Course progress

- Previously
  - PCA: dimensionality reduction (simple unsupervised learning)
  - OLS, ridge regression,*conjugate gradient*
  - *Convex optimization*, linear programming, Lagrange multipliers, duality and minimax games
  - Sparse regression (LASSO); NMF, Sparse PCA,
  - *Gradient descent, dual ascent, dual decomposition, augmented Lagrangians, ADMM* (method to solve lasso, NMF)
  - Random sampling and randomized QR and SVD factorizations
  - Compressed sensing and matrix completion
  - DFT and FFT, shift invariant and circulant matrices/2D Fourier transform/filters
  - Graphs and their matrix representation, clustering, stochastic gradient descent
- Today
  - Adaptive methods: ADAGrad, ADAM

SGD

- For a model which outputs a label $F(x, a)$ given features a and parameters x, loss $\ell(y)$ and data $(a_1, b_1), \ldots, (a_n, b_n)$, let

$$\ell_i(x) = \ell(F(x, a_i) - b_i)$$

- The training, i.e., learning x given the data, can be done by minimizing

$$L(x) = \frac{1}{|B|} \sum_{i \in B} \ell_i(x)$$

- Use a GD algorithm

$$x_{k+1} = x_k - s_k \nabla L(x_k)$$

where a batch of size of $|B|$ is chosen uniformly at random (*minibatch GD*).

SGD

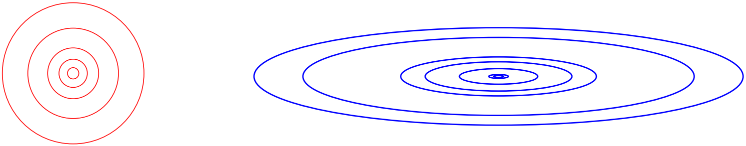
- When $B = 1$, we have *SGD*
- If the step size is constant

$$s_k = s = \text{constant}/\sqrt{T}$$

then

$$\min_{1 \le k \le T} \mathbb{E}\|\nabla L(x_k)\|^2 \le \frac{C}{\sqrt{T}}$$

- Assign the same learning rate to all features: is this a good idea?



ADAGRAD

- Downscale the step size by sqrt of the sum the squares of the gradients from previous steps

$$s_k = \frac{\alpha}{\sqrt{k}} \left[\frac{1}{k} \mathrm{diag}\Big(\sum_{i=1}^{k} \nabla L_i \odot \nabla L_i \Big) \right]^{-\frac{1}{2}}$$

  where $\odot$ is the entry-wise product.
- Shrinking features that have large partial derivatives seems like a good idea when L is convex.

RMSProp

- When L is not convex, make the gradient history more "local" by using exponential weights

$$s_k = \frac{\alpha}{\sqrt{k}}\left[(1-\beta)\mathrm{diag}\left(\sum_{i=1}^{k}\beta^{k-i}\nabla L_i \odot \nabla L_i\right)\right]^{-\frac{1}{2}}$$

- "Root mean square propagation" had been apparently proposed in a slide in Geoffrey Hinton's coursera course
- ADAM adds to the above the so-called "momentum"

Momentum

- For $0 < b \leq 1$, consider

$$f(x, y) = \frac{1}{2}(x^2 + by^2)$$

- When b is small, e.g., $1/10$, GD with the exact line search exhibits a zig-zag pattern

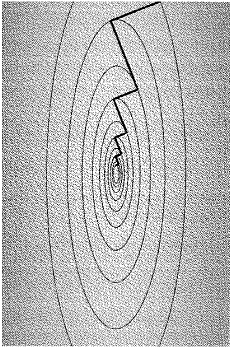


Figure: from p 349 in [1]).

Momentum

- Try to avoid the above pattern by adding a momentum, i.e. a multiple of the previous direction, to the gradient

$$x_{k+1} = x_k - s_k z_k$$
$$z_k = \nabla L(x_k) + \beta z_{k-1}$$

- So-called "heavy ball" methods (Polyak)

ADAM

- for

$$x_{k+1} = x_k - s_k D_k$$

- Adjusting the step size as in RMSProp

$$s_k = \frac{\alpha}{\sqrt{k}} \left[(1-\beta)\text{diag}\left(\sum_{i=1}^{k} \beta^{k-i} \nabla L_i \odot \nabla L_i \right) \right]^{-\frac{1}{2}}$$

and add momentum-like terms

$$D_k = (1-\delta) \sum_{i=1}^{k} \beta^{k-i} \nabla L(x_i)$$

- Equivalently, the above terms can be expressed recursively

$$\begin{aligned} s_k^2 &= \beta s_{k-1}^2 + (1-\beta)\text{diag}(\nabla L(x_k) \odot L(x_k)) \\ D_k &= \delta D_{k-1} + (1-\delta)\nabla L(x_k) \end{aligned}$$

- where $s^2 = s \odot s$.

Next steps

- Construction of deep neural networks (Sec. VII.1)
- Convolutional neural nets (Sec. VII.2)