VI.5 Stochastic Gradient Descent + ADAM

In data science, different loss functions are used to measure how well our model fits the given training data:

$$(v_i, y_i) \in \mathbb{R}^m \times \mathbb{R}^k, \quad i = 1, \ldots, N.$$

We want to find a model

$$\hat{y} = F(x, v)$$

that depends on the parameters $x \in \mathbb{R}^n$ so that

$$y_i \approx F(x, v_i), \quad i = 1, \ldots, p.$$

Minimizing a loss function over the parameters x ensures this is a good approximation.

Square loss: (General $y_i \in \mathbb{R}^k$)

$$L(x) = \frac{1}{N} \sum_{i=1}^{N} \| F(x, v_i) - y_i \|_2^2$$

Hinge loss: ($y_i = 1$ or -1)

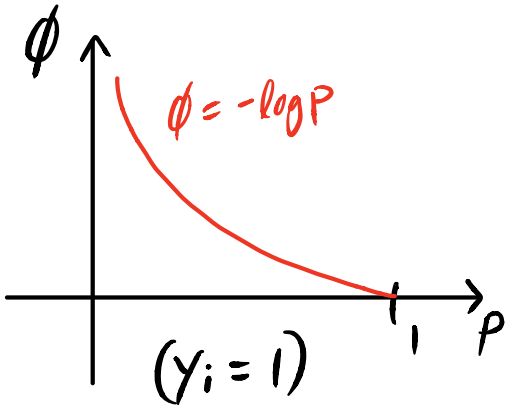
$$L(x) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i F(x, v_i))$$

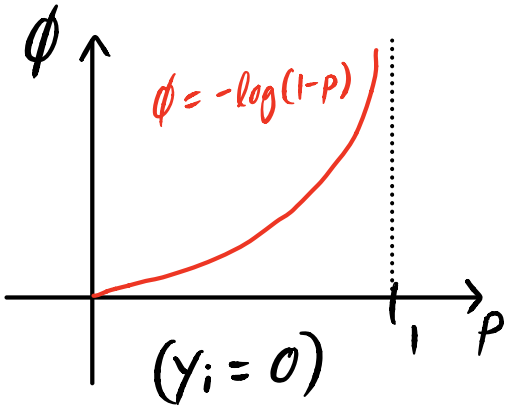
Cross-entropy loss: ($y_i = 0$ or 1)

(aka "logistic loss")

$$L(x) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log \hat{y}_i(x) + (1 - y_i) \log (1 - \hat{y}_i(x)) \right]$$

where $\hat{y}_i(x) = F(x, v_i) \in (0, 1)$.





In each case $L: \mathbb{R}^n \to \mathbb{R}$ and we want to solve

$$\min_x L(x).$$

To do this, we need to compute the gradient $\nabla L(x)$ so we can use a gradient method.

However, in applications the number of data points N is <u>very large</u>, so computing $L(x)$ and $\nabla L(x)$ is <u>too expensive</u>.

Instead of evaluating $L(x)$ and $\nabla L(x)$ using <u>all</u> the data, we only use <u>some</u> of the data in each iteration of the optimization algorithm.

Stochastic Gradient Descent (SGD):

Each such subset of the data is called a <u>minibatch</u> of B data points.

Let S_k be the minibatch that is used in the k^{th} iteration. We define

$$L_k(x) = \frac{1}{B} \sum_{i \in S_k} \ell_i(x),$$

where $\ell_i(x)$ is the loss of x on the i^{th} data point, such as:

$$\ell_i(x) = \| F(x, v_i) - y_i \|_2^2,$$

$$\ell_i(x) = \max\left(0, 1 - y_i F(x, v_i)\right),$$

$$\ell_i(x) = y_i \log \hat{y}_i(x) + (1 - y_i) \log\left(1 - \hat{y}_i(x)\right).$$

In iteration k, we use $\nabla L_k(x_k)$ instead of $\nabla L(x_k)$ to compute the next iterate x_{k+1}.

These minibatches are selected randomly (or stochastically), so this is a stochastic gradient descent method.

The theoretical convergence analysis assumes minibatches are drawn randomly from the data, with replacement. However, in practice the minibatches are obtained by randomly shuffling the data and then traversing it in that order.

Each traversal through all the minibatches (ie all data points) is refered to as an epoch. Training a neural network typically requires many epochs.

Initially we expect the loss function to decrease rapidly, but as it gets closer to a local minimum, we observe the loss function fluctuating and the iterates moving around randomly in a region of confusion. Doing a weighted average of the iterates in this region can help, and is called stochastic weight averaging (SWA).

Stochastic gradient descent converges in expectation:

$$\min_{1 \le k \le T} \mathbb{E}\left[\|\nabla L(x_k)\|_2^2\right] \le \frac{C}{\sqrt{T}}$$

after T iterations. As $T \to \infty$, the minumum expected value of the norm of the gradient approaches zero.

Adaptive methods

The most popular SGD methods for training neural networks are adaptive:

$$d_k = d(\nabla L_k, \nabla L_{k-1}, \ldots, \nabla L_0)$$

$$s_k = s(\nabla L_k, \nabla L_{k-1}, \ldots, \nabla L_0).$$

That is, the direction d_k and step length s_k use all the previous gradients, where

$$x_{k+1} = x_k - s_k d_k.$$

In the ADAM (adaptive moment estimation) method, d_k and s_k are given by

$$d_k = \delta d_{k-1} + (1-\delta)\nabla L_k(x_k),$$
$$s_k^2 = \beta s_{k-1}^2 + (1-\beta)\|\nabla L_k(x_k)\|^2,$$

where $0<\delta<1$ and $0<\beta<1$ are the "forgetting factors" (eg $\delta = 0.9$ and $\beta = 0.999$) [see stochastic gradient descent on Wikipedia].