Introduction to stochastic optimization for large-scale problems

Dr. Ir. Valentin Leplat

Skoltech
Center for Artificial Intelligence and Technology

Email: v.leplat@skoltech.ru
Website: https://sites.google.com/view/valentinleplat

ISP Seminar - Part 2

24 th Jan. 2023

Table of Contents

Context

- Here we tackle the following problems:

$$\min_{w} F(w) + g(w)$$

  where $F(w)$ can be the *expected risk* or the **empirical risk**, $g(w)$ is a regularization function for promoting structure on the solution $w^\star$ or avoid over-fitting (ex: Tikhonov regularization).
- **Strongly-convex assumption**: given $F(w)$ convex, using a strongly-convex function $g(w)$ makes the objective function $F(w) + g(w)$ strongly convex, hence ensuring good rates for SGD, and a unique minimizer.
- **Moreover**: for general functions, it improves the local smoothness, in particular we may get "locally Lipschitz" function, even if it is not globally, improving local convergence.

Assumptions

For simplicity, we assume:

- g has a simple proximal operator (in the sense that it can be computed easily).
- **Recall**: the proximal operator associated to a function g with parameter $\lambda > 0$, is a mapping $\mathbf{prox}_{g,\lambda} : \mathbb{R}^d \to \mathbb{R}^d$, calculated as follows:

$$\mathbf{prox}_{g,\lambda}(v) = \arg\min_{w \in \mathcal{W}} \{g(w) + \frac{1}{2\lambda}\|w - v\|^2\}$$

  $\to$ an unconstrained opt. prob. on its own !!
- Some proximal operators can be computed in closed form :) ! $\to$ **Example**: $g(w) = \|w\|_1$, see ▸ ADMM lecture notes (slides 39 -47) for an introduction and examples.

Proximal Stochastic Gradient Descent Algorithm

Initialization

Set $w_1 \in \mathcal{W}$.

Iteration ($k \geqslant 1$):

1. Generate a realization of the random variable ξ_k
2. Compute a stochastic vector $\nabla f(w_k; \xi_k)$
3. Choose a step size $\alpha_k > 0$
4. Set

$$v_{k+1} \leftarrow w_k - \alpha_k \nabla f(w_k; \xi_k) \tag{1}$$

5. Set

$$w_{k+1} \leftarrow \mathbf{prox}_{g,\alpha_k}(v_{k+1}) = \arg\min_{w \in \mathcal{W}} \{g(w) + \frac{1}{2\alpha_k} \|w - v_{k+1}\|^2\} \tag{2}$$

Proximal Stochastic Gradient Descent Algorithm

Interpretation:

$$w_{k+1} \leftarrow \mathbf{prox}_{g,\alpha_k}(w_k - \alpha_k \nabla f(w_k; \xi_k))$$

- from the definition of the proximal operator:

$$w_{k+1} := \arg\min_{w \in \mathcal{W}} \{g(w) + \frac{1}{2\alpha_k} \|w - w_k + \alpha_k \nabla f(w_k; \xi_k)\|^2\}$$

$$\arg\min_{w \in \mathcal{W}} \{g(w) + f(w_k; \xi_k) + \langle \nabla f(w_k; \xi_k), w - w_k \rangle + \frac{1}{2\alpha_k} \|w - w_k$$

- w_{k+1} minimizes $g(w)$ plus a simple quadratic local model of $f(w; \xi_k)$ around w_k.

Proximal Stochastic Gradient Descent Algorithm

Interpretation:

$$w_{k+1} \leftarrow \mathbf{prox}_{g,\alpha_k}(w_k - \alpha_k \nabla f(w_k; \xi_k))$$

- The term **proximal** refers to the presence of the third term in the minimization problem on the right-hand side,
- encourages the new iterate to be close to w_k,
- Notice that if the regularization (i.e., last) term were not present, we would exactly recover the stochastic gradient method

Prox. SGD for strongly cvx functions - decreasing α_k

Here: we give a brief convergence result for Proximal SGD, first we make several assumptions

Assumptions 3

- g is a proper convex, closed and proper function,
- $F(w)$ is c-strongly convex and has a L-Lipschitz gradient,
- there exists $M > 0$ such that $\mathcal{V}_{\xi_k}[\nabla f(w_k; \xi_k)] \leqslant M$.
- here exists $w^\star \in \arg\min F + g$,
- the sequence α_k is deterministic, satisfies $\alpha_k = \frac{a}{c(k+b)}$ for given $a > 1$ and $b > 0$ such that $\alpha_1 = \frac{a}{cb} \leqslant \frac{1}{2L}$

Prox. SGD for strongly cvx functions - decreasing α_k

Theorem 5 (Proof Part 3)

Under assumptions 3, the iterates of the proximal stochastic gradient algorithm satisfy the convergence rate (in terms of distance to the optimum):

$$\mathbb{E}[\|w_k - w^\star\|^2] \leqslant \frac{\frac{4a^2M}{(a-1)c^2}}{k+b} \sim O(\frac{1}{k}) \tag{3}$$

Table of Contents

Motivation

In **previous** sections:

- The theoretical arguments developed have led many in the machine learning community to view SGD as the ideal optimization approach for large-scale applications.
- **however**: that this is far from settled !
- **Indeed**: SGD suffers from, among other things, the adverse effect of noisy gradient estimates:
  1. prevents it from converging to the solution when fixed stepsizes are used,
  2. leads to a "slow", sublinear rate of convergence when a diminishing stepsize sequence $\{\alpha_k\}_{k+1}^{\infty}$ is employed.

