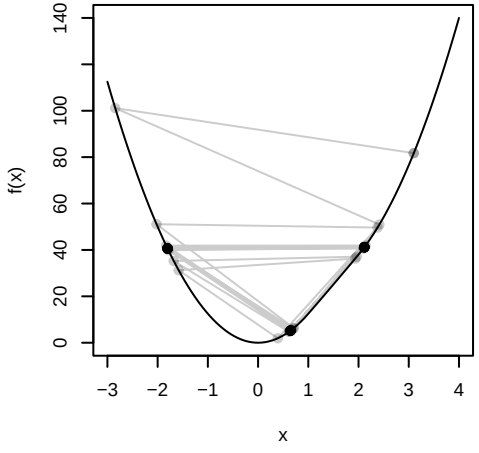


Figure 4: Initialized at $x_0 = 3.2$, the algorithm fails to converge.

Nesterov Momentum

Algorithm 3: GD with Nesterov Momentum

Data: $t > 0$, $\mu \in [0, 1)$

for $k \leftarrow 1, 2, \ldots$ **do**

$v_k \leftarrow x_{k-1} - t\nabla f(x_{k-1})$;

$x_k \leftarrow v_k + \mu(v_k - v_{k-1})$;

Overcomes convergence problem of classical (Polyak) momentum.

First appeared in article by Nesterov (1983).

Consider two iterations of Nesterov algorithm:

$$
\begin{aligned}
v_k &= x_{k-1} - t\nabla f\left(x_{k-1}\right) \\
x_k &= v_k + \mu\left(v_k - v_{k-1}\right) \\
v_{k+1} &= x_k - t\nabla f\left(x_k\right) \\
x_{k+1} &= v_{k+1} + \mu\left(v_{k+1} - v_k\right)
\end{aligned}
$$

Idea: focus on interim step:

$$
\begin{aligned}
x_k &= v_k + \mu\left(v_k - v_{k-1}\right) \\
v_{k+1} &= x_k - t\nabla f\left(x_k\right)
\end{aligned}
$$

Reindex:

$$
\begin{aligned}
x_k &= v_{k-1} + \mu\left(v_{k-1} - v_{k-2}\right) \\
v_k &= x_k - t\nabla f\left(x_k\right)
\end{aligned}
$$

But since $x_k = v_k$ for $k = 1$ by construction, we can swap x_k for v_k and get the update

$$
x_k = x_{k-1} + \mu(x_{k-1} - x_{k-2}) - t\nabla f(x_k + \mu(x_{k-1} - x_{k-2})).
$$

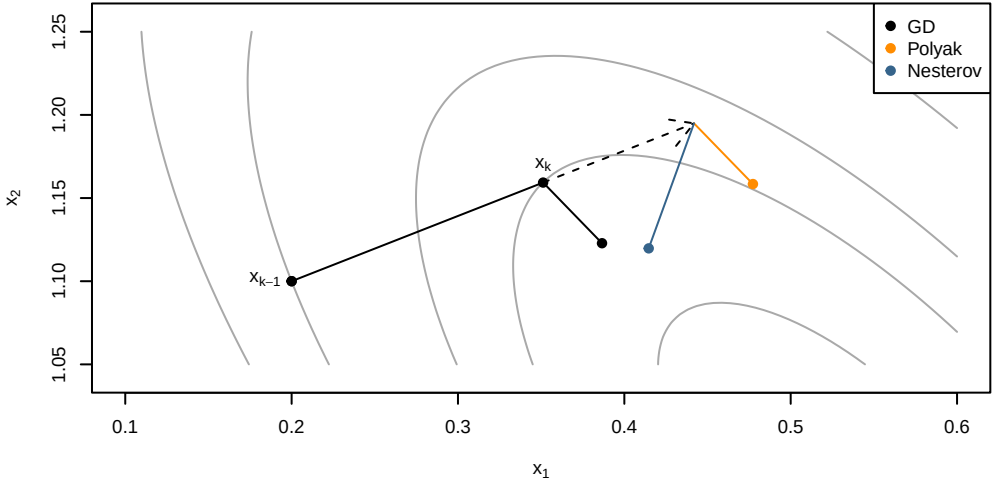


Figure 5: Illustration of Nesterov and Polyak momentum

Optimal Momentum

For gradient descent with $t = 1/L$, the optimal choice of μ_k for general convex and smooth f is

