Introduction to Neural Networks and Deep Learning

Optimization in Deep Learning

Andres Mendez-Vazquez

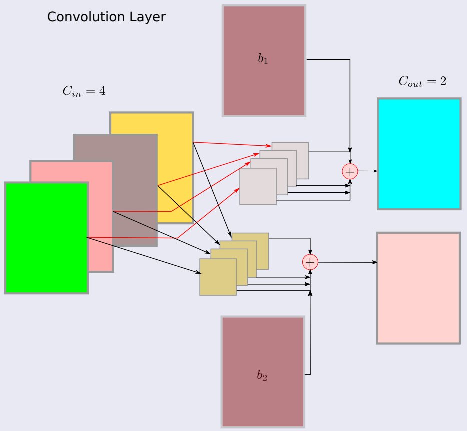
April 16, 2021

Outline

Beyond Simple Optimization [2]

As always Optimization is a Problem in Deep Learning

- We have a huge composition of linear and non-linear functions [1]



Thus

It is not possible to talk about

- That classic optimization theory [3, 4, 5] can explain totally the complexities on those deep architectures

For example, we have

- The well-known "exploding/vanishing" gradient

Actually, we have

The following Issues

$$\text{Opt Problems}\begin{cases}\text{Local} & \Rightarrow \begin{cases}\text{Convergence Issue} & \Rightarrow \text{Vanishing/Exploding Gradient}\\ \text{Convergence Speed Issue} & \Rightarrow \text{As always problematic}\end{cases}\\ \text{Global} & \Rightarrow \text{Bad Local Minima, Plateaus, etc}\end{cases}$$

We have a data set

We have the following

- Data points $x_i \in \mathbb{R}^{d_x}$ and labels $y_i \in \mathbb{R}^{d_y}$ for $i = 1...n$

Thus, we want the architecture to predict y_i based on x_i

$$f_\theta = W^L \phi \left[W^{L-1} \cdots \phi \left[W^2 \phi \left[W^1 x_i + b_1 \right] + b_2 \right] + b_{L-1} \right] + b_L$$

This is "trained"

By the use of Backpropagation

- With Gradient Descent, Stochastic Gradient Descent, ADAM, etc

Thus

- It is a good idea to review those techniques

Gradient Descent [6, 7, 5, 4]

The basic procedure is as follow

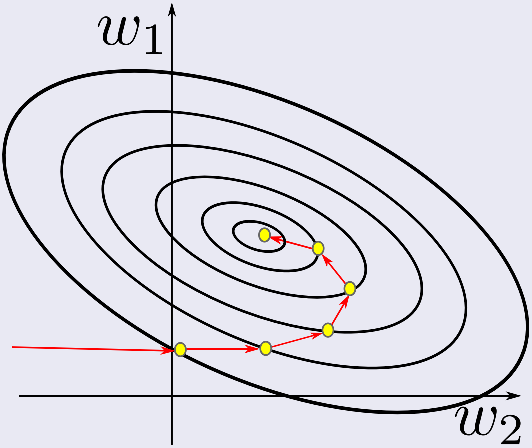
1. Start with a random weight vector $\boldsymbol{w}_0$.
2. Compute the gradient vector $\nabla J\left(\boldsymbol{w}_0\right)$.
3. Obtain value $\boldsymbol{w}_1$ by moving from $\boldsymbol{w}_0$ in the direction of the steepest descent:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta_n \nabla J\left(\boldsymbol{w}_n\right) \tag{1}$$

η_n is a positive scale factor or learning rate!!!

Geometrically

We have the following



$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - \eta(k)\nabla J(\boldsymbol{w}(k))$

The Problems of Gradient Descent with Large Data Sets

It is possible to prove

- That the gradient direction gives the greatest increase direction!!!

We have a problem in cost functions like in Deep Neural Networks

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \left(y_i - f(\boldsymbol{w}, \boldsymbol{x}_i)\right)^2$$

- Where, we have that $f(\boldsymbol{w}, \boldsymbol{x}_i) = f_1 \circ f_2 \circ f_3 \circ \cdots \circ f_T(\boldsymbol{w}, \boldsymbol{x}_i)$

Even though

Gradient Descent could be discarded easily

- For the Deep Learning Architectures

It is good to analyze some of its properties

- Given how they apply to other optimization algorithms for Deep Learning

Thus, we need to talk about

Convergence Rate

- Important for Speed Ups
  - After all you want to avoid slow algorithms

Convex Functions

- The most basic stuff
  - A unique minima

Attempts to Accelerate Gradient Descent

- To obtain better performances
  - Momentum, Nesterov... and Stochatic Gradient Descent

Do you remember the problem of the η step size?

Gradient Descent with fixed step size

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta \nabla J\left(\boldsymbol{w}_n\right)$$

Why to worry about this?

- Because, we want to know how fast Gradient Descent will find the answer...

We have

Lipschitz Continuous [8]

- Lipschitz continuity, named after Rudolf Lipschitz, is a strong form of uniform continuity for functions.

Uniform continuity

- The function $f : A \to \mathbb{R}$ is said to be uniformly continuous on A iff for every $\epsilon > 0$, $\exists \delta > 0$ such that $|\boldsymbol{x} - \boldsymbol{y}| < \delta$ implies $|f(\boldsymbol{x}) - f(\boldsymbol{y})| < \epsilon$.

Lipschitz Continuous

Definition

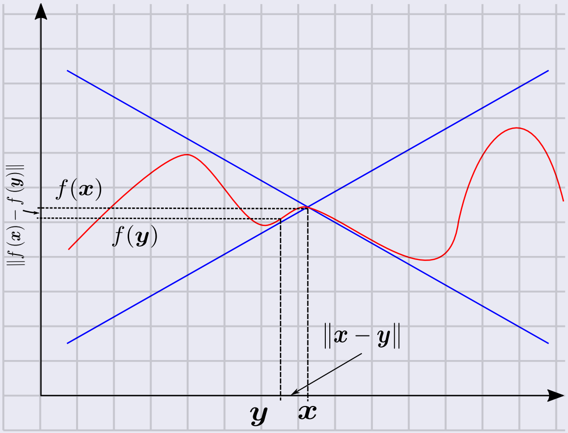
- A function $f : S \subset \mathbb{R}^n \to \mathbb{R}$ satisfies the Lipschitz Continuous at $\boldsymbol{x} \in S$, if there is a such constant $L > 0$ such that

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq L \|\boldsymbol{x} - \boldsymbol{y}\|$$

for all $\boldsymbol{y} \in S$ sufficiently near to x. **Lipschitz continuity can be seen as a refinement of continuity.**

Example when you see L as the slope

Here the function $f : \mathbb{R} \to \mathbb{R}$



$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\|$
$f(\boldsymbol{x})$
$f(\boldsymbol{y})$
$\|\boldsymbol{x} - \boldsymbol{y}\|$
$\boldsymbol{y}$
$\boldsymbol{x}$

An interesting property of such setup

The derivative of the function cannot exceed L (Example, $f:\mathbb{R}\to\mathbb{R}$)

$$f'(x) = \lim_{\delta\to\infty} \frac{f(x+\delta) - f(x)}{\delta}$$

Then, we have that

$$f'(x) = \lim_{\delta\to\infty} \frac{f(x) - f(y)}{x-y} \leq \lim_{\delta\to\infty} \frac{|f(x) - f(y)|}{|x-y|} \leq L$$

Therefore

Lipschitz Continuity implies

$$|f'(x)| < L$$

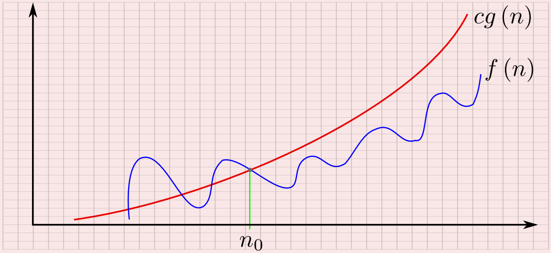
Convergence Idea

Definition (Big O - Upper Bound) [9]

For a given function $g(n)$:

$$O(g(n)) = \{f(n) \mid \text{There exists } c > 0 \text{ and } n_0 > 0 \text{ s.t. } 0 \leq f(n) \leq cg(n)\ \forall n \geq n_0\}$$

Example



What are the implications?

Definition [8]

- Suppose that the sequence $\{x_n\}$ converges to the number L:

We say that this sequence converges linearly to L, if there exists a number $\frac{1}{n} \in (0,1)$ such that

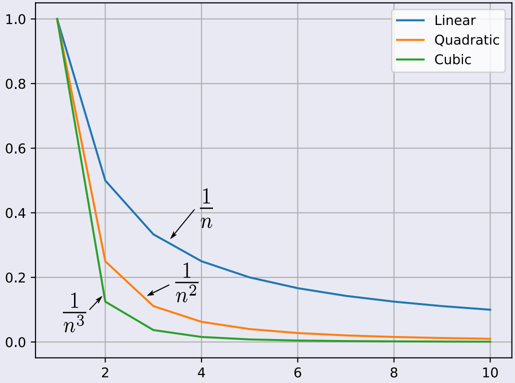
$$\lim_{k \longrightarrow \infty} \frac{|x_{n+1} - L|}{|x_n - L|} = \frac{1}{n}$$

Thus, Gradient Descent has a linear convergence speed

- If you do a comparison with quadratic convergence...

Example

As you can see the quadratic is faster than linear in convergence



Why the importance of Convex Functions? [4, 5, 3]

There is an interest on the rates of convergence for many optimization algorithms

- And they are affected by the different cost function that can be used:
  - Lipschitz-continuity, convexity, strong convexity, and smoothness

There are different rates of convergence for the Gradient Descent

- For example, when a function is strongly convex with $\alpha > 0$

$$\nabla^2 f(x) \succ \alpha I \Longleftrightarrow \nabla^2 f(x) - \alpha I \succ 0 \text{ (Matrix greater)}$$

Actually

You have $\nabla^2 f(x)$ is a squared symmetric matrix

- Thus, it is positive definite

This means that

- The curvature of $f(\boldsymbol{x})$ is not very close to zero, making possible to accelerate the convergence

Convex Sets

Definition

- For a convex set X, for any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ such that $\boldsymbol{x}, \boldsymbol{y} \in X$, the line between them lies within the set

$$\boldsymbol{z} = \lambda \boldsymbol{x} + (1 - \lambda) \boldsymbol{y}, \forall \theta \in (0, 1) \text{ then } \boldsymbol{z} \in X$$

  - The sum $\lambda \boldsymbol{x} + (1 - \lambda) \boldsymbol{y}$ is termed as convex linear combination.

Convex Functions

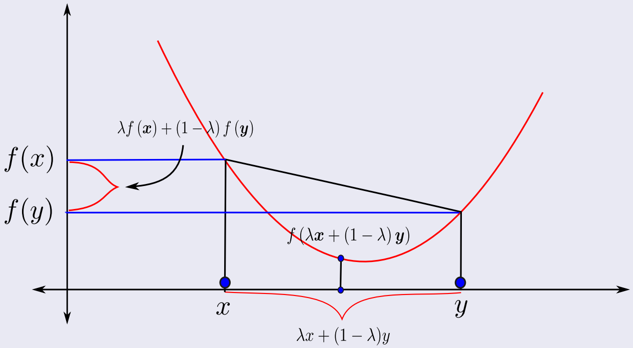
Definition

- . A function $f(\boldsymbol{x})$ is convex if the following holds:
  1. The Domain of f is convex
  2. $\forall \boldsymbol{x}, \boldsymbol{y}$ in the Domain of f and $\lambda \in (0,1)$

$$f\left(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}\right) \leq \lambda f\left(\boldsymbol{x}\right) + (1-\lambda) f\left(\boldsymbol{y}\right)$$

Graphically

We have the following


$\lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$
$f(x)$
$f(y)$
$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y})$
x
y
$\lambda x + (1-\lambda) y$

Convergence of gradient descent with **fixed step size**

Theorem

- Suppose the function $f : \mathbb{R}^d \to \mathbb{R}$ is convex and differentiable, and we have that $\left\|\nabla f\left(\boldsymbol{x}\right) - \nabla f\left(\boldsymbol{y}\right)\right\|_2 \leq L \left\|\boldsymbol{x} - \boldsymbol{y}\right\|$ (Lipschitz Continuous Gradient) for any $\boldsymbol{x}, \boldsymbol{y}$ and $L > 0$.

We have that

- Then, if we run the **gradient descent** for n iterations with a fixed step size $\eta \leq \frac{1}{L}$, it will yield a solution f_n which satisfies

$$f\left(x_n\right) - f\left(x^*\right) \leq \frac{\left\|x_{(0)} - x^*\right\|_2^2}{2\eta n}$$

where $f\left(x^*\right)$ is the optimal value.

Proof

$f\left(\boldsymbol{x}\right)$ is Lipschitz continuous with constant L implies $\left(\left\|\boldsymbol{y}-\boldsymbol{x}\right\|^2 = \left\|\boldsymbol{y}-\boldsymbol{x}\right\|_2^2\right)$

$$\nabla^2 f\left(x\right) - LI \text{ as semi-definite matrix}$$

We have the following inequality

$$\begin{aligned} f\left(\boldsymbol{y}\right) &= f\left(\boldsymbol{x}\right) + \nabla f\left(\boldsymbol{x}\right)^T\left(\boldsymbol{y}-\boldsymbol{x}\right) + \frac{1}{2}\nabla^2 f\left(\boldsymbol{x}\right)\left\|\boldsymbol{y}-\boldsymbol{x}\right\|^2 \\ &\leq f\left(\boldsymbol{x}\right) + \nabla f\left(\boldsymbol{x}\right)^T\left(\boldsymbol{y}-\boldsymbol{x}\right) + \frac{1}{2}L\left\|\boldsymbol{y}-\boldsymbol{x}\right\|^2 \end{aligned}$$

Proof

Now, if we apply the Gradient update $\boldsymbol{y} = \boldsymbol{x}^{+} = \boldsymbol{x} - \eta \nabla f(\boldsymbol{x})$

$$\begin{aligned} f\left(\boldsymbol{x}^{+}\right) &\leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{T}\left(\boldsymbol{x}^{+} - \boldsymbol{x}\right) + \frac{1}{2} L \left\|\boldsymbol{x}^{+} - \boldsymbol{x}\right\|^{2} \\ &= f(\boldsymbol{x}) - \eta \left\|\nabla f(\boldsymbol{x})\right\|^{2} + \frac{1}{2} L \eta^{2} \left\|\nabla f(\boldsymbol{x})\right\|^{2} \\ &= f(\boldsymbol{x}) - \left(1 - \frac{1}{2} L \eta\right) \eta \left\|\nabla f(\boldsymbol{x})\right\|^{2} \end{aligned}$$

Using $\eta \leq \frac{1}{L}$

$$-\left(1 - \frac{1}{2} L \eta\right) \leq -\frac{1}{2}$$

Therefore

We have that

$$f\left(\boldsymbol{x}^{+}\right) \leq f(\boldsymbol{x}) - \frac{1}{2}\eta \left\|\nabla f(\boldsymbol{x})\right\|^{2} \quad (2)$$

Implying that

- This inequality implies that the objective function value strictly decreases until it reaches the optimal value

This only holds when η is small enough

- This explains why we observe in practice that gradient descent diverges when the step size is too large.

Since f is convex

We can write

$$f\left(\boldsymbol{x}^*\right) \geq f\left(\boldsymbol{x}\right) + \nabla f\left(\boldsymbol{x}\right)^T\left(\boldsymbol{x}^* - \boldsymbol{x}\right)$$
$$f\left(\boldsymbol{x}\right) \leq f\left(\boldsymbol{x}^*\right) + \nabla f\left(\boldsymbol{x}\right)^T\left(\boldsymbol{x} - \boldsymbol{x}^*\right)$$

This comes from the “First order condition for convexity”

$$f\left(\boldsymbol{y}\right) \geq f\left(\boldsymbol{x}\right) + \nabla f\left(\boldsymbol{x}\right)^T\left(\boldsymbol{y} - \boldsymbol{x}\right)$$

Then

Plugging this in to (Equation 2)

$$f\left(\boldsymbol{x}^{+}\right) \leq f\left(\boldsymbol{x}^{*}\right) + \nabla f\left(\boldsymbol{x}\right)^{T}\left(\boldsymbol{x} - \boldsymbol{x}^{*}\right) - \frac{1}{2}\eta\left\|\nabla f\left(\boldsymbol{x}\right)\right\|^{2}$$

Therefore

$$f\left(\boldsymbol{x}^{+}\right) - f\left(\boldsymbol{x}^{*}\right) \leq \frac{1}{2\eta}\left[\left\|\boldsymbol{x} - \boldsymbol{x}^{*}\right\|^{2} - \left\|\boldsymbol{x} - \eta\nabla f\left(\boldsymbol{x}\right) - \boldsymbol{x}^{*}\right\|^{2}\right]$$

Then plugging this $\boldsymbol{x}^{+} = \boldsymbol{x} - \eta\nabla f\left(\boldsymbol{x}\right)$ into

$$f\left(\boldsymbol{x}^{+}\right) - f\left(\boldsymbol{x}^{*}\right) \leq \frac{1}{2\eta}\left[\left\|\boldsymbol{x} - \boldsymbol{x}^{*}\right\|^{2} - \left\|\boldsymbol{x}^{+} - \boldsymbol{x}^{*}\right\|^{2}\right]$$

Then

Summing over all iterations and the telescopic sum in the right side

$$\sum_{i=1}^{n}\left[f\left(\boldsymbol{x}_i\right)-f\left(\boldsymbol{x}^*\right)\right] \leq \frac{1}{2\eta}\left[\left\|\boldsymbol{x}_0-\boldsymbol{x}^*\right\|^2\right]$$

Finally, using the fact that f decreasing on every iteration

$$f\left(\boldsymbol{x}_n\right)-f\left(\boldsymbol{x}^*\right) \leq \frac{1}{n}\sum_{i=1}^{n}\left[f\left(\boldsymbol{x}_i\right)-f\left(\boldsymbol{x}^*\right)\right] \leq \frac{1}{2\eta n}\left[\left\|\boldsymbol{x}_0-\boldsymbol{x}^*\right\|^2\right]$$

Therefore

It converges with rate

$$O\left(\frac{1}{n}\right)$$

Accelerating the Gradient Descent

It is possible to modify the Batch Gradient Descent

- In order to accelerate it several modifications have been proposed

Possible Methods

- Polyak's Momentum Method or Heavy-Ball Method (1964) [10]
- Nesterov's Proposal (1983) [11]
- Stochastic Gradient Descent (1951) [12]

Polyak's Momentum Method

Polyak's Step Size

- He Proposed that the step size could be modified to

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \alpha \nabla f(\boldsymbol{w}_n) + \mu (\boldsymbol{w}_n - \boldsymbol{w}_{n-1}) \text{ with } \mu \in [0,1], \alpha > 0$$

Basically, the method uses the previous gradient information through the step difference $(\boldsymbol{w}_n - \boldsymbol{w}_{n-1})$

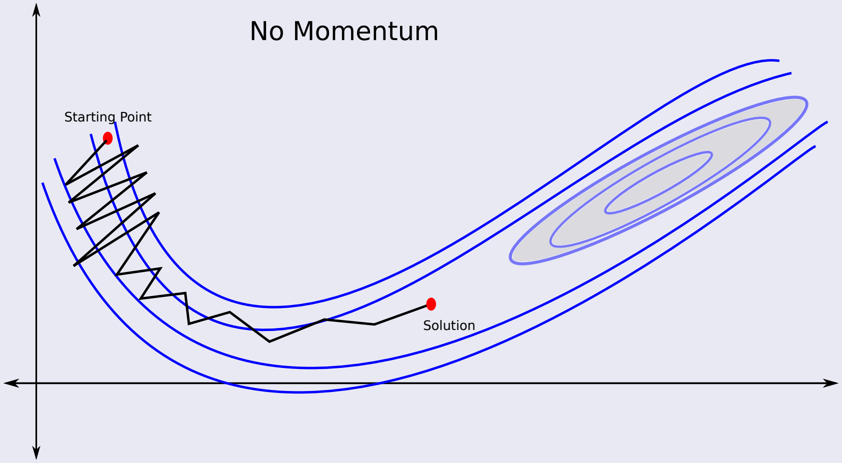
- By the discretization of the second order ODE

$$\ddot{\boldsymbol{w}} + a\dot{\boldsymbol{w}} + b\nabla f(\boldsymbol{w}) = 0$$

  - **which models the motion of a body in a potential field given by f with friction.**

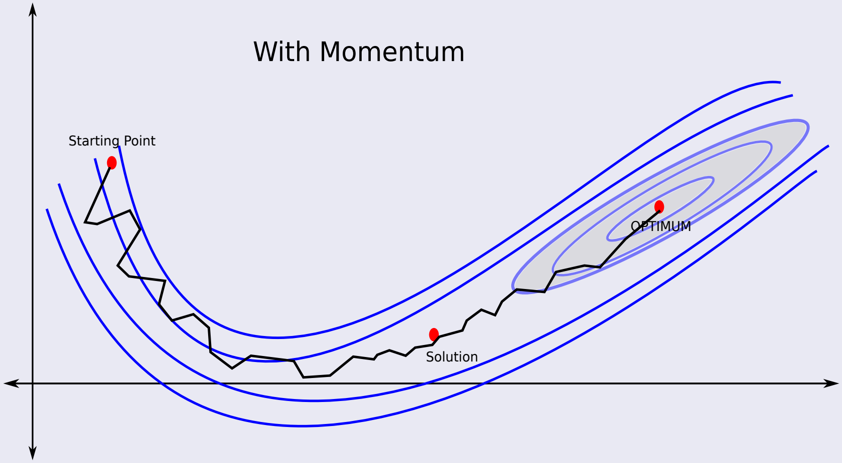
The Momentum helps to stabilize the GD

If we do not have Momentum



Then, with Momentum

If we have Momentum



Problem

It has been proved that the method has problems

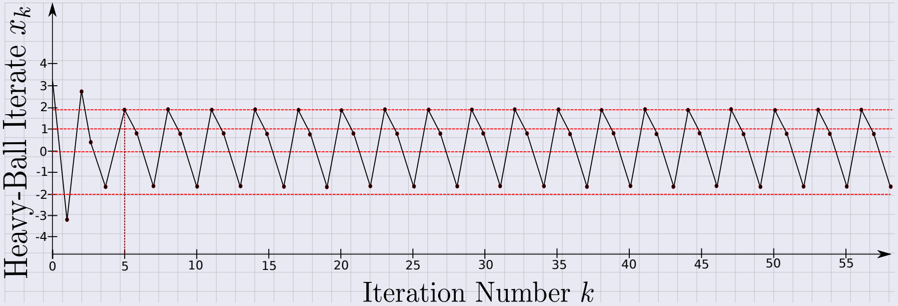
- L. Lessard, B. Recht, and A. Packard. Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints. ArXiv e-prints, Aug. 2014.

Under the function

$$\nabla f(x) = \begin{cases} 25x & \text{if } x < 1 \\ x + 24 & \text{if } 1 \leq x \leq 2 \\ 25x - 24 & \text{if otherwise} \end{cases}$$

In Lessard et al.

We have a non-convergence (Original Lessard et al.) [13]



Nesterov's Proposal to solve the issue

He proposed a Quasi-Convex Combination

- Instead to use

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \alpha \nabla f\left(\boldsymbol{w}_n\right) + \mu\left(\boldsymbol{w}_n - \boldsymbol{w}_{n-1}\right)$$

Have an intermediate step to update $\boldsymbol{w}_{n+1}$

$$\boldsymbol{w}_{n+1} = \left(1 - \gamma_n\right) \boldsymbol{y}_{n+1} + \gamma_n \boldsymbol{y}_n$$

This allow to weight the actual original gradient change

- with the previous gradient change... making possible to avoid the original problem by Polyak... Which is based in Lyapunov Analysis

Nesterov's Proposal [11]

Nesterov's Accelerated Gradient Descent (A Quasi-Convex Modification)

$$\begin{aligned}\boldsymbol{y}_{n+1} =& \boldsymbol{w}_n - \frac{1}{\beta}\nabla J\left(\boldsymbol{w}_n\right)\\ \boldsymbol{w}_{n+1} =& \left(1-\gamma_n\right)\boldsymbol{y}_{n+1} + \gamma_n \boldsymbol{y}_n\end{aligned}$$

Where, we use the following constants

$$\begin{aligned}\lambda_0 =& 0\\ \lambda_n =& \frac{1+\sqrt{1+4\lambda_{n-1}^2}}{2}\\ \gamma_n =& \frac{1-\lambda_n}{\lambda_{n+1}}\end{aligned}$$

Nesterov's Algorithm

Nesterov Accelerated Gradient

Input: Training Time T, Learning Rate β, an initialization $\boldsymbol{w}_0$

```
1  $y_0 \leftarrow \boldsymbol{w}_0$
2  $\lambda_0 \leftarrow 0$
3  for $t = 0$ to $T - 1$ do
4          $\boldsymbol{y}_{n+1} = \boldsymbol{w}_n - \frac{1}{\beta} \nabla J(\boldsymbol{w}_n)$
5          $\lambda_n = \frac{1+\sqrt{1+4\lambda_{n-1}^2}}{2}$
6          $\lambda_{n+1} = \frac{1+\sqrt{1+4\lambda_n^2}}{2}$
7          $\gamma_n = \frac{1-\lambda_n}{\lambda_{n+1}}$
8          $\boldsymbol{w}_{n+1} = (1-\gamma_n)\,\boldsymbol{y}_{n+1} + \gamma_n \boldsymbol{y}_n$
```

With the following complexity

Theorem (Nesterov 1983)

- Let f be a convex and β-smooth function (∇f is β-Lipschitz continous), then Nesterov's Accelerated Gradient Descent satisfies:

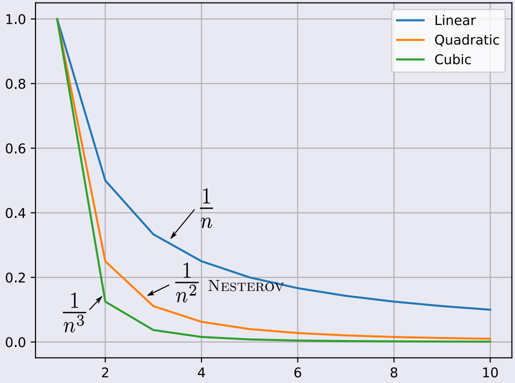
$$f\left(\boldsymbol{y}_{n+1}\right) - f\left(\boldsymbol{w}^{*}\right) \leq \frac{2\beta \left\| \boldsymbol{w}_1 - \boldsymbol{w}^{*} \right\|^2}{n^2}$$

It converges with rate

$$O\left(\frac{1}{n^2}\right)$$

Example

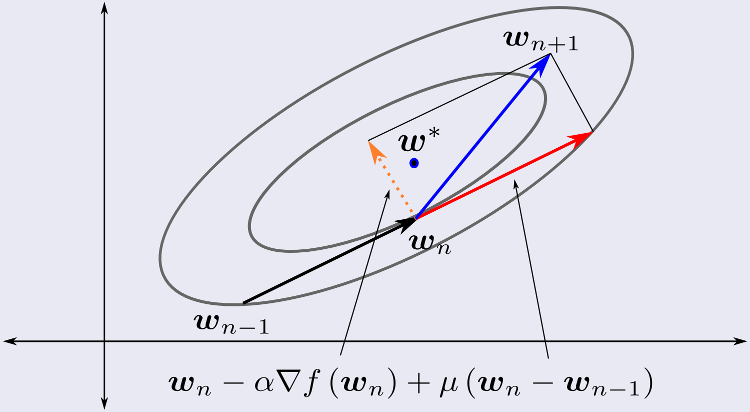
As you can see Nesterov is faster...



Remark, Polyak vs Nesterov

We have a remarkable difference

- The gradient descent step (orange arrow) is perpendicular to the level set before applying momentum to $\boldsymbol{w}_1$ (red arrow) in Polyak's algorithm



$\boldsymbol{w}_{n+1}$
$\boldsymbol{w}^*$
$\boldsymbol{w}_n$
$\boldsymbol{w}_{n-1}$
$\boldsymbol{w}_n - \alpha \nabla f(\boldsymbol{w}_n) + \mu(\boldsymbol{w}_n - \boldsymbol{w}_{n-1})$

In the case of Nesterov

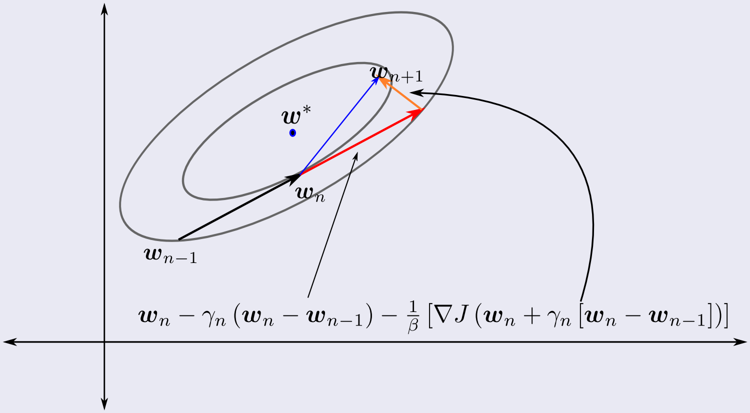
If we rewrite the equations

$$\begin{aligned}
\boldsymbol{w}_{n+1} =& (1-\gamma_n)\left[\boldsymbol{w}_n - \frac{1}{\beta}\nabla J(\boldsymbol{w}_n)\right] + \gamma_n \boldsymbol{y}_n \\
=& \boldsymbol{w}_n - \gamma_n \boldsymbol{w}_n - \frac{1}{\beta}\nabla J(\boldsymbol{w}_n) + \frac{\gamma_n}{\beta}\nabla J(\boldsymbol{w}_n) + \gamma_n \boldsymbol{w}_{n-1} - \frac{\gamma_n}{\beta}\nabla J(\boldsymbol{w}_{n-1}) \\
=& \boldsymbol{w}_n - \gamma_n(\boldsymbol{w}_n - \boldsymbol{w}_{n-1}) - \frac{1}{\beta}\left[\nabla J(\boldsymbol{w}_n) + \gamma_n \nabla J(\boldsymbol{w}_n) - \gamma_n \nabla J(\boldsymbol{w}_{n-1})\right] \\
=& \boldsymbol{w}_n - \gamma_n(\boldsymbol{w}_n - \boldsymbol{w}_{n-1}) - \frac{1}{\beta}\left[\nabla J(\boldsymbol{w}_n + \gamma_n[\boldsymbol{w}_n - \boldsymbol{w}_{n-1}])\right]
\end{aligned}$$

In Nesterov

We have a remarkable difference

- it is perpendicular to the level set after applying momentum to $\boldsymbol{w}_1$ in Nesterov's algorithm.



$\boldsymbol{w}_{n+1}$
$\boldsymbol{w}^*$
$\boldsymbol{w}_n$
$\boldsymbol{w}_{n-1}$
$\boldsymbol{w}_n - \gamma_n (\boldsymbol{w}_n - \boldsymbol{w}_{n-1}) - \frac{1}{\beta} [\nabla J (\boldsymbol{w}_n + \gamma_n [\boldsymbol{w}_n - \boldsymbol{w}_{n-1}])]$

Basically

Nesterov new Momemtum

- It tries to move towards the optimum because the dampening term $\gamma_n \left(\boldsymbol{w}_n - \boldsymbol{w}_{n-1}\right)$

Even with these attempts

- The Gradient Descent is highly dependent on the type of function you are trying to optimize

There is a dependence with respect with different properties of f

In this table, we can see upper bounds for the convergences $D = \|\boldsymbol{x}_1 - \boldsymbol{x}^*\|_2$ and λ regularization term [3]

Properties of the Objective Function	**Upper Bound for Gradient Descent**
convex and L-Lipschitz	$\frac{D_1 L}{\sqrt{n}}$
convex and β-smooth	$\frac{\beta D_1^2}{n}$
α-strongly convex and L-Lipschitz	$\frac{L^2}{\alpha n}$
α-strongly convex and β-smooth	$\beta D_1^2 \exp\left(-\frac{4n}{\beta/\lambda}\right)$

A Hierarchy can be established (Black Box Model)

Based on the following idea

- A black box model assumes that the algorithm does not know the objective function f being minimized.

Not only that

- Information about the objective function can only be accessed by querying an oracle.

Remarks

- The oracle serves as a bridge between the unknown objective function and the optimizer.

Furthermore

At any given step, the optimizer queries the oracle with a guess x

- The oracle responds with information about the function around x

For Example

- Value of the Cost function, Gradient, Hessian, etc.

Then, we have

Zeroth Order Methods [14, 15, 16]

- These methods only require the value of function f at the current guess $\boldsymbol{x}$.
  - The Bisection, Genetic Algorithms, Simulated Annealing and Metropolis-Hastings methods

First Order Methods

- These methods can inquire the value of the function f and its first derivative [5, 10, 11].
  - Gradient descent, Nesterov's and and Polyak's

Second Order Methods

- These methods require the value of the function f, its first derivative ∇f, and its second derivative $\nabla^2 f$ [5, 17, 3, 11].
  - Newton's method. Improving the efficiency of these algorithms is an active area of research.

One of the Last Hierarchy

Adaptive Methods and Conjugate Gradients

- The methods we mentioned until this point assume that all dimensions of a vector-valued variable have a common set of hyperparameters.

Adaptive methods relax this assumption

- They allow for every variable to have its own set of hyper-parameters.

Some popular methods under this paradigm

- AdaGrad, AdaDelta and ADAM

Finally, but not less important

Lower Bounds

- Lower bounds are useful because they tell us what's the best possible rate of convergence we can have given a category of optimizer.

Something Notable

- Without lower bounds, an unnecessary amount of research energy would be spent in designing better optimizers
  - Even if convergence rate improvement is impossible within this category of algorithm

However, if we prove that each procedure has a lower bounded rate of convergence

- We do not know if a specific method reaches this bound.

However

Please, take a look

- Convex Optimization: Algorithms and Complexity by Sébastien Bubeck - Theory Group, Microsoft Research [3]

In our classic Convex Scenario [7]

Least Square Problem locking to minimize the average of the LSE

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f\left(\boldsymbol{x}\right) = \min_{\boldsymbol{x}\in\mathbb{R}^d} \frac{1}{2M}\sum_{m=1}^{M}\left(\boldsymbol{w}^T\boldsymbol{x}_m - y_m\right)^2 = \min_{\boldsymbol{x}\in\mathbb{R}^d} \frac{1}{2M}\left\|X\boldsymbol{w} - Y\right\|^2$$

Therefore

Calculating the Gradient

$$\nabla_{\boldsymbol{w}} f\left(\boldsymbol{x}\right) = \frac{1}{M}\sum_{i=1}^{M}\left(\boldsymbol{w}^T\boldsymbol{x}_m - y_m\right)\boldsymbol{x}_m$$

Observations

It is easy to verify that the complexity per iteration is $O(dM)$

- With M is for the sum and d is for $\boldsymbol{w}^T \boldsymbol{x}_m$.

Drawbacks

When the number of samples M is Large

- Even with a rate of linear convergence, Gradient Descent

Not only that but in the On-line Learning scenario

- The data $(\boldsymbol{x}_i, y_i)$ is coming one by one making the gradient not computable.

Therefore

Thus, the need to look for something faster

- Two possibilities:
  - Accelerating Gradient Decent Using Stochastic Gradient Descent!!!
  - Accelerating Gradient Descent Using The Best of Both World, Min-Batch!!!

Using the Mean Squared Error (MSE)

It is used to measure how good our estimators are

- The average squared difference between the estimated values and what is estimated

We have the following equation

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \widehat{y}_i\right)^2 = E\left[\left(y - \widehat{y}\right)^2\right]$$

Then, we have that

This Measure is equal to (We know this as the Variance-Bias Trade-off)

$$MSE = \underbrace{Var_D\left(\widehat{y}|\boldsymbol{x} \in D\right)}_{Variance} + \underbrace{\left(E_D\left[\widehat{y} - y|\boldsymbol{x} \in D\right]\right)^2}_{BIAS}$$

If the MSE is small

- We expect that, on average, the resulting estimates to be close to the true value.

Furthermore

What will happen if we can decrease the Variance at MSE

- In such a way that the bias does not produce a too simplistic $\widehat{y}$?

Then, we want as the process MSE_t evolves over time

- $Var_D^{(t)}\left(\widehat{y}|\boldsymbol{x}\in D\right)\rightarrow V>0$ as $t\rightarrow\infty$ to avoid over-fitting
- $\left(E_D\left[\widehat{y}-y|\boldsymbol{x}\in D\right]\right)^2\rightarrow B>0$ as $t\rightarrow\infty$ to avoid over-fitting

Therefore, if we think in the parameters $\boldsymbol{w}$ of a Linear Model

We have a function

$$L\left(\boldsymbol{w}\right)=\left(E_D\left[\boldsymbol{w}^T x-y|\boldsymbol{x}\in D\right]\right)^2$$

We can see that the optimal $\boldsymbol{w}^*$ as the root of the function ∇L the minimal possible for L

$$\nabla_{\boldsymbol{w}} L\left(\boldsymbol{w}^*\right)=\nabla_{\boldsymbol{w}}\left(E_D\left[\boldsymbol{w}^{*T} x-y|\boldsymbol{x}\in D\right]\right)^2=0+\epsilon \text{ with } \epsilon\sim p\left(\epsilon|\theta\right)$$

- and ϵ is small enough

The MSE Linear Estimation, the Normal Equations

It was proved in slide set 2

- The optimal **Mean-Square Error estimate** of y given the value $X = \boldsymbol{x}$ is

$$E\left[y|\boldsymbol{x}\right] = \widehat{y}$$

  - In general, a nonlinear function.

For Linear Estimators, in $(\boldsymbol{x}, y) \in \mathbb{R}^d \times \mathbb{R}$ joint distributed random variables of zero mean values

- Our goal is to obtain an estimate of $\boldsymbol{w} \in \mathbb{R}^d$ (Our Unknown θ) in the linear estimator model

$$\widehat{y} = \boldsymbol{w}^T \boldsymbol{x}$$

Thus, using MSE as the Cost Equation

Cost Function

$$J\left(\boldsymbol{w}\right) = E\left[\left(y - \widehat{y}\right)^2\right]$$

Thus, we are looking for an estimator that minimize the variance of the error

$$\epsilon = y - \widehat{y}$$

We want to Minimize the cost function $J\left(\boldsymbol{w}\right)$ by finding an optimal $\boldsymbol{w}^*$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J\left(\boldsymbol{w}\right)$$

Then, we can simply use $\nabla J(\boldsymbol{w}) = 0$

We have

$$\begin{aligned}\nabla J(\boldsymbol{w}) =& \nabla E\left[\left(y-\boldsymbol{w}^T\boldsymbol{x}\right)^2\right]\\ =& \nabla E\left[\left(y-\boldsymbol{w}^T\boldsymbol{x}\right)\left(y-\boldsymbol{w}^T\boldsymbol{x}\right)\right]\\ =& \nabla\left\{E\left[y^2\right]-2\boldsymbol{w}^T E[\boldsymbol{x}y]+\boldsymbol{w}^T E\left[\boldsymbol{x}\boldsymbol{x}^T\right]\boldsymbol{w}^T\right\}\\ =& -2\boldsymbol{p}+2\Sigma_x\boldsymbol{w}=0\end{aligned}$$

Where, we have

$$\begin{aligned}\boldsymbol{p} =& \left[E[yx_1], E[yx_2], ..., E[yx_d]\right] = E[\boldsymbol{x}y]\\ \Sigma_x =& E\left[\boldsymbol{x}\boldsymbol{x}^T\right]\end{aligned}$$

This generates what is know as

Then, we get the Normal Equations

$$\Sigma_x \boldsymbol{w}^* = \boldsymbol{p}$$

We can use our gradient method[7]

Therefore, we have

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \mu\left[-\boldsymbol{p} + \Sigma_x \boldsymbol{w}_n\right]$$

Finally, we have that

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu\left[\boldsymbol{p} - \Sigma_x \boldsymbol{w}_{n-1}\right]$$

Then, the final idea is to find a μ

- Which allows for convergence!!!
- This is the first step in the idea of Stochastic Gradient Descent (SGD)
  - Given that SGD depends on specifics μ

How can we do this?

We can use our error to measure the convergence by μ

$$c_n = \boldsymbol{w}_n - \boldsymbol{w}^*$$

Thus, we obtain

$$\boldsymbol{w}_n - \boldsymbol{w}^* = \boldsymbol{w}_{n-1} + \mu \left[\boldsymbol{p} - \Sigma_x \boldsymbol{w}_{n-1}\right] - \boldsymbol{w}^*$$

Then

$$c_n = c_{n-1} + \mu \left[\boldsymbol{p} - \Sigma_x \left(c_{n-1} + \boldsymbol{w}^*\right)\right]$$

Therefore

Remembering $\Sigma_x \boldsymbol{w}^* = \boldsymbol{p}$

- We can try to guess the rate of convergence:

$$c_n = I c_{n-1} - \mu \left[\Sigma_x c_{n-1}\right] = \left[I - \mu \Sigma_x\right] c_{n-1}$$

Remember that

$$\Sigma_x = Q \Lambda Q^T \text{ with } Q Q^T = I$$

Then, we can build the following iterative process

We have

$$c_n = \left[QQ^T - \mu Q\Lambda Q^T\right] c_{n-1} = Q\left[I - \mu\Lambda\right] Q^T c_{n-1}$$

Finally, using $\boldsymbol{v}_n = Q^T c_n$

$$\boldsymbol{v}_n = \left[I - \mu\Lambda\right] \boldsymbol{v}_{n-1}$$

Iterating over all the sequence

We have by using recursion

$$\boldsymbol{v}\left(i\right)=\left[I-\mu\Lambda\right]^{i}\boldsymbol{v}\left(0\right)$$

Thus, for each component

$$\boldsymbol{v}_{ji}=\left(1-\mu\lambda_{j}\right)^{i}\boldsymbol{v}_{j0}$$

Now, we have that

$$\left|1-\mu\lambda_{j}\right|<1 \text{ for all } j=1,2,...,d$$

Or in an equivalent way

We have that

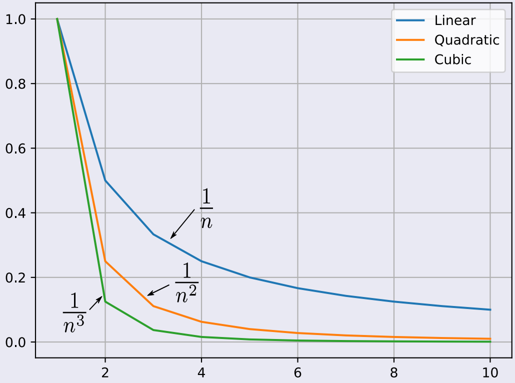
$$\begin{aligned}
-1 <& 1 - \mu\lambda_{max} < 1 \\
-1 <& -\mu\lambda_{max} < 0 \\
0 <& \mu\lambda_{max} < 2
\end{aligned}$$

Finally, we obtain a convergence condition

$$0 < \mu < \frac{2}{\lambda_{max}}$$

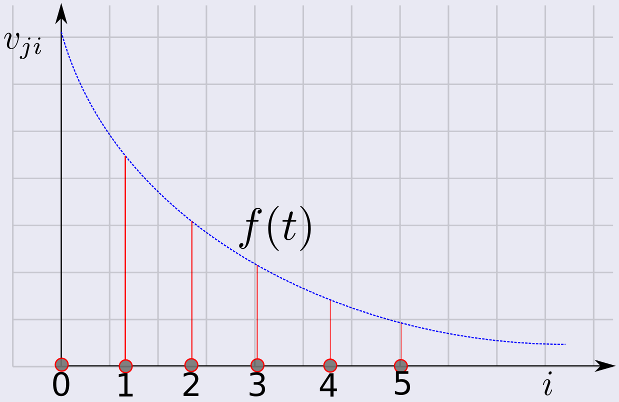
What about the Rate of Convergence?

As you can see the quadratic is faster than linear in convergence



What about the Rate of Convergence?

Assume an ideal case for the evolution of v_{ji} as it converges



Given the evolution of this curve, $f(t)$

Then, we can assume $f(t) = \exp\left\{-{}^{t}/_{\tau_j}\right\}$

- We can try to guess the rate of convergence τ_i.

Then we have $t = iT$ and $t = (i-1)T$

- Assuming a step size of T

Then, using $\boldsymbol{v}_{ji} = [1 - \mu\lambda_j]\,\boldsymbol{v}_{ji-1}$

$$\exp\left\{-{}^{iT}/_{\tau_j}\right\} = [1 - \mu\lambda_j] \exp\left\{-{}^{(i-1)T}/_{\tau_j}\right\}$$

Then, Solving the Equation

We have applying the function $\ln$

$$-\frac{iT}{\tau_j} = \ln\left[1 - \mu\lambda_j\right] - \frac{(i-1)\,T}{\tau_j}$$

Solving, we have

$$\tau_j = -\frac{1}{\ln\left(1 - \mu\lambda_j\right)}$$

The time constant results as

$$\tau_j \approx \frac{1}{\mu\lambda_j} \text{ for } \mu \ll 1$$

- **The slowest rate of convergence is associated with the component that corresponds to the smallest eigenvalue.**

However

However, this is only true for small enough values of μ

- Therefore, we need to consider something different

Therefore, we take two extreme vases

Let us consider as an example the case of μ taking a value

$$\mu \simeq \frac{2}{\lambda_{\max}}$$

The value of $|1-\mu\lambda_j|$ corresponding to the maximum eigenvalue

- It will have an absolute value very close to one.

$$|1-\mu\lambda_{\max}| = \left|1-\frac{2}{\lambda_{\max}}\lambda_{\max}\right| = 1$$

Now, we have

On the other hand, when using the minimum eigenvalue in the previous formula

$$|1-\mu\lambda_{\min}| = \left|1-\frac{2}{\lambda_{\max}}\lambda_{\min}\right| \ll 1$$

In such a case

- The maximum eigenvalue exhibits slower convergence.

The Optimal Value

We can use the following cost function

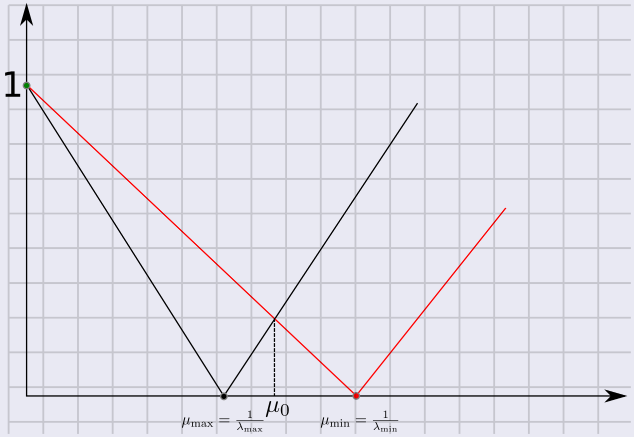
$$\mu_0 = \arg\min_{\mu}\max_{j}|1-\mu\lambda_j|$$
$$s.t.\ \ |1-\mu\lambda_j| < 1\ j = 1, 2, ..., d$$

This has the following solution

$$\mu_0 = \frac{2}{\lambda_{max}+\lambda_{min}}$$

Graphically

We have the following situation



The solution

This has the following solution

$$\mu_0 = \frac{2}{\lambda_{max} + \lambda_{min}}$$

Focusing on the mean-square error.

Adding and Subtracting $\boldsymbol{w}^{*T}\Sigma_x\boldsymbol{w}^*$ and taking the definition

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w}) \text{ and } \Sigma_x\boldsymbol{w}^* = \boldsymbol{p}$$

Therefore, we have

$$J(\boldsymbol{w}) = J(\boldsymbol{w}^*) + (\boldsymbol{w} - \boldsymbol{w}^*)^T \Sigma_x (\boldsymbol{w} - \boldsymbol{w}^*)$$

Where we have that at the optimal

It is possible to see that

$$J\left(\boldsymbol{w}^{*}\right)=\sigma_{y}^{2}-\boldsymbol{p}^{T}\Sigma_{x}^{-1}\boldsymbol{p}=\sigma_{y}^{2}-\boldsymbol{w}^{*T}\Sigma_{x}^{-1}\boldsymbol{w}^{*}=\sigma_{y}^{2}-\boldsymbol{p}\boldsymbol{w}^{*}$$

- The minimum at the optimal solution!!!

Taking the orthonormality of the eigenvectors

Taking in account that Σ_x is a diagonal matrix

$$J(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \sum_{j=1}^{d} \lambda_j |v_{ji}|^2$$

Therefore, we have

$$J\left(\boldsymbol{w}\right) = J\left(\boldsymbol{w}^{*}\right) + \sum_{j=1}^{d} \lambda_j \left(1 - \mu\lambda_j\right)^{2i} \left|v_{j0}\right|^{2}$$

Convergence

This converges to the minimum value $J\left(\boldsymbol{w}^{*}\right)$ asymptotically

- This convergence is monotonic, because $\lambda_j\left(1-\mu\lambda_j\right)^2$ is positive.

The rates of convergence are finally

$$\tau_j = \frac{-1}{2\ln\left(1-\mu\lambda_j\right)} \approx \frac{1}{2\mu\lambda_j}$$

We have a problem

In general, once we go away of the fix size step

- After all a fix size depends on having Convex Functions.

In general functions as

$$f_\theta = W^L \phi\left[W^{L-1} \cdots \phi\left[W^2 \phi\left[W^1 x_i + b_1\right] + b_2\right] + b_{L-1}\right] + b_L$$

But we have a card in the sleeve

- Stochastic Gradient Descent

There, using Gradient Descent Convergence

We know Gradient Descent Algorithm convergences if

$$\mu_i \to 0, \text{ as } i \to \infty$$
$$\sum_{i=1}^{\infty} \mu_i = \infty$$

This can be rephrased for Stochastic Gradient Descent as

$$\sum_{i=1}^{\infty} \mu_i^2 < \infty \text{ and } \sum_{i=1}^{\infty} \mu_i = \infty$$

As for example, if we choose

$$\mu_i = \frac{1}{i}$$

Solving for the normal equations as well as using the gradient descent

There is a small problem

- You are required to have access to the analytical model.

Additionally

- You need to have access to the second order statistics of the involved variables
  - The Covariance Matrix Σ_x

$$\Sigma_x \boldsymbol{w}^* = \boldsymbol{p}$$

Furthermore

We have a problem

- This is not known and it has to be approximated using a set of measurements.

But, we can solve the problem

- By using stochastic methods resembling Monte Carlo ideas!!!

We have that the Robbins-Monro Theorem[12]

The origins of such techniques are traced back to 1951

- When Robbins and Monro introduced the method of stochastic approximation
    - DARPA project!!!

Setup, given a function $M(\boldsymbol{w})$ and a constant α such that the equation

$$M(\boldsymbol{w}) = \alpha$$

- It has a unique root $\boldsymbol{w} = \boldsymbol{w}^*$

Goal

We want to compute the root, $\boldsymbol{w}$, of such equation

$$M\left(\boldsymbol{w}^{*}\right)=\alpha$$

Then, we want to generate values $\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_{n-1}$ thus, we generate $\boldsymbol{w}_n$ from

1. $M\left(\boldsymbol{w}_1\right), M\left(\boldsymbol{w}_2\right), ..., M\left(\boldsymbol{w}_{n-1}\right)$
2. and the possible derivatives $M'\left(\boldsymbol{w}_1\right), M'\left(\boldsymbol{w}_2\right), ..., M'\left(\boldsymbol{w}_{n-1}\right)$

Thus, we would love that

$$\lim_{n\to\infty}\boldsymbol{w}_n=\boldsymbol{w}^{*}$$

Instead, we suppose that for each $\boldsymbol{w}$ corresponds a Random Variable $Y = Y(\boldsymbol{w})$

This Random Variable has a distribution function

$$Pr\left[Y(\boldsymbol{w}) \leq y\right] = H(y|\boldsymbol{w})$$

Such that

$$M(\boldsymbol{w}) = \int_{-\infty}^{\infty} y dH(y|\boldsymbol{w})$$

We Postulate

First a bound to the $M(\boldsymbol{w})$

$$|M(\boldsymbol{w})| \leq C < \infty, \ \int_{-\infty}^{\infty} (y - M(\boldsymbol{w}))^2 \, dH(y|\boldsymbol{w}) \leq \sigma^2 < \infty$$

IMPORTANT

Neither the exact nature of $H\left(y|\boldsymbol{w}\right)$ nor that of $M\left(\boldsymbol{w}\right)$ is known

- But an important assumption is that

$$M\left(\boldsymbol{w}\right)-\alpha=0$$

It has only one root

Here is we use the α value to generate the root by assuming

- $M\left(\boldsymbol{w}\right)-\alpha\leq 0$ for $\boldsymbol{w}\leq\boldsymbol{w}^{*}$ and $M\left(\boldsymbol{w}\right)-\alpha\geq 0$ for $\boldsymbol{w}>\boldsymbol{w}^{*}$.

Now, For a positive δ

$M(\boldsymbol{w})$ is strictly increasing if

$$\|\boldsymbol{w}^* - \boldsymbol{w}\| < \delta$$

And Finally

$$\inf_{\|\boldsymbol{w}^* - \boldsymbol{w}\| \geq \delta} |M(\boldsymbol{w}) - \alpha| > 0$$

Now choose a sequence $\{\mu_i\}$

Such that

$$\sum_{i=1}^{\infty} \mu_i^2 = A < \infty \text{ and } \sum_{i=1}^{\infty} \mu_i = \infty$$

Now, we define a non-stationary Markov Chain $\{\boldsymbol{w}_n\}$

$$\boldsymbol{w}_{n+1} - \boldsymbol{w}_n = \mu_n \left(\alpha - y_n\right)$$

Where y_n is a random variable such that

$$Pr\left[y_n \leq y | \boldsymbol{w}_n\right] = H\left(y | \boldsymbol{w}_n\right)$$

Using the expected value!!!

Here, we define b_n

$$b_n = E\left[\boldsymbol{w}_n - \boldsymbol{w}^*\right]^2$$

We want conditions where this variance goes to zero

$$\lim_{n\to\infty} b_n = 0$$

- No matter what is the initial value $\boldsymbol{w}_0$.

We have then

Based on

$$\boldsymbol{w}_{n+1} - \boldsymbol{w}_n = \mu_n \left(\alpha - y_n\right)$$

We have then

$$\begin{aligned}
b_{n+1} =& E\left[\boldsymbol{w}_{n+1} - \boldsymbol{w}^*\right]^2 = E\left[E\left[\boldsymbol{w}_{n+1} - \boldsymbol{w}^*\right]^2 | \boldsymbol{w}_n\right] \\
=& E\left[\int_{-\infty}^{\infty} \left[\boldsymbol{w}_n - \boldsymbol{w}^* - \mu_n \left(y - \alpha\right)^2\right] dH\left(y|\boldsymbol{w}_n\right)\right] \\
=& b_n + \mu_n E\left[\int_{-\infty}^{\infty} \left(y - \alpha\right)^2 dH\left(y|\boldsymbol{w}_n\right)\right] - 2\mu_n E\left[\left(\boldsymbol{w}_n - \boldsymbol{w}^*\right)\left(M\left(\boldsymbol{w}_n\right) - \alpha\right)\right] \\
=& b_n + \mu_n^2 e_n - 2\mu_n d_n
\end{aligned}$$

With Values

We have

$$d_n = E\left[\left(\boldsymbol{w}_n - \boldsymbol{w}^*\right)\left(M\left(\boldsymbol{w}_n\right) - \alpha\right)\right]$$
$$e_n = E\left[\int_{-\infty}^{\infty} \left(y - \alpha\right)^2 dH\left(y|\boldsymbol{w}_n\right)\right]$$

From $M\left(\boldsymbol{w}\right) \leq \alpha$ for $\boldsymbol{w} \leq \boldsymbol{w}^*$ and $M\left(\boldsymbol{w}\right) \geq \alpha$ for $\boldsymbol{w} > \boldsymbol{w}^*$

$$d_n \geq 0$$

Additionally

Now, assuming that exist C such that

$$Pr\left[\left|Y\left(\boldsymbol{w}\right)\right|\leq C\right]=\int_{-C}^{C}dH\left(y|\boldsymbol{w}\right)=1\ \forall x$$

We can prove that

$$0\leq e_n\leq\left[C+\left|\alpha\right|^2\right]<\infty$$

Now, given

$$\sum_{i=1}^{\infty}\mu_i^2=A<\infty \text{ and } \textstyle\sum_{i=1}^{\infty}\mu_i{=}\infty$$

Therefore $\sum_{i=1}^{\infty} \mu_i^2 e_i$ converges

Then, summing over i we obtain

$$b_{n+1} = b_1 + \sum_{i=1}^{n} \mu_i^2 e_i - 2 \sum_{i=1}^{n} \mu_i d_i$$

Since $b_{n+1} \geq 0$

$$\sum_{i=1}^{n} \mu_i d_i \leq \frac{1}{2} \left[b_1 + \sum_{i=1}^{n} \mu_i^2 e_i \right] < \infty$$

Then

Hence the positive-term series

$$\sum_{i=1}^{\infty} \mu_i d_i \text{ converges}$$

Then, $\lim_{n\to\infty} b_n$ exists and...

$$\lim_{n\to\infty} b_n = b_1 + \sum_{i=1}^{\infty} \mu_i^2 e_i - 2\sum_{i=1}^{\infty} \mu_i d_i = b$$

Therefore

If a sequence of $\{k_i\}$ of non-negative constants such that

$$d_i \geq k_i b_i, \ \sum_{i=1}^{\infty} \mu_i k_i = \infty$$

We want to prove that

$$\sum_{i=1}^{\infty} \mu_i k_i b_i < \infty$$

For this

We know that

$$\sum_{i=1}^{\infty} \mu_i d_i \text{ converges}$$

Therefore

$$k_i b_i \leq d_i \Rightarrow \mu_i k_i b_i \leq \mu_i d_i$$

Then

We have that

$$\sum_{i=1}^{\infty} \mu_i k_i b_i \leq \sum_{i=1}^{\infty} \mu_i d_i < \infty$$

Then, we have that

$$\sum_{i=1}^{\infty} \mu_i k_i b_i < \infty, \ \textstyle\sum_{i=1}^{\infty} \mu_i k_i = \infty$$

Finally

For any $\epsilon > 0$ there must be infinitely values i such that $b_i < \epsilon$

- Therefore given that $\lim_{n\to\infty} b_n = b$ then $b = 0$.

Robbins and Monro Theorem (Original)

If $\{\mu_n\}$ is of type $\frac{1}{n}$

- Given a family of conditional probabilities

$$\{H(y|\boldsymbol{w}) = Pr(Y(\boldsymbol{w}) \leq y|\boldsymbol{w})\}$$

We have the following Expected Risk

$$M(\boldsymbol{w}) = \int_{-\infty}^{\infty} y dH(y|\boldsymbol{w})$$

Now

If we additionally have that

$$Pr\left(|Y\left(\boldsymbol{w}\right)| \leq C\right) = \int_{-C}^{C} dH\left(y|\boldsymbol{w}\right) = 1 \tag{3}$$

Then under the following constraints

For some $\delta > 0$

$$\begin{aligned} M(\boldsymbol{w}) &\leq \alpha - \delta \text{ for } \boldsymbol{w} < \boldsymbol{w}^* \\ M(\boldsymbol{w}) &\geq \alpha + \delta \text{ for } \boldsymbol{w} > \boldsymbol{w}^* \end{aligned} \tag{4}$$

Or Else

$$\begin{aligned} M(\boldsymbol{w}) &< \alpha \text{ for } \boldsymbol{w} < \boldsymbol{w}^* \\ M(\boldsymbol{w}^*) &= \alpha \\ M(\boldsymbol{w}) &> \alpha \text{ for } \boldsymbol{w} > \boldsymbol{w}^* \end{aligned} \tag{5}$$

Next

Furthermore

$$M\left(\boldsymbol{w}\right) \text{ is strictily increasing if } \left|\boldsymbol{w}-\boldsymbol{w}^{*}\right|<\delta \tag{6}$$

And

$$\inf_{\left|\boldsymbol{w}-\boldsymbol{w}^{*}\right|\geq\delta}\left|M\left(\boldsymbol{w}\right)-\alpha\right|>0 \tag{7}$$

And Let $\{\mu_i\}$ be a sequence of positive numbers such that

$$\sum_{n=1}^{\infty}\mu_n=\infty \text{ and } \sum_{n=1}^{\infty}\mu_n^2<\infty \tag{8}$$

Then

Let x_1 an arbitrary number, then under the recursion

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \mu_n \left(\alpha - y_n\right)$$

- Where $y_n \sim P\left(y|\boldsymbol{w}_n\right)$

Theorem

- If (3) and (8), either (4) or (5,6,7) hold, then $\boldsymbol{w}_n$ converges stochastically to $\boldsymbol{w}^*$ given that $b = 0$.

Recap of Robbins-Monro Proposal

Given the following function

$$f\left(\boldsymbol{w}\right) = E\left[\phi\left(\boldsymbol{w}, \eta\right)\right], \ \boldsymbol{w} \in \mathbb{R}^{d+1}$$

Given a series of i.i.d. observations $x_0, x_1, \cdots$

- The following iterative procedure (Robbins-Monro Scheme)

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} - \mu_n \phi\left(\boldsymbol{w}_{n-1}, \boldsymbol{x}_n\right)$$

Robbins-Monro Proposal

Starting from an arbitrary initial condition, $\boldsymbol{w}_0$

- It converges to a root of $M\left(\boldsymbol{w}\right) = \alpha$

Under some general conditions about the step size

$$\sum_{i=0}^{\infty} \mu_i^2 < \infty$$

$$\sum_{i=0}^{\infty} \mu_i \to \infty$$

Mean-Square Error [7]

Cost function for MSE

$$J(\boldsymbol{w}) = E\left[\mathcal{L}(\boldsymbol{w}, \boldsymbol{x}, y)\right]$$

- Also known as the expected risk or the expected loss.

Then, our objective is the reduction of the Expected Risk!!!

- Thus, the simple thing to do is to derive the function and make such gradient equal to zero.

Therefore

We can get the Gradient of the Expected Cost Function

$$\nabla J(\boldsymbol{w}) = E\left[\nabla \mathcal{L}(\boldsymbol{w}, \boldsymbol{x}, y)\right]$$

- where the expectation is w.r.t. the pair $(\boldsymbol{x}, y)$

Therefore, everything depends on the form of the Loss function

$$\mathcal{L}_1(\boldsymbol{w}, \boldsymbol{x}, y) = \frac{1}{2}\left\|\boldsymbol{w}^T\boldsymbol{x} - y\right\|_2^2 \text{ (Least Squared Loss)}$$

$$\mathcal{L}_2(\boldsymbol{w}, \boldsymbol{x}, y) = \left[\frac{1}{1+\exp\left\{\boldsymbol{w}^T\boldsymbol{x}\right\}}\right]^{1-y}\left[\frac{\exp\left\{\boldsymbol{w}^T\boldsymbol{x}\right\}}{1+\exp\left\{\boldsymbol{w}^T\boldsymbol{x}\right\}}\right]^{y} \text{ (Logistic Loss)}$$

$$\mathcal{L}_3(\boldsymbol{w}, \boldsymbol{x}, y) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\log\left(y_{nk}^{(l)}\right) \text{ (Cross-Entropy Loss)}$$

Therefore

We simply take $\alpha = 0$ then

$$\nabla J(\boldsymbol{w}) = E\left[\nabla \mathcal{L}(\boldsymbol{w}, \boldsymbol{x}, y)\right] = 0$$

Then, we apply the Robbins-Monroe Schema to the function

$$f(\boldsymbol{w}) = \nabla J(\boldsymbol{w}) = 0$$

Then

Given the sequence of observations $\{(\boldsymbol{x}_i, y_i)\}_{i=1,2,\ldots}$ and values $\{\mu_i\}_{i=1,2,\ldots}$

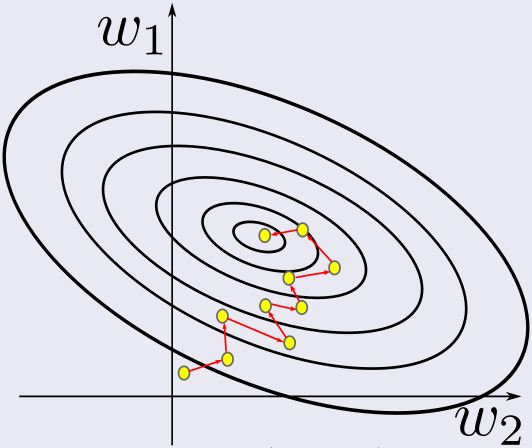
- We have that the iterative procedure becomes:

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} - \mu_n \nabla \mathcal{L}\left(\boldsymbol{w}_n, \boldsymbol{x}_n, y_n\right)$$

  - The Well known Vanilla Stochastic Gradient Descent (SGD)

Geometrically

We have the following



$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} - \mu_n \nabla \mathcal{L}\left(\boldsymbol{w}_n, \boldsymbol{x}_n, y_n\right)$$

Therefore

However, although the theorem is important

- it is not by itself enough.

One has to know something more concerning

- The rate of convergence of such a scheme.

It has been shown that

$$\mu_n = O\left(\frac{1}{n}\right)$$

Additionally

Assuming that iterations have brought the estimate close to the optimal value

$$E(\boldsymbol{w}_n) = \boldsymbol{w}^* + \frac{1}{n}\boldsymbol{c}$$

And

$$Cov(\boldsymbol{w}_n) = \frac{1}{n}V + O\left(\frac{1}{n^2}\right)$$

- Where $\boldsymbol{c}$ and V are constants that depend on the form of the expected risk.

Meaning

Therefore

- These formulas indicate that the parameter vector estimate fluctuates around the optimal value.

However

- Low complexity requirements makes this algorithmic family to be the one that is selected in a number of practical applications.
    - Given the problem with Batch Gradient Descent (BGD)

Remarks Stochastic Gradient Descent

It has become the corner stone

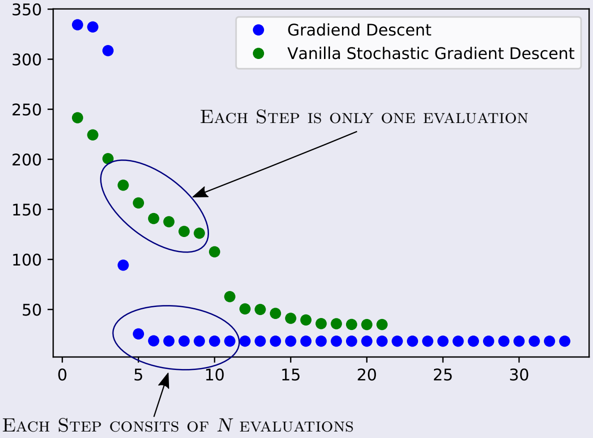
- For the development of new methods of Optimization

Something

Properties

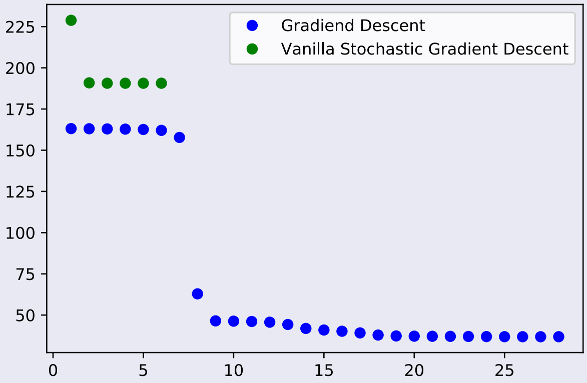
Example of SGD for, $\frac{1}{2}\sum_{i=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{x}-\boldsymbol{y}\right)^2$

We can see how from the Vanilla SGD improves over the Batch GD with respect to Speed of Evaluation



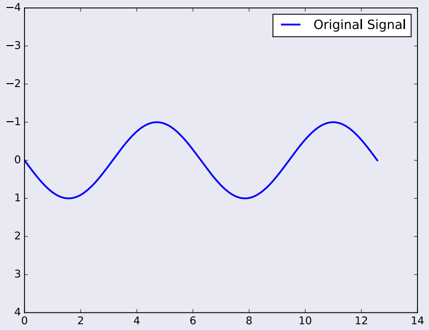
Problems

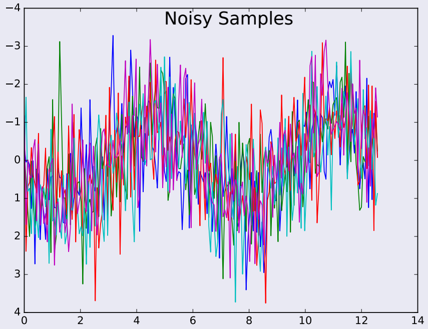
However, we need to improve such Vanilla Stochastic Gradient Descent



Do you Remember?

Imagine the following signal from $\sin(\theta)$





What if we know the noise?

Given a series of observed samples $\{\widehat{\boldsymbol{x}}_1, \widehat{\boldsymbol{x}}_2, ..., \widehat{\boldsymbol{x}}_N\}$ with noise $\epsilon \sim N(0,1)$

We could use our knowledge on the noise, for example additive:

$$\widehat{\boldsymbol{x}}_i = \boldsymbol{x}_i + \epsilon$$

We can use our knowledge of probability to remove such noise

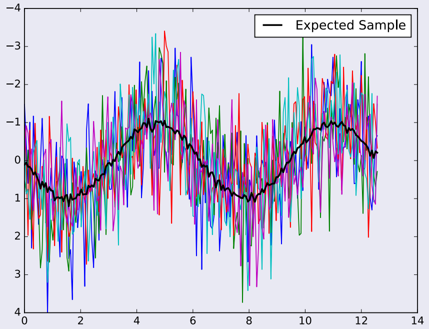
$$E\left[\widehat{\boldsymbol{x}}_i\right] = E\left[\boldsymbol{x}_i + \epsilon\right] = E\left[\boldsymbol{x}_i\right] + E\left[\epsilon\right]$$

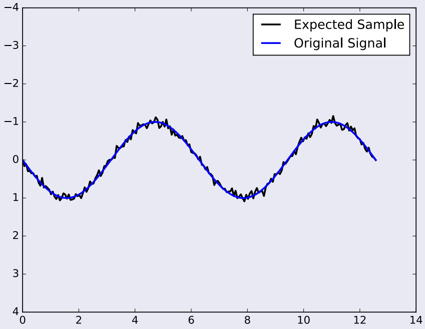
Then, because $E\left[\epsilon\right] = 0$

$$E\left[\boldsymbol{x}_i\right] = E\left[\widehat{\boldsymbol{x}}_i\right] \approx \frac{1}{N}\sum_{i=1}^{N} \widehat{\boldsymbol{x}}_i$$

In our example

We have a nice result





Thus

Using a similar idea, you could use an average [18]

$$\nabla J\left(\boldsymbol{w}_{k-1}\middle|\boldsymbol{x}_{i:i+m}, y_{i:i+m}\right) = \ldots$$
$$\frac{1}{m}\sum_{i=1}^{m}\nabla J\left(\boldsymbol{w}_{k-1}, \boldsymbol{x}_i, y_i\right)$$

This allows to reduce the variance of the original Stochastic Gradient

- It reduces the variance of the parameter updates, which can lead to more stable convergence.
- It can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

There are other more efficient options

We can update the $\boldsymbol{w}(k)$

- By Batches per epoch...

Therefore

1. for i in batch k

$$\boldsymbol{w}_k = \boldsymbol{w}_{k-1} - \alpha \nabla J\left(\boldsymbol{w}_{k-1}, \boldsymbol{x}_i, y_i\right)$$

Mini-batch gradient descent finally takes the best of both worlds

Min-Batch(X)

Input:

- Initialize $\boldsymbol{w}_0$, Set number of epochs, L, Set learning rate α

```
1 for $k = 1$ to $L$:
2     Randomly pick a mini batch of size $m$.
3     for $i = 1$ to $m$ do:
4         Evaluate $g(k) = \nabla J(\boldsymbol{w}_{k-1}, \boldsymbol{x}_i, y_i)$
5         $\boldsymbol{w}_k = \boldsymbol{w}_{k-1} - \alpha g(k)$
```

Notes

Remark, for $\alpha = \frac{1}{m}$, the method is equivalent to average sample way

$$\begin{aligned}\boldsymbol{w}_k =& \boldsymbol{w}_{k-1} - \alpha \nabla J\left(\boldsymbol{w}_{k-1}, \boldsymbol{x}_i, y_i\right) - ... \\ & \alpha \nabla J\left(\boldsymbol{w}_{k-1}, \boldsymbol{x}_{i+1}, y_{i+1}\right) - ... \\ & \alpha \nabla J\left(\boldsymbol{w}_{k-1}, \boldsymbol{x}_{i+m}, y_{i+m}\right) \\ =& \boldsymbol{w}_{k-1} - \frac{1}{m} \sum_{i=1}^{m} \nabla J\left(\boldsymbol{w}_{k-1}, \boldsymbol{x}_i, y_i\right)\end{aligned}$$

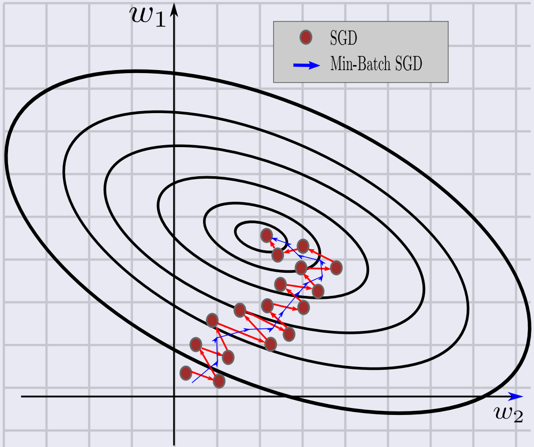
Notes

We have the following

- Common mini-batch sizes range between 50 and 256, but can vary for different applications.
- Mini-batch gradient descent is typically the algorithm of choice when training a neural network.

A Small Intuition

We have smoother version of the Stochastic Gradient Descent



$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} - \mu_n \nabla \mathcal{L}\left(\boldsymbol{w}_n, \boldsymbol{x}_n, y_n\right)$$

Drawbacks

Choosing a proper learning rate can be difficult

- A learning rate that is too small leads to painfully slow convergence,
- Too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

Learning Rate Schedules

- To adjust the learning rate during training by e.g. annealing
- These schedules and thresholds, however, have to be defined in advance not on-line

Another key challenge of minimizing highly non-convex error functions

- For example, neural networks, it is avoiding getting trapped in their numerous suboptimal local minima.

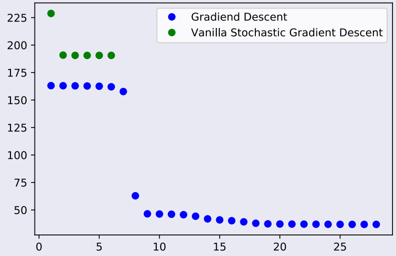
Observations

Using Traditional Methods used in Gradient Descent

- Golden Ratio
- Bisection Method
- etc

Nevertheless

- Experiments with the Bisection Method has produced not so great results!!!



Adaptive Rate Speeds in SGD [19]

Structure of SGD with an adaptive learning rate

$$\begin{aligned} \boldsymbol{w}(t+1) &= \boldsymbol{w}(t) - \eta(t)\, g(t) \\ \eta(t) &= h(t) \end{aligned}$$

Where

- $g(t) = \nabla L(\boldsymbol{w}(t))$
- $h(t)$ is a continuous function

First Order Methods

Gradient descent on the learning rate

- Introducing the following function:

$$\begin{aligned} f : &\mathbb{R}^n \to \mathbb{R} \\ &\eta \to L\left(\boldsymbol{w}\left(t\right) - \eta g\left(t\right)\right) \end{aligned}$$

This comes a simple intuition

- At time t using $\eta\left(t\right)$, we suffer a loss of $L\left(\boldsymbol{w}\left(t\right) - \eta g\left(t\right)\right)$ in the next iteration:
  - So f represents such loss in the future if we choose $\boldsymbol{w}\left(t+1\right) = \boldsymbol{w}\left(t\right) - \eta g\left(t\right)$

Therefore

The first-order method is written as

$$
\begin{aligned}
\boldsymbol{w}(t) &= \boldsymbol{w}(t) - \eta(t)\, g(t) \\
\eta(t+1) &= \eta(t) - \alpha f'(\eta(t))
\end{aligned}
$$

Remark

- This method introduces a new "meta" learning rate α.

The final $f'(\eta(t))$

We have that $\forall \eta$

$$f'(\eta) = -g(t)^T \cdot \nabla L(\boldsymbol{w}(t) - \eta g(t))$$

We can rewrite this as

$$f'(\eta) = -g(t)^T \cdot g(t+1)$$

Intuition

If we continue in a similar direction

- We increase the learning rate, if we backtrack then we decrease it.

However

- The algorithm is not scale invariant anymore:

$$\text{Different scales } L'(\boldsymbol{w}) = \lambda L(\boldsymbol{w}) \text{ different results}$$

Second Order Methods

Remark

- The previous method presents the problem of choosing another meta-learning rate for optimizing the actual learning rate.

In order to avoid such problems

- We can use a second-order Newton-Raphson optimization method

$$\boldsymbol{w}(t) = \boldsymbol{w}(t) - \eta(t)\, g(t)$$
$$\eta(t+1) = \eta(t) - \frac{f'(\eta(t))}{f''(\eta(t))}$$

We get rid of the meta or hyper-parameter α

- However, the second derivative of f requires building the loss Hessian matrix

Hessian Matrix

We have

$$f''(\eta) = -g(t)^T H_L(\boldsymbol{w}(t) - \eta g(t))$$

Here, we can use an approximation

- "Deep learning via hessian-free optimization" by James Martens
  - They are actually know as finite Calculus ("Calculus of Finite Differences" by Charles Jordan)

$$f'(\eta+\epsilon) = \frac{f(\eta+2\epsilon) - f(\eta)}{2\epsilon} \text{ (Forward Difference)}$$
$$f'(\eta-\epsilon) = \frac{f(\eta) - f(\eta-2\epsilon)}{2\epsilon} \text{ (Backward Difference)}$$

Then

We have that

$$f''(\eta) = \frac{f(\eta + 2\epsilon) + f(\eta - 2\epsilon) - 2f(\eta)}{4\epsilon^2}$$

Now, using the previous differences, we have

$$f'(\eta) = \frac{f(\eta + \epsilon) - f(\eta - \epsilon)}{2\epsilon}$$

Finally

We have an approximation to the η hyper-parameter

$$\eta(t+1) = \eta(t) - 2\epsilon \frac{f(\eta+\epsilon) - f(\eta-\epsilon)}{f(\eta+2\epsilon) + f(\eta-2\epsilon) - 2f(\eta)}$$

Meaning

- When slightly increasing, the learning rate corresponds to a lower loss than slightly reducing it, then the numerator is negative.

In consequence

- The learning rate is raised at this update, as pushing in the ascending direction for the learning rate seems to help reducing the loss.

Some Considerations

As you have notice in the second order method, we can have an underflow

1. If $f\left(\eta + 2\epsilon\right) + f\left(\eta - 2\epsilon\right) - 2f\left(\eta\right) \approx 0$
2. $$\eta\left(t+1\right) = \eta\left(t\right) - 2\epsilon \frac{f(\eta+\epsilon) - f(\eta-\epsilon)}{f(\eta+2\epsilon) + f(\eta-2\epsilon) - 2f(\eta) + \delta^{-6}}$$

A typical value for δ is 10^{-6}

- Furthermore, the order of operations needs to be maintained...

At k Iteration,

we have a loss value $L^{(k)}$ and a learning rate value $\eta^{(k)}$

- At the $k+1$ step, we have the five loss values $f\left(\eta^{(k)}+\epsilon\right)$, $f\left(\eta^{(k)}-\epsilon\right)$, $f\left(\eta^{(k)}+2\epsilon\right)$, $f\left(\eta^{(k)}-2\epsilon\right)$ and $f\left(\eta^{(k)}\right)$
  - Actually five passes over the function f

Then, we calculate $L^{(k+1)}$ by

$$L^{(k+1)} \leftarrow f\left(\eta^{(k)}\right)$$

Then the $\eta\left(k+1\right)$ update

$$\eta\left(t+1\right)=\eta\left(t\right)-2\epsilon\frac{f\left(\eta+\epsilon\right)-f\left(\eta-\epsilon\right)}{f\left(\eta+2\epsilon\right)+f\left(\eta-2\epsilon\right)-2f\left(\eta\right)}$$

Final Remark

Something Notable

- First-order and second-order updates of the learning rate do not guarantee positive learning rates

A simple way to avoid this problem is to use

$$\eta\left(k+1\right)=\max\left\{\eta\left(t+1\right),\delta\right\}$$

- With an appropriate smoothing δ value.

Introduction

We have been able to accelerate the speed with SGD

- However, Is this enough?
  - After all, we are dealing with large data sets that are costly to train on them.

Therefore

- We introduce the concept of regret which is used in on-line learning...
  - After all SGD is a way of doing on-line learning!!!

What is regret?

- It measures how “sorry” the learning algorithm is, in retrospect, of not having followed the predictions of some hypothesis $h \in \mathcal{H}$.

A Better Intuition

Imagine you are playing a game where data is given to you

$$X_1, X_2, ..., X_t$$

Your task

- To guess X_{t+1} and an estimator of X, $\widehat{X}$

Clearly, you have looses

- They could be exemplified by the square distance between $\left(\widehat{X} - X_{t+1}\right)^2$

Strategies to minimize the regret

In the case of least squared error

$$\widehat{X} = \frac{1}{T}\sum_{t=i}^{T} X_t$$

Something Notable

- This is actually a good estimate given, if we assume $X \sim N\left(\mu, \sigma^2\right)$
- The maximum likelihood estimator of $\widehat{X} = \frac{1}{N}\sum_{t=1}^{N} X_t$

Furthermore

$$E\left[\widehat{X}\right] = \mu$$

Nevertheless

A common question in statistics

- **How well can I do using the information from my samples compared to how well I could have done had I known the distribution in advance?**

A simple function

$$Cost_T\left(Alg\right) - Cost\left(OPT\right)$$

Regret

Definition

- The sum of all the previous difference between the on-line prediction $f_i\left(\boldsymbol{w}_i\right)$ and the best optimal parameter $f_i\left(\boldsymbol{w}^*\right)$

$$R\left(T\right) = \sum_{i=1}^{N}\left[f_i\left(\boldsymbol{w}_i\right) - f_i\left(\boldsymbol{w}^*\right)\right] = f\left(T\right)$$

  - Where $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in\mathcal{X}} \sum_{i=1}^{n} f_t\left(\boldsymbol{w}\right)$

What do we want?

We want $f(T) = o(T)$ (Little o) i.e.

$$\frac{f(T)}{T} \to 0$$

Example

The Expert Advice Model

- On a sequence of rounds $t = 1, ..., T$ a player choose an action $i_t \in \{1, ..., n\}$
- The adversary chooses cost or loses for each action $l_t(1), ..., l_t(n) \in \{0, 1\}$

It looks like a Min-Max Play from Artificial Intelligence

- Theorem (Von Neumann Minimax Theorem)

$$\min_{\boldsymbol{y} \in \Delta^n} \max_{\boldsymbol{x} \in \Delta^m} \boldsymbol{y}^t A \boldsymbol{x} = V = \max_{\boldsymbol{x} \in \Delta^m} \min_{\boldsymbol{y} \in \Delta^n} \boldsymbol{y}^t A \boldsymbol{x}$$

However, we want something more flexible

The player instead of picking highest cost

- The player pick a distribution over the actions $\{1, ..., n\}$

Then, the player pays $E\left[l_t\left(I\right)\right]$ observes l_t

- Updates $p_{t+1} \in \Delta_n$, where Δ_n is the probability simplex over the n actions.

The probability simplex is the $(n-1)$-dimensional simplex determined by the unit vectors $e_1, ..., e_n \in R$

- It is the set of vectors that satisfy $\boldsymbol{x} \succcurlyeq 0$ with $\mathbf{1}^T \boldsymbol{x} = 1$

Furthermore

This is typically called the “Expert” or “Hedge” setting with regret

$$Regret = \sum_{t=1}^{T} p_t l_t - \min_{i \in \{1,\ldots,N\}} \sum_{t=1}^{T} l_t(i)$$

We now introduce the Weighted Majority Algorithm

- We define $L_t(i) = \sum_{s=1}^{t} l_s(i)$ to be the vector of cumulative losses of the experts at time t.

The algorithm chooses an expert at time t by distribution p_t where

- $w_t(i) = \exp\{-\eta L_t(i)\}$ Weight assigned to expert i at time t and $\eta > 0$ is a parameter of the algorithm.
- $p_t(i) = \frac{w_t(i)}{\sum_{j=1}^{n} w_t(i)}$ Probability of choosing expert i at time t.

Randomized Weighted-Majority(n experts)

Algorithm

Input: **Penalty** $\beta \in \left[\frac{1}{2}, 1\right)$

```
1  for $i = 1$ to $n$
2      $w_1(i) = 1$
3      $p_1(i) = \frac{1}{N}$
4  for $t = 1$ to $T$
5      for $i = 1$ to $n$
6          if $l_t(i) = 1$:
7              $w_{t+1}(i) = \beta w_t(i)$
8          else $w_{t+1}(i) = w_t(i)$
9      $W_{t+1} = \sum_{i=1}^{n} w_{t+1}(i)$
10     for $i = 1$ to $n$
11         $p_{t+1}(i) = \frac{w_{t+1}(i)}{W_{t+1}}$
12 return $w_{T+1}$
```

Then

Theorem

- Then, for any $T \geq 1$, the expected cumulative loss of Randomized Weighted-Majority can be bounded as follows

$$\mathcal{L}_T \leq \frac{\log n}{1-\beta} + (2-\beta)\,\mathcal{L}_T^{\min}$$

- with $\mathcal{L}_T = \sum_{t=1}^{T} p_t l_t,\ \ \mathcal{L}_T^{\min} = \min_{i\in\{1,\ldots,N\}} \sum_{t=1}^{T} l_t\,(i)$
- For $\beta = 1 - \frac{\sqrt{\log n}}{T}$ when $1 - \frac{\sqrt{\log n}}{T} \geq \frac{1}{2}$,

$$\mathcal{L}_T \leq \mathcal{L}_T^{\min} + 2\sqrt{T \log N}$$

Now, the proof

We define the following function

$$W_t = \sum_{i=1}^{n} w_t(i)$$

Where

We have that

$$W_{t+1} = \sum_{i:l_t(i)=0} w_t\left(i\right) + \beta \sum_{i:l_t(i)=1} w_t\left(i\right)$$

Then

$$W_{t+1} = \sum_{i:l_t(i)=0} w_t\left(i\right) + \sum_{i:l_t(i)=1} w_t\left(i\right) - \sum_{i:l_t(i)=1} w_t\left(i\right) + \beta \sum_{i:l_t(i)=1} w_t\left(i\right)$$

Then

We have

$$W_{t+1} = W_t + (\beta - 1) \sum_{i:l_t(i)=1} w_t(i) \times \frac{W_t}{W_t}$$

Then by using $p_t(i) = \frac{w_t(i)}{W_t}$ and assuming that

$$W_{t+1} = W_t + (\beta - 1) W_t \sum_{i:l_t(i)=1} p_t(i)$$

Finally

$$W_{t+1} = W_t + (\beta - 1) W_t L_t = W_t (1 - (1 - \beta) L_t)$$

Then, we have an upper bound

We have by recursion

$$W_{T+1} = n \prod_{t=1}^{T} \left(1 - (1-\beta) L_t\right)$$

- With $W_1 = \sum_{i=1}^{n} 1$ which correspond to the initialization of the algorithm

Now, we have a lower bound lower bound

$$W_{T+1} \geq \max_{i \in \{1,\dots,N\}} w_{T+1}(i) = \beta^{\mathcal{L}_T^{\min}}$$

Finally, we have that

Using $\beta^{\mathcal{L}_T^{\min}} \leq n \prod_{t=1}^{T} \left[1 - (1-\beta) L_T\right]$

$$\mathcal{L}_T^{\min} \log \beta \leq \log n + \sum_{t=1}^{T} \log \left[1 - (1-\beta) L_T\right]$$

Then, we have by using the inequality $\forall x < 1, \log(1-x) \leq -x$

$$\mathcal{L}_T^{\min} \log \beta \leq \log n - (1-\beta) \sum_{t=1}^{T} L_T$$

Furthermore

We have that

$$\mathcal{L}_T^{\min} \log \beta \leq \log n - (1-\beta)\,\mathcal{L}_T$$

After a small math manipulation we have

$$\mathcal{L}_T \leq \frac{\log n}{1-\beta} - \frac{\log\left(1-(1-\beta)\right)}{1-\beta}\mathcal{L}_T^{\min}$$

Then using $\forall x \in \left[0, \frac{1}{2}\right], -\log\left(1-x^2\right) \leq x + x^2$

$$\mathcal{L}_T \leq \frac{\log n}{1-\beta} - (2-\beta)\,\mathcal{L}_T^{\min}$$

Finally

We have that $\mathcal{L}_T^{\min} = \min_{i \in \{1,\dots,N\}} \sum_{t=1}^{T} l_t(i) \leq T$

$$\mathcal{L}_T \leq \frac{\log n}{1-\beta} - (1-\beta)\,T + \mathcal{L}_T^{\min}$$

I leave this to you, please remember

- For For $\beta = 1 - \frac{\sqrt{\log n}}{T}$ when $1 - \frac{\sqrt{\log n}}{T} \geq \frac{1}{2}$,

$$\mathcal{L}_T \leq \mathcal{L}_T^{\min} + 2\sqrt{T \log N}$$

The Stochastic Gradient Descent

Imagine the follow

- We assume that the covariance matrix and the cross-correlation vector are unknown.

We have that for a single sample

$$\mathcal{L}\left(\boldsymbol{w}, y, \boldsymbol{x}\right) = \frac{1}{2}\left(\boldsymbol{w}^T \boldsymbol{x} - y\right)^2$$

Therefore

We know

- The solution corresponds to the root of the gradient of the cost function:

$$\Sigma_x \boldsymbol{w} - \boldsymbol{p} = E\left[\boldsymbol{x}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)\right] = 0$$

We have

$$\nabla J\left(\boldsymbol{w}\right) = \Sigma_x \boldsymbol{w} - \boldsymbol{p} = E\left[\boldsymbol{x}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)\right] = 0$$

Then

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu_n \boldsymbol{x}_n\left(\boldsymbol{x}_n^T\boldsymbol{w}_{n-1} - y_n\right)$$

The Least-Mean Squares Adaptive Algorithm

The stochastic gradient algorithm for MSE

- It converges to the optimal mean-square error solution provided that μ_n satisfies the two convergence conditions.

Once the algorithm has converged

- It "locks" at the obtained solution.

In a case where the statistics of the involved process changes

- The algorithm cannot track the changes.

Therefore

if such changes occur, the error term

$$e_n = y_n - \boldsymbol{x}_n^T \boldsymbol{w}_{n-1}$$

- It will get larger values.

However

- Because μ_n is very small, the increased value of the error will not lead to corresponding changes of the estimate at time n.

Solution

This can be overcome if one sets the value of μ_n

- To a preselected fixed value, μ.

The celebrated Least-Mean-Squares Algorithms

- Algorithm LMS

```
1  $\boldsymbol{w}_{-1} = 0 \in \mathbb{R}^d$
2  Select a value $\mu$
3  **for** $n = 0, 1, ...$ **do**
4          $e_n = y_n - \boldsymbol{x}_n^T \boldsymbol{w}_{n-1}$
5          $\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu e_n \boldsymbol{x}_n$
```

Complexity

Something Notable

- The complexity of the algorithm amounts to $2d$ multiplications/additions (MADs) per time update.

However

- As the algorithm converges close the solution

Thus

- The error term is expected to take small values making the updates to remain close the solution

Important

Given that μ has a constant value

- The algorithm has now the "agility" to update the estimates
  - In an attempt to "push" the error to lower values.

Something Notable

- This small variation of the iterative scheme has important implications.

No More a Robbins-Monro stochastic family

- The resulting algorithm is no more a member of the Robbins-Monro stochastic approximation family.

AdaGrad

Adaptive Gradient Algorithm (AdaGrad) [20]

- It is a variation of the SGD based on the subgradient idea

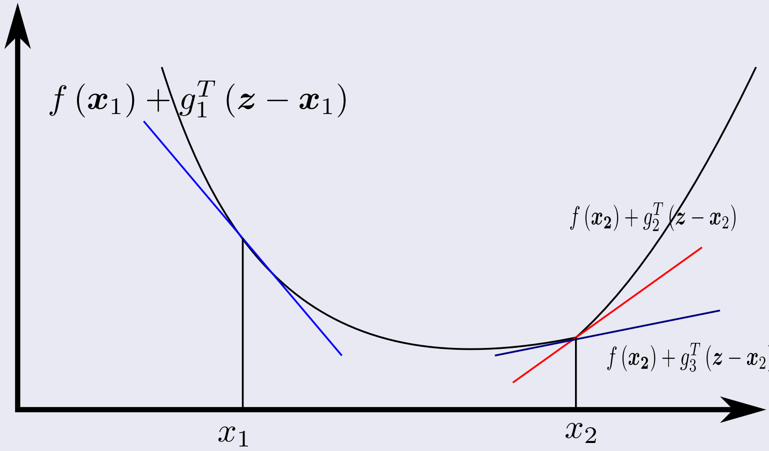
Definition (Subgradient) [4]

- A vector g is a subgradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ at a point $\boldsymbol{x} \in dom f$, if for all $\boldsymbol{z} \in dom f$

$$f\left(\boldsymbol{z}\right) \geq f\left(\boldsymbol{x}\right) + g^T\left(\boldsymbol{z} - \boldsymbol{x}\right)$$

Then

Example


$f(\boldsymbol{x}_1) + g_1^T(\boldsymbol{z} - \boldsymbol{x}_1)$
$f(\boldsymbol{x_2}) + g_2^T(\boldsymbol{z} - \boldsymbol{x}_2)$
$f(\boldsymbol{x_2}) + g_3^T(\boldsymbol{z} - \boldsymbol{x}_2)$
x_1
x_2

Standard Subgradient Algorithms

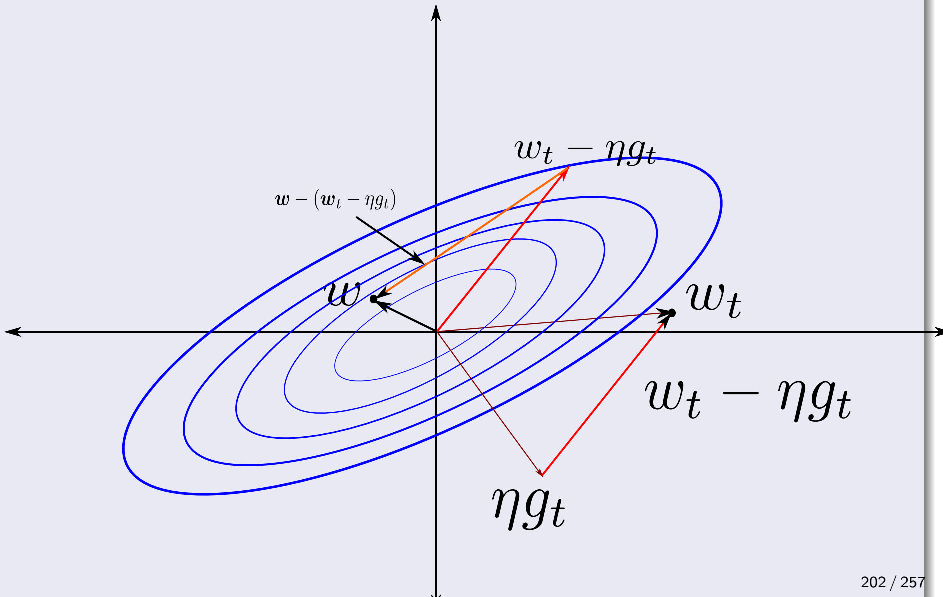
At Every Timestamp t, the learner gets the subgradient information $g_t \in \partial f_t\left(\boldsymbol{w}_t\right)$

- They move the predictor x_t in the opposite direction of g_t while projecting the gradient update

$$\boldsymbol{w}_{t+1} = \Pi_X\left(\boldsymbol{x}_t - \eta g_t\right) = \arg\min_{\boldsymbol{w} \in X} \left\|\boldsymbol{w} - \left(\boldsymbol{w}_t - \eta g_t\right)\right\|_2^2$$

Graphically

As we can see the traditional setup does not get a faster convergence



We need something faster

It has a problem when searching for the best $\boldsymbol{w}$

- Then, we need to have something way better and simpler!!!

We can do that by accumulating the gradients and use them for mapping

$$G_{1:t} = \begin{bmatrix} g_1 & g_2 & \cdots & g_t \end{bmatrix}$$

- It is the the matrix obtained by concatenating the sub-gradient sequence in row format...

We denote the i^{th} row of this matrix

- The concatenation of the i^{th} component of each sub-gradient by $g_{1:t,i}$

A First Approach

The Covariance matrix

$$G_t = \sum_{i=1}^{T} g_i g_i^T$$

It is an accumulation into the past of the previous gradients

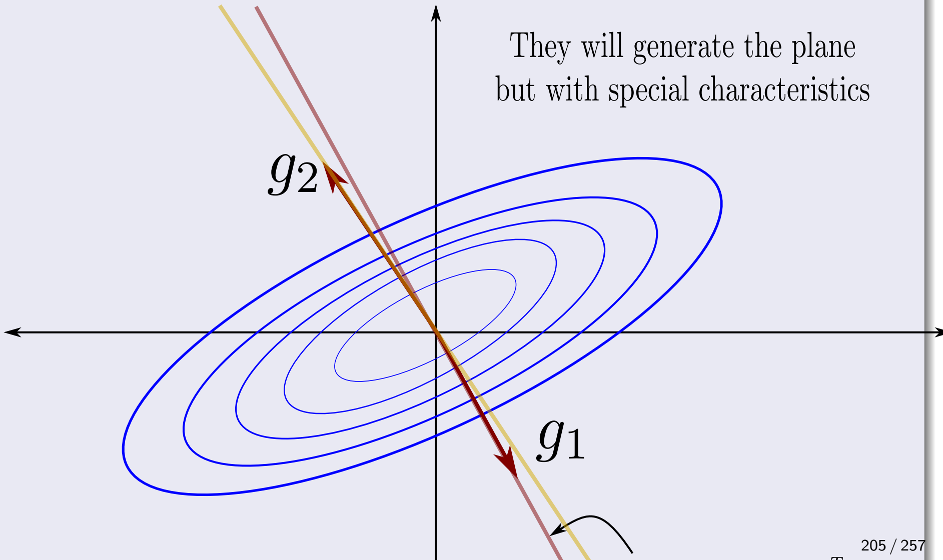
- Therefore, the larger changes happen at the beginning of the updates
  - Not only that $g_1 g_1^T$ has rank 1

Therefore as we go into the building process of G_t

- We might add new dimensions if the g_t is not in the subspace of the G_{t-1}

Graphically

The gradient descent iterations start building a possible space of projection


They will generate the plane
but with special characteristics
g_2
g_1

Mahalanobis Idea

If we think in the Mahalanobis Norm $\|\cdot\|_A = \sqrt{\langle \cdot, A\cdot \rangle}$

- Denoting the projection of a point y onto X according to A

$$\Pi_{\mathcal{X}}^{A}\left(\boldsymbol{y}\right) = \arg\min_{\boldsymbol{w}\in\mathcal{X}} \|\boldsymbol{w}-\boldsymbol{y}\|_A^2 = \arg\min_{\boldsymbol{w}\in\boldsymbol{X}} \langle \boldsymbol{w}-\boldsymbol{y}, A\left(\boldsymbol{w}-\boldsymbol{y}\right)\rangle$$

In Mahalonobis, the A generate a subspace where you are mapping

- So, you can change the distance to obtain a better performance

Using this, we can define

Therefore, we can use the inverse of such a covariance matrix

$$\boldsymbol{w}_{t+1} = \Pi_{\mathcal{X}}^{G_t^{1/2}} \left(\boldsymbol{w}_t - \eta G_t^{-\frac{1}{2}} g_t \right)$$

- $g_t = \nabla f\left(\boldsymbol{w}_t\right)$
- $G = \sum_{\tau=1}^{t} g_\tau g_\tau^T$

Remark

Actually, the inverse of the term G is related with the Hessian

- When you have a Gaussian $\boldsymbol{w}$ vector, we have that

$$H\left(\boldsymbol{w}^{*}\right)=\Sigma_{\theta}^{-1}$$

And given that $G=\sum_{\tau=1}^{t} g_{\tau} g_{\tau}^{T}$

- It can be seen as a covariance matrix for the gradient with centered data

Remarks

Given that $G_t^{-\frac{1}{2}}$ is computationally intensive $O\left(d^3\right)$

- And the diagonal has the necessary information!!! We can choose the information at the diagonal $O\left(d\right)$:

$$\boldsymbol{w}_{t+1} = \Pi_X^{diag(G)^{\frac{1}{2}}} \left[\boldsymbol{w}_t - \eta diag\left(G\right)^{-\frac{1}{2}} g_t\right]$$

Basically, it looks as a normalization

- G acts as memory for the variance of g_t

Remarks

Given that the diagonal elements $G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2$, the parameters are updated

$$w_j^{t+1} = w_j^t - \frac{\eta}{\sqrt{G_{j,j}}} g_j$$

Something Notable

- Since the denominator in this factor, $\sqrt{G_{j,j}} = \sqrt{\sum_{\tau=1}^{t} g_{\tau,j}^2}$ is the $L2$ norm.

We have that

- Extreme parameter updates get dampened, while parameters that get few or small updates receive higher learning rates.

Improving over AdaGrad

Becker and LecCun [17]

- They proposed a diagonal approximation to the Hessian.

An interesting thing

- This diagonal approximation can be computed with one additional forward and back-propagation through the model

They have the following update

$$\Delta \boldsymbol{w}_t = -\frac{1}{|diag\left(H_t\right)| + \mu} g_t$$

Even with such improvements

There are drawbacks [21]

1. The continual decay of learning rates throughout training,
2. The need for a manually selected global learning rate.

Thus

- Zeiler tried to improve this drawbacks

Idea 1: Accumulate Over Window

Something Notable

- In the AdaGrad method the denominator accumulates the squared gradients from each iteration.

This accumulation is problematic

- It continues to grow throughout training

Thus

- The learning rate will become infinitesimally small

$$w_j^{t+1} = w_j^t - \underbrace{\frac{\eta}{\sqrt{G_{j,j}}} g_j}_{\Delta w_j}$$

Thus, the modification

Use a window instead of taking all time elements and compute

$$E\left[g^2\right]_t = \rho E\left[g^2\right]_{t-1} + (1-\rho)\, g_t^2$$

- where ρ is a decay constant similar to the one in the momentum method.

Since this require the square root, this become the Root Mean Square

$$RMS\left[g\right]_t = \sqrt{E\left[g^2\right]_t + \epsilon}$$

We have the following update

$$\Delta \boldsymbol{w}_t = -\frac{\eta}{RMS\left[g\right]_t} g_t$$

Idea 2: Correct Units with Hessian Approximation

When considering the parameter updates

- "If the parameter had some hypothetical units, the changes to the parameter should be changes in those units as well"

SGD and Momentum has the following problem

$$\text{units } \Delta \boldsymbol{w} \propto \text{units } g \propto \text{units } \frac{\partial f}{\partial \boldsymbol{w}} \propto \frac{1}{\text{units } \boldsymbol{w}}$$

Hessian methods, a different story

We have that

$$\Delta \boldsymbol{w} \propto H^{-1} g \propto \frac{\frac{\partial f}{\partial \boldsymbol{w}}}{\frac{\partial^2 f}{\partial^{\mathbf{2}} \boldsymbol{w}}} \propto \text{units } \boldsymbol{w}$$

Zeiler notices that the Second Newton's Method

$$\Delta \boldsymbol{w} = \frac{\frac{\partial f}{\partial \boldsymbol{w}}}{\frac{\partial^2 f}{\partial^{\mathbf{2}} \boldsymbol{w}}} \Rightarrow \frac{1}{\frac{\partial^2 f}{\partial^{\mathbf{2}} \boldsymbol{w}}} = \frac{\Delta \boldsymbol{w}}{\frac{\partial f}{\partial \boldsymbol{w}}}$$

The RMS of the previous gradient

$$\Delta \boldsymbol{w}_t = -\frac{\eta}{RMS\left[g\right]_t} g_t$$

Even though

$\Delta \boldsymbol{w}_t$ is not know at the current time t

- But we can assume that the curvature is locally smooth (Linear)

It is possible to compute an approximation to the $\Delta \boldsymbol{w}_t$

- By computing the exponentially decaying RMS over a window of certain size by

Then, we have that

$$\frac{\Delta \boldsymbol{w}}{\frac{\partial f}{\partial \boldsymbol{w}}} \approx \frac{RMS\left[\Delta \boldsymbol{w}\right]_{t-1}}{RMS\left[g\right]_{t}}$$

The final update is

Then for the new update, we have

$$\Delta \boldsymbol{w}_t \approx -\frac{RMS\left[\Delta \boldsymbol{w}\right]_{t-1}}{RMS\left[g\right]_t} g_t$$

As in MSE [22]

We are interested in minimizing the expected value of f

$$E\left[f\left(\boldsymbol{w}\right)\right]$$

Now, assuming $g_t = \nabla_{\boldsymbol{w}} f_t\left(\boldsymbol{w}\right)$

- The algorithm updates moving averages of the gradient m_t and the squared gradient v_t.

Using combinations with $\beta_1, \beta_2 \in [0, 1)$

$$m_t = \beta_1 m_{t-1} + \left(1 - \beta_1\right) g_t$$
$$v_t = \beta_2 v_{t-1} + \left(1 - \beta_2\right) g_t^2$$

Basically, they are the following quantities

You could thing on the following concepts

$$m_t = \sum_{t=1}^{n} \tau_n g_t \approx E\left[g_t\right] \text{ and } v_t = \sum_{t=1}^{n} \tau_n g_t^2 \approx E\left[(g_t - 0)^2\right]$$

Therefore, given the decays by the following formulas

$$\widehat{m}_t = \frac{m_t}{(1-\beta_1^t)} \text{ and } \widehat{v}_t = \frac{v_t}{(1-\beta_2^t)}$$

The algorithm tries to control the step size Δ_t

$$\Delta_t = \alpha \frac{\widehat{m}_t}{(\sqrt{\widehat{v}_t})}$$

Therefore

We have two upper bounds

- When $1 - \beta_1 > \sqrt{1 - \beta_2}$

$$|\Delta_t| \leq \alpha \frac{(1 - \beta_1)}{\sqrt{1 - \beta_2}}$$

Otherwise

$$|\Delta_t| \leq \alpha$$

Therefore

Something Notable

- Since α sets (an upper bound of) the magnitude of steps in parameter space
  - We can often deduce the right order of magnitude of α for the problem at hand.

Furthermore, $\frac{\widehat{m}_t}{\left(\sqrt{\widehat{v}_t}\right)}$ can be seen as a Signal to Noise Ration (SNR)

- This value becomes zero when reaching to the optimal.

Leading to smaller effective steps in parameter space

- A form of automatic annealing.

Finally, ADAM Algorithm

Adam Algorithm

Input: α step size, $\beta_1, \beta_2 \in [0,1)$, $f(\boldsymbol{w})$ objective function, $\boldsymbol{w}_0$ Initial Parameter

```
1  $m_0 = 0, v_0 = 0$, 1st and 2nd moment vector respectively.
2  $t = 0$ initial time step
3  while $w_t$ not converged do
4          $t = t + 1$
5          $g_t = \nabla f(\boldsymbol{w}_{t-1})$ ←Get gradients w.r.t. stochastic objective at timestep $t$
6          $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ ←Update raw first moment
7          $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ ←Update raw second moment
8          $\widehat{m}_t = \frac{m_t}{(1-\beta_1^t)}$ ←Bias correction pf the first moment
9          $\widehat{v}_t = \frac{v_t}{(1-\beta_2^t)}$ ←Bias correction pf the seconf moment
10         $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \alpha \frac{\widehat{m}_t}{\left(\sqrt{\widehat{v}_t + \epsilon}\right)}$
11 Return $\boldsymbol{w}_t$
```

Regret in ADAM

The adaptive method ADAM achieves

$$R(T) = O\left(\log d\sqrt{n}\right)$$

Compared with the Online Gradient Descent

- Hazan, Elad, Alexander Rakhlin, and Peter L. Bartlett. "Adaptive online gradient descent." Advances in Neural Information Processing Systems. 2008.

$$R(T) = O\left(\sqrt{dn}\right)$$

Looking into the past

If we look at the following equations

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2$$

with the update

$$\widehat{m}_t = \frac{m_t}{(1-\beta_1^t)} \text{ and } \widehat{v}_t = \frac{v_t}{(1-\beta_2^t)}$$

Now, we have

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \alpha \frac{\widehat{m}_t}{\left(\sqrt{\widehat{v}_t} + \epsilon\right)}$$

Then, if we apply the recursion to it

We have

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-2} - \alpha \left[\frac{\widehat{m}_{t-1}}{\left(\sqrt{\widehat{v}_{t-1}} + \epsilon\right)} + \frac{\widehat{m}_t}{\left(\sqrt{\widehat{v}_t} + \epsilon\right)} \right]$$

We notice that the term $\sqrt{\widehat{v}_{t+1}} + \epsilon$

- It works as a variance that if $\nabla f\left(\boldsymbol{w}_{t-1}\right) \longrightarrow 0$ works as a dampener in the search

Then, the final recursion takes to the point 0

$$\boldsymbol{w}_t = \boldsymbol{w}_0 - \alpha \left[\sum_{k=1}^{t} \frac{\widehat{m}_k}{\left(\sqrt{\widehat{v}_k} + \epsilon\right)} \right]$$

Doing some Math work

We have that the last updating term look like when making $\epsilon = 0$

$$\sum_{k=1}^{t} \frac{\widehat{m}_k}{\left(\sqrt{\widehat{v}_k}\right)} = \sum_{k=1}^{t} \frac{\frac{m_k}{\left(1-\beta_1^k\right)}}{\left(\sqrt{\frac{v_k}{\left(1-\beta_2^k\right)}}\right)} = \sum_{k=1}^{t} \frac{\left(1-\beta_2^k\right)^{\frac{1}{2}}}{\left(1-\beta_1^k\right)} \times \frac{m_k}{\sqrt{v_k}}$$

Clearly

$$\left(1-\beta_2^k\right)^{\frac{1}{2}} \to 1 \text{ and } 1-\beta_1^k \to 1$$

But the first one faster than the second one

- Therefore the steps modifications depend on the different values for the betas.

We have different cases

For Example, we could have

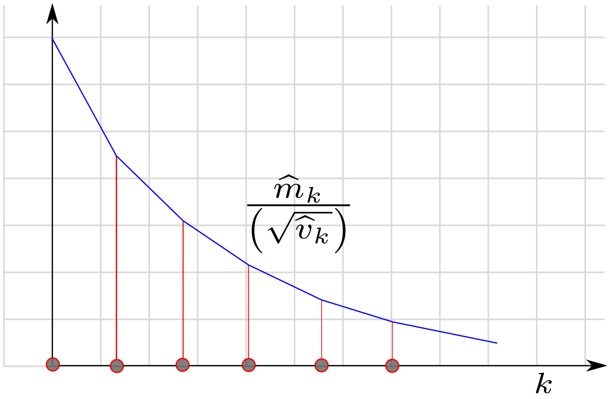
- $\beta_1 = 0.9$ and $\beta_2 = 0.9$
  - Making going to zero slower than when values are near to 0.
  - A more detailed analysis is needed!!!

However, if we assume that they cancel each other, and if v_k tend to zero at slower pace

- The terms in the past could be more important than the present ones

Actually we need to analyze the convergence

We could have something like



Simulated Annealing and Adam

Simulated_Annealing$(\omega, M_k, \epsilon_t, \epsilon, t_k, f)$

```
1  $\Delta E = \infty$
2  while $|\Delta E| > \epsilon$
3      for $i = 0, 1, 2, ..., M_k$
4          Randomly select $\omega'$ in $N(\omega)$
5          $\Delta E = f(\omega') - f(\omega)$
6          if $\Delta E \leq 0$
7              $\omega = \omega'$
8          if $\Delta E > 0$
9              $\omega = \omega'$ with probability $Pr\{Accepted\} = \exp\left\{\frac{-\Delta E}{t_k}\right\}$
10     $t_k = t_k - \epsilon_t$ # We can also use $t_k = \epsilon_t \cdot t_k$
```

In accordance with the Simulated Annealing part

This makes ADAMS adaptive

- But with a limitation on the change because you always take the step
  - Remember the step $\omega = \omega'$ **with probability** $Pr\left\{Accepted\right\} = \exp\left\{\frac{-\Delta E}{t_k}\right\}$

Making the past more important than the present

- When updating the values

Actually it is form of cooling as in Simulated Annealing by the term

$$\sum_{k=1}^{t} \frac{\left(1-\beta_2^k\right)^{\frac{1}{2}}}{\left(1-\beta_1^k\right)}$$

The Problem with such approach

You require more information from your surroundings optimization landscape

- After all, the ADAM is a compromise between adaptation and the Zeroth methods

Making it quite light for problem as

- Deep Neural Networks

However

It could be a good idea to add such adaptivness to ADAM

I could result in something heavier, but more effective to obtain better performance

A naive idea would be to substitute the term $\frac{\alpha}{\left(\sqrt{\widehat{v}_t}+\epsilon\right)}$ by the Fisher Information matrix [23]

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - E\left[\frac{\partial \log f\left(X|\theta\right)}{\partial \theta}|\theta\right]^{-1} \widehat{m}_t$$

Remarks about ADAM

ADAM is favored in Deep Learning given that

1. Given the use of stochastic gradient update:
    1. It is Computationally Efficient
    2. It requires Little memory.
    3. It is suited for problems that are large in terms of data and/or parameters.
2. Invariant to diagonal rescale of the gradients.
3. Appropriate for non-stationary objectives.
4. Appropriate for problems with very noisy/or sparse gradients.

Finally and most important

- Hyper-parameters have intuitive interpretation and typically require little tuning.

Natural Gradient Descent

In 1998 Amari et al. [24, 25, 26, 23] started to integrate the Fisher Information Matrix into the gradient step

$$E\left[\frac{\partial \log p\left(x, \boldsymbol{w}\right)}{\partial w_i} \times \frac{\partial \log p\left(x, \boldsymbol{w}\right)}{\partial w_j}\right]$$

We will look at the developments of Amari in his paper

- "Natural Gradient Works Efficiently in Learning"

Natural Gradient

Let $S = \left\{ \boldsymbol{w} \in \mathbb{R}^d \right\}$ be the parameter function for a loss function $L(\boldsymbol{w})$

- And we can actually define the norm of the small increment $d\boldsymbol{w}$ (The Steep Direction) as

$$\|d\boldsymbol{w}\|_2^2 = \sum_{i=1}^{d} (dw_i)^2$$

This is perfect for an orthonormal coordinate system

- But there is a more interesting thing happens when we use any other coordinate system

As in the Mahalonobis distance

$$\|d\boldsymbol{w}\|_{non-2}^2 = \sum_{i=1}^{d} g_{ij} dw_i dw_j$$

This happens in Curved Manifolds

Still a Riemann Space

- And it allows to define a new matrix $n \times n$ $G = (g_{ij})$

Where

$$g_{ij}(\boldsymbol{w}) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

- When you talk of the orthonormal Euclidean case

Amari was able to establish

Theorem

- The steepest descent direction of $L\left(\boldsymbol{w}\right)$ in a Riemannian space is given by

$$-\widetilde{\nabla} L\left(\boldsymbol{w}\right) = -G^{-1}\left(\boldsymbol{w}\right)\nabla L\left(\boldsymbol{w}\right)$$

  - where $G^{-1} = \left(g^{ij}\right)$ is the inverse of the metric $G = \left(g_{ij}\right)$ and ∇L is

$$\nabla L\left(\boldsymbol{w}\right) = \left(\frac{\partial L\left(\boldsymbol{w}\right)}{\partial w_1}, \ldots, \frac{\partial L\left(\boldsymbol{w}\right)}{\partial w_n}\right)$$

Proof

For this, we assume $d\boldsymbol{w} = \epsilon a$

$$L\left(\boldsymbol{w} + d\boldsymbol{w}\right) = L\left(\boldsymbol{w}\right) + \epsilon \nabla L\left(\boldsymbol{w}\right)^T a$$

Under constraint given that we are only interested in the direction

$$\|a\|^2_{non-2} = \sum_{i=1}^{d}\sum_{j=1}^{d} g_{ij} a_i a_j = a^T G a = 1$$

By the Lagrangean method

$$\frac{\partial\left(L\left(\boldsymbol{w}\right) + \epsilon \nabla L\left(\boldsymbol{w}\right)^T a - \lambda\left(a^T G a - 1\right)\right)}{\partial a} = 0$$

Further

Using our well know matrix derivatives

$$\epsilon \nabla L(\boldsymbol{w}) - \lambda 2 G a = 0$$

Then, we have

$$\nabla L(\boldsymbol{w}) = \frac{2\lambda}{\epsilon} G a$$

Therefore

$$\nabla L(\boldsymbol{w}) = \lambda' G a$$

Finally, we have

The following

$$a = \frac{1}{\lambda'} G^{-1} \nabla L(\boldsymbol{w})$$

We call the following term, the natural gradient in the Riemmannian Space

$$\widetilde{\nabla} L(\boldsymbol{w}) = G^{-1}(\boldsymbol{w}) \nabla L(\boldsymbol{w})$$

Suggesting that

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta_t \widetilde{\nabla} L(\boldsymbol{w})$$

Here, Expected Value Magic

Consider that you have a distribution generating samples as always independently

$$z_1, z_2, ..., z_t \sim q(z)$$

Then, you assume a loss function $l(\boldsymbol{w}, z)$ to process the $z's$

- Then, the average risk is $L(\boldsymbol{w}) = E[l(\boldsymbol{w}, z)]$

Now, we want to approximate the probability distribution $q(z)$

- By using an estimation $p(z, \widehat{\boldsymbol{w}})$

For this

We can use the following loss function

$$l\left(z,\boldsymbol{w}\right)=-\log p\left(z,\boldsymbol{w}\right)$$

The expected loss is then given by

$$L\left(\boldsymbol{w}\right)=-E\left[\log p\left(z,\boldsymbol{w}\right)\right]=E_q\left[\log\frac{q\left(z\right)}{p\left(z,\boldsymbol{w}\right)}\right]+H_Z$$

- where H_Z is the entropy of $q\left(z\right)$ not depending on $\boldsymbol{w}$.

Thus, we can use Kullback-Leibler divergence

We can minimize such function

$$D\left[q\left(z\right):p\left(z,\boldsymbol{w}\right)\right]=\int q\left(z\right)\log\frac{q\left(z\right)}{p\left(z,\boldsymbol{w}\right)}dz$$

When the true distribution $q\left(z\right)$ is written as $q\left(z\right)=p\left(z,\boldsymbol{w}^{*}\right)$

- This is equivalent to obtain the maximum likelihood estimator $\widehat{\boldsymbol{w}}$

And Here the important part

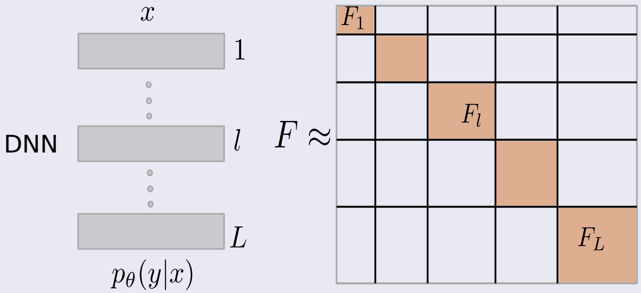
The Riemannian structure of the parameter space of a statistical model is defined by the Fisher information [27, 28]

$$g_{ij}(\boldsymbol{w}) = E\left[\frac{\partial \log p(x, \boldsymbol{w})}{\partial w_i} \times \frac{\partial \log p(x, \boldsymbol{w})}{\partial w_j}\right]$$

Although, the computations of the matrix

They are $O\left(d^3\right)$

- There are proposal in the fact that you have a layered structure in the Neural Network
  - The Fisher has a box structure



However

Even though, we would love to look more for this

- This is for another time...

Conclusions

In Machine Learning

- We need to have the best speedups to handle the problem dealing with Big Data...

As we get more and more algorithms

- It is clear that optimization for Big Data is one of the hottest trends in Machine Learning

[1] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[2] R.-Y. Sun, "Optimization for deep learning: An overview," *Journal of the Operations Research Society of China*, vol. 8, no. 2, pp. 249–294, 2020.

[3] S. Bubeck, "Convex optimization: Algorithms and complexity," *arXiv preprint arXiv:1405.4980*, 2014.

[4] J. Nocedal and S. Wright, *Numerical optimization*.
Springer Science & Business Media, 2006.

[5] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: theory and algorithms*.
John Wiley & Sons, 2013.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*.
Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[7] S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*.
Academic Press, 1st ed., 2015.

[8] W. Rudin, *Real and Complex Analysis, 3rd Ed.*
New York, NY, USA: McGraw-Hill, Inc., 1987.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*.
The MIT Press, 3rd ed., 2009.

[10] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.

[11] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*.
Springer Publishing Company, Incorporated, 1 ed., 2014.

[12] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 09 1951.

[13] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 57–95, 2016.

[14] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[15] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*, pp. 7–15, Springer, 1987.

[16] D. B. Hitchcock, "A history of the metropolis–hastings algorithm," *The American Statistician*, vol. 57, no. 4, pp. 254–257, 2003.

[17] S. Becker, Y. Le Cun, *et al.*, "Improving the convergence of back-propagation learning with second order methods," in *Proceedings of the 1988 connectionist models summer school*, pp. 29–37, 1988.

[18] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, (New York, NY, USA), pp. 661–670, ACM, 2014.

[19] M. Ravaut, "Faster gradient descent via an adaptive learning rate," 2017.

[20] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[21] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] J. Martens, "New insights and perspectives on the natural gradient method," *arXiv preprint arXiv:1412.1193*, 2014.

[24] G. Zhang, J. Martens, and R. Grosse, "Fast convergence of natural gradient descent for overparameterized neural networks," *arXiv preprint arXiv:1905.10961*, 2019.

[25] S.-I. Amari and S. C. Douglas, "Why natural gradient?," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, vol. 2, pp. 1213–1216, IEEE, 1998.

[26] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[27] S.-i. Amari, "Information geometry of the em and em algorithms for neural networks," *Neural networks*, vol. 8, no. 9, pp. 1379–1408, 1995.

[28] C. R. Rao, "Information and the accuracy attainable in the estimation of statistical parameters," in *Breakthroughs in statistics*, pp. 235–247, Springer, 1992.