Motivation

In **this** section:

- **Solution**: develop methods endowed with noise reduction capabilities (reduce the errors in the gradient estimates and/or iterate sequence).
- **Three main classes**:
  1. dynamic sampling methods, achieve noise reduction by gradually increasing the minibatch size used in the gradient computation $\rightarrow$ employing increasingly more accurate gradient estimates
  2. Gradient aggregation methods: improve the quality of the search directions by storing gradient estimates corresponding to samples employed in previous iteration.
  3. iterate averaging methods: accomplish noise reduction not by averaging gradient estimates, but by maintaining an average of iterates computed during the optimization process.

high-level summary for convergence rates

- dynamic sampling methods: *linear* rate of convergence to the optimal value using a *fixed* stepsize.
- Gradient aggregation methods: *linear* rate of convergence to the optimal value using a *fixed* stepsize.
- iterate averaging methods: uses a more agressive stepsize sequence, of order $O(\frac{1}{\sqrt{k}})$ instead of $O(\frac{1}{k})$
  $\longrightarrow$ really appealing because:....it is this sequence of averaged iterates that converges to the solution !

Organization of this section

1. formal motivation of noise reduction concept: a fundamental result that stipulates a rate of decrease in noise that allows an SGD-type method to converge at a linear rate.
2. three gradient aggregation methods — SVRG, SAGA, and SAG.
3. discussion of the practical and theoretical properties of iterate averaging methods.

Reducing Noise at a Geometric Rate

Let us recall Lemma 1:

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \leqslant -\alpha_k \langle \nabla F(w_k), \mathbb{E}_{\xi_k}[\nabla f(w_k;\xi_k)] \rangle + \frac{\alpha_k^2 L}{2} \mathbb{E}_{\xi_k}[\|\nabla f(w_k;\xi_k)\|^2] \tag{4}$$

Intuitively:

1. if $-\nabla f(w_k;\xi_k)$ is a descent direction in expectation (first term of RHS is negative),
2. and if we "manage" to decrease $\mathbb{E}_{\xi_k}[\|\nabla f(w_k;\xi_k)\|^2$ "fast enough",

then: the effect of having noisy directions will not impede a fast rate of convergence !

Reducing Noise at a Geometric Rate

Formally: Consider SGD with fixed step sizes for strongly convex functions; if $\mathcal{V}_{\xi_k}[\nabla f(w_k; \xi_k)]$ decreases *geometrically*, then sequence of expected optimality gaps vanishes at *linear* rate.

Theorem 6 (Proof on the board)

Suppose that assumptions 1, 2 and 3, but with the existence of constants $M > 0$ and $\zeta \in (0, 1)$ such that for all $k \in \mathcal{N}$:

$$\mathcal{V}_{\xi_k}[\nabla f(w_k; \xi_k)] \leqslant M\zeta^{k-1} \tag{5}$$

Moreover, suppose SGD is ran with fixed step size $0 < \alpha \leqslant \min\{\frac{\mu}{L\mu_G^2}, \frac{1}{c\mu}\}$. Then the expected optimality gap satisfies:

$$\mathbb{E}[F(w_k) - F^\star] \leqslant \omega\rho^{k-1} \tag{6}$$

with $\omega := \max\{\frac{\alpha LM}{c\mu}, F(w_1) - F^\star\}$, and $\rho := \max\{1 - \frac{\alpha c\mu}{2}, \zeta\} < 1$.

Gradient Aggregation

- **Main idea**: achieve a lower variance by *reusing* and or/ *revising* previously computed information.
- **Key**: say that the current iterate has not been displaced too far from previous iterates,
  $\longrightarrow$ stochastic gradient information from previous iterates may still be useful !
- **Moreover**: say one maintains indexed gradient estimates in storage, then one can revise specific estimates as new information is collected.
- **Attention**: we focus on the min. of the empirical risk.

Stochastic variance-reduced gradient (SVRG)

Main principle

- operates in cycles.
- At the beginning of each cycle: compute a batch gradient: $\nabla F(w_k) = \frac{1}{n} \sum_i^n \nabla f_i(w_k)$
- Initialize: $\tilde{w}_1 \leftarrow w_k$
- Perform a set of m inner iterations indexed by j as follows:

$$\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \left(\nabla f_{i_j}(\tilde{w}_j) - \left(\nabla f_{i_j}(w_k) - \nabla F(w_k) \right) \right)$$

with $i_j \in \{1, ..., n\}$ chosen random.

SVRG

Initialization: Set $w_1 \in \mathcal{W}$, stepsize $\alpha > 0$ and positive integer m.

