Stochactic Gradient Descent
AdaGrad
ADAM

M. Fatih Amasyalı

Finite sum of functions

- In ML optimization, objective function has the form:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

n : training set size

$f_i(x)$: loss value for i.th training point according to parameter x

Some examples of finite sum

- Training set: i=1:n a_i: i.th input, b_i: i.th output
- x: parameters of a model to be trained
- Least squares:

$$\min_x \frac{1}{n}\sum_{i=1}^{n}(a_i^\top x - b_i)^2 = \min_x \frac{1}{n}\sum_{i=1}^{n}(f_i(x))^2$$

- Lasso:

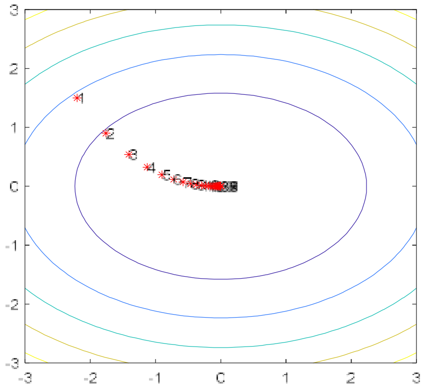
$$\min_x \frac{1}{n}\sum_{i=1}^{n}(a_i^\top x - b_i)^2 + \lambda\sum_{j=1}^{d}|x_j|$$

- ANN:

$$\min_x \frac{1}{n}\sum_{i=1}^{n} loss(ANN(x, a_i), b_i))$$

Gradient Descent

- $x_{t+1} = x_t - \eta_t \nabla f(x_t) = x_t - \eta_t \sum_{i=1}^{n} \nabla f_i(x_t)$
- $f(x_1, x_2) = x_1^2 + 2x_2^2$
- Learning rate (η_t) =0.1
- it converges at 40.th step
- Code: hessian_gradyan.m

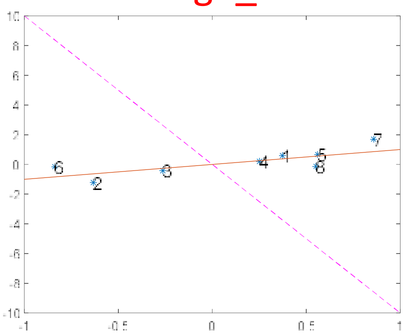


Large scale problems

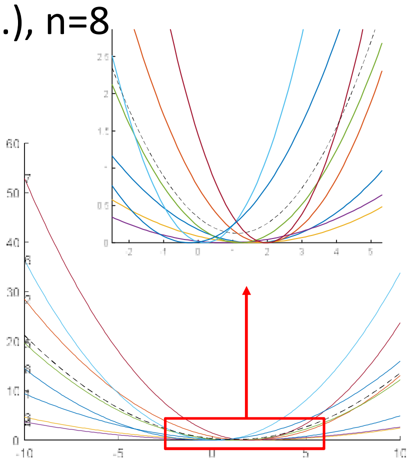
- Zillions of parametres (d)
- Zillions of data points (n)
- Each GD iteration requires n*d calculations

A simple model

- $\min_x \frac{1}{2n} \sum_{i=1}^{n} (a_i^\top x - b_i)^2 = \min_x \frac{1}{2n} \sum_{i=1}^{n} (f_i(x))^2$
- ai, bi, x : scalar (1 dim.), n=8
- Code: sgd_how.m

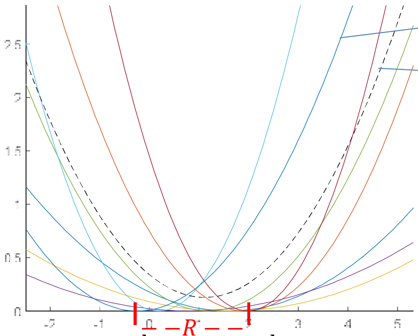


Training points, - true model b=1a,
-- current model b=-10a



Corresponding $f_i(x)$ s = $(a_i x - b_i)^2$

A simple model



$f_i(x) = \frac{1}{2}(a_i x - b_i)^2$

$f(x) = \frac{1}{2n}\sum_{i=1}^{n}(a_i x - b_i)^2$

$$x^* = \min_x \frac{1}{2n}\sum_{i=1}^{n}(a_i x - b_i)^2$$

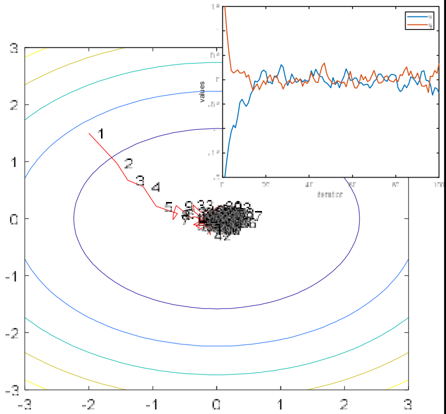
Region R=[$\min \frac{b_i}{a_i}$, $max \frac{b_i}{a_i}$]

$x^* \in R$

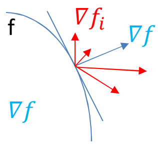
Obverse that:
For all i, $\nabla f_i(x)$ and $\nabla f(x)$ have the same sign outside R
So, if we use $\nabla f_i(x)$, we guarantee the improvement outside R
In R, we get chaos ☺
Lets see

Stochastic Gradient Descent

- $x_{t+1} = x_t - \eta_t \nabla f_i(x_t)$
- $f(x_1, x_2) = x_1^2 + 2x_2^2$
- Learning rate (η_t) =0.1
- it does not converge
- Code: sgd.m



Region R : an area (in 2 dim)
Outside R, GD and SGD are very similar
Inside R (near x*), SGD fluctuates
Usage of mean(x_t) or mean (x_{t-k} … x_t) instead of the last x_t make sense
Here, the last x_t=[-0.2692, 0.3260], mean(x_t)=[-0.1192, 0.0452]



$E[\nabla f_i] = \nabla f$

Each GD iteration requires d*n calculations
Each SGD iteration requires d calculations ☺

Stabilization of SGD

- Gradient clipping[*]: rescale gradients so that their norm is at most a predetermined value (c).

if $\|\nabla f_i(x_t)\| \geq c$ then $\nabla f_i(x_t) = c\,\frac{\nabla f_i(x_t)}{\|\nabla f_i(x_t)\|}$

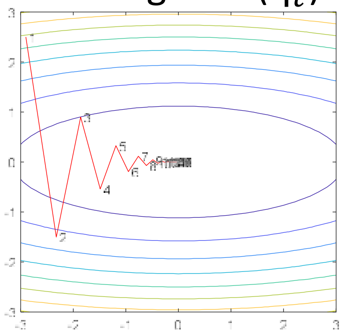
- Mini-batch: instead of one example, use k example. Reduce variance of $\nabla f_i(x_t)$

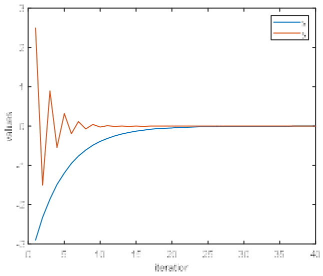
$$x_{t+1} = x_t - \frac{\eta_t}{|K_t|}\sum_{j \in K_t} \nabla f_j(x_t)$$

[*] https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48

Gradient Descent

- $f(x_1, x_2) = x_1^2 + 8x_2^2$
- Learning rate (η_t) =0.1





GD wastes time. 2.order information helps but requires d*d calculations
There is an another way, mimic Hessian (quasi-Newton methods)
Observe all previous gradients

AdaGrad[*]: A quasi-Newton method

- $G_t = \sum_{j=1}^{t} \nabla f(x_j)\,\nabla f(x_j)^{\mathsf{T}}$
- Gt (sum of outer product of all previous gradients) mimic Hessian
- Gt: d*d matrix
- 2 versions:
- $x_{t+1} = x_t - \eta_t G_t^{-1/2} \nabla f(x_t)$

Full matris inversion and square root requires d*d calculations

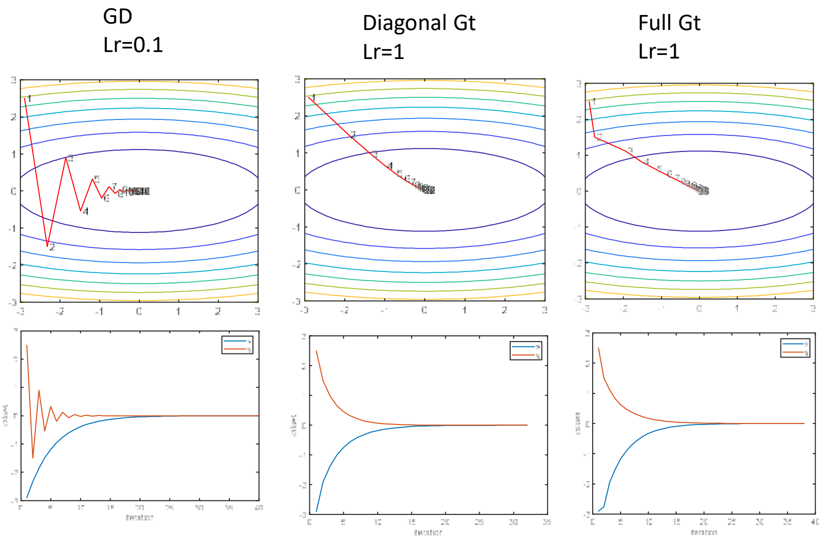
- $x_{t+1} = x_t - \eta_t diag(G_t)^{-1/2} \nabla f(x_t)$

Diagonal matris (vector in d dim) inversion and square root requires d calculations

[*] http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

code: hessian_gradyan.m

AdaGrad

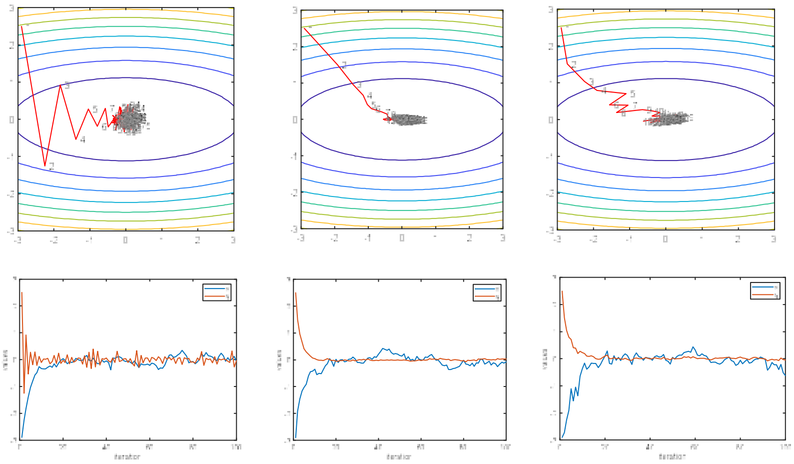


Stochastic AdaGrad

GD
Lr=0.1

Diagonal Gt
Lr=1

Full Gt
Lr=1



ADAM[*]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

[*] Kingma, D. P., & Ba, J. Adam: A method for stochastic optimization. ICLR 2015, *arXiv:1412.6980*.

Mehmet Fatih AMASYALI Optimization Techniques Lecture Notes YILDIZ TECHNICAL UNIVERSITY

code: moving_avg.m

ADAM

$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ Adagrad version of v_t: $v_t = v_{t-1} + g_t^2$

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ Adagrad version of m_t : $m_t = g_t$

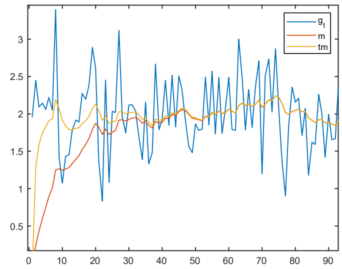
$\hat{m}_t = m_t/(1-\beta_1^t)$

$\hat{v}_t = v_t/(1-\beta_2^t)$ Corrections

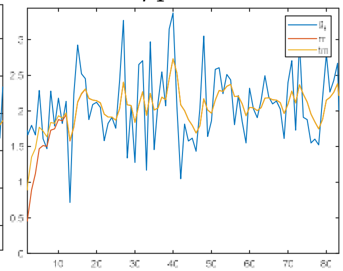
$\theta_t = \theta_{t-1} - \alpha \hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon)$ Update

$g_t \sim N(2,1)$

$\beta_1 = 0.9$

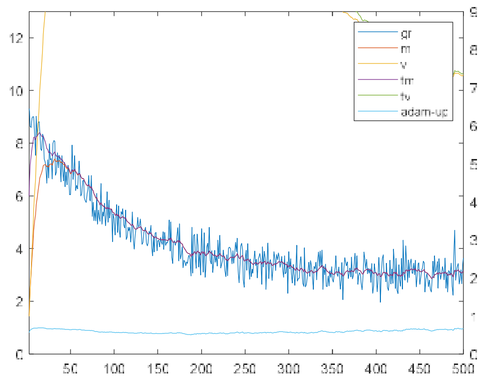


$\beta_1 = 0.7$



Adam update

Robust to magnitute of gradients, noisy gradients (stochastic opt.), sparse gradients

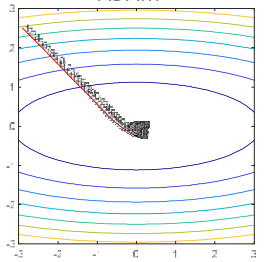


Half of gradients are 0

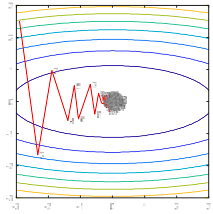
code: myADAM.m

ADAM in 2d

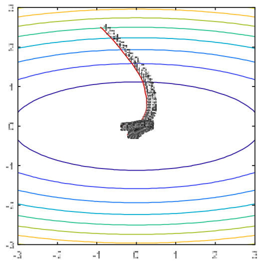
ADAM



SGD



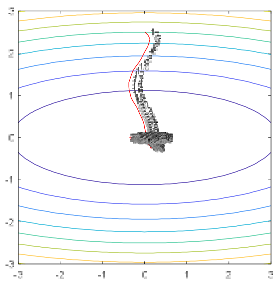
It directly goes to the optimal point.
Does it know the optimum point ☺



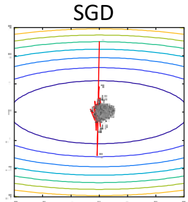
No ☺
Its update values are equal for all dimensions

Disadvantage of ADAM

ADAM



SGD



When if gradient is very close to 0 for a dimension,
it can make it bigger.

Other quasi Newton methods

- Broyden–Fletcher–Goldfarb–Shanno (BFGS)
- Davidon–Fletcher–Powell (DFP)

References

- SGD related sections:
  - Survit Sra https://www.youtube.com/watch?v=k3AiUhwHQ28
  - Gilbert Strang https://www.youtube.com/watch?v=AeRwohPuUHQ
- AdaGrad:
  - Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7). http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf