Stochastic gradient descent

Like usual gradient descent, Stochastic gradient descent is motivated to minimize an average of functions:

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{\beta}).$$

[For regression: $f_i(\boldsymbol{\beta}) = (y_i - \mathbf{z}_i^T \boldsymbol{\beta})^2$, where $x_i = (y_i, z_i)$.]

The 'exact' gradient of the above objective function will be $\frac{1}{n} \sum_{i=1}^{n} \frac{\partial f_i(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$ and thus the gradient descent algorithm will repeat the following steps:

$$\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k-1)} - t_k \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{\partial f_i(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k-1)}}$$

However, gradient descent will be very costly if we have n very large or have streaming data with increasing n.

Then, we can apply the following algorithm:

$$\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k-1)} - t_k \cdot \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k-1)}}$$

This algorithm is called the stochastic gradient descent or SGD (or incremental gradient descent). The main appeal and motivations of using SGD are: 1) The iteration cost of SGD is independent of m 2) SGD can be a big savings in terms of memory usage.

Note: SGD is not necessarily a descent method!

Two rules for choosing index i_k at iteration k:

- Randomized rule: choose $i_k \in 1 \ldots, n$ uniformly at random.
- Cyclic rule: choose $i_k = 1, 2, \ldots, n, 1, 2, \ldots, n, \ldots$.

Randomized rule is more common in practice. For randomized rule, note that $\mathbb{E}\left(\frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right) = \mathbb{E}\left(\frac{\partial f_i(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right) \approx \frac{1}{n} \sum_i \frac{\partial f_i(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$ using the sample mean-population mean argument. So we can view SGD as using an unbiased estimate of the gradient at each step.

1 Step Sizes

It is standard to use diminishing step sizes in SGD (e.g., $t_k = 1/k$). The short (and intuitive) explanation for why we do not use fixed step sizes is the following. Suppose we

take cyclic rule for simplicity. We have $t_k = t$ for n updates in a row, so for SGD we get $\boldsymbol{\beta}^{(k+n)} = \boldsymbol{\beta}^{(k)} + t\sum_{i=1}^{n} \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k+i-1)}}$.

Meanwhile, for full gradient method with step size nt, we have $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + t\sum_{i=1}^{n} \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k)}}$.

Consider the error between the above two updates: $t[\sum_{i=1}^{n} \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k+i-1)}} - \sum_{i=1}^{n} \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k)}}]$.

If we keep t constant, this difference will in general not go to zero. Hence, usually using a diminishing step sizes will make stochastic update a more accurate estimate of the full gradient estimate.

2 Mini-batch SGD

Since people in machine learning often want better performance on the samples outside the training sets, which makes fully optimization of the objectives unnecessary, therefore we can always stop early on an "okay" solution. To people actually want the optimization, we show a common way to improve the convergence rate of SGD, which the Mini-batch SGD. Mini-batch SGD For the optimization problem described early, during the k-th update, we choose a random subset $I_k \subseteq \{1, \ldots, n\}$, where $|I_k| = b \ll n$, and use the following update rule:

$$\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k-1)}}$$

As before, the gradient estimate is unbiased as $\frac{1}{b}\sum_{i \in I_k} \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathbb{E}\left(\frac{\partial f_i(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right) \approx \frac{1}{n}\sum_i \frac{\partial f_i(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$ using the sample mean-population mean argument. and its variance is reduced by a factor $1/b$, though at the cost of b times more expensive running time at each iteration. Therefore, one may regard Mini-batch SGD as a compromise between SGD and full gradient descent.

Theoretically, it does not lead to any computational gain in terms of convergence rate. Though it is not convincing to use Mini-batch SGD from the theory aspect, Mini-batch SGD turns out performing not too bad in practice.

3 SGD with momentum

SGD is often augmented with a momentum term, so that the update in,

$$\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k-1)} - t_k \cdot \frac{\partial f_{i_k}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k-1)}} + \boldsymbol{\beta}(\boldsymbol{\beta}^{(k-1)} - \boldsymbol{\beta}^{(k-2)})$$

where $\boldsymbol{\beta}$ is an exponential decay factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients to the weight change.

Remark As we have discussed, SGD can be very effective in terms of resources (per iteration cost, memory), and for this reason it is still the standard for solving large problems. However, SGD is slow to converge, and does not benefit from strong convexity unlike full gradient descent. Mini-batch SGD can be an approach to trade off effectiveness and convergence rate: it is not that beneficial in terms of ops, but it can still be useful in practice in settings where the cost of computation over a batch can be brought closer to the cost for a point. Still, while mini-batches help reducing the variance, they do not necessarily lead to faster convergence in the theoretical domain. Due to these properties, it was thought for a while that SGD would not be commonly useful. It was recognized very recently that these lower bounds (which were established for a more general stochastic problem) do not apply to finite sums, and with the new wave of variance reduction methods, it was shown that we can modify SGD to converge much faster for finite sums.

4 Nesterov Acceleration

Nesterov's accelerated gradient (NAG) is a momentum-based optimization technique that improves upon the classical momentum method discussed above. The key intuition is that instead of computing the gradient at the current parameters, NAG looks *ahead* at a projected future position based on the momentum term.

Let $f(\boldsymbol{\beta})$ denote the loss function and $\boldsymbol{\beta}^{(t)}$ the parameters at iteration t. The classical momentum update is:

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \nabla f(\boldsymbol{\beta}^{(t)}), \\ \boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} + v_{t+1}, \end{aligned}$$

where η is the learning rate and μ is the momentum parameter.

Nesterov acceleration modifies this to compute the gradient at the lookahead point:

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \nabla f(\boldsymbol{\beta}^{(t)} + \mu v_t), \\ \boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} + v_{t+1}. \end{aligned}$$

This results in faster convergence because it anticipates the effect of the accumulated momentum before applying the gradient correction.

5 Batch Normalization

Batch normalization (BN) is a normalization layer designed to stabilize and accelerate non-parametric and nonlinear regression problems, such as deep neural networks or even GLM.

It normalizes the data, reducing the problem of internal covariate shift. **Normalization is an essential step for any nonparametric or nonlinear regression.** For illustration, we consider the standard scaling as the normalization where each predictor is scaled as $(\mathbf{x}-\mu)/\sigma$ where μ =Mean($\mathbf{x}$) and σ =SD($\mathbf{x}$). However, we run this batch-wise and further update the associated parameters (like the μ and σ) with every passing batch. The updated running mean and sd are applied in the evaluation stage on the test set.

Given a mini-batch of activations

$$B = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}, \quad \mathbf{x}_i \in \mathbb{R}^d,$$

BN first computes the batch mean and variance for each feature dimension:

$$\boldsymbol{\mu}_B = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i, \qquad \boldsymbol{\sigma}_B^2 = \{\frac{1}{m}\sum_{i=1}^{m}(x_{k,i}-\mu_{k,B})^2\}_{k=1}^{d}.$$

Then normalize:

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}},$$

where $\epsilon > 0$ is a small constant for numerical stability.

Finally, a learnable affine transformation is applied:

$$\mathbf{z}_i = \gamma\hat{\mathbf{x}}_i + \beta,$$

where γ (scale) and β (shift) are trainable parameters learned jointly with the model. Then $\mathbf{z}_i$ becomes the input.

5.1 Running Mean and Variance Updates

During training, each mini-batch has its own mean and variance (μ_B, σ_B^2), which may fluctuate across iterations due to stochastic sampling.

To ensure stable behavior during inference, BN maintains *running estimates* of the population mean and variance using an **exponential moving average (EMA)**:

$$\text{running\_mean}_t = \rho\,\text{running\_mean}_{t-1} + (1-\rho)\,\mu_B,$$

$$\text{running\_var}_t = \rho\,\text{running\_var}_{t-1} + (1-\rho)\,\sigma_B^2,$$

where $\rho \in [0, 1)$ is the **momentum** (typically 0.9).

This update rule ensures that the running statistics become a smoothed estimate of the true dataset-wide statistics:

$$\text{running\_mean}_t = (1-\rho)\sum_{k=0}^{t-1}\rho^k \mu_{B_{t-k}},$$

which weights recent batches more strongly but still remembers older ones.

5.2 Training vs. Inference Behavior

During Training: BN uses the **current batch statistics** (μ_B, σ_B^2) to normalize activations:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}.$$

The running estimates are updated in the background but not used directly for normalization.

During Test set prediction: BN switches to a deterministic mode, using the stored running estimates instead of batch statistics:

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \text{running\_mean}}{\sqrt{\text{running\_var} + \epsilon}}.$$

This ensures consistent behavior regardless of batch composition. We use $\hat{\mathbf{x}}_i$ as the input.

6 Few more extensions

There are some Newton-like adjustments to set parameter-specific different learning rates. You may check those out. Specifically, these methods use the sequence of updates to quantify different aspects of the loss functions.

6.1 AdaGrad (Adaptive Gradient Algorithm)

Scale each parameter's learning rate by the inverse root of the sum of its past squared gradients, giving larger steps for rarely-updated (sparse) features and smaller steps for frequently-updated ones.

For parameters $\boldsymbol{\beta} \in \mathbb{R}^d$ and gradient $g_t = \nabla_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}^{(t)})$,

$$G_t \;=\; G_{t-1} + g_t \odot g_t \quad \text{(elementwise square, } G_0 = 0\text{)},$$

$$\boldsymbol{\beta}^{(t+1)} \;=\; \boldsymbol{\beta}^{(t)} \;-\; \eta\,\frac{g_t}{\sqrt{G_t}+\epsilon},$$

where division is elementwise, η is the base learning rate, and $\epsilon > 0$ avoids division by zero.

6.2 RMSProp (Root Mean Square Propagation)

Fix AdaGrad's ever-growing accumulator by using an exponentially decaying average of past squared gradients.

$$v_t \;=\; \rho\, v_{t-1} + (1-\rho)\, g_t \odot g_t \quad (\text{EMA of } g^2,\ v_0 = 0),$$
$$\boldsymbol{\beta}^{(t+1)} \;=\; \boldsymbol{\beta}^{(t)} \;-\; \eta\,\frac{g_t}{\sqrt{v_t}+\epsilon}.$$

6.3 Adam (Adaptive Moment Estimation)

Combine momentum (Exponential Moving Average of gradients) with RMSProp-style scaling (Exponential Moving Average of squared gradients), plus bias corrections for the EMAs.

With $m_0 = 0$, $v_0 = 0$, hyperparameters $\theta_1, \theta_2 \in [0, 1)$:

$$\begin{aligned}
m_t &= \theta_1 m_{t-1} + (1-\theta_1)\, g_t && (\text{EMA of } g),\\
v_t &= \theta_2 v_{t-1} + (1-\theta_2)\, g_t \odot g_t && (\text{EMA of } g^2),\\
\hat{m}_t &= \frac{m_t}{1-(\theta_1)^t}, \qquad \hat{v}_t = \frac{v_t}{1-(\theta_2)^t} && (\text{bias corrections}),\\
\boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} - \eta\,\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}.
\end{aligned}$$