Iteration ($k \geqslant 1$):

1. Compute the batch gradient $\nabla F(w_k)$
2. Initialize $\tilde{w}_1 \leftarrow w_k$
3. **for** $j = 1 : m$

$$\text{Choose } i_j \sim \mathcal{U}(1, ..., n)$$
$$\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \left(\nabla f_{i_j}(\tilde{w}_j) - \left(\nabla f_{i_j}(w_k) - \nabla F(w_k) \right) \right)$$

   end for

4. Option (a): $w_{k+1} := \tilde{w}_{m+1}$
5. Option (b): $w_{k+1} := \frac{1}{m} \sum_{j=1}^{m} \tilde{w}_{j+1}$
6. Option (c): Choose $j \sim \mathcal{U}(1, ..., m)$, and $w_{k+1} := \tilde{w}_{j+1}$

SVRG

Insights on step 3, for simplicity pose
$\tilde{g}_j := \left(\nabla f_{i_j}(\tilde{w}_j) - \left(\nabla f_{i_j}(w_k) - \nabla F(w_k)\right)\right)$

- $\mathbb{E}_{i_j \in \{1,\dots,n\}}[\nabla f_{i_j}(w_k)] = \nabla F(w_k)$,
- $\nabla f_{i_j}(w_k) - \nabla F(w_k)$ seen as the bias in the gradient estimate $\nabla f_{i_j}(w_k)$
- in every iteration: SVRG draws a stochastic gradient $\nabla f_{i_j}(\tilde{w}_j)$ evaluated at current inner iterate $\tilde{w}_j$ and correct it based on perceived bias
- *Overall*: the stochastic vector $\tilde{g}_j$ represents an unbiased estimator of $\nabla F(\tilde{w}_j)$
- *but* with a variance that one can expect to be smaller than if one were simply to chose $\tilde{g}_j := \nabla f_{i_j}$ (as for SGD)
  $\rightarrow$ "Variance reduced method"

SVRG

Comments on "Options"

- For both Options (b) and (c) and for min. of strongly convex functions, SVRG achieves a linear rate of convergence.
- **Formally**: if we choose m (number of inner loops) and α such that:

$$\rho := \frac{1}{1 - 2\alpha L}\left(\frac{1}{mc\alpha} + 2L\alpha\right) < 1 \tag{7}$$

then we obtain:

$$\mathbb{E}[F(w_{k+1}) - F^\star] \leqslant \rho\mathbb{E}[F(w_k) - F^\star] \tag{8}$$

- This result applies for the outer iterates $\{w_k\}$ where step from w_k to w_{k+1} requires $2m + n$ evaluations of the gradients.
- **Therefore**: one iteration of SVRG is much more expensive than one of SGD, and in fact is comparable to a full gradient iteration.

SAGA

Main principle

- iteration that is closer in form to SGD: does not operate in cycles, nor does it compute batch gradients (except possibly at the initial point).
- **At each it.**: computes a stochastic vector g_k as the average of stochastic gradients evaluated at previous iterates.
- **Specifically**: at it. k, method will have stored $\nabla f_i(w_{[i]})$ for all $i \in \{1, ..., n\}$, where $w_{[i]}$ represents the latest iterate at which ∇f_i was evaluated.
- **Then**: choose $j \sim \mathcal{U}(1, ..., n)$, the stochastic vector is set to:

$$g_k \leftarrow \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_{[i]}) \tag{9}$$

SAGA

Initialization: Set $w_1 \in \mathcal{W}$, stepsize $\alpha > 0$.

Storage:

for $i = 1 : n$

1. Compute $\nabla f_i(w_1)$
2. Store $\nabla f_i(w_{[i]}) \leftarrow \nabla f_i(w_1)$

end for

Iteration ($k \geqslant 1$):

1. Choose $j \sim \mathcal{U}(1, ..., n)$
2. Compute $\nabla f_j(w_k)$
3. Set $g_k \leftarrow \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_{[i]})$
4. Store $\nabla f_j(w_{[j]}) \leftarrow \nabla f_j(w_k)$
5. Set $w_{k+1} \leftarrow w_k - \alpha g_k$

SAGA

Insights:

- Taking the expectation of g_k with respect to all choices of $j \in \{1, ..., n\}$: $\mathbb{E}[g_k] = \nabla F(w_k)$
- **Hence**: method employs unbiased gradient estimates, but with variances less than the stochastic gradients employed by SGD
- Putting aside the "storage" phase: the per-iteration cost of SAGE is the same as in a basic SGD method.
- **However**: SAGA enjoys linear rate of convergence for min. of strongly convex $F(w)$ (=*empirical risk*).
- **Result**: with $\alpha = \frac{1}{2(cn+L)}$:

$$\mathbb{E}[\|w_k - w^\star\|^2] \leqslant (1 - \frac{1}{2(cn+L)})^k (\|w_1 - w^\star\|^2 + C) \qquad (10)$$

for some constant C (not detailed here).

SAGA

Insights:

- **Tips**: previous result requires knowledge of the strong convexity constant c and L. If c is not known, then the stepsize can instead be chosen to be $\alpha = \frac{1}{3L}$ and a similar convergence result can be established.
- **important drawback**: need to store n stochastic gradient vectors, which would be prohibitive in many large-scale applications.
- **History**:SAGA has its origins in the stochastic average gradient (SAG) algorithm, the main difference:

$$g_k \leftarrow \frac{1}{n}\left(\nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \sum_{i=1}^{n} \nabla f_i(w_{[i]})\right)$$

Gradient Aggregation - commentary

- Even if linear rates have been proved (for strongly convex case), Gradient Aggregation methods should not be considered as superior to SGD !
- One can prove that the computing time:
  1. for SGD: $T(n, \epsilon) \sim \frac{(\frac{L}{c})^2}{\epsilon}$,
  2. for SVRG, SAGA and SAG: $T(n, \epsilon) \sim (n + \frac{L}{c}) \log(1/\epsilon)$
     $\rightarrow$ increase with n!
- **Hence**: for large n, gradient aggregation methods are comparable to batch methods, and cannot beat SGD.
- Moreover: for $L/c = 1$, SGD is optimal.
- but for $L/c \gg n$, gradient aggregation methods may be superior.
- **Anyway**: interesting methods due to clever use of past info.

