Big Data Analytics

(Stochastic) Gradient descent

Souhaib Ben Taieb

March 25, 2021

University of Mons

Table of contents

Introduction

Gradient descent

Stochastic gradient descent

Other optimization methods

How to minimize E_{in}

For logistic regression,

$$E_{\text{in}}(\mathbf{w}) \;=\; \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right) \qquad \longleftarrow \text{ iterative solution}$$

Compare to linear regression:

$$E_{\text{in}}(\mathbf{w}) \;=\; \frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n\right)^2 \qquad \longleftarrow \text{ closed-form solution}$$

© AM Creator: Yaser Abu-Mostafa - LFD Lecture 9

Optimization for machine learning

Much of machine learning can be written as the following optimization problem:

$$\min_{\boldsymbol{w}} \frac{1}{N} \sum_{n=1}^{N} e_n(\boldsymbol{w}; (y_n, \boldsymbol{x}_n)) \equiv \ell(\boldsymbol{w})$$

where e_n is the error on the nth data point.

Types of optimization problems:

- **Convex optimization**
  - Many classes of convex optimization problems admit polynomial-time algorithms
  - Includes logistic regression, linear regression, etc.
- **Non-convex optimization**
  - NP-hard in general
  - Includes neural networks (deep learning)

Table of contents

Introduction

Gradient descent

Stochastic gradient descent

Other optimization methods

Optimization for machine learning

We want to minimize a convex and differentiable loss function $\ell(\boldsymbol{w})$ or $\mathsf{E}_{\mathsf{in}}(\boldsymbol{w})$. In some cases, it is possible to **analytically** compute $\boldsymbol{w}^*$ such that $\nabla\ell(\boldsymbol{w}^*) = 0$.

More commonly the condition that the gradient equal zero will not have an analytical solution. We will need **iterative methods**.

How can you minimize a function if you don't know much about it? The trick is to assume it is much simpler than it really is. This can be done by approximating the function using **Taylor's approximation** and **minimizing this approximation**.

Taylor's approximation

Let us approximate the function $\ell(\cdot)$ around $\boldsymbol{w}$, i.e. we want to approximate $\ell(\boldsymbol{w}+\boldsymbol{s})$ where $\|\boldsymbol{s}\|_2$ is small (i.e. $\boldsymbol{w}+\boldsymbol{s}$ is very close to $\boldsymbol{w}$). In that case, we can approximate the function $\ell(\boldsymbol{w}+\boldsymbol{s})$ by its first derivatives as

$$\ell(\boldsymbol{w}+\boldsymbol{s}) \approx \ell(\boldsymbol{w}) + g(\boldsymbol{w})^T \boldsymbol{s},$$

where $g(\boldsymbol{w}) = \nabla \ell(\boldsymbol{w})$ is the gradient.

Using its first and second derivatives, we can also write

$$\ell(\boldsymbol{w}+\boldsymbol{s}) \approx \ell(\boldsymbol{w}) + g(\boldsymbol{w})^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T H(\boldsymbol{w})\boldsymbol{s},$$

where $H(\boldsymbol{w}) = \nabla^2 \ell(\boldsymbol{w})$ is the Hessian of ℓ.

Both approximations are valid if $\|\boldsymbol{s}\|_2$ is small, but the second one assumes that ℓ is **twice differentiable** and is more expensive to compute but also more accurate than only using gradient.

Table of contents

Introduction

Gradient descent

Stochastic gradient descent

Other optimization methods

Gradient descent

In an iterative method, given $\boldsymbol{w}(t)$, we want to find $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \boldsymbol{s}$ such that $\ell(\boldsymbol{w}(t+1)) - \ell(\boldsymbol{w}(t))$ is minimized.

Let us write $\boldsymbol{s} = \eta \boldsymbol{v}$ where $\eta > 0$ is the step-size in the direction $\boldsymbol{v}$ with $\|\boldsymbol{v}\|_2 = 1$. We want to find the direction $\boldsymbol{v}$ which minimizes $\ell(\boldsymbol{w}(t) + \eta \boldsymbol{v}) - \ell(\boldsymbol{w}(t))$, i.e.

$$\begin{aligned}
\min_{\boldsymbol{v}, \|\boldsymbol{v}\|_2 = 1} \ell(\boldsymbol{w}(t) + \eta \boldsymbol{v}) - \ell(\boldsymbol{w}(t)) &\equiv \min_{\boldsymbol{v}, \|\boldsymbol{v}\|_2 = 1} g(\boldsymbol{w}(t))^T \boldsymbol{v} \\
&\equiv \min_{\boldsymbol{v}, \|\boldsymbol{v}\|_2 = 1} \|g(\boldsymbol{w}(t))\| \, \|\boldsymbol{v}\| \cos(\theta) \\
&\equiv \min_{\boldsymbol{v}, \|\boldsymbol{v}\|_2 = 1} \cos(\theta),
\end{aligned}$$

where θ is the angle between the vectors $g(\boldsymbol{w}(t))$ and $\boldsymbol{v}$.

Gradient descent

This quantity is minimized when $\cos(\theta) = -1$, i.e. $\theta = 180°$, where $\boldsymbol{v}$ is pointing in the opposite direction of the gradient, i.e. $-g(\boldsymbol{w}(t)$, and since $\boldsymbol{v}$ is a unit vector, we can write

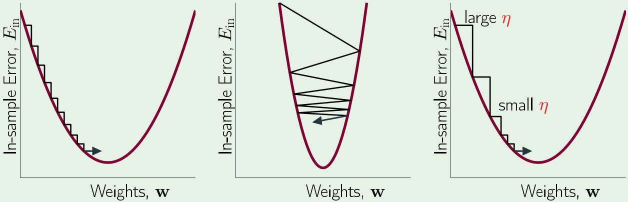
$$\boldsymbol{v} = \frac{-g(\boldsymbol{w}(t))}{\|g(\boldsymbol{w}(t))\|}.$$

In other words, we have

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta \frac{g(\boldsymbol{w}(t))}{\|g(\boldsymbol{w}(t))\|}$$

Fixed-size step?

How η affects the algorithm:



η too small | η too large | variable η – just right

η should increase with the slope

Gradient descent - from step size to learning rate

Instead of

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta \frac{g(\boldsymbol{w}(t))}{\|g(\boldsymbol{w}(t))\|},$$

we use

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta_t \frac{g(\boldsymbol{w}(t))}{\|g(\boldsymbol{w}(t))\|},$$

where $\eta_t = \eta \|g(\boldsymbol{w}(t))\|$, i.e. the step size is proportional to the length of the gradient.

We obtain

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta g(\boldsymbol{w}(t)),$$

where η is now a (redefined) fixed **learning rate**.

Gradient descent algorithm

Fixed learning rate gradient descent:

```
Initialize the weights at time step t = 0 to w(0).
for t = 0, 1, 2, ... do
    Compute the gradient g_t = ∇E_in(w(t)).
    Set the direction to move, v_t = −g_t.
    Update the weights: w(t + 1) = w(t) + η v_t.
    Iterate to the next step until it is time to stop.
Return the final weights.
```

$\boldsymbol{v}_t$ is a direction that is no longer restricted to unit length.

Logistic regression algorithm

Initialize the weights at $t = 0$ to $\mathbf{w}(0)$

for $t = 0, 1, 2, \ldots$ **do**

  Compute the gradient

$$\nabla E_{\text{in}} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{\,y_n \mathbf{w}^{\mathsf{T}}(t) \mathbf{x}_n}}$$

  Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E_{\text{in}}$

  Iterate to the next step until it is time to stop

Return the final weights $\mathbf{w}$

Gradient descent

Another way to retrieve the gradient descent algorithm consists in minimizing a specific quadratic approximation of the function[1].

The second-order Taylor expansion of ℓ is given by

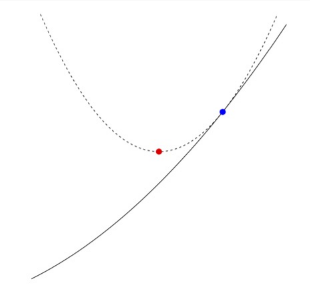
$$\begin{aligned}\ell(\boldsymbol{w}(t+1)) \approx \ell(\boldsymbol{w}(t)) &+ g(\boldsymbol{w}(t))^T(\boldsymbol{w}(t+1) - \boldsymbol{w}(t)) \\ &+ \frac{1}{2}(\boldsymbol{w}(t+1) - \boldsymbol{w}(t))^T H(\boldsymbol{w}(t))(\boldsymbol{w}(t+1) - \boldsymbol{w}(t)).\end{aligned}$$

Consider the quadratic approximation of ℓ, replacing $H(\boldsymbol{w}(t))$ by $\frac{1}{\eta}\boldsymbol{I}$ (replacing the curvature given by the Hessian with a much simpler notion of curvature). We can write

$$\begin{aligned}\ell(\boldsymbol{w}(t+1)) \approx \ell(\boldsymbol{w}(t)) &+ g(\boldsymbol{w}(t))^T(\boldsymbol{w}(t+1) - \boldsymbol{w}(t)) \\ &+ \frac{1}{2\eta}\left\|(\boldsymbol{w}(t+1) - \boldsymbol{w}(t))\right\|^2.\end{aligned}$$

[1] Note that trying to directly minimize a linear approximation to our function wouldn't be very useful since the solution is infinity.

Gradient descent



$\ell(\boldsymbol{w}(t+1))$ is approximated by a convex quadratic, so we know we can minimize it just by setting its gradient to 0.

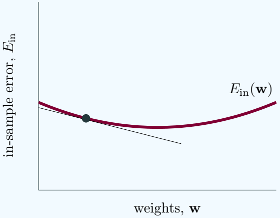
We have

$$\nabla \ell(\boldsymbol{w}(t+1)) \approx g(\boldsymbol{w}(t)) + \frac{1}{\eta}(\boldsymbol{w}(t+1) - \boldsymbol{w}(t)) = 0.$$

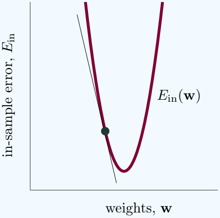
$$\implies \boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta g(\boldsymbol{w}(t))$$

Choosing the learning rate

In gradient descent, the learning rate η multiplies the negative gradient to give the move $-\eta \nabla E_{\text{in}}$. The size of the step taken is proportional to η . The optimal step size (and hence learning rate η) depends on how **wide** or **narrow** the error surface is near the minimum.



wide: use large η.



narrow: use small η.

When the surface is wider, we can take larger steps without overshooting; since $\|\nabla E_{\text{in}}\|$ is small, we need a large η. Since we do not know ahead of time how wide the surface is, it is easy to choose an inefficient value for η.

Variable learning rate gradient descent

A simple heuristic that adapts the learning rate to the error surface works well in practice. If the error drops, increase η; if not, the step was too large, so reject the update and decrease η.

Variable Learning Rate Gradient Descent:

Initialize $\mathbf{w}(0)$, and η_0 at $t = 0$. Set $\alpha > 1$ and $\beta < 1$.
while stopping criterion has not been met **do**
    Let $\mathbf{g}(t) = \nabla E_{\text{in}}(\mathbf{w}(t))$, and set $\mathbf{v}(t) = -\mathbf{g}(t)$.
    if $E_{\text{in}}(\mathbf{w}(t) + \eta_t \mathbf{v}(t)) < E_{\text{in}}(\mathbf{w}(t))$ **then**
        accept: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta_t \mathbf{v}(t)$; $\eta_{t+1} = \alpha\eta_t$
    else
        reject: $\mathbf{w}(t+1) = \mathbf{w}(t)$; $\eta_{t+1} = \beta\eta_t$.
    Iterate to the next step, $t \leftarrow t + 1$.

I is also called *Backtracking line search*.

Steepest descent – gradient descent with (exact) line search

Once the direction in which to move, $\boldsymbol{v}_t$, has been determined, why not simply continue along that direction until the error stops decreasing? This leads us to *steepest descent – gradient descent with (exact) line search*.

Steepest Descent (Gradient Descent + Line Search):

1: Initialize $\mathbf{w}(0)$ and set $t = 0$;
2: **while** stopping criterion has not been met **do**
3: Let $\mathbf{g}(t) = \nabla E_{\text{in}}(\mathbf{w}(t))$, and set $\mathbf{v}(t) = -\mathbf{g}(t)$.
4: Let $\eta^* = \text{argmin}_\eta\, E_{\text{in}}(\mathbf{w}(t) + \eta\mathbf{v}(t))$.
5: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta^*\mathbf{v}(t)$.
6: Iterate to the next step, $t \leftarrow t + 1$.

Stopping criterion

Typically the initial point $\boldsymbol{w}(0)$ is picked randomly, or we use prior knowledge about the problem. But when to stop the algorithm?

Some common choices (ϵ is a small prescribed threshold):

- $\|\nabla E_{\text{in}}(\boldsymbol{w}(t))\| < \epsilon$
- $|E_{\text{in}}(\boldsymbol{w}(t+1)) - E_{\text{in}}(\boldsymbol{w}(t))| < \epsilon$
- $\|\boldsymbol{w}(t+1) - \boldsymbol{w}(t)\| < \epsilon$
- $\frac{|E_{\text{in}}(\boldsymbol{w}(t+1)) - E_{\text{in}}(\boldsymbol{w}(t))|}{\max\{1,\ |E_{\text{in}}(\boldsymbol{w}(t))|\}} < \epsilon$
- $t > T$

Table of contents

Introduction

Gradient descent

Stochastic gradient descent

Other optimization methods

About gradient descent

Many machine learning problems involve the following optimization problem The in-sample error is given by

$$\min_{\boldsymbol{w}} E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} e(y_n, h(\boldsymbol{x}_n)), \tag{1}$$

e.g. for logistic regression, we have $e(y_n, h(\boldsymbol{x}_n)) = \ln(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n})$.

Minimizing (1) using **gradient descent** requires to compute

$$\nabla E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \nabla e(y_n, h(\boldsymbol{x}_n)),$$

In other words, $\nabla E_{\text{in}}(\boldsymbol{w})$ is based on <u>all</u> examples $(\boldsymbol{x}_n, y_n)$, also called **batch GD**.

- Computing the **full gradient** is slow for big data
- Stuck at stationary points (non-convex optimization)

Stochastic gradient descent

1. Pick one $(\boldsymbol{x}_n, y_n)$ at a time (uniformly at random)
2. Apply GD to $e(y_n, h(\boldsymbol{x}_n))$, i.e. compute

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \nabla e(y_n, h(\boldsymbol{x}_n)).$$

What is the average direction?

$$\begin{aligned}
\mathbb{E}_n[-\nabla e(y_n, h(\boldsymbol{x}_n))] &= \sum_{n=1}^{N} \frac{1}{N}[-\nabla e(y_n, h(\boldsymbol{x}_n))] \\
&= -\frac{1}{N}\sum_{n=1}^{N} \nabla e(y_n, h(\boldsymbol{x}_n)) \\
&= -\nabla E_{\text{in}}(\boldsymbol{w})
\end{aligned}$$

Stochastic gradient descent (SGD) is an unbiased estimate of GD with a **higher variance**.

Benefits of SGD

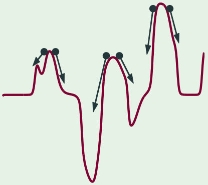
- Cheaper computation
- Randomization
- Simple

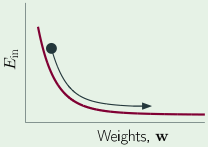
Benefits of SGD

1. cheaper computation
2. randomization
3. simple

Rule of thumb:

$\eta = 0.1$ works





randomization helps

© Creator: Yaser Abu-Mostafa - LFD Lecture 10

Mini-batch gradient descent

Compute the gradient using $1 \leq b \leq N$ data points.

1. Pick b data points ($1 \leq b \leq N$)
2. Apply batch GD to these b points

- $b = N$ is GD and $b = 1$ is SGD
- Bias and variance tradeoff
- A single pass through the entire training data is called an *epoch*. With mini-batches of size b, we update the parameters N/b times per epoch.
- We often need multiple epochs to obtain a good training accuracy.

Table of contents

Introduction

Gradient descent

Stochastic gradient descent

Other optimization methods

Newton's method

Let us assume ℓ is twice differentiable, and minimize the second-order Taylor expansion of ℓ, given by

$$\begin{aligned}\ell(\boldsymbol{w}(t+1)) \approx\ & \ell(\boldsymbol{w}(t)) + g(\boldsymbol{w}(t))^T(\boldsymbol{w}(t+1)-\boldsymbol{w}(t)) \\ & + \frac{1}{2}(\boldsymbol{w}(t+1)-\boldsymbol{w}(t))^T H(\boldsymbol{w}(t))(\boldsymbol{w}(t+1)-\boldsymbol{w}(t)).\end{aligned}$$

We have

$$\begin{aligned}\nabla\ell(\boldsymbol{w}(t+1)) &= g(\boldsymbol{w}(t)) + H(\boldsymbol{w}(t))(\boldsymbol{w}(t+1)-\boldsymbol{w}(t)) : \\ &\Longrightarrow g(\boldsymbol{w}(t)) + H(\boldsymbol{w}(t))(\boldsymbol{w}(t+1)-\boldsymbol{w}(t)) = 0 \\ &\Longrightarrow \boldsymbol{w}(t+1) = \boldsymbol{w}(t) - H(\boldsymbol{w}(t))^{-1} g(\boldsymbol{w}(t))\end{aligned}$$

The *damped* Newton's method with a small step size $0 < \eta < 1$:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta\ H(\boldsymbol{w}(t))^{-1} g(\boldsymbol{w}(t)).$$

See also Quasi-Newton methods which approximate the Hessian.

Other optimization methods

- Momentum and Acceleration
- Adaptive Gradient Algorithm (AdaGrad), Root Mean Square Propagation (RMSProp), Adam
- Stochastic Average Gradient (SAG), Stochastic Variance Reduced Gradient (SVRG)
- Conjugate gradient
- ...

An overview of gradient descent optimization algorithms: `https://ruder.io/optimizing-gradient-descent/`

Additional considerations

- **Gradient descent**
  - Simple idea, and each iteration is (usually) cheap
  - Fast for well-conditioned, strongly convex problems
  - Can often be slow, because many interesting problems aren't strongly convex or well-conditioned
  - Can't handle nondifferentiable functions
- **Stochastic Gradient Descent**
  - In many ML problems we don't care about optimizing to high accuracy, it doesn't pay off in terms of statistical performance
  - Can be super effective in terms of iteration cost, memory.
  - Can be slow to converge.
  - Popular in large-scale, continous, nonconvex optimization, but it is still not well-understood (e.g. implicit regularization)
- **Newton's method**
  - Requires more memory and computation per iteration
  - Not affected by a problem's conditioning,