Beginning of Sept. 1, 2022

For non-convex objectives, however, a stationary point may be a saddle point, and there exist functions and saddle points around which GD will get stuck. Even worse, it is *NP-hard* to distinguish saddle points from local minimum.

Stochastic Gradient Descent

Instead of always moving in the negative gradient direction, we consider an algorithm that moves in the *noisy* negative gradient direction given by

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \tilde{\nabla} F(w^{(t)})$$

where $\tilde{\nabla} F(w^{(t)})$ is an unbiased random variable called the **stochastic gradient** such that

$$\mathbb{E}[\tilde{\nabla} F(w^{(t)})] = \nabla F(w^{(t)}).$$

SGD usually needs more iterations to obtain convergence but each iteration takes less time. In some examples SGD can be significantly faster, e.g. when GD fails due to "bad" saddle points.

Second-Order Methods

Taylor approximation gives

$$F(w) = F(w^{(t)}) + \nabla F(w^{(t)})^T (w - w^{(d)}) + \frac{1}{2}(w - w^{(t)})^T H_t (w - w^{(t)}) + \mathcal{O}(\|w - w^{(t)}\|)^3$$

where H_t is the Hessian $\left\{ \frac{\partial^2 F(w)}{\partial w_i \partial w_j} \right\}_{i,j}$ evaluated at $w = w^{(t)}$. We define the second-order approximation

$$\tilde{F}(w) := F(w^{(t)}) + \nabla F(w)^T (w - w^{(t)}) + \frac{1}{2}(w - w^{(t)})^T H_t (w - w^{(t)}).$$

Differentiating $\tilde{F}$ gives

$$\nabla \tilde{F}(w) = \nabla F(w^{(t)}) + H_t w - H_t w^{(t)}/2 - H_t w^{(t)}/2.$$

Setting this to zero we obtain

$$H^{(t)} w = H_t w^{(t)} - \nabla F(w^{(t)})$$

and so

$$w = w^{(t)} - H_t^{-1} \nabla F(w^{(t)}).$$

This iteration ($w \leftarrow w - H_t^{-1} \nabla F(w) = w - (\nabla^2 F(w))^{-1} \nabla F(w)$) is called **Newton's method**.

Newton's Method v.s. Gradient Descent

- Step size: Newton's method doesn't have any but GD requires one.
- Rate/order of convergence: Newton's method converges much faster (quadratic convergence).
- Complexity: Newton's method requires inversion of Hessians which is of $\mathcal{O}(d^3)$, but GD only takes $\mathcal{O}(d)$.

0.1 Linear Classifiers

Recall we have:

- input $x \in \mathbb{R}^d$,
- output (label): $y \in [C] := \{1, 2, ...C\}$, and
- goal: a mapping $f : \mathbb{R}^d \to [C]$ that predicts reliable outcomes based on input.

We first look at **binary classification**, where $C = 2$.

Binary Classification

Let $\mathcal{F}$ be the class of linear functions. Conveniently when $C = 2$ we make use of the sign function.

Definition: Separating Hyperplanes & Linear Predictors

In $\mathbb{R}^d$, we define the function class of **separating hyperplanes** to be

$$\mathcal{F} := \{f : \mathbb{R}^d \to \{-1, 1\} : f(x) = \text{sgn}(w^T x) \text{ where } w \in \mathbb{R}^d\}.$$

For notational simplicity we denote it as $\{f(x) = \text{sgn}(w^T x) : w \in \mathbb{R}^d\}$ and similarly onward.

Picking a Loss Function

With this setup, we now need to choose an appropriate loss function. The most common choice for binary classification is

$$\ell(f(x), y) := \mathbf{1}[f(x) \neq y]$$

(i.e. the inficator function). The **0/1 loss function** is

$$\ell_{0-1}(yw^T x) := \mathbf{1}[(yw^T x \leqslant 0)].$$

The function returns 1 when $w^T x \neq y$ (i.e., 1 and -1 or reversed). However, such function is non-convex and inconvenient mathematically and computationally. We instead consider a **convex surrogate loss.**

- The **perceptron loss** is defined as $\ell(z) := \max\{0, -z\}$ (used in Perceptron),
- The **hinge loss** is defined as $\ell(z) := \max\{0, 1 - z\}$ (used in SVM and many others), and
- The **logistic loss** $\ell(z) := \log(1 + \exp(-z))$ (used in logistic regression).

GD on Perceptron Loss

We now try to find the ERM

$$w^* = \underset{w \in \mathbb{R}^d}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i w^T x_i)$$

where ℓ is a convex surrogate loss. In general there will not be a closed form solution but we have numerical toolboxes.

Perceptron Loss and the Perceptron Algorithm

We first focus on Perceptron loss:

$$F(w) = \frac{1}{n}\sum_{i=1}^{n} \max\{0, -y_i w^T x_i\}. \tag{*}$$

The gradient of $\max\{0, -z\}$ is 0 if $z \geqslant 0$ and -1 otherwise.
Applying GD to (*) gives

$$\nabla F(w) = -\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[y_i w^T x_i \leqslant 0] y_i x_i,$$

so we iteratively define

$$w \leftarrow w + \frac{\eta}{n}\sum_{i=1}^{n} \mathbf{1}[y_i w^T x_i \leqslant 0] y_i x_i. \tag{GD.1}$$

After every iteration, we need to update the entire training set, which can be costly.
Alternatively we can consider SGD, as follows:

- Pick one index $i \in [n]$ uniformly at random, and let

$$\tilde{\nabla} F(w^{(t)}) = -\mathbf{1}[y_i w^T x_i \leqslant 0] y_i x_i.$$

- Use $\tilde{\nabla}$ as the stochastic gradient.

- To see it is unbiased:

$$\mathbb{E}[\tilde{\nabla} F(w^{(t)})] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[y_i w^T x_i \leqslant 0] y_i x_i = \nabla F(w^{(t)}).$$

- We update the SGD by

$$w \leftarrow w + \eta \mathbf{1}[y_i w^T x_i \leqslant 0] y_i x_i. \tag{SGD.1}$$

In comparison, (SGD.1) is significantly faster than (GD.1) as each update only changes one data point at most.
The **Perceptron algorithm** is simply (SDG.1) with $\eta = 1$. That is, we initialize $w = 0$, and repeat the following until convergence:

- Pick $x_i \sim$ unif$(x_1, ..., x_n)$,

- If $\text{sgn}(w^T x_i) \neq y_i$:

$$w \leftarrow w + y_i x_i.$$

Intuition. Say our w makes mistake on (x_i, y_i), i.e., $y_i w^T x_i < 0$ or equivalently $\text{sgn}(w^T x_i) \neq y_i$. We consider $w' := w + y_i x_i$. At least for this one data point,

$$y_i (w')^T x_i = y_i w^T x_i + y_i^2 x_i^T x = y_i w^T x_i + y_i^2 \|x_i\|^2.$$

If $x_i \neq 0$ then $\|x_i\| > 0$ and $y_i (w')^T x_i > y_i w^T x_i$. That is, our Perceptron algorithm pushes $y_i w^T x_i$ towards the positive numbers, which we want.

Logistic Loss

We now look at logistic loss, which is given by

$$F(w) = \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i w^T x_i)).$$

Instead of $\{\pm 1\}$, we "predict the probability", i.e., regress on probability. One way to model probability using the sigmoid function is

$$\mathbb{P}(y = \pm 1 \mid x; w) = \sigma(w^T x) \qquad \text{where} \qquad \sigma(z) = \frac{1}{1+e^{-z}}.$$

Some immediate properties of the sigmoid function:

- $\lim_{x\to-\infty}\sigma(x) = 0$ and $\lim_{x\to\infty}\sigma(x) = 1$. Monotone increasing and differentiable; good for probability.
- $\sigma(w^T x) \gtrless 0$ if and only if $w^T x \gtrless 0$, consistent with predicting the label with $\operatorname{sgn}(w^T x)$.
- The larger $w^T x$ is, the larger $\sigma(w^T x)$ is, and thus the higher *confidence* in labelling the point as 1.
- $\sigma(z) + \sigma(-z) = 1$ for all z, corresponding to

$$\mathbb{P}(y = -1 \mid x; w) + \mathbb{P}(y = 1 \mid x; w) = \sigma(-w^T x) + \sigma(w^T x) = 1.$$

Because of these, it is natural to model $\mathbb{P}(y \mid x; w)$ by

$$\mathbb{P}(y \mid x; w) := \sigma(y w^T x) = \frac{1}{1+\exp(-y w^T x)}.$$

MLE on Sigmoid v.s. Minimizing Logistic Loss

What we observe as labels, *not* probabilities. Taking a probabilistic view, we consider "what is the probability of seeing labels $y_1, ..., y_n$ given $x_1, ..., x_n$, when all of these are generated by some w? The **maximum likelihood estimator**, MLE, maximizes the following probability

$$\mathbb{P}(w) = \prod_{i=1}^{N}\mathbb{P}(y_i \mid x_i; w).$$

Some math using monotonicity of $\log$:

$$\begin{aligned}
w^* &= \operatorname*{argmax}_w P(w) = \operatorname*{argmax}_w \prod_{i=1}^{N}\mathbb{P}(y_i \mid x_i; w) \\
&= \operatorname*{argmax}_w \sum_{i=1}^{n}\log\mathbb{P}(y_i \mid x_i; w) \\
&= \operatorname*{argmin}_w \sum_{i=1}^{n} -\log\mathbb{P}(y_i \mid x_i; w) \\
&= \operatorname*{argmin}_w \sum_{i=1}^{n}\log(1+\exp(-y_i w^T x_i)) \\
&= \operatorname*{argmin}_w \sum_{i=1}^{n}\ell_{\text{logistic}}(y_i w^T x_i) = \operatorname*{argmin}_w F(w).
\end{aligned}$$

That is, minimizing the logistic loss is *exactly* finding MLE for the sigmoid model.

SGD on Logistic Loss

Recall we can sample one index uniformly at random and make the corresponding loss function on $y_i w^T x_i$ the stochastic gradient. That is,

$$\begin{aligned} w &\leftarrow w - \eta \tilde{\nabla} F(w) \\ &= w - \eta \nabla_w \ell_{\text{logistic}}(y_i w^T x_i) \\ &= w - \eta \cdot \left[\left. \frac{\partial \ell_{\text{logistic}}(z)}{\partial z} \right|_{z = y_i w^T x_i} \right] y_i x_i \\ &= w - \eta \cdot \left[\left. \frac{-e^{-z}}{1 + e^{-z}} \right|_{z = y_i w^T x_i} \right] y_i x_i \\ &= w + \eta \sigma(-y_i w^T x_i) y_i x_i \\ &= w + \eta \mathbb{P}(-y_i \mid x_i; w) y_i x_i \end{aligned}$$

using our previously defined probability model for $\mathbb{P}(y \mid x; w)$. Note that this is a "soft version" of Perceptron (where $\mathbf{1}[\text{sgn}(w^T x_i) \neq y_i]$ is discontinuous, $\mathbb{P}(-y_i \mid x_i; w)$ is smooth; for the normal Perceptron we either move a lot or stay, whereas in this SGD we always move, though sometimes very little).