Iterate Averaging Methods

- **Starting point**: SGD generates noisy iterate sequences that tend to oscillate around minimizers.
- **Natural idea**: compute a corresponding sequence of iterate averages (automatically possess less noisy behavior).
- **Algorithmic principle**: employ the iteration

$$\begin{aligned} w_{k+1} &\leftarrow w_k - \alpha_k \nabla f(w_k; \xi_k) \\ \tilde{w}_{k+1} &\leftarrow \frac{1}{k+1} \sum_{j=1}^{k+1} w_j \end{aligned} \tag{11}$$

- **Hope**: the auxiliary sequence $\{\tilde{w}_k\}_{k=1}^{\infty}$ possesses better convergence properties than SGD iterates.
- **But..** with diminishing step size sequence with rate $O(1/k)$, not the case !!

Iterate Averaging Methods

- **Fundamental advancement** - the work of (Polyak, 1991): rate for diminishing step size should be $O(\frac{1}{k^a})$ with $a \in (\frac{1}{2}, 1)$.
- See works of (Juditsky el al., 1992), (Ruppert,1988) and (Nemirovski et al., 1978) for more details.
- **Results**:

$$\begin{aligned} \mathbb{E}[\|w_k - w^\star\|^2] &\sim O(\frac{1}{k^a}) \\ \mathbb{E}[\|\tilde{w}_k - w^\star\|^2] &\sim O(\frac{1}{k}) \end{aligned} \tag{12}$$

- **Fruitful** consequences: this idea of sequence of averaged iterates has been fundamental in many central and even recent works.
  Main goal: allow longer steps while maintaining desired rates of convergence.

So many things we skipped :)

More stochastic methods:

- Stochastic quasi-Newton methods: estimate the inverse of the Hessian B_k and use the step:

$$w_{k+1} \leftarrow w_k - \alpha_k B_k \nabla f(w_k; \xi_k) \tag{13}$$

- Adaptive step-sizes: step-size sequence adapted to the problem at stake, most famous methods are Adagrad and Adam.
- Coordinate descent methods: update a subset of variables at a time.
- Accelerated SGD; see, e.g., Ghadimi and Lan (2016), Accelerated gradient methods for nonconvex nonlinear and stochastic programming. Mathematical Programming, 156(1-2), 59-99.

Table of Contents

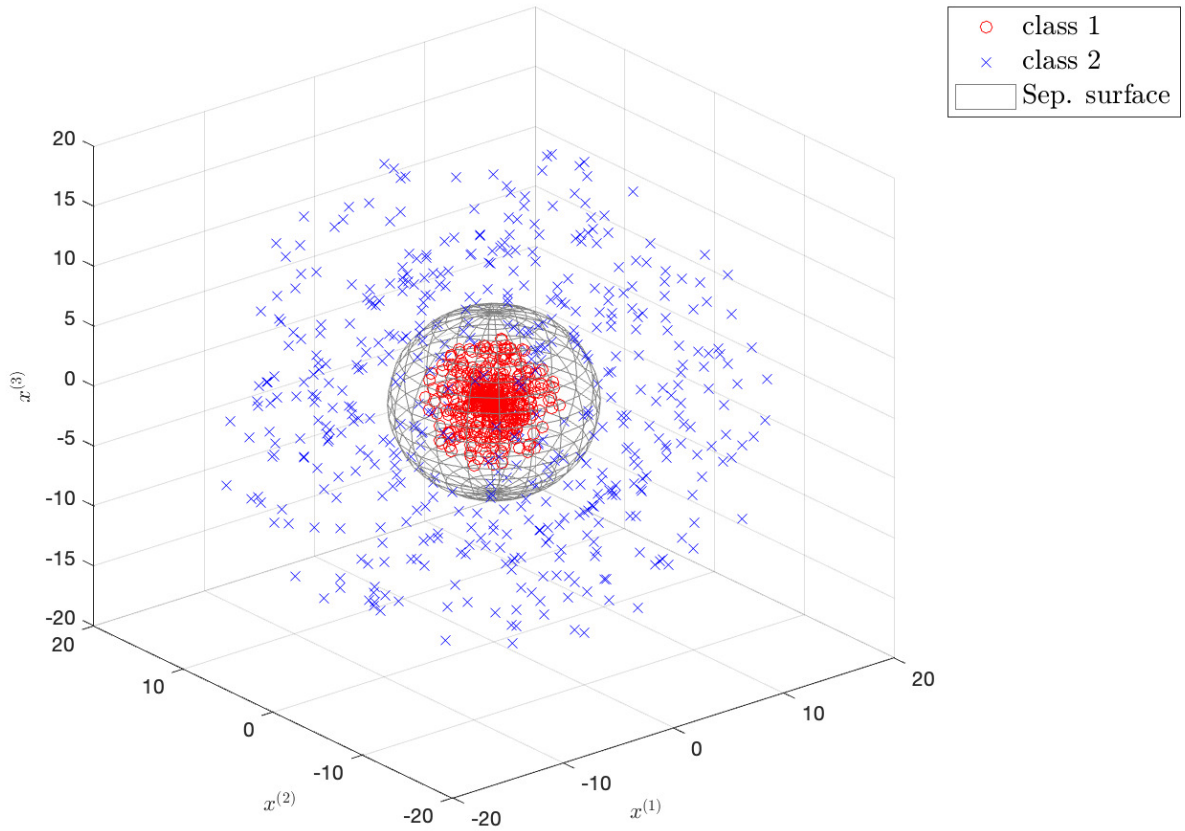
Numerical tests

Goals:

1. implement and benchmark some methods presented in this course,
2. on a small and hand-able NN: build the so-called "back-propagation" along with the selected stochastic method

Test setup

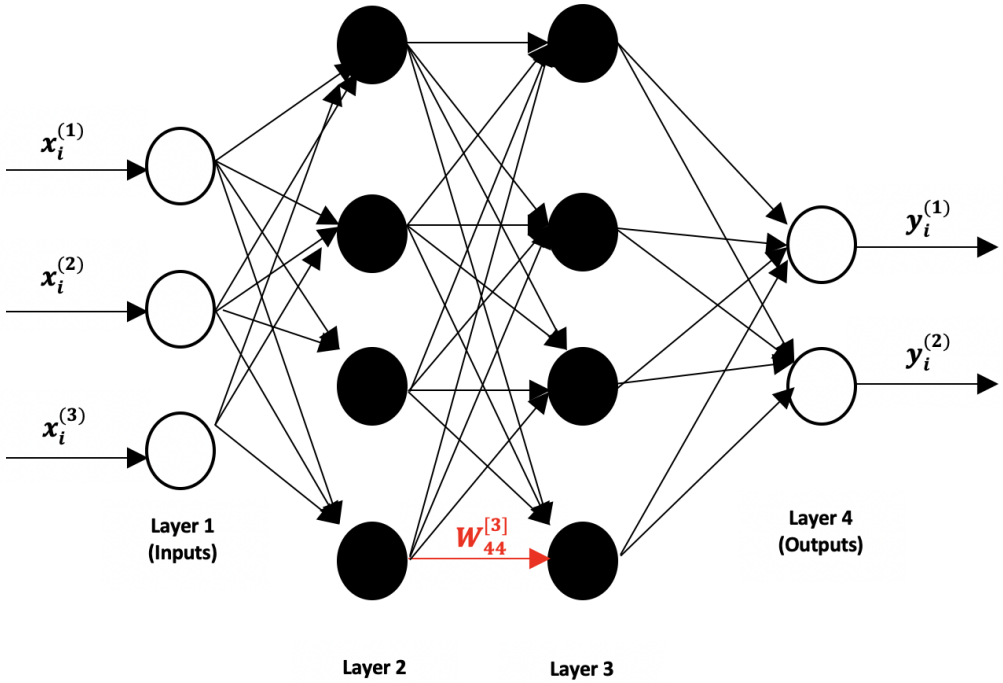
Context: consider the set of points: labeled data—some points are in class 1, indicated by red circles, and the rest are in class 2, indicated by blue crosses.



Test setup

Our job: construct a prediction function $h(x_i; w)$ that takes any point in $x_i \in \mathbb{R}^3$ and returns either a circle or a cross.

Form of h: before *training* h to do this task, we need to *fix* its form: we consider the network presented in the figure below:



Forward Propagation

If new for you: here-under a very simple and step by step explanation:

- **Layer 1**: represented by three circles, since our input vector x_i have three components.
- **Layer 2**: four solid circles, indicating that fours neurons are being employed.
- **First connections**: arrows from layer 1 to layer 2 indicate that all the three components of the input data are made available to the four neurons in layer 2.
- **Weight and bias**: input data has the form $x_i \in \mathbb{R}^3$: weights and biases for layer 2 may be represented by a matrix $W^{[2]} \in \mathbb{R}^{4\times 3}$ and a vector $b^{[2]} \in \mathbb{R}^4$ resp.
- **Layer output**: output from layer 2 has the form:

$$\Phi(W^{[2]}x_i + b^{[2]}) \in \mathbb{R}^4 \tag{14}$$

with $\Phi(.)$ the so-called activation function of the layer at hand, here we choose the sigmoid function $\Phi(x) = \frac{1}{1+e^{-x}}$ applied component-wise. $(\Phi'(x) = \Phi(x)(1-\Phi(x)))$

Forward Propagation - more

- **Layer 3**: has four neurons, each receiving input in $\mathbb{R}^4$, so the weights and biases for layer 3 may be represented by a matrix $W^{[3]} \in \mathbb{R}^{4\times 4}$ and a vector $b^{[3]} \in \mathbb{R}^4$ resp. The ouput has the form:

$$\Phi(W^{[3]}\Phi(W^{[2]}x_i + b^{[2]}) + b^{[3]}) \in \mathbb{R}^4 \tag{15}$$

- **Layer 4 - Output**: the output from layer 4, and hence from the overall network, has the form:

$$h(x_i; w) = \Phi(W^{[4]}\Phi(W^{[3]}\Phi(W^{[2]}x_i + b^{[2]}) + b^{[3]}) + b^{[4]}) \in \mathbb{R}^2 \tag{16}$$

  with $w = \{(W^{[l]}, b^{[l]})\}_{l=2}^{4}$ the parameters, the variables.
- Denoting $\{x_i\}_{i=1}^{n}$ our input data points, we use y_i for the target output; that is: $y_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ if x_i is in class 1, and $y_i = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ if x_i is in class 2.

Forward Propagation - loss function

- In this numerical test, we have access to n realizations $\xi_{[i]} = (x_i, y_i)$ with $i = 1, ..., n$.
- **Now**: it is time to *train* the network, that is finding values for parameters/variables w such that $h : \mathbb{R}^3 \to \mathbb{R}^2$ such that $h(x_i; w) = y_i$ for most i $(1 \leqslant i \leqslant n)$.
- **Loss function**: with the notations of Part 1, we consider the loss function $l(h(x_i; w), y_i) = \frac{1}{2}\|h(x_i; w) - y_i\|_2^2$
- **Optimization Problem**: we want to min. the empirical risk function, that is we want to solve :

$$\min_w F(w) := \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\|h(x_i; w) - y_i\|_2^2 \tag{17}$$

Forward Propagation - generalization

General notation: for a general network

- The network has L layers, with layers 1 and L being the input and output layers,
- Suppose that layer l, for $l = 1, 2, 3, ..., L$ contains n_l neurons; n_1 is the dimension of the input data (we used d_x in Introduction of Part 1).
- **Overall**: the network maps from $h : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^{n_L}$.
- **Weight and biases**: use $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ (weight matrix at layer l) and $b^{[l]} \in \mathbb{R}^{n_l}$ the vector of biases for layer l (neuron j at layer l uses the bias $b_j^{[l]}$).
- **Activation**: given an input $x_i \in \mathbb{R}^{n_1}$, we summarize the action of the network by letting $a_j^{[l]}$ denote the output, or *activation*, from neuron j at layer l:

$$\begin{aligned} a^{[1]} &= x_i \in \mathbb{R}^{n_1} \\ a^{[l]} &= \Phi(W^{[l]} a^{[l-1]} + b^{[l]}) \in \mathbb{R}^{n_l} \quad \text{for } l = 2, 3, ..., L \end{aligned} \tag{18}$$

Back Propagation

- To apply our stochastic methods to min. Problem (17) and then train the network, we need compute the partial derivatives (the gradients for each parameter) of the objective function w.r.t each $W_{jk}^{[l]}$ and $b_j^{[l]}$.
- For a selected and *fixed* input point x_i, denote $a^{[L]} := h(x_i; w)$.
- To ease the expressions of the partial derivatives, we introduce:

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l} \quad \text{for } l = 2, 3, ..., L \tag{19}$$

- **Fundamental induced relations**:
  1. $a^{[l]} = \Phi(z^{[l]})$,
  2. let $\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}$ for $1 \leqslant j \leqslant n_l$ and $2 \leqslant l \leqslant L$. (measures the sensitivity of the loss function to the weighted input for neuron j at layer l), with $C = f_i(w; \xi_{[i]}) = \frac{1}{2}\|y_i - a^{[L]}\|_2^2$

Back Propagation - Partial derivatives

Useful operator: the Hadamard product of two vectors $u, v \in \mathbb{R}^n$, that is $(u \odot v)^{(i)} = u_i v_i$.
We obtain the following results that are a consequence of the *chain rule*:

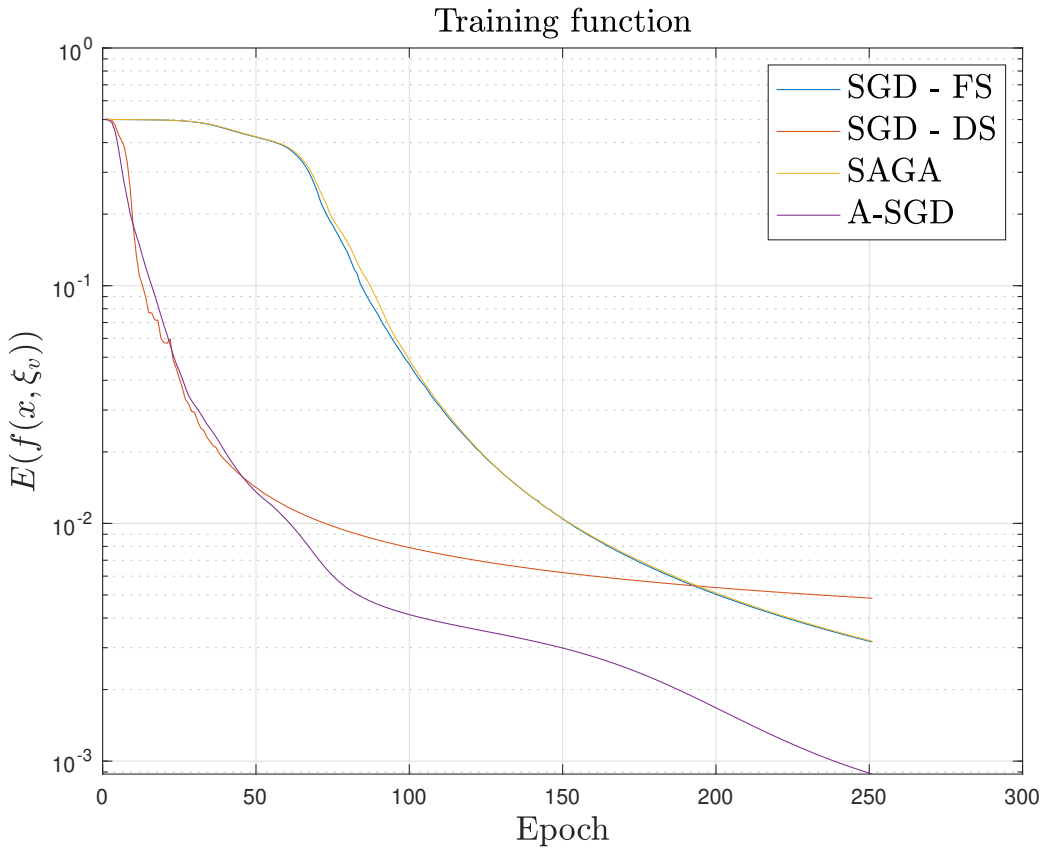
Lemma 3

We have:

1. $\delta^{[L]} = \Phi'(z^{[L]}) \odot (a^{[L]} - y_i)$
2. $\delta^{[l]} = \Phi'(z^{[l]}) \odot ((W^{[l+1]})^T \delta^{[l+1]})$ for $2 \leqslant l < L$
3. $\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}$ for $2 \leqslant l \leqslant L$,
4. $\frac{\partial C}{\partial W_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]}$ for $2 \leqslant l \leqslant L$.

Demo - SGD (FS+DS) vs SAGA vs A-SGD

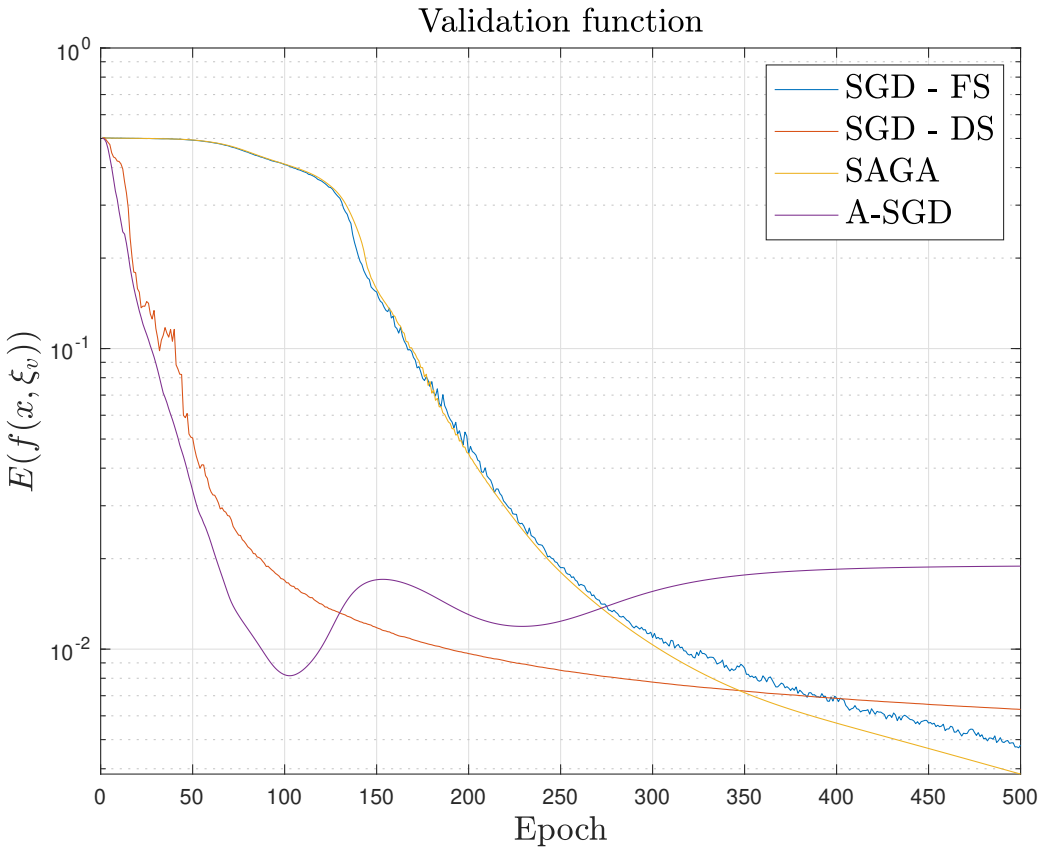
Test case: $n = 1000$ (split into 80-20 % training-validation points), $m = 10$ (size mini-batches), seed $= 5000$



Code ▸ Link, Code Ocean ▸ Link

Demo - SGD (FS+DS) vs SAGA vs A-SGD

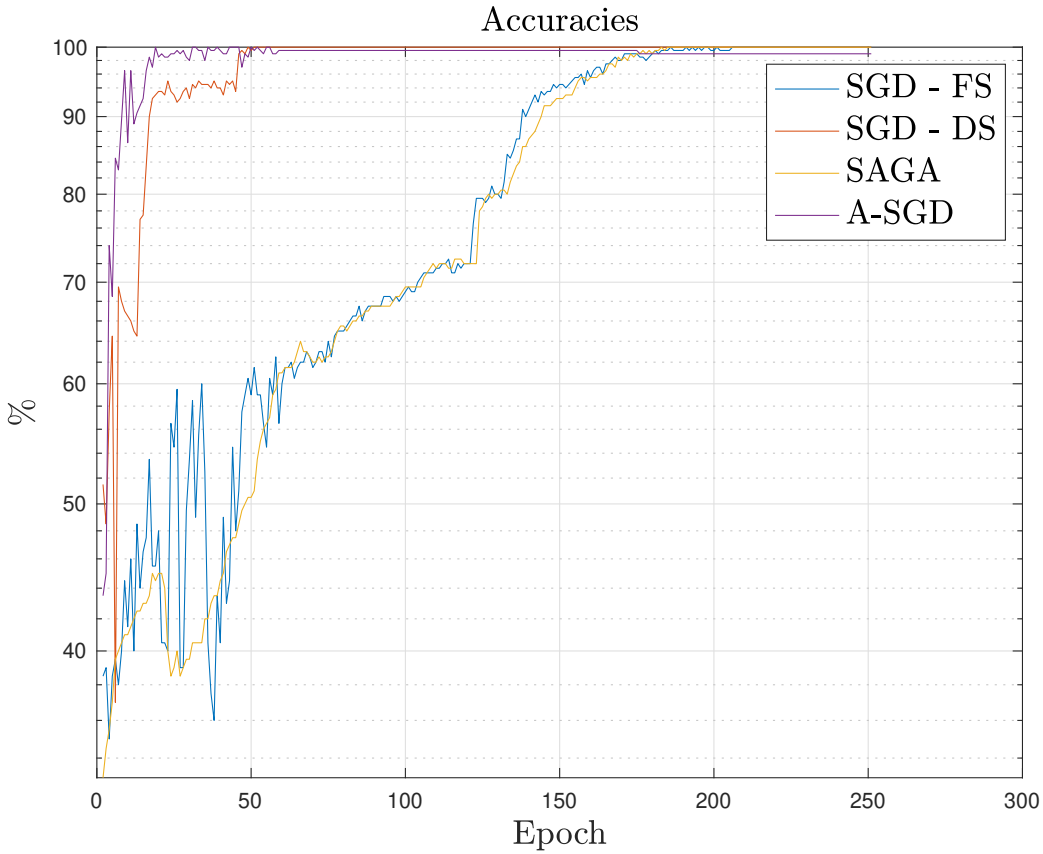
Test case: $n = 1000$ (split into 80-20 % training-validation points), $m = 10$ (size mini-batches), seed $= 5000$



Code ▸ Link, Code Ocean ▸ Link

Demo - SGD (FS+DS) vs SAGA vs A-SGD

Test case: $n = 1000$ (split into 80-20 % training-validation points), $m = 10$ (size mini-batches), seed $= 5000$



Code ▸ Link, Code Ocean ▸ Link

References

Boyd, Parikh, Chu, Peleato and Eckstein (2010)

Distributed Optimization and Statistical Learning via the Alternating Method of Multipliers

Machine Learning Vol. 3, No. 1 (2010) 1–122

Y.Nesterov. (2018)

Lectures on Convex Optimization.

Springer.

G. E. Hinton. (2012)

A practical guide to training restricted Boltzmann machines.

Neural Networks: Tricks of the Trade, volume 7700 of Lecture Notes in Computer Science pages 599–619. Springer, 2012.

References

N. Gillis (2021)

First-order methods for large-scale optimization

Lectures UMons - Advanced Opt. course

Bottou, Curtis, and Nocedal (2018)

Optimization Methods for Large-Scale Machine Learning.

SIAM REVIEW Vol. 60, No. 2, pp. 223–311, 2018.

G. Papamakarios (2014)

Comparison of Modern Stochastic Optimization Algorithms.

Technical report, University of Edinburgh

N. Keskar et al. (2017)

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.

arXiv:1609.04836

References

A. Agarwal et al. (2012)

Information- theoretic lower bounds on the oracle complexity of stochastic convex optimization

IEEE Trans. Inform. Theory, 58, pp. 3235-3249

B.T. Polyak. (1991)

New method of stochastic approximation type

Automat. Remote Control, 51, pp. 937-946

B.T. Polyak and A.B. Juditsky (1992)

Acceleration of stochastic approximation by averaging

SIAM J. Control Optim., 30, pp. 838-855

D. Ruppert (1988)

Efficient Estimations from a Slowly Convergent Robbins-Monro Process

Tech report, 781.

References

A.S. Nemirovski and D.B. Yudin (1978)

On Cesaro's convergence of the steepest descent method for approximating saddle point of convex-concave functions

Soviet Math. Dokl., 19

C.F. Higham and D.J. Higham (2019)

Deep Learning: An Introduction for Applied Mathematicians

SIAM REVIEW Vol. 61, No. 4, pp. 860–891, 2019.


That's all Folks!