$$\mu_k = \frac{a_{k-1} - 1}{a_k}$$

for a series of

$$a_k = \frac{1 + \sqrt{4a_{k-1}^2 + 1}}{2}$$

with $a_0 = 1$ (and hence $\mu_1 = 0$).

First step ($k = 1$) is just standard gradient descent.

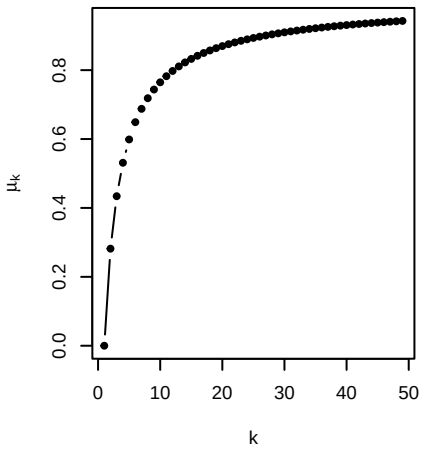


Figure 6: Optimal momentum for Nesterov acceleration (for GD).

Convergence rate with Nesterov acceleration goes from $O(1/k)$ to $O(1/k^2)$

This is **optimal** for a first-order method. But it is no longer a *descent* method!

Convergence improves further for quadratic and strongly convex!

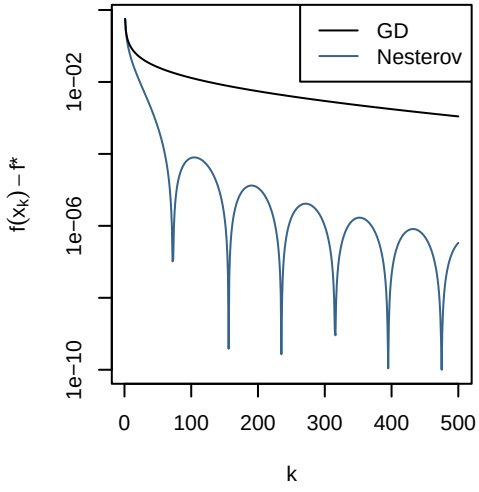
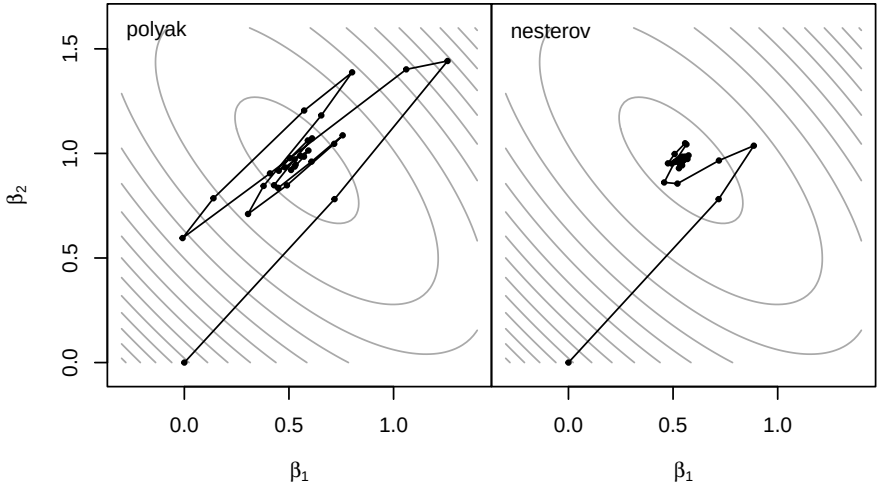


Figure 7: Suboptimality plot for a logistic regression problem with $n = 1000$, $p = 100$.


polyak
nesterov
β_2
β_1
β_1
0.0
0.5
1.0
1.5
0.0
0.5
1.0
0.0
0.5
1.0

What About SGD?

So far, we have mostly talked about standard GD, but we can use momentum (Polyak or Nesterov) for SGD as well.

For standard GD, Nesterov is the dominating method for achieving acceleration; for SGD, Polyak momentum is actually quite common.

Stochastic Gradient Descent
In term of convergence, all bets are now off.

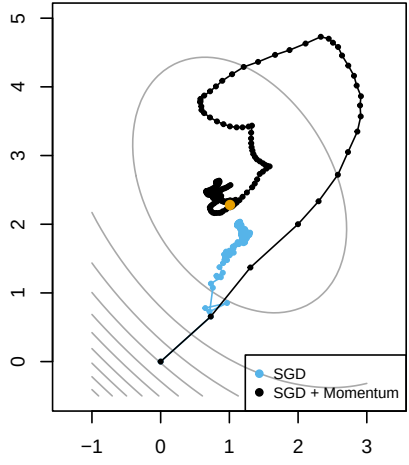
No optimal rates anymore, just heuristics.

But momentum helps in another way: it **reduces variance** of the iterates.

Variance Reduction

Momentum smooths out the updates, reducing variance of the iterates.

And also adds inertia, accelerating progress in low-variance directions.



Iterate Averaging (Polyak–Ruppert)

Keep a running average of SGD iterates to reduce variance:

$$\bar{x}_k = \frac{1}{k} \sum_{i=1}^{k} x_i.$$

Update incrementally:

$$\bar{x}_k = \bar{x}_{k-1} + \frac{1}{k}(x_k - \bar{x}_{k-1}).$$

Algorithm 4: SGD with Polyak–Ruppert averaging

Data: x_0, schedule t_k

$\bar{x}_0 \leftarrow x_0$;

for $k \leftarrow 1, 2, \ldots$ **do**

- Sample i_k;
- $g_k \leftarrow \nabla f_{i_k}(x_{k-1})$;
- $x_k \leftarrow x_{k-1} - t_k g_k$;
- $\bar{x}_k \leftarrow \bar{x}_{k-1} + \frac{1}{k}(x_k - \bar{x}_{k-1})$;

Improves asymptotic variance at negligible cost.

Works with momentum and batches.

