ECE 586 BH: Interplay between Control and Machine Learning **Fall 2023**

Lecture 11

Control Tools for Stochastic Optimization and Supervised Learning, Part I

Lecturer: Bin Hu, Date:09/26/2023

In this lecture, we will talk about stochastic optimization methods for the following empirical risk minimization (ERM) problem in supervised learning:

$$\underset{x\in\mathbb{R}^p}{\text{minimize}} \;\; f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{11.1}$$

where $f:\mathbb{R}^p\to\mathbb{R}$ is a strongly-convex objective function. We will introduce several popular stochastic optimization algorithms and discuss how these methods can be represented as feedback dynamical systems as shown in Figure 11.1. In the next lecture, we will discuss how the dissipation inequality approach covered in the previous lecture can be used to unify the analysis of stochastic optimization methods.

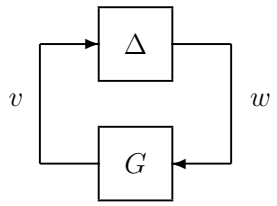

Figure 11.1. The Block-Diagram Representation for Feedback Interconnection $F_u(G,\Delta)$

11.1 Motivations: ERM and Supervised Learning

ERM is a key paradigm in machine learning and naturally leads to the optimization problem (11.1). Specifically, many supervised learning tasks, including ridge regression, logistic regression, and support vector machines, can be formulated as the following problem

$$\underset{x\in\mathbb{R}^p}{\text{minimize}}\, f(x) := \frac{1}{n}\sum_{i=1}^{n} l_i(x) + \lambda\Omega(x)$$

Here one wants to fit some prediction/classification model parameterized by x using the training data. The training set consists of n data points. The task is to fit a model x that works well for all the "unseen" data.

Interpretations for $l_i(x)$. The loss function $l_i(x)$ measures how well x performs on the i-th data point in the training set. If $l_i(x)$ is large, it means that the "loss" on the i-th data point is big and the model x works poorly on this data point. If $l_i(x)$ is small, it means the "loss" on the i-th data point is small and the model x works well on this data point. By minimizing the empirical risk $\frac{1}{n}\sum_{i=1}^n l_i(x)$, one expects the model x to work reasonably well for training data. This prevents underfitting.

Importance of $\Omega(x)$. If we allow the model to be arbitrarily complicated, we can obtain zero loss on training data but the model will work poorly on the data that have not been seen. This is called over-fitting. Roughly speaking, the difference between the model performance on the training data and the unseen data is called generalization error. One way to prevent overfitting and induce generalization is to add a regularizer $\Omega(x)$ that measures the model complexity. By adding such a term in the cost function, one expects the complexity of the resultant x is somehow controlled and hence the model x should "generalize" to the unseen data.[1] For example, one can choose $\Omega(x) = \|x\|^2$, and there exists some learning theory (e.g. stability theory) that can be used to explain how such ℓ_2-regularization induces generalization. Confining the search of x on small norm models can help generalization in many situations. Sometime $\Omega(x)$ is used to induce other desired structures. For example, the ℓ_1-regularization is typically used to induce sparsity.

What is λ? In ERM, λ is a hyperparameter which is tuned to trade off training performance and generalization. For the purpose of this course, let's say λ is a fixed positive number. In practice, λ is typically set as a small number between 10^{-8} and 0.1.

Example 1: Ridge regression. The ridge regression is formulated as an ERM problem with the following objective function

$$f(x) = \frac{1}{n}\sum_{i=1}^n (a_i^\mathsf{T} x - b_i)^2 + \frac{\lambda}{2}\|x\|^2 \tag{11.2}$$

where $a_i \in \mathbb{R}^p$ and $b_i \in R$ are data points used to fit the linear model x.

- What is this problem about? The purpose of this problem is to fit a linear relationship between a and b. One wants to predict b from a as $b = a^\mathsf{T} x$. The ridge regression gives a way to find such x based on the observed pairs of (a_i, b_i).
- Why is there a term $\frac{\lambda}{2}\|x\|^2$? Again, the term $\frac{\lambda}{2}\|x\|^2$ is just the ℓ_2-regularizer. It confines the complexity of the linear predictors you want to use. The high-level idea is that you want x to work for all (a, b), not just the observed pairs (a_i, b_i). Again, this is called generalization in machine learning. So adding such a term can induce the so-called stability and helps the predictor x to "generalize" for the data you have not seen. You need to take a machine learning course if you want to learn about generalization.

[1] What we mean is that the model x should work similarly on the training data and the unseen data.

