Lab 3 - Gradient Descent GD vs Stochastic Gradient Descent SGD

Main Topics:

- Gradient Descent
- Stochastic Gradient Descent
- Computational aspects of GD vs SGD

In this lab we will see how to solve the ERM using iterative methods such as GD and SGD. We will focus on the computational aspects that make SGD a compelling algorithm w.r.t. GD. Most of this lab will focus on linear models (since they are very easy to be handled and theoretically studied), nonetheless many ideas follow mutatis mutandis to non linear models and therefore to non convex optimization.
We cannot stress enough how important SGD is in real world Machine Learning: it is the workhorse algorithm for convex and non convex optimization (especially in the large data regime) and has a countless number of applications ranging from pure ML, to robotics, to Computer Vision, to Recomendation Systems etc ...

Notation

Recall the notation we used so far to denote *generalization error* $L_{\mathcal{D}}(h)$ and training error $L_{\mathcal{S}}(h)$. The learning problem requires to find a function $\hat{h}$ that belongs to the model class $\mathcal{H}$ such that the generalization error is minimized. In practice since we do not have access to $L_{\mathcal{D}}(h)$ we use a the training error as a proxy to the generalization and find a candidate model $\hat{h}_{ERM} \in \mathcal{H}$. We find $\hat{h}_{ERM}$ by solving the following optimization problem:

$$\hat{h}_{ERM} = \arg\min_{h \in \mathcal{H}} L_S(h) = \arg\min_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} l(y_i, h(x_i)) \tag{1}$$

This optimization problem is pretty general, the way we solve it **highly** depends on the model class and the loss we choose. Only in very *special* cases we can find the optimal solution in closed form. See e.g. the Least-square problem in which the loss is the squared loss and the model class is the class of linear models, do not forget in this case we are facing a convex optimization problem, which is usually very well studied and understood. Nonetheless in a general ERM problema we are not guaranteed to have neither closed form solutions nor convexity, for example:

- logistic regression, in which we have a convex problem whose solution is not obtainable in closed form
- Deep Learning, in which the model class is the class of Deep Neural Networks and the loss is the Cross-Entropy Loss or squared loss (depending on the problem)

For the sake of simplicity let's assume the model class $\mathcal{H}$ is parametrized by a vector of parameters w (hence we have h_w and we can simply write $L_{\mathcal{S}}(w)$), moreover we assume that $L_{\mathcal{S}}(w)$ is **differentiable** w.r.t. w (this assumption is important since we are going to compute gradients, but be aware it is possible to extend some of the results also to the non-differentiable case).

Gradient Descent

In order to solve ERM using GD follow these steps:

- initialize $w^{(0)}$ (you can choose it randomly or based on your prior knowledge or to zero)
- at each iteration (indexed by t) update the parameter $w^{(t)}$ using the following update rule

$$w^{(t+1)} = w^{(t)} - \eta \nabla L_{\mathcal{S}}(w^{(t)}) \qquad t = 1, 2, \dots$$

Where η is a positive number (which is usually chosen "small", what we consider "small" depends on the optimization problem you are solving.

- the sequence of updates could be infinite but usually we explicitly choose a termination condition:
  - the number of iterations to perform
  - the value of the loss under a certain threshold
  - the magnitude of the gradient below a certain threshold.

Once the termination condition is meet we return the optimal parameter we found along the iterations (call T the number of iterations performed), once again we can choose different rules to return the best parameter:

- the last parameter
- the average parameter of the last K iterations
- the best performing vector we found ($\arg\min_{t \in [T]} L_S(w^{(t)})$)

Problem: Local minima, you will not avoid local minima since you are following the steepest descent direction everytime. Therefore if you get stuck into a local minimum whose loss value is high, it would be impossible to find a better solution to the optimization problem.

Stochastic Gradient Descent

The main idea behind SGD is that we can apply the very same update rule we described for GD but with a different gradient than $\nabla L_{\mathcal{S}}(w^{(t)})$, let's call it $\nabla \hat{L_{\mathcal{S}}}(w^{(t)})$. Which are the properties that this different gradient must satisfy?
We only require that $\nabla \hat{L_{\mathcal{S}}}(w^{(t)})$ is an unbiased estimate of $\nabla L_{\mathcal{S}}(w^{(t)})$. What does it mean that SGD gradient is an unbiased estimator of the GD gradient ($\mathbb{E}[\nabla \hat{L_{\mathcal{S}}}(w^{(t)})] = \nabla L_{\mathcal{S}}(w^{(t)})$)? Where is the randomness which we considering?
After all we are optimizing the training error which is a **deterministic number** (since the training set is fixed). This randomness is introduced by SGD itself and has nothing to do with the randomness of our data. To make it simple, a possible example of SGD gradient is the following: suppose you know the GD gradient $\nabla L_{\mathcal{S}}(w^{(t)})$ and in place of the GD update rule you decide to apply the following one:

$$w^{(t+1)} = w^{(t)} - \eta \underbrace{(\nabla L_{\mathcal{S}}(w^{(t)}) + e_t)}_{\text{this is the SGD gradient}} \qquad t = 1, 2, \dots$$

where $e_t \sim \mathcal{N}(0, I)$ which is i.i.d. (i.e. no correlation with parameters or data). As you can see the gradients used to perform the update are now noisy (but the noise has been introduced only by the optimization algorithm, it is not present on the ERM before we choose to solve it using SGD).
If you now take the expectation $\mathbb{E}_{SGD}[\nabla L_{\mathcal{S}}(w^{(t)}) + e_t] = \mathbb{E}_{SGD}[\nabla L_{\mathcal{S}}(w^{(t)})] + \mathbb{E}_{SGD}[e_t] = \nabla L_{\mathcal{S}}(w^{(t)})$ the SGD gradient is an unbiased estimate of the GD gradient (with $\mathbb{E}_{SGD}$ we stressed the fact that this expected value is computed w.r.t. to the random variables used to generate the noise by the SGD algorithm).

Depending on the noise SGD introduces we have different SGD algorithms, for example if you add uncorrelated gaussian noise you are using SGLD (Stochastic Gradient Langevin Dynamics).

We are now ready to consider the standard noise used to compute SGD updates in real world ML applications. Typically in ML applications (as you have seen in class) the losses we use are written as the sum of m terms each one associated to one single datum (e.g. $L_{\mathcal{S}}(w) := \frac{1}{m}\sum_{i=1}^{m} l(y_i, h_w(x_i))$). When m is *very* large we might not be able to compute the sum over all the data (e.g. we might incur in time or memory constraints). SGD solves the cumpuational and memory burden introducing a special kind of noise which allows us to consider at each iteration a subset of the dataset available (we call this subset minibatch, we shall refer to its size as B).

The SGD gradient is:

$$\nabla L_{\mathcal{S},\mathcal{B}^{(t)}}(w^{(t)}) = \frac{1}{B}\sum_{b \in \mathcal{B}^{(t)}} l(y_b, h_w(x_b))$$

The way we model such a noise is by means of indicator random variables $\mathbb{1}_b$ which weight each single datum in the loss sum:

$$\nabla L_{\mathcal{S},\mathcal{B}^{(t)}}(w^{(t)}) = \frac{1}{B}\sum_{b=1}^{m} \mathbb{1}_b \nabla l(y_b, h_w(x_b)) \qquad \text{where} \quad \sum_{b=1}^{m} \mathbb{1}_b = B$$

Choosing $\mathbb{1}_b \quad b = 1, \dots, m$ with different distribution we get different sampling schemes and therefore different types of injected noise (typically you choose a subset of your dataset with or without replacement). If we choose $\mathbb{1}_b$ as a Bernoulli random variable with mean (or probability) $\frac{1}{m}$ we are describing extraction without replacement (i.e. no repeating indeces are allowed in the estimate of the gradient).

TODO: Show $\mathbb{E}_{SGD}[\frac{1}{B}\sum_{b=1}^{m} \mathbb{1}_b \nabla l(y_b, h_w(x_b)) | \sum_{j=1}^{m} \mathbb{1}_j = B] = \nabla L_{\mathcal{S}}(w^{(t)})$, assume $\mathbb{1}_b$ are i.i.d. Bernoulli random variables with mean $\frac{1}{m}$.

Hints to solve the TODO:

- Recall the property of expected value of Bernoulli r.v. (link with probability of success).
- Compute the $\mathbb{P}[\mathbb{1}_b | \sum_{j=1}^{m} \mathbb{1}_j = B]$ using Bayes rule
- To solve previous step recall Binomial distribution probability expression (you will need it twice)

Therefore also this SGD update rule provides an unbiased estimator of the GD gradient!

Up to now we established notation and intuition behind SGD, how to perform it in practice?

In order to solve ERM using SGD follow these steps:

- initialize $w^{(0)}$ (you can choose it randomly or based on your prior knowledge or to zero)
- generate the stochastic estimate of the gradient of $L_{\mathcal{S}}(w^{(t)})$:
  - it depends on the kind of noise you are willing to inject in the optimization, now assume we are using indicators variables in order to reduce the computational burdens
  - Choose B indexes $\mathcal{B}^{(t)}$ from the train dataset indexes $[m]$ (you can choose them with or without replacement)
- at each iteration (indexed by t) update the parameter $w^{(t)}$ using the following update rule

$$w^{(t+1)} = w^{(t)} - \eta \nabla L_{\mathcal{S},\mathcal{B}^{(t)}}(w^{(t)}) \qquad t = 1, 2, \dots$$

When to stop? Same as GD (we will prefer to perform a fixed number of interations: T).

How to choose the final optimal parameter? Same as GD (we will prefer to choose the last obtained parameter at time T).

Note 1: Differently from GD using SGD update rule you can avoid local minima (think a little bit about it, it is pretty intuitive), at least there exists a non zero probability on the event that your SGD run does not get stuck into a local minimum of the training error.

Note 2: Since SGD does not properly converge it is usually assumed the learning rate η is shurnk as the number of iterations grows. We will briefly explore how to schedule the learning rate later in this lab.

Gradients notation

The following code is implemented in a very general fashion. This means we need to use a pretty general notation (i.e. we will not assume linear models only).
In the following we will be interested in the following quantity: $\nabla_w L_{\mathcal{S}}(w)$ where $L_{\mathcal{S}}(w) = \frac{1}{m}\sum_{i=1}^m l(y_i, h_w(x_i))$.

A very simple and general way to write such a gradient is the following:
$\nabla_w L_{\mathcal{S}}(w) = J_X \nabla_{h_w(X)} L_{\mathcal{S}}(w)$
where J_X is the jacobian of the model outputs and $L_{\mathcal{S}}(w)$ is simpy the gradient of the loss w.r.t. model outputs (at all input location).

How are these quantities defined?

$$J_X := \begin{bmatrix} \frac{\partial h_w(x_1)}{\partial w_1} & \frac{\partial h_w(x_2)}{\partial w_1} & \dots & \frac{\partial h_w(x_n)}{\partial w_1} \\ \frac{\partial h_w(x_1)}{\partial w_2} & \frac{\partial h_w(x_2)}{\partial w_2} & \dots & \frac{\partial h_w(x_n)}{\partial w_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_w(x_1)}{\partial w_p} & \frac{\partial h_w(x_2)}{\partial w_p} & \dots & \frac{\partial h_w(x_n)}{\partial w_p} \end{bmatrix} = [\nabla_w h_w(x_1) \quad \nabla_w h_w(x_2) \quad \dots \quad \nabla_w h_w(x_n)] \tag{2}$$

$$\nabla_{h_w(X)} L_{\mathcal{S}}(w) := \begin{bmatrix} \frac{L_{\mathcal{S}}(w)}{\partial h_w(x_1)} \\ \frac{L_{\mathcal{S}}(w)}{\partial h_w(x_2)} \\ \vdots \\ \frac{L_{\mathcal{S}}(w)}{\partial h_w(x_m)} \end{bmatrix} \tag{3}$$

Note: $J_X \in \mathbb{R}^{p,n}$ and $\nabla_{h_w(X)} L_{\mathcal{S}}(w) \in \mathbb{R}^n$

Let's see why this relation holds (it's only a matter of computations and wise use of chain rule, do not skip it and understand what is going on).
Let's focus on the first element of the vector $\nabla_{h_w(X)} L_{\mathcal{S}}(w)$:

$$\frac{\partial L_{\mathcal{S}}(w)}{\partial w_1} = \frac{\partial L_{\mathcal{S}}(h_w(x_1), h_w(x_2), \dots, h_w(x_m))}{\partial w_1} = \underbrace{\sum_{i=1}^m \frac{\partial L_{\mathcal{S}}(w)}{\partial h_w(x_i)} \frac{\partial h_w(x_i)}{\partial w_1}}_{\text{Chain rule}} \tag{4}$$

$$= \begin{bmatrix} \frac{\partial h_w(x_1)}{\partial w_1} \\ \frac{\partial h_w(x_2)}{\partial w_1} \\ \vdots \\ \frac{\partial h_w(x_m)}{\partial w_1} \end{bmatrix}^T \begin{bmatrix} \frac{\partial L_{\mathcal{S}}(w)}{\partial h_w(x_1)} \\ \frac{\partial L_{\mathcal{S}}(w)}{\partial h_w(x_2)} \\ \vdots \\ \frac{\partial L_{\mathcal{S}}(w)}{\partial h_w(x_m)} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial h_w(x_1)}{\partial w_1}, \frac{\partial h_w(x_2)}{\partial w_1}, \dots, \frac{\partial h_w(x_m)}{\partial w_1} \end{bmatrix}}_{\text{First row of the Jacobian} J_X} \nabla_{h_w(X)} L_{\mathcal{S}}(w) \tag{5}$$

GD and SGD for Least Squares regression

Recall the usual notation of Least Squares.

$$L_S(w) = \frac{1}{m}\|Y - Xw\|^2 \tag{6}$$

where Y and X are the matrices whose $i-$th row are, respectively, the output data y_i and the input vectors $x_i^\top$.

Let's compute the gradient of the training error:

$$\nabla_w L_{\mathcal{S}}(w) = J_X \nabla_{h_w(X)} L_{\mathcal{S}}(w) = \underbrace{X^T}_{\text{Jacobian for a linear model}} \underbrace{\left[-\frac{2}{m}(Y - Xw)\right]}_{\nabla_{h_w(X)} L_{\mathcal{S}}(w)} = \frac{2}{m}(-X^T Y + X^T X w) = \frac{2}{m}\Big(-\sum_{i=0}^m x_i y_i + \sum_{i=0}^m x_i x_i^T w \Big)$$

What are the update equations in the case of GD and SGD?

For GD:

$$w^{(t+1)} = w^{(t)} - \eta \tfrac{2}{m}(-X^T Y + X^T X w^{(t)}) = \Big(I - \tfrac{2\eta}{m} X^T X\Big) w^{(t)} - \eta \tfrac{2}{m} X^T Y \qquad t = 1, 2, \dots$$

For SGD let's explicitly write the summations:

$$w^{(t+1)} = w^{(t)} - \eta \tfrac{2}{m}\Big(-\sum_{b \in \mathcal{B}^{(t)}} x_b y_b + \sum_{b \in \mathcal{B}^{(t)}} x_b x_b^T w \Big) = \Big(I - \tfrac{2\eta}{m} \sum_{b \in \mathcal{B}^{(t)}} x_b x_b^T\Big) w^{(t)} - \eta \tfrac{2}{m} \sum_{b \in \mathcal{B}^{(t)}} x_b y_b \qquad t = 1, 2, \dots$$

Creating the regression dataset

Let's draw inspiration from the last lab and create a toy dataset for regression so that we can easily see what the theory suggests.

Let's take the following linear model:
$y := Ax + e$
where assume $x \sim \mathcal{N}(\mu_x, \Sigma_x)$ and $e \sim \mathcal{N}(0, \sigma^2)$ and x is independent from e (this implies $p(y|x) \sim \mathcal{N}(Ax, \sigma^2)$).

We also have $y \sim \mathcal{N}(A\mu_x, A\Sigma_x A^T + \sigma^2)$

Instability of GD for large eta

Why for high eta we have instability? We know that GD dynamics equivalent to (have a look at previous LS Markdown cell) a linear dynamical systems whose state transition matrix is $(I - \frac{2\eta}{m} X^T X)$, if the spectrum of this matrix has some non stable eigenvalue then GD dynamics will become unstable.

Let $\Lambda(I - \frac{2\eta}{m} X^T X)$ be the spectrum of the matrix $I - \frac{2\eta}{m} X^T X$, the following realtion is trivially true:
$\Lambda(I - \frac{2\eta}{m} X^T X) = I - \Lambda(\frac{2\eta}{m} X^T X) = I - \frac{2\eta}{m} \Lambda(X^T X)$

Now remember $\Lambda(X^T X) \geq 0$ (since this is a p.s.d. matrix). This means that it exists a certain η from which the system will not be anymore stable (for larger η). In fact the linear system will have a pole smaller that -1 (what matters here is that the pole's absolute value is greater than 1) and therefore the dinamics will not be stable anymore.