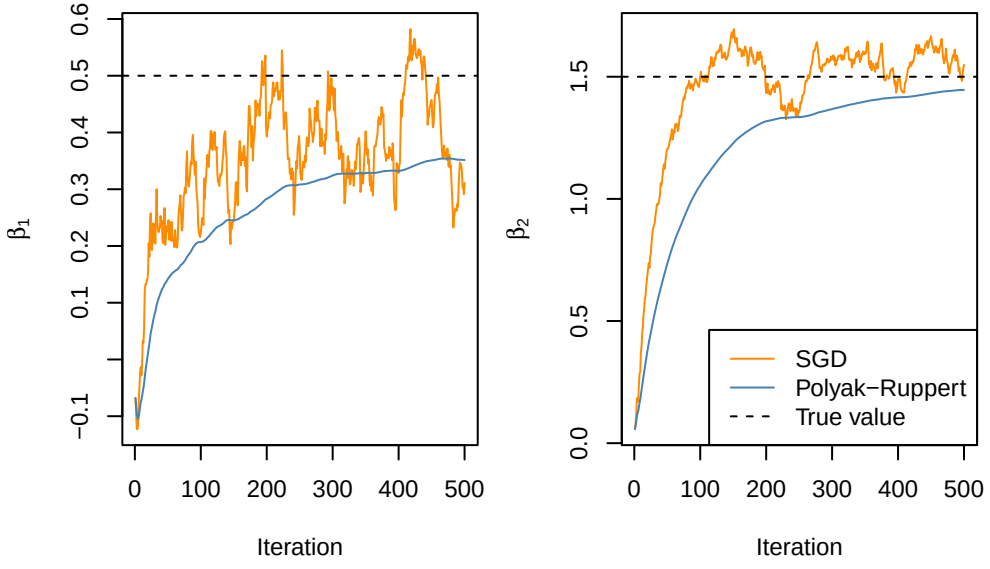


Figure 9: Comparison of SGD with and without Polyak-Ruppert averaging for a logistic regression problem.

General Idea

Some directions may be important, but feature information is **sparse**.

AdaGrad

Store matrix of gradient history,

$$G_k = \sum_{i=1}^{k} \nabla f(x_i) \nabla f(x_i)^\top,$$

and update by multiplying gradient with $G_k^{-1/2}$.

Algorithm 5: AdaGrad

Data: $t > 0$, $G = 0$

for $k \leftarrow 1, 2, \ldots$ **do**

$G_k \leftarrow G_{k-1} + \nabla f(x_{k-1}) \nabla f(x_{k-1})^\top$;

$x_k \leftarrow x_{k-1} - t G_k^{-1/2} \nabla f(x_{k-1})$;

Effects

- Larger learning rates for sparse features
- Step-sizes adapt to curvature.

AdaGrad In Practice

Simplified Version

Computing $\nabla f \nabla f^\top$ is $O(p^2)$: expensive!

Replace $G_k^{-1/2}$ with $\operatorname{diag}(G_k)^{-1/2}$

Avoid Singularities

Add a small ϵ to diagonal.

Algorithm 6: Simplified AdaGrad

Data: $t > 0$, $\epsilon > 0$

for $k \leftarrow 1, 2, \ldots$ **do**

$G_k \leftarrow G_{k-1} + \operatorname{diag}\left(\nabla f(x_{k-1})^2\right)$;

$x_k \leftarrow x_{k-1} - t \operatorname{diag}\left(\epsilon I_p + G_k\right)^{-1/2} \nabla f(x_{k-1})$;

RMSProp

Acronym for **R**oot **M**ean **S**quare **Prop**agation (Hinton 2018).

Idea
Divide learning rate by running average of magnitude of recent gradients:

$$v(x,k) = \xi v(x,k-1) + (1-\xi)\nabla f(x_k)^2$$

where ξ is the **forgetting factor**.

Similar to AdaGrad, but uses **forgetting** to gradually decrease influence of old data.

Algorithm 7: RMSProp. (Note that $\odot$ is element-wise product.)

Data: $t > 0$, $\xi > 0$
for $k \leftarrow 1, 2, \ldots$, $\xi \in [0, 1)$ **do**
    $v_k = \xi v_{k-1} + (1-\xi)\nabla f(x_{k-1})$;
    $x_k \leftarrow x_{k-1} - \frac{t}{\sqrt{v_k}} \odot \nabla f(x_{k-1})$;

Adam

Acronym for **Ada**ptive **m**oment estimation (Kingma and Ba 2015)

Basically RMSProp + *momentum* (for both gradients and second moments thereof)

Popular and still in much use today.

Not covered here, but see the course literature (Hansen 2024) for details.

Gradient Clipping

Idea
If gradient is too large, scale it down to avoid too large steps.

Replace gradient $\nabla f(x_k)$ by

$$\tilde{\nabla} f(x_k) = \begin{cases} \nabla f(x_k) & \text{if } \|\nabla f(x_k)\|_2 \leq c, \\ c \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|_2} & \text{otherwise.} \end{cases}$$

How do you Pick c?
Some strategies:

1. **Rule of thumb**: $c \in [1, 10]$ often works well in practice
2. **Gradient norm monitoring**: Track $\|\nabla f(x_k)\|_2$ during training and set c based on observed distribution
3. **Percentile-based**: Set c to be the 95th or 99th percentile of observed gradient norms
4. **Problem-specific**: Use domain knowledge about expected gradient magnitudes

Practical considerations:

Early Stopping

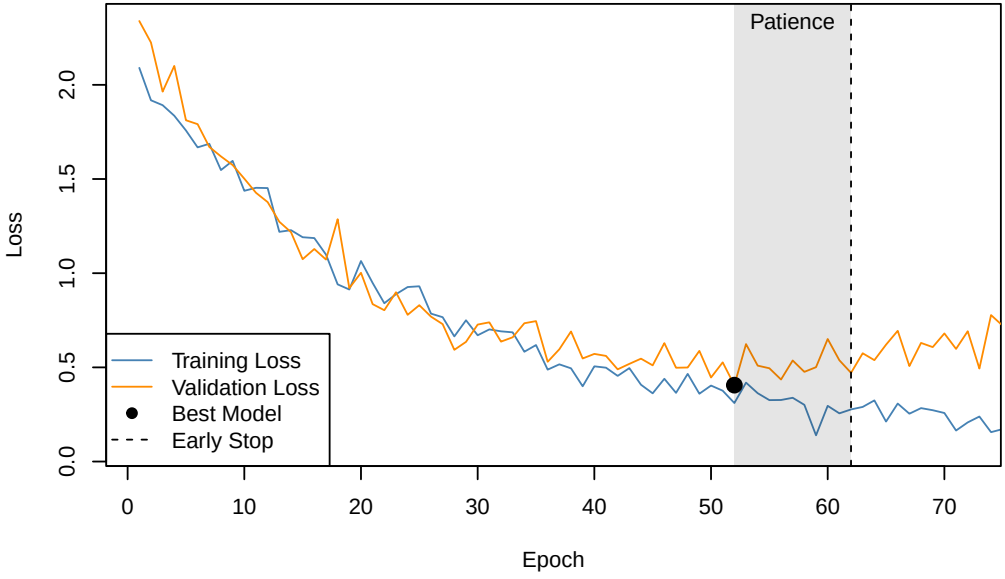
Idea
We are not really interested in **training error**.

Early Stopping
Stop training when validation metric stops improving.

Use a held-out validation set; never the training set, and avoid **overfitting**.

Algorithm 8: Early stopping with patience

Data: Patience P, min improvement $\delta \geq 0$
$x \leftarrow x_0$, $x_{\text{best}} \leftarrow x$, $b_{\text{best}} \leftarrow \infty$, $p \leftarrow 0$;
for $k = 1, 2, \ldots$ **do**
    Train for some epochs;
    Compute validation loss b_k;
    if $b_k \leq b_{\text{best}} - \delta$ **then**
        $b_{\text{best}} \leftarrow b_k$, $x_{\text{best}} \leftarrow x$, $p \leftarrow 0$;
    else
        $p \leftarrow p + 1$;
        if $p \geq P$ **then**
            break
return x_{best}


Patience
Loss
0.0
0.5
1.0
1.5
2.0
0
10
20
30
40
50
60
70
Epoch
Training Loss
Validation Loss
Best Model
Early Stop

Example: Nonlinear Least Squares

Let's assume we're trying to solve a least-squares type of problem:

$$f(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - g(\theta; x_i)\right)^2$$

with $\theta = (\alpha, \beta)$ and

$$g(\theta; x) = \alpha \cos(\beta x).$$

Then

$$\nabla_\theta f(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - g(\theta; x_i)) \begin{bmatrix} -\cos(\beta x_i) \\ \alpha x_i \sin(\beta x_i) \end{bmatrix}.$$

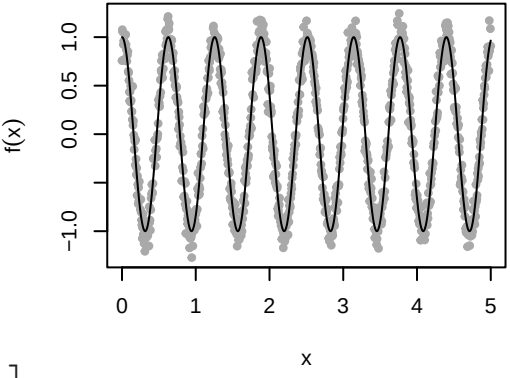


Figure 11: Simulation from problem

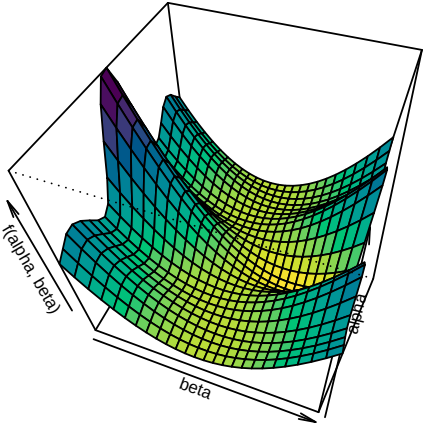


Figure 12: Perspective plot of function

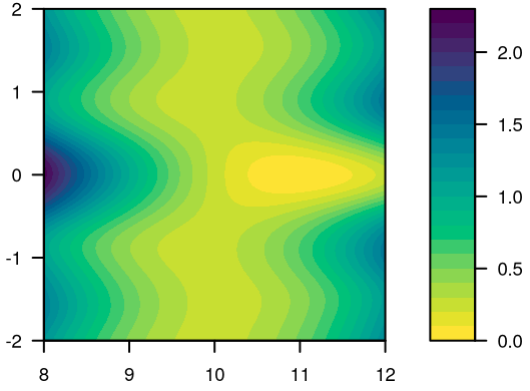


Figure 13: Contour plot of f

Variants

We will consider three variants:

- Batch gradient descent
- Batch gradient descent **with momentum**
- Adam

In each case, we'll use a batch of size $m = 50$.

We initialize at different starting values and see how well the algorithm converges.

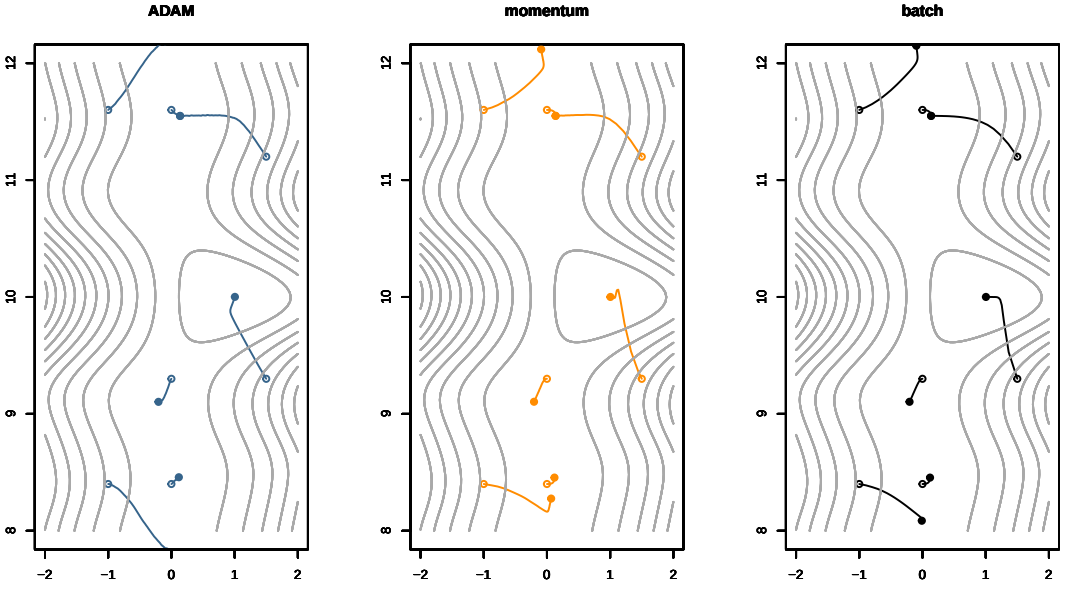


Figure 14: Convergence of different algorithms for different starting values.

