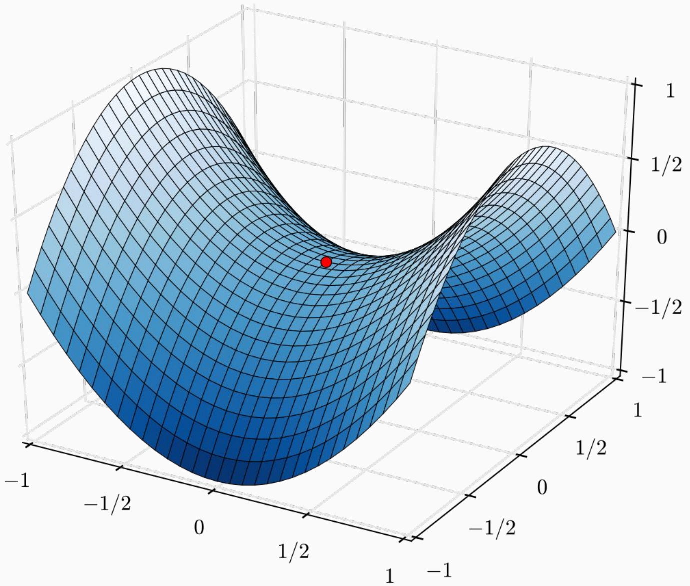
AMSC 663 Project Proposal

Marco Bornstein
Advisor: Dr. Furong Huang

Problem Formulation

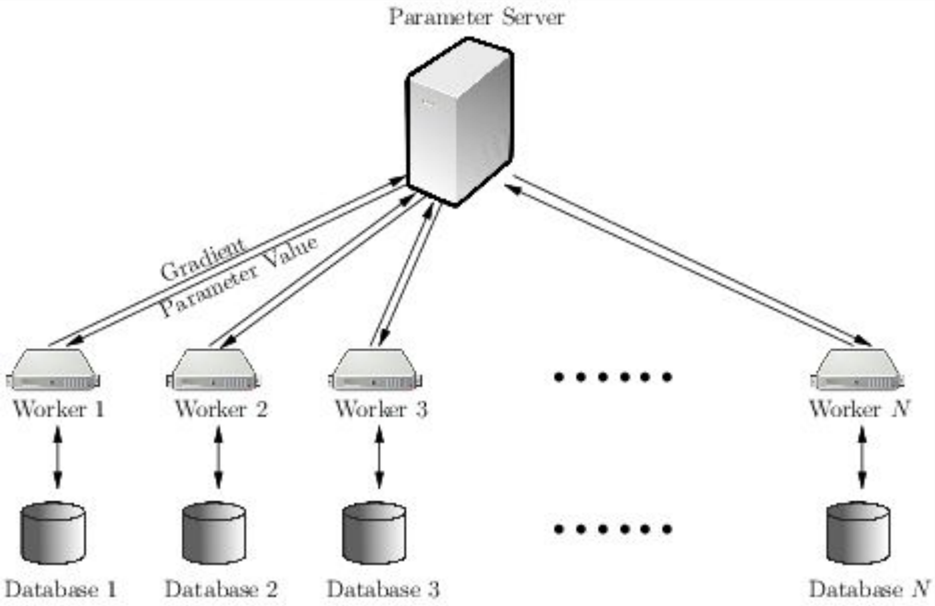
Given a non-convex function f potentially having many saddle points, what properties will guarantee asynchronous coordinate descent to escape from strict saddle points and converge to a local minima?

Non-convex Issues



- In non-convex settings, convergence to first-order stationary points is not satisfactory
- Saddle points are the main cause culprit, as they are first-order stationary yet correspond to highly suboptimal solutions
- For many non-convex problems, it is sufficient to find a local minimum

Synchronization Issues



- Parallel computing breaks data up and processes it simultaneously by multiple workers
- Algorithms (like SGD) require all computed gradients be returned to the global server before next iterate
- The speed of parallel computing thus relies on the slowest worker

Current Literature

Non-convex Optimization:

- ➔ How to Escape Saddle Points Efficiently, Jin et al. (gradient descent) https://arxiv.org/pdf/1703.00887.pdf
- ➔ On Nonconvex Optimization for Machine Learning: Gradients, Stochasticity, and Saddle Points, Jin et al. (GD/SGD) https://arxiv.org/pdf/1902.04811.pdf

Asynchronous Coordinate Descent (ACD):

- ➔ Asynchronous Coordinate Descent under More Realistic Assumptions, Sun et al. https://arxiv.org/pdf/1705.08494.pdf

Methods

Escaping saddle points:

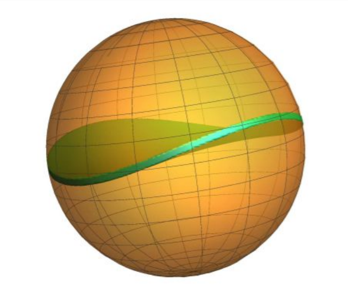
➔ Jin et al. shows that perturbing a point at a potential saddle is successful (no Hessian information needed)

Asynchronous Coordinate Descent (ACD) with delays:

➔ Sun et al. provides framework to prove that asynchronous block coordinate descent converges for bounded delays

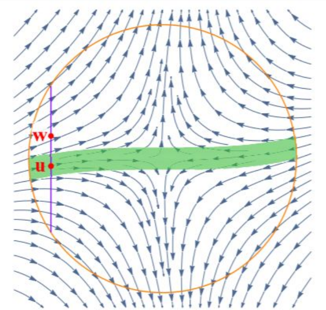
Escaping Saddle Points

- ➔ Say that a point x^s is stuck at a saddle point
- ➔ Taking a ball of radius r (perturbation ball) centered at x^s, select a point over a uniform distribution to be a perturbed point x^p
- ➔ The volume of the perturbation ball largely consists of regions where points will <u>not</u> be pulled back towards the saddle point
- ➔ Thus, it is likely that x^p can escape the saddle point if perturbed correctly

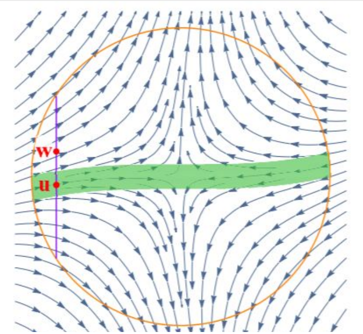


Definition II.4. *For a ρ-Hessian Lipschitz function $f(\cdot)$, we say that x is a* **second-order stationary point** *if $\|\nabla f(x)\| = 0$ and $\lambda_{min}(\nabla^2 f(x)) \geq 0$; we also say x is* **ϵ-second-order stationary point** *if:*

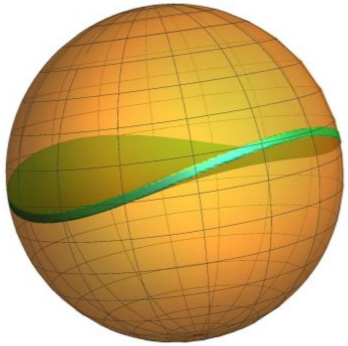
$$\|\nabla f(x)\| \leq \epsilon, \text{ and } \lambda_{min}(\nabla^2 f(x)) \geq -\sqrt{\rho\epsilon}$$



Escaping Saddle Points



- Bounding the thickness of this “stuck” region is an important theoretical result which is key to the Improved-or-Localized property
  - Any point stuck during the course of ACD undergoes perturbation. This leads to two possible results: either the perturbed point decreases the objective function, or it is close to a second-order stationary point



$$\frac{Vol(R_{stuck})}{Vol(B^{(d)}_{\tilde{x}}(r))} \leq \frac{Vol(B^{(d-1)}_{\tilde{x}}(r))\left(\frac{\eta r \lambda \sqrt{\pi}}{\sqrt{d}}\right)}{Vol(B^{(d)}_{\tilde{x}}(r))}$$

Asynchronous Coordinate Descent

Asynchronous coordinate descent is defined by the following update rule:

$$x_i^{j+1} = x_i^j - \eta \nabla_i f(\hat{x}^j)$$

x^j - Global point within ACD (x^{j+1} is the subsequent point)
η - Learning rate (step size)
i - The selected block (each worker assigned a block, can also be chosen at random)
x̂^{j} - Decayed point (a worker's point may be outdated by the update is complete)

$$\hat{x}^j = \left(x_1^{j-d(j,1)},\ x_2^{j-d(j,2)},\ \ldots,\ x_N^{j-d(j,N)}\right) \qquad d(j) = \max_{1 \le n \le N} \{d(j,n)\} \le \tau$$

Note: delays cause a loss of monotonicity!

Project Goals

Main Goals:

- ➔ Implement the Saddle Escaping Asynchronous Coordinate Descent algorithm
  - ◆ Includes optimizing the selection of hyper-parameters within the algorithm
- ➔ Test and analyze the convergence of SEACD
  - ◆ Compare with both regular gradient descent (GD) and perturbed gradient descent (PGD)
    - This comparison isn't necessarily "fair", as GD/PGD are not asynchronous

Approach: SEACD Algorithm

The Saddle Escaping Asynchronous Coordinate Descent (SEACD) algorithm consists of three inner algorithms:

- ➔ Single Worker Asynchronous Coordinate Descent (SWACD)
- ➔ Global Asynchronous Coordinate Descent (GACD)
- ➔ Perturbed Asynchronous Coordinate Descent (PACD)

Approach: SWACD Algorithm

Single Worker Asynchronous Coordinate Descent (SWACD)

Algorithm 1: s = SWACD($\hat{x}, f, \eta, i$)

Input: Shared point $\hat{x} \in \mathbb{R}^N$ (the read coordinate information that may be outdated by the end of the algorithm), objective function f, learning rate (step-size) η, updating block i (containing coordinates c)

Output: The update s to the shared solution (product of the gradient and step size)

$\bar{x} \leftarrow \hat{x}$;

for $c \in i$ **do**

  $\bar{x} \leftarrow \bar{x} - \eta \nabla_c f(\bar{x}) \mathbf{e}_c$;

end

$s \leftarrow \bar{x} - \hat{x}$;

return s

Approach: GACD Algorithm

Global Asynchronous Coordinate Descent (GACD)

Algorithm 2: $(n, x^{j+n}, E_{j+n}) = \mathrm{GACD}(x^j, f, \eta, \tau, M, L)$

Input: A starting point $x^j \in \mathbb{R}^N$, objective function f, learning rate (step-size) η, delay bound τ, momentum threshold M, gradient-Lipschitz L

Output: Total iterations performed n, the point x^{j+n}, energy function E_{j+n} at that point

$\gamma \leftarrow j + \tau$;
while $j < \gamma$ **do**
  Choose Block i;
  $x^{j+1} - x^j \leftarrow \mathrm{SWACD}(x^j, f, \eta, i)$;
  if $||x^j - x^{j+1}||_2 \geq M$ **then**
    $j \leftarrow j + 1$;
    break;
  end
  $j \leftarrow j + 1$;
end
$n \leftarrow (j + \tau - \gamma)$;
$E_j = f(x^j) + \frac{L}{2} \sum_{k=j-\tau}^{j-1} (k - (j - \tau) + 1) ||x^{k+1} - x^k||_2^2$;
return n, x^j, E_j

(Lines of the while loop: In Parallel)

Approach: PACD Algorithm

Perturbed Asynchronous Coordinate Descent (PACD)

Algorithm 3: $(x^{j+1}, E_{j+1}) = \text{PACD}(x^j, f, \eta, \tau, r, T, L)$

Input: A starting point $x^j \in \mathbb{R}^N$, objective function f, learning rate (step-size) η, delay bound τ, perturbation radius r, escaping time bound T, gradient-Lipschitz L

Output: The following point x^{j+1} (after T steps of perturbation), energy function E_{j+1} at that point

```
ξ ← uniformly ~ B(0, r);
y^0 ← x^j + ξ;
t ← 0;
while t < T do
    Choose Block i;                               }
    y^{t+1} − y^t ← SWACD(y^t, f, η, i);          }  In Parallel
    t ← t + 1;                                    }
end
E_{j+1} = f(y^T) + (L/2) Σ_{k=T−τ}^{T−1} (k − (T − τ) + 1) ||y^{k+1} − y^k||_2^2;
x^{j+1} = y^T;
return x^{j+1}, E_{j+1}
```

$$\xi \leftarrow \text{uniformly} \sim \mathbb{B}(0, r);$$
$$y^0 \leftarrow x^j + \xi;$$
$$t \leftarrow 0;$$

while $t < T$ **do** (In Parallel:)

- Choose Block i;
- $y^{t+1} - y^t \leftarrow \text{SWACD}(y^t, f, \eta, i);$
- $t \leftarrow t + 1;$

end

$$E_{j+1} = f(y^T) + \frac{L}{2} \sum_{k=T-\tau}^{T-1} (k - (T - \tau) + 1) \|y^{k+1} - y^k\|_2^2;$$
$$x^{j+1} = y^T;$$

return x^{j+1}, E_{j+1}

Approach: SEACD Algorithm

Saddle Escaping Asynchronous Coordinate Descent (SEACD)

Algorithm 4: $(x_\epsilon^*) = \text{SEACD}(x^0, f, \eta, r, \tau, T, \mathscr{F}, M, L)$

Input: An initial point $x^0 \in \mathbb{R}^N$, objective function f, learning rate (step-size) η, perturbation radius r, delay bound τ, escaping time bound T, function change threshold $\mathscr{F}$, momentum threshold M, gradient-Lipschitz L

Output: Returns an ϵ-second-order stationary point x_ϵ^*

```
E_0 ← f(x^0);
j ← 0;
for s = 1, 2, 3, ... do
    n, x^{j+n}, E_{j+n} ← GACD(x^j, