Machine Learning

Linear Models

Fabio Vandin

October 25^{th}, 2022

Perceptron: A Modern View

The previous presentation of the Perceptron is the standard one.

However, we can derive the Perceptron in a different way...

Assume you want to solve a:

- binary classification problem: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$
- with linear models
- with loss $\ell(\mathbf{w}, (\mathbf{x}, y)) = \max\{0, -y\langle \mathbf{w}, \mathbf{x}\rangle\}$.

Approach: ERM $\Rightarrow$ need to find the model/hypothesis with smallest training error

How?

Note: this is a common framework in all of machine learning!

Gradient Descent (GD)

General approach for *minimizing* a differentiable convex function $f(\mathbf{w})$

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function

Definition

The *gradient* $\nabla f(\mathbf{w})$ of f at $\mathbf{w} = (w_1, \ldots, w_d)$ is

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)$$

Intuition: the gradient points in the direction of the greatest rate of increase of f around $\mathbf{w}$

Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

GD algorithm:

$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$; // or $\vec{w}^{(0)} \leftarrow$ random vector in $\mathbb{R}^d$

for $t \leftarrow 0$ *to* $T - 1$ **do**

$\quad \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$;

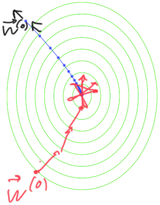
return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;

Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

$\vec{w} \in \mathbb{R}^2$

GD algorithm:

$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;
for $t \leftarrow 0$ *to* $T-1$ **do**
$\quad \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$;
return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;



Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

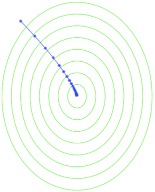
GD algorithm:

$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;

for $t \leftarrow 0$ *to* $T - 1$ **do**

$\quad \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$;

return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;



Notes:

- output vector could also be $\mathbf{w}^{(T)}$ or $\arg\min_{\mathbf{w}^{(t)} \in \{1,\ldots,T\}} f(\mathbf{w}^{(t)})$
- returning $\bar{\mathbf{w}}$ is useful for nondifferentiable functions (using *subgradients* instead of gradients...) and for stochastic gradient descent...
- η: *learning rate*; sometimes a time dependent $\eta^{(t)}$ is used (e.g., "move" more at the beginning than at the end)

Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

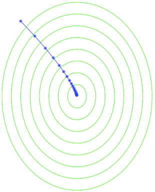
GD algorithm:

$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;

for $t \leftarrow 0$ *to* $T-1$ **do**

  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$;

return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;



Notes:

- output vector could also be $\mathbf{w}^{(T)}$ or $\arg\min_{\mathbf{w}^{(t)} \in \{1,\ldots,T\}} f(\mathbf{w}^{(t)})$
- returning $\bar{\mathbf{w}}$ is useful for nondifferentiable functions (using *subgradients* instead of gradients...) and for stochastic gradient descent...
- η: *learning rate*; sometimes a time dependent $\eta^{(t)}$ is used (e.g., "move" more at the beginning than at the end)

Note: there are guarantees on the number of iterations required by GD to return a *good* value of $\bar{\mathbf{w}}$ under some assumptions on f (see the book for details)

Stochastic Gradient Descent (SGD)

Idea: instead of using exactly the gradient, we take a (random) vector with *expected value* equal to the gradient direction.

SGD algorithm:

$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$; // or $\vec{w}^{(0)} \leftarrow$ random vector in $\mathbb{R}^d$

for $t \leftarrow 0$ *to* $T-1$ **do**

  choose $\mathbf{v}_t$ at random from distribution such that $\mathbf{E}[\mathbf{v}_t|\mathbf{w}^{(t)}] \in \nabla f(\mathbf{w}^{(t)})$;

  /* $\mathtt{v}_t$ has *expected value* equal to the gradient of $f(\mathtt{w}^{(t)})$ */

  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \mathbf{v}_t$;

return $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}^{(t)}$;

Stochastic Gradient Descent (SGD)

Idea: instead of using exactly the gradient, we take a (random) vector with *expected value* equal to the gradient direction.

SGD algorithm:

$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;

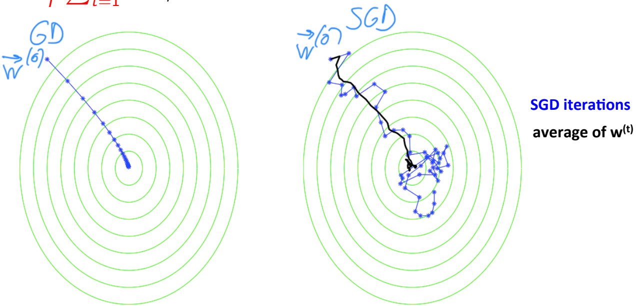
for $t \leftarrow 0$ *to* $T-1$ **do**

    choose $\mathbf{v}_t$ at random from distribution such that $\mathbf{E}[\mathbf{v}_t | \mathbf{w}^{(t)}] \in \nabla f(\mathbf{w}^{(t)})$;

    `/*` $\mathtt{v}_t$ `has` *expected value* `equal to the gradient of` $f(\mathtt{w}^{(t)})$ `*/`

    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \mathbf{v}_t$;

return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;



Note: there are guarantees on the number of iterations required by SGD to return a *good*, *in expectation*, value of $\bar{\mathbf{w}}$ under some assumptions on f (see the book for details)

Why should we use SGD instead of GD?

$S = \{(\vec{x}_1, y_1), \dots (\vec{x}_m, y_m)\}$

Question: when do we use GD in the first place?

Answer: for example to find $\mathbf{w}$ that minimizes $L_S(\mathbf{w})$

That is: we use GD for $f(\mathbf{w}) = L_S(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \ell(h_{\vec{w}}, (\vec{x}_i, y_i))$

Why should we use SGD instead of GD?

Question: when do we use GD in the first place?

Answer: for example to find $\mathbf{w}$ that minimizes $L_S(\mathbf{w})$

That is: we use GD for $f(\mathbf{w}) = L_S(\mathbf{w})$
$\Rightarrow \nabla f(\mathbf{w})$ depends on all pairs $(\mathbf{x}_i, y_i) \in S, i = 1, \ldots, m$: may require long time to compute it!

What about SGD?

We need to pick $\mathbf{v}_t$ such that $\mathbf{E}[\mathbf{v}_t|\mathbf{w}^{(t)}] \in \nabla f(\mathbf{w}^{(t)})$: **how**?

Pick a random $(\mathbf{x}_i, y_i) \in S \Rightarrow$ pick $\mathbf{v}_t \in \nabla \ell(\mathbf{w}^{(t)}, (\mathbf{x}_i, y_i))$:

- satisfies the requirement!
- requires much less computation than GD

Analogously we can use SGD for regularized losses, etc.

Back to Our Linear Classification Problem

- binary classification problem: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$
- with linear models
- with loss $\ell(\mathbf{w}, (\mathbf{x}, y)) = \max\{0, -y\langle \mathbf{w}, \mathbf{x}\rangle\}$.

How to find the ERM solution? SGD!

SGD for Linear Classification

SGD: take i uniformly at random from $\{1, \dots, m\}$. Let $(\vec{x}', y')$ be the corresponding point in the training set, and consider the vector $\nabla \ell(\vec{w}, (\vec{x}', y'))$

Note that GD considers (as gradient of the function to minimize):

$$\nabla L_S(\vec{w}) = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(\vec{w}, (\vec{x}_i, y_i))$$

and for SGD we have:

$$\mathbb{E}\left[\nabla \ell(\vec{w}, (\vec{x}', y'))\right] = \sum_{i=1}^{m} \overbrace{\Pr\left[(\vec{x}', y') = (\vec{x}_i, y_i)\right]}^{\frac{1}{m}\ \forall i \text{ (uniform distribution)}} \cdot \nabla \ell(\vec{w}, (\vec{x}_i, y_i))$$

$$= \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(\vec{w}, (\vec{x}_i, y_i))$$

$$= \nabla L_S(\vec{w})$$

SGD algorithm:

$\vec{w}^{(0)} \leftarrow \vec{0}$;

for $t \leftarrow 0$ to $T-1$ do{

pick i uniformly at random from $\{1, \dots, m\}$;

$$\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta \underbrace{\nabla \ell(\vec{w}^{(t)}, (\vec{x}_i, y_i))}_{?} \qquad (\star)$$

}

return $\bar{w} = \left(\sum_{t=1}^{T} \vec{w}^{(t)}\right) / T$;

To simplify the notation, we are going to look at $\nabla \ell(\vec{w}, (\vec{x}_i, y_i))$

$$\nabla \ell(\vec{w}, (\vec{x}_i, y_i)) = \begin{cases} \vec{0} & \text{if } y_i \langle \vec{w}, \vec{x}_i \rangle > 0 \\ \nabla(-y_i \langle \vec{w}, \vec{x}_i \rangle) & \text{otherwise} \end{cases}$$

(if $\vec{x}_i$ is correctly classified by $\vec{w}$)

(if $\vec{x}_i$ is missclassified by $\vec{w}$)

Assume $y_i \langle \vec{w}, \vec{x}_i \rangle < 0$:

$$\nabla\left(-y_i \langle \vec{w}, \vec{x}_i \rangle\right) = \left[\frac{\partial(-y_i \langle \vec{w}, \vec{x}_i \rangle)}{\partial w_1}, \dots, \frac{\partial(-y_i \langle \vec{w}, \vec{x}_i \rangle)}{\partial w_d} \right]^T$$

Let $\vec{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{id} \end{bmatrix}$. Since $-y_i \langle \vec{w}, \vec{x}_i \rangle = -y_i \cdot \sum_{j=1}^{d} (w_j x_{ij})$

$$\Rightarrow \frac{\partial(-y_i \langle \vec{w}, \vec{x}_i \rangle)}{\partial w_j} = -y_i x_{ij}$$

$$\Rightarrow \nabla \ell(\vec{w}, (\vec{x}_i, y_i)) = \left[-y_i x_{i1}, -y_i x_{i2}, \dots, -y_i x_{id} \right]^T$$

$$= -y_i \vec{x}_i$$

Therefore, in the pseudocode, $(*)$ is replaced by:

$\quad$ if $y_i \langle \vec{w}^{(t)}, \vec{x}_i \rangle < 0$ then {

$\quad\quad \vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} + \eta\, y_i \vec{x}_i$

$\quad$ }

Comparison: "perceptron" vs "SGD perceptron"

perceptron	vs	SGD perceptron
1) choose a missclassified point		choose a point at random (not only a missclassified)
2) $\eta = 1$		η
3) return "best" $\vec{w}^{(t)}$		return $\vec{w}$

Main difference: 1), but you can "speed up" the "SGD perceptron" by, at each iteration, pick a missclassified point at random

$\Rightarrow$ "SGD perceptron" is the "perceptron"

PROBABILITY DISTRIBUTION

$\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$

data

$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots$

$$\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^{d+1}$$

training data:

$(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)$

test data

$(\vec{x}_{t_1}, y_{t_1}), \ldots, (\vec{x}_{t_n}, y_{t_n})$

HYPOTHESIS/MODEL SET:

$\mathcal{H}$

$$h_{\vec{w}}(\vec{x}) = \mathrm{sign}\left(w_0 + \sum_{i=1}^{d} x_i w_i\right) = \mathrm{sign}(\langle \vec{w}, \vec{x} \rangle)$$

LOSS FUNCTION

$\ell()$

$$\ell(h_{\vec{w}}, (\vec{x}, y)) = \max\{0, -y\langle \vec{w}, \vec{x} \rangle\}$$

ALGORITHM

$\mathcal{A}$

SGD

(perceptron)

FINAL HYPOTHESIS/MODEL

$h_{\vec{w}^*}$ (fix $\vec{w}^*$)

estimate of the generalization error for $h_{\vec{w}^*}$ ($L_{\mathcal{D}}(h_{\vec{w}^*})$)

ERM: pick $\vec{w}^*$ that gives the minimum training error