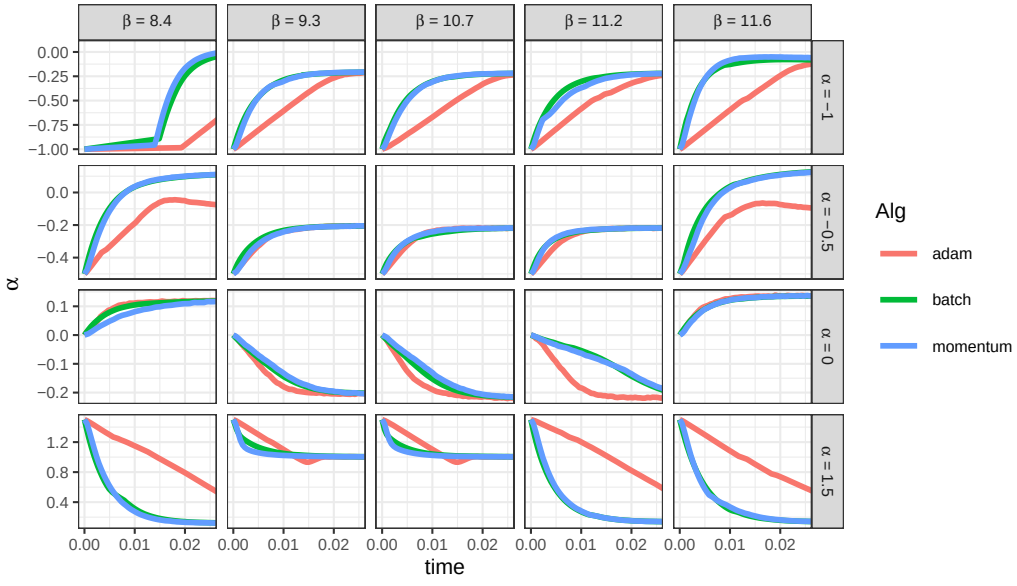


Figure 15: Updates of α parameter over time for the different algorithms over different starting values.

