دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

مقاله‌ی مصوب شده: A Stochastic Quasi-Newton Method for Non-convex Optimization with Non-uniform Smoothness

بهینه‌سازی پیشرفته

نام و نام‌خانوادگی: سیدرضا مسلمی

شماره دانشجویی: ۸۱۰۱۰۳۳۲۶

پاییز ۱۴۰۳

فهرست مطالب

فهرست تصاویر

فهرست جداول

گزارش ۱

گزارش روش‌های بهینه‌سازی: $ClippedSQN$ و $MinibatchSGD$

۱.۱ مقدمه

در این گزارش، دو روش بهینه‌سازی که برای حل مسائل غیرمحدب که اغلب در یادگیری ماشین با آن‌ها مواجه می‌شویم، مقایسه می‌شوند: $ClippedSQN$ و $MinibatchSGD$. این دو روش به دو مسئله مهم در یادگیری ماشین اعمال می‌شوند: رگرسیون خطی مقاوم ($RobustLinearRegression$) و رگرسیون لجستیک ($LogisticRegression$). هدف از این آزمایش‌ها بررسی عملکرد این الگوریتم‌ها از نظر سرعت همگرایی و کاهش خطای آموزش است.

در این گزارش، تعاریف توابع هزینه استفاده‌شده، گرادیان‌های مربوطه، قواعد به‌روزرسانی برای هر دو روش بهینه‌سازی و مقایسه عملکرد آن‌ها در وظایف ذکرشده ارائه می‌شود.

۲.۱ تابع هزینه رگرسیون خطی مقاوم

تابع هزینه رگرسیون خطی مقاوم برای کاهش اثر داده‌های پرت طراحی شده است، به‌طوری که در سناریوهایی که تابع هزینه مربعات کمینه ممکن است به داده‌های نویزی حساس باشد، مناسب‌تر است. این تابع به صورت

زیر تعریف می‌شود:

$$\text{Loss}_{\text{RLR}}(x) = \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{1}{2}(b_i - \mathbf{a}_i^T x)^2 + 1\right)$$

۳.۱ تابع هزینه رگرسیون لجستیک

تابع هزینه رگرسیون لجستیک همان تابع هزینه متداول کراس‌انتروپی است که برای مسائل طبقه‌بندی دودویی استفاده می‌شود. این تابع به صورت زیر تعریف می‌شود:

$$\text{Loss}_{\text{LR}}(x) = -\frac{1}{n}\sum_{i=1}^{n} \big(b_i \log(\sigma(\mathbf{a}_i^T x)) + (1-b_i)\log(1-\sigma(\mathbf{a}_i^T x))\big)$$

که در آن $\sigma(z)$ تابع سیگموید است:

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

۴.۱ گرادیان‌های توابع هزینه

۱.۴.۱ گرادیان رگرسیون خطی مقاوم

گرادیان تابع هزینه رگرسیون خطی مقاوم به صورت زیر محاسبه می‌شود:

$$\nabla L_{\text{RLR}}(x) = -\frac{1}{n}\sum_{i=1}^{n} \frac{\mathbf{a}_i^T (b_i - \mathbf{a}_i^T x)}{\frac{1}{2}(b_i - \mathbf{a}_i^T x)^2 + 1}$$

۲.۴.۱ گرادیان رگرسیون لجستیک

گرادیان تابع هزینه رگرسیون لجستیک به صورت زیر است:

$$\nabla L_{\mathrm{LR}}(x) = \frac{1}{n} \sum_{i=1}^{n} \left(\sigma(\mathbf{a}_i^T x) - b_i\right) \mathbf{a}_i$$

۵.۱ به‌روزرسانی $ClippedSQN$

روش $ClippedSQN$ از مفهوم تقریبی ماتریس هسین برای بهبود همگرایی در بهینه‌سازی غیرمحدب استفاده می‌کند. اندازه‌گام استفاده‌شده در این روش به صورت زیر تعریف می‌شود:

$$\text{size step} = \min\left(\frac{h_\beta}{2L_0\lambda_M^2}, \frac{h_\beta\epsilon}{L_0\lambda_M^2\|\nabla L\| + 1e-5}, \frac{h_\beta\epsilon}{L_1\lambda_M^2\|\nabla L\|^2 + 1e-5}\right)$$

که در آن: - $h_\beta = \lambda_m - \frac{\beta\lambda_M^2}{4}(2 + 3L_1c)$ - λ_m و λ_M با استفاده از فرمول‌های زیر محاسبه می‌شوند:

فرمول λ_m:

$$\lambda_m = \left(\delta + \frac{kappa^2 p}{q\gamma_0^4}\left(\frac{1}{\delta} + \frac{\delta\gamma_0 + kappa^2}{\gamma_0^3} + \frac{p + 2q\gamma_0^4}{2p\gamma_0}\right)\right)^{-1}$$

فرمول λ_M:

$$\lambda_M = \frac{1}{\delta\gamma_0^2 kappa^2 q}\left(\frac{a^p - 1}{a - 1}\right)$$

که در آن

$$a = 1 + \frac{2}{\delta\gamma_0 kappa^2 q} + \left(\frac{1}{\delta\gamma_0 kappa^2 q}\right)^2$$

1 Clipped SQN Function

```
def clipped_stochastic_quasi_newton():
    x_k = x_0
    grad_k = grad_func(x_k, A, b, batch_size_S1)
    grad_k_prev = grad_k
    loss_history = []

    gamma_0 = 0.9
    gamma_1 = 0.5
    L0 = np.sqrt(2*(gamma_0**2 + gamma_1**2))
    L1 = gamma_1 * np.sqrt(2)
    step_size = 1
    q = 10
    r = 5

    for k in range(num_iterations):
        if k % r == 0:
            grad_k = grad_func(x_k, A, b, batch_size_S1)
        else:
            grad_k_prev = grad_k
            grad_k = grad_k_prev + grad_func(x_k, A, b, \
                batch_size_S2) - grad_func((x_k + step_size)/\
                    grad_k, A, b, batch_size_S2)

        PI = gamma_0 * (1 + (np.exp(gamma_1/L0))) + (gamma_1**2)\
             * np.dot(grad_k_prev.T, grad_k)
        w_k = kappa**2 / PI**2

        q_k = q * PI**4 # with update

        lambda_m = compute_lambda_m(delta, kappa, p, q, gamma_0)
        lambda_M = compute_lambda_M(delta, gamma_0, kappa, q, p)

        h_beta = h_beta_c(lambda_m, lambda_M, beta, L1, c)
        step_size = min(
            h_beta / (2 * L0 * lambda_M**2),
            h_beta * epsilon / (L0 * lambda_M**2 * \
                np.linalg.norm(grad_k) + 1e-5),
            h_beta * epsilon / (L1 * lambda_M**2 * \
                np.linalg.norm(grad_k)**2 + 1e-5)
        )

        update_direction = stochastic_adaptive_lbfgs(
            x_k, (x_k + step_size)/grad_k, grad_k, grad_k_prev,\
                memory_size, delta, q, kappa, k, w_k
        )

        x_k = x_k + step_size * update_direction
        loss_history.append(loss_func(x_k, A, b))

    return x_k, loss_history
```

۶.۱ به‌روزرسانی L–BFGS تطبیقی تصادفی

روش L–BFGS تطبیقی تصادفی معکوس ماتریس هسین را با استفاده از گرادیان‌ها و به‌روزرسانی‌های قبلی مدل تقریبی می‌کند. قانون به‌روزرسانی به صورت زیر است:

$$\mathbf{H}_k \nabla L_k \approx \sum_{i=1}^{P} \rho_i s_i^T \nabla L_{k-i}$$

که در آن: - $\rho_i = \frac{1}{s_i^T y_i}$ (فاکتور مقیاس برای هر به‌روزرسانی)، - $s_i = x_{k-i} - x_{k-i-1}$ (تفاوت در پارامترها)، - $y_i = \nabla L_{k-i} - \nabla L_{k-i-1}$ (تفاوت در گرادیان‌ها).

۱.۶.۱ مراحل الگوریتم

- محاسبه s_k و y_k

تفاوت بین پارامترهای مدل فعلی و قبلی به صورت $s_k = x_k - x_{k-1}$ محاسبه می‌شود که میزان تغییرات پارامترهای مدل را اندازه‌گیری می‌کند. به طور مشابه، تفاوت در گرادیان‌ها $y_k = \text{grad}_k - \text{grad}_{k-1}$ محاسبه می‌شود.

- مقداردهی اولیه تقریب ماتریس هسین H_k

ابتدا، نرمال y_k و ضرب داخلی بین s_k و y_k محاسبه می‌شود. این مقادیر برای تعیین ثابت مقیاس c_k استفاده می‌شوند. یک تقریب قطری از ماتریس هسین H_k با مقدار $\frac{1}{c_k}$ مقداردهی اولیه می‌شود.

- محاسبه θ (شرط انحنا)

اسکالر μ_k به عنوان ضرب داخلی بین s_k و $H_k \cdot s_k$ محاسبه می‌شود. اگر ضرب داخلی $s_k^T y_k$ کوچکتر از $q \cdot \mu_k$ باشد، مقدار θ_k بر اساس شرایط انحنا محاسبه می‌شود تا گام به درستی تنظیم شود.

- محاسبه y'_k

عبارت y'_k نسخه‌ای مقیاس‌شده از تفاوت گرادیان‌ها y_k است که توسط θ_k و تقریب معکوس ماتریس هسین تنظیم می‌شود.

- به‌روزرسانی‌های حافظه L–BFGS

بردارهای s_k و y'_k به بافرهای حافظه s و $y'_p rime$ اضافه می‌شوند. مقدار ρ_k معکوس ضرب داخلی بین s_k و y'_k است و در لیست ρ ذخیره می‌شود.

- حلقه اول (تصحیح حافظه)

حلقه اول از حافظه برای تصحیح گرادیان u استفاده می‌کند و اصلاحات مربوط به گرادیان‌ها و به‌روزرسانی‌های پارامترهای گذشته را اعمال می‌کند.

- اعمال ماتریس هسین اولیه

تقریب اولیه ماتریس هسین H_k به گرادیان تصحیح‌شده u اعمال می‌شود تا یک جهت به‌روزرسانی پارامتر جدید تولید شود.

- حلقه دوم (مقیاس‌دهی)

حلقه دوم اصلاح α را بیشتر مقیاس‌دهی می‌کند با استفاده از مقادیر ρ و تفاوت‌های بین گرادیان‌های گذشته.

- بازگشت

این روش در نهایت α را برمی‌گرداند که تقریب معکوس ماتریس هسین اعمال‌شده به گرادیان است.

۲.۶.۱ فرمول‌های ریاضی

- فرمول‌های کلیدی الگوریتم

$$s_k = x_k - x_{k-1}$$

$$y_k = \nabla L_k - \nabla L_{k-1}$$

$$H_k = \frac{1}{c_k} \cdot I$$

$$c_k = \max\left(\delta, \frac{w_{k-1} y_k^T y_k}{s_k^T y_k}\right)$$

$$\theta_k = \frac{(1-q)\mu_k}{\mu_k - s_k^T y_k} \quad \text{اگر} \quad s_k^T y_k < q\mu_k$$

۶.۱. به‌روزرسانی L-BFGS تطبیقی تصادفی

$$y'_k = w_{k-1} \left(\theta_k y_k + (1-\theta_k) H_k s_k\right)$$

$$\mathbf{H}_k \nabla L_k \approx \sum_{i=1}^{P} \rho_i s_i^T \nabla L_{k-i}$$

2 Stochatsic Adaptive LBFGS Function

```
def stochastic_adaptive_lbfgs(x_k, x_k_minus_1, grad_k,\
     grad_k_minus_1, memory_size, delta, q, kappa, k, w_k_minus_1):

    global rho, s, y_prime

    s_k = x_k - x_k_minus_1
    s.append(s_k)
    y_k = grad_k - grad_k_minus_1

    y_norm = np.dot(y_k.T, y_k)
    s_y_dot = np.dot(s_k.T, y_k)

    c_k = max(delta, w_k_minus_1 * y_norm / s_y_dot)
    H_k_0 = np.eye(len(x_k)) / c_k

    mu_k = np.dot(s_k.T, np.dot(H_k_0, s_k))
    if s_y_dot < q * mu_k:
        theta_k = ((1 - q) * mu_k) / (mu_k - s_y_dot)
    else:
        theta_k = 1

    y_prime_k = w_k_minus_1 * (theta_k * y_k + (1 - theta_k)\
         * np.dot(H_k_0, s_k))

    y_prime.append(y_prime_k)

    rho_k = [1 / np.dot(s_k.T, y_prime_k)]
    rho.append(rho_k)
    u = grad_k.copy()
    nu = []
    P = min(memory_size, k)

    for i in range(P):
        nu_i = rho[k-i-1][0] * np.dot(u.T, s[k-i-1])
        nu.append(nu_i)
        u = u - nu_i * y_prime[k-i-1]

    alpha = np.dot(1/c_k, u)

    for i in range(P):
        beta = rho[k-P+i] * np.dot(alpha[i].T, y_prime[k-P+i])
        alpha = alpha + (nu[P-i-1] - beta) * s[k-P+i]

    return alpha  # H_k * grad_k
```

۷.۱ به‌روزرسانی $MinibatchSGD$

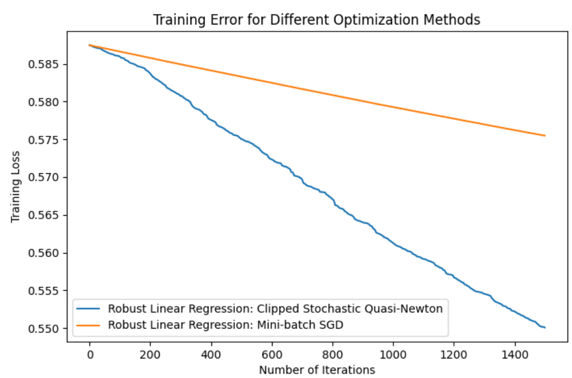
در $MinibatchSGD$، قانون به‌روزرسانی پارامترهای مدل به صورت زیر است:

$$x_{k+1} = x_k - \eta \nabla L(x_k)$$

که در آن: – η نرخ یادگیری است، – $\nabla L(x_k)$ گرادیان تابع هزینه در تکرار k است.

۸.۱ مقایسه نتایج: رگرسیون خطی مقاوم (RLR)

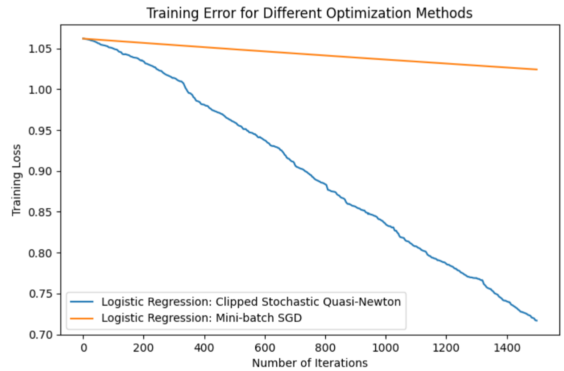
در شکل زیر، نتایج آموزش برای رگرسیون خطی مقاوم (RLR) با استفاده از دو روش بهینه‌سازی مختلف نشان داده شده است. در اینجا، $ClippedSQN$ (خط آبی) همگرایی سریع‌تری نسبت به $MinibatchSGD$ (خط نارنجی) دارد، که نشان‌دهنده عملکرد بهتر $ClippedSQN$ در این مسئله است.



شکل ۱.۱: خطای آموزش برای رگرسیون خطی مقاوم

۹.۱ مقایسه نتایج: رگرسیون لجستیک (LR)

در شکل زیر، نتایج آموزش برای رگرسیون لجستیک نیز با استفاده از دو روش بهینه‌سازی مختلف ارائه شده است. همان‌طور که در رگرسیون خطی مقاوم مشاهده می‌شود، $ClippedSQN$ (خط آبی) عملکرد بهتری دارد و سریع‌تر به پایین‌ترین خطا می‌رسد.



شکل ۲.۱: خطای آموزش برای رگرسیون لجستیک

۱۰.۱ نتیجه‌گیری

در این گزارش، روش $ClippedSQN$ بررسی شد که پیچیدگی آن از مرتبه‌ی $O(\epsilon^3)$ می‌باشد. علاوه بر این، یک روش L-BFGS تطبیقی معرفی شد که به کمک آن مقادیر ویژه تقریب معکوس هسین قابل کنترل است، و این امکان را به ما می‌دهد که سرعت همگرایی را به‌طور مؤثری تنظیم کنیم. مقایسه بین دو روش بهینه‌سازی نشان می‌دهد که $ClippedSQN$ نسبت به $MinibatchSGD$ در هر دو مسئله رگرسیون خطی مقاوم و رگرسیون لجستیک همگرایی سریع‌تری داشته و خطای آموزش کمتری را در تعداد کمتری از تکرارها به دست می‌آورد.

کتاب‌نامه