- What is λ? λ is a hyperparameter which is tuned to trade off training performance and generalization. Again, λ is typically set as a small number between 10^{-8} and 0.1.

It is worth mentioning that f is L-smooth and m-strongly convex in this case. It is straightforward to verify that

$$f(x) = \frac{1}{2}x^\mathsf{T}\left(\frac{2}{n}\sum_{i=1}^n a_i a_i^\mathsf{T} + \lambda I\right)x - \left(\frac{2}{n}\sum_{i=1}^n b_i a_i\right)^\mathsf{T} x + \frac{1}{n}\sum_{i=1}^n b_i^2$$

which is a special case of the positive definite quadratic minimization problem. Notice $\frac{2}{n}\sum_{i=1}^n a_i a_i^\mathsf{T} + \lambda I > 0$ and hence f is m-strongly convex and L-smooth (why?). Therefore, we can apply the gradient method to ridge regression, and obtain a convergence rate $\rho = 1 - \frac{1}{\kappa}$ where κ is the condition number of the positive definite matrix $\frac{2}{n}\sum_{i=1}^n a_i a_i^\mathsf{T} + \lambda I$.

Example 2: ℓ_2-Regularized Logistic regression. The ℓ_2-regularized logistic regression is formulated as an ERM problem with the following objective function

$$f(x) = \frac{1}{n}\sum_{i=1}^n \log(1 + e^{-b_i a_i^\mathsf{T} x}) + \frac{\lambda}{2}\|x\|^2 \tag{11.3}$$

where $a_i \in \mathbb{R}^p$ and $b_i \in \{-1, 1\}$ are data points used to fit the linear model x.

- What is this problem about? The purpose of this problem is to fit a linear "**classifier**" between a and b. Let's say you have collected a lot of images of cats and dogs. You augment the pixels of any such image into a vector a and wants to predict whether the image is a cat or a dog. Let's say $b = 1$ if the image is a cat, and $b = -1$ if the image is a dog. So you want to predict b based on a. You want to find x such that $b = 1$ when $a^\mathsf{T} x \geq 0$, and $b = -1$ when $a^\mathsf{T} x < 0$. The logistic regression gives a way to find such x based on the observed feature/label pairs of (a_i, b_i). You may want to take a statistics course or a machine learning course if you want to learn more about logistic regression.

- Why is there a term $\frac{\lambda}{2}\|x\|^2$? Again, the term $\frac{\lambda}{2}\|x\|^2$ is the ℓ_2-regularizer. It is used to induce generalization and help x work on all the (a, b) not just the observed data points (a_i, b_i).

The function (11.3) is also L-smooth and m-strongly convex.

Other examples. There are many other convex examples including multi-class logistic regression, support vector machines, elastic nets, and PCA. The ERM problems in deep learning involve non-convex loss functions. The optimization of deep learning has not been fully understood and it is an important research topic.

Finite-sum Structure of ERM. The ERM problem is in the form of the finite-sum optimization (11.1) where $f = \frac{1}{n}\sum_{i=1}^{n} f_i$. If we apply the gradient method or Nesterov's method, we need to evaluate $\nabla f = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i$ for each iteration. In other words, we need to evaluate the gradient on all the data points. The computation cost for each iteration scales with $O(n)$. For big data applications, n is typically very large. The per iteration cost of the gradient method and Nesterov's method is high. This motivates the use of stochastic optimization methods that sample one or a small batch of data points for gradient estimate at every iteration. In stochastic optimization, the computation cost for each iteration does not depend on n and scales with $O(1)$. The hope is that there will be a lot of redundancy between data points and these stochastic methods will work well in some average sense in the long run. We will talk about various stochastic optimization methods and represent them as feedback interconnections in the next lecture.

11.2 Stochastic Optimization Methods for ERM

A classical way to solve (11.1) is the gradient method, which uses the following iteration:

$$x_{k+1} = x_k - \alpha \nabla f(x_k) \tag{11.4}$$

Since f is strongly convex, the gradient method with a well-chosen constant stepsize converges at a linear rate. To achieve accelerated convergence, we can apply Nesterov's method. Both the gradient method and Nesterov's method require computing a full gradient $\nabla f(x_k)$ at each step. Hence the iteration cost of these methods scale linearly with n. This leads to a high iteration cost when the size of the training set is large.

Consequently, stochastic optimization methods become more popular for large-scale ERM problems. Now we give a few examples.

- Stochastic gradient method: The baseline algorithm for large-scale learning tasks is the stochastic gradient (SG) method that iterates as

  $$x_{k+1} = x_k - \alpha \nabla f_{i_k}(x_k) \tag{11.5}$$

  where for each step k, the index i_k is sampled uniformly from the set $\{1, 2, \ldots, n\}$. The per iteration cost of the SG method is independent of n. At every step k, only one (or a small batch of) data point is sampled for gradient evaluation. The stochastic gradient $\nabla f_{i_k}(x_k)$ is an estimate for the true gradient $\nabla f(x_k)$. The hope is that in the long run the stochastic gradient method leads to a solution that works reasonably well in some average sense. The SG method is the most popular optimization method for large-scale learning tasks. However, one issue is that the SG method only converges linearly to some tolerance of the optimum for a well-chosen constant stepsize. Just think about that initializing the SG method from the optimal point x^* satisfying $\nabla f(x^*) = 0$. Notice $\nabla f(x^*) = 0$ does not mean $\nabla f_i(x^*) = 0$. Hence the SG method will not stay at this optimal point even if it is initialized there. The issue is that x^* is not a fixed

point for the SG method. If diminishing stepsize is used, the SG method will converge to the optimum at a sublinear rate.

- Stochastic average gradient (SAG): Compared with the stochastic gradient $\nabla f_{i_k}(x_k)$, an average gradient may be used to provide a better estimate for the true gradient. The basic idea is that one can use a vector y_k to memorize the gradient on each data point as follows

$$y_{k+1}^{(i)} := \begin{cases} \nabla f_i(x_k) & \text{if } i = i_k \\ y_k^{(i)} & \text{otherwise} \end{cases} . \tag{11.6}$$

where at each step k, a random training example i_k is drawn uniformly from the set $\{1, 2, \ldots, n\}$. Hence, at each k, one still only samples one data point and updates $y_k^{(i)}$ for that data point. Since the vector y has memorized the gradient on all the data points, averaging y should lead to a better estimate for the full gradient ∇f. SAG uses such an average gradient and iterates as

$$x_{k+1} = x_k - \alpha \left(\frac{1}{n} \sum_{i=1}^{n} y_{k+1}^{(i)} \right) \tag{11.7}$$

Therefore, at every step k, SAG first updates y for the average gradient evaluation and then updated x using the average gradient. With well-chosen constant stepsize α, SAG converges to the optimal solution of ERM. Why does SAG work? Intuitively, as x_k converges to the optimal point x^*, the change in x_k becomes smaller and smaller. Hence the average value $\left(\frac{1}{n} \sum_{i=1}^{n} y_{k+1}^{(i)} \right)$ approximates the true gradient better and better, and eventually converges to the true gradient. The detailed analysis for SAG is quite lengthy. This motivates the development of SAGA.

- SAGA: The idea is similar to SAG, but the update for x_{k+1} is modified as

$$x_{k+1} = x_k - \alpha \left(\nabla f_{i_k}(x_k) - y_k^{(i_k)} + \frac{1}{n} \sum_{i=1}^{n} y_k^{(i)} \right) \tag{11.8}$$

To see the difference between SAG and SAGA, just notice SAG's update rule (11.7) can be rewritten as

$$x_{k+1} = x_k - \alpha \left(\frac{\nabla f_{i_k}(x_k) - y_k^{(i_k)}}{n} + \frac{1}{n} \sum_{i=1}^{n} y_k^{(i)} \right) \tag{11.9}$$

Although the update rules for SAG and SAGA are similar, the convergence rate proof for SAGA is much simpler. This partially explains why SAGA gets more popular than SAG. The LMI tools in the controls field can be used to tell which method is easier to analyze at the early stage of algorithm developments. We will come back to this point later.

In this lecture, we focus on the above methods. In the next section, we will represent the above methods as feedback interconnections and comment on other stochastic methods, e.g. SVRG, Finito, and SDCA. Hopefully you will be convinced that stochastic optimization methods for ERM are just feedback dynamical systems.

11.3 Stochastic Methods as Feedback Systems

To model stochastic optimization methods as feedback systems, we need to allow either Δ or G to depend on the sampling index i_k. This leads to the following two formulations.

1. We can use an LTI system G and a stochastic perturbation Δ to form a feedback model for the SG method (and SVRG-like methods).
2. We can use a dynamical jump system G and a deterministic static nonlinearity Δ to form a feedback model for SAGA-like methods.