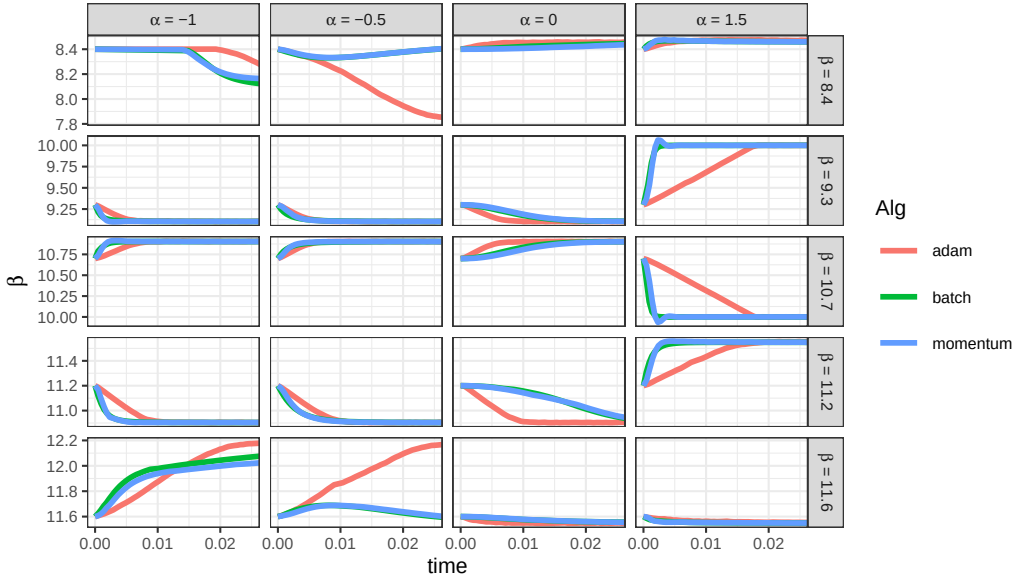


Figure 16: Updates of β parameter over time for the different algorithms over different starting values.

Very attractive for stochastic methods due to all the loop constructs and slicing.

However, Rcpp lacks linear algebra functions.

Approaches

Still use only Rcpp (but then you need to write your own linear algebra functions[1])

Use RcppEigen or RcppArmadillo.

[1] Not recommended!

Exercise: Rosenbrock's Banana

Steps

1. Minimize $f(x_1, x_2) = (a - x_1)^2 + b(x_2 - x_1^2)^2$ with $a = 1$ and $b = 100$ using gradient descent. Optimum is (a, a^2).
2. Add Polyak momentum.
3. Experiment with different step sizes and momentum parameters.

Algorithm 9: GD with Polyak Momentum

Data: $t > 0$, $\mu \in [0, 1)$

for $k \leftarrow 1, 2, \ldots$ **do**

$x_k \leftarrow x_{k-1} - t\nabla f(x_{k-1}) + \mu(x_{k-1} - x_{k-2})$;

Plot Contours

You can use the following code to plot the contours of the function (after defining `f`):

```
x1 <- seq(-2, 2, by = 0.05)
x2 <- seq(-1, 3, by = 0.05)
z <- outer(x1, x2, f)
contour(x1, x2, z, nlevels = 20)
```

Summary

We introduced several new concepts:

- Polyak momentum,
- Nesterov acceleration (momentum),
- Polyak-Ruppert averaging,
- adaptive gradients (AdaGrad, RMSProp),
- gradient clipping, and
- early stopping.

We practically implemented versions of gradient descent and stochastic gradient descent with momentum.

Additional Resources
(Goh 2017) is an interactive article on momentum in gradient descent with lots of pedagogical illustrations.

Next Time

R Packages
We build an R package.

Summary
We summarize the course.

Exam Advice
We talk about the upcoming oral examinations.

References i

Goh, Gabriel. 2017. "Why Momentum Really Works." *Distill*. https://doi.org/10.23915/distill.00006.

Hansen, Niels Richard. 2024. "Computational Statistics with R." Online Book. Computational Statistics with R. August 9, 2024. https://cswr.nrhstat.org/.

Hinton, Geoffrey. 2018. "Lecture 6." Oral Presentation presented at the Coursera course: Neural networks for machine learning, Virtual. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture/%5Fslides/%5Flec6.pdf.

Kingma, Diederik P., and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization." In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, edited by Yoshua Bengio and Yann LeCun. http://arxiv.org/abs/1412.6980.

Lessard, Laurent, Benjamin Recht, and Andrew Packard. 2016. "Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints." *SIAM Journal on Optimization* 26 (1): 57–95. https://doi.org/10.1137/15M1009597.

References ii

Nesterov, Yuri. 1983. "A Method of Solving a Convex Programming Problem with Convergence Rate O(1/k^2)." *Doklady Akademii Nauk SSSR* 269 (3): 543–47. https://cir.nii.ac.jp/crid/1570572699326076416.

Polyak, B. T. 1964. "Some Methods of Speeding up the Convergence of Iteration Methods." *USSR Computational Mathematics and Mathematical Physics* 4 (5): 1–17. https://doi.org/10.1016/0041-5553(64)90137-5.