11.3.1 Using stochastic Δ to model SG

The SG method can be modeled as a feedback interconnection $F_u(G, \Delta)$ shown in Figure 11.1 if we choose $w = \Delta(v)$ as a stochastic nonlinear mapping $w_k = \nabla f_{i_k}(v_k)$ and set G to be the following LTI system

$$\begin{aligned} \xi_{k+1} &= \xi_k - \alpha w_k \\ v_k &= \xi_k \end{aligned}$$

To see this, we just set $\xi_k = x_k$. Then the first equation in the above LTI model becomes $x_{k+1} = x_k - \alpha w_k = x_k - \alpha \nabla f_{i_k}(v_k) = x_k - \alpha \nabla f_{i_k}(x_k)$. Notice in the modeling for the gradient method $x_{k+1} = x_k - \alpha \nabla f(x_k)$, we choose Δ as a static nonlinearity ∇f. For the SG method, the perturbation Δ depends on i_k. Therefore, it is not surprising that the convergence rate proofs for the gradient method and the SG method are quite similar. Notice that the dissipation inequality approach presented in Lecture 2 can be used to handle various types of Δ. Actually the stochastic mapping ∇f_{i_k} can also be directly handled via dissipation inequality as long as we are able to construct some informative supply rates for such a mapping.

In later lectures, we will show that standard assumptions (smoothness, convexity, etc) on f_i can be manipulated as quadratic supply rate conditions $\mathbb{E} \begin{bmatrix} \xi_k \\ w_k \end{bmatrix}^{\mathsf{T}} X \begin{bmatrix} \xi_k \\ w_k \end{bmatrix} \leq M$ with well-chosen X and M. Such supply rate conditions can be used to recover standard rate bounds for the SG method via our analysis routine.

Extensions. Many other stochastic methods can also be modeled as feedback interconnections of an LTI system G and a stochastic perturbation Δ. To handle stochastic gradient with momentum, we only need to modify the matrices (A, B, C) in the LTI model of G. To handle SVRG-like methods, we only need to modify Δ.

11.3.2 Jump system models for SAGA-like methods

SAGA and SAG can be rewritten as special cases of the following general jump system

$$\xi_{k+1} = A_{i_k}\xi_k + B_{i_k}w_k \tag{11.10}$$

$$v_k = C\xi_k \tag{11.11}$$

$$w_k = \begin{bmatrix} \nabla f_1(v_k) \\ \nabla f_2(v_k) \\ \vdots \\ \nabla f_n(v_k) \end{bmatrix} \tag{11.12}$$

The above general jump system model is just an interconnection of a linear jump system G and a static nonlinearity Δ that maps v to w as defined in (11.12). Here, Δ depends on the gradient information of all the data points in the training set. It seems that the computation of w_k at each k requires gradient information on all the data points. However, B_{i_k} is typically sparse for SAGA-like methods. Therefore, $B_{i_k}w_k$ only involves gradient evaluation on one data point, ensuring the low per-iteration cost of SAGA-like methods.

The feedback model (11.10) provides a unification for SAGA-like methods. We can rewrite SAG, SAGA, and many other variants in this form by properly choosing (A_{i_k}, B_{i_k}, C) for the linear jump system G. Now we show how to choose (A_{i_k}, B_{i_k}, C) for SAG and SAGA.

- **Jump system model for SAG:** Note that the gradient update rule for SAG is (11.6): $y_{k+1}^{(i)} = \nabla f_i(x_k)$ if $i = i_k$ and $y_{k+1}^{(i)} = y_k^{(i)}$ otherwise. Define the following stacked vector:

$$y_k = \begin{bmatrix} y_k^{(1)} \\ y_k^{(2)} \\ \vdots \\ y_n^{(k)} \end{bmatrix} \tag{11.13}$$

  At every step, the y information is almost unchanged except on the i_k-th data point. This can be summarized by the jump system iteration:

$$y_{k+1} = \left((I_n - e_{i_k}e_{i_k}^T) \otimes I_p\right) y_k + \left((e_{i_k}e_{i_k}^T) \otimes I_p\right) w_k \tag{11.14}$$

  where $w_k = \left[\nabla f_1(x_k)^T \cdots \nabla f_n(x_k)^T\right]$, and e_i is an n-dimensional vector whose i-th entry is 1 and other entries are 0. Here the notation "$\otimes$" denotes the Kronecker product. Clearly, $e_i e_i^{\mathsf{T}}$ is a matrix whose (i,i)-th entry is 1 and all other entries are 0. Now we can rewrite (11.9) as

$$\begin{aligned} x_{k+1} &= x_k - \frac{\alpha}{n}(e_{i_k}^T \otimes I_p)(w_k - y_k) - \frac{\alpha}{n}(e^T \otimes I_p)y_k \\ &= x_k - \frac{\alpha}{n}\left((e - e_{i_k})^T \otimes I_p\right) y_k - \frac{\alpha}{n}(e_{i_k}^T \otimes I_p)w_k \end{aligned} \tag{11.15}$$

where e is a vector whose entries are all 1. Since $v_k = x_k$, we can combine (11.14) and (11.15) to obtain the following jump system G mapping from w to v:

$$\begin{aligned}\begin{bmatrix} y_{k+1} \\ x_{k+1} \end{bmatrix} &= \begin{bmatrix} (I_n - e_{i_k} e_{i_k}^T) \otimes I_p & \tilde{0} \otimes I_p \\ -\frac{\alpha}{n}(e - e_{i_k})^T \otimes I_p & I_p \end{bmatrix} \begin{bmatrix} y_k \\ x_k \end{bmatrix} + \begin{bmatrix} (e_{i_k} e_{i_k}^T) \otimes I_p \\ (-\frac{\alpha}{n} e_{i_k}^T) \otimes I_p \end{bmatrix} w_k \\ v_k &= \begin{bmatrix} \tilde{0}^T \otimes I_p & I_p \end{bmatrix} \begin{bmatrix} y_k \\ x_k \end{bmatrix} \end{aligned} \tag{11.16}$$

Since we already have $w = \Delta(v)$, we can represent SAG as $F_u(G, \Delta)$ where G is described by the above linear jump system model. Notice in this case the state of G is $\xi_k := \begin{bmatrix} y_k \\ x_k \end{bmatrix}$.

- **Jump system model for SAGA:** Notice the update of y_k is still captured by (11.14) with $w_k = \begin{bmatrix} \nabla f_1(x_k)^T \cdots \nabla f_n(x_k)^T \end{bmatrix}^T$. Now we can rewrite (11.8) as

$$\begin{aligned} x_{k+1} &= x_k - \alpha(e_{i_k}^T \otimes I_p)(w_k - y_k) - \frac{\alpha}{n}(e^T \otimes I_p) y_k \\ &= x_k - \frac{\alpha}{n}\left((e - n e_{i_k})^T \otimes I_p\right) y_k - \alpha(e_{i_k}^T \otimes I_p) w_k \end{aligned} \tag{11.17}$$

Since $v_k = x_k$, we can combine (11.14) and (11.17) to obtain the following jump system G mapping from w to v:

$$\begin{aligned}\begin{bmatrix} y_{k+1} \\ x_{k+1} \end{bmatrix} &= \begin{bmatrix} (I_n - e_{i_k} e_{i_k}^T) \otimes I_p & \tilde{0} \otimes I_p \\ -\frac{\alpha}{n}(e - n e_{i_k})^T \otimes I_p & I_p \end{bmatrix} \begin{bmatrix} y_k \\ x_k \end{bmatrix} + \begin{bmatrix} (e_{i_k} e_{i_k}^T) \otimes I_p \\ (-\alpha e_{i_k}^T) \otimes I_p \end{bmatrix} w_k \\ v_k &= \begin{bmatrix} \tilde{0}^T \otimes I_p & I_p \end{bmatrix} \begin{bmatrix} y_k \\ x_k \end{bmatrix} \end{aligned} \tag{11.18}$$

Again, we have $\xi_k = \begin{bmatrix} y_k \\ x_k \end{bmatrix}$. Putting the above model for G in a feedback loop with Δ directly realizes SAGA as a special case of (11.10).

Fixed points of the jump system models for SAG and SAGA. Suppose x^* satisfies $\nabla f(x^*) = 0$. Then we define $w^* = \begin{bmatrix} \nabla f_1(x^*)^T \cdots \nabla f_n(x^*)^T \end{bmatrix}$. Next we can set $\xi^* = \begin{bmatrix} (w^*)^T & (x^*)^T \end{bmatrix}^T$, and $v^* = x^*$. Using the fact that $\sum_{i=1}^n \nabla f_i(x^*) = 0$, we can verify that (ξ^*, w^*, v^*) provides a fixed point for the jump system model of SAG and SAGA. If ξ_k converges to ξ^*, then x_k converges to x^* and $y_k^{(i)}$ converges to $\nabla f_i(x^*)$. It SAGA and SAG are initialized at such fixed points, they are going to stay there. This partially fixes the issue of the SG method.

Generality. Many other SAGA-like methods including Finito, SDCA, and point-SAGA can be also modeled using the above jump system model if we modify (A_{i_k}, B_{i_k}, C) properly.

What is next? In the next lecture, we will tailor the quadratic constraint approach to provide a unified analysis for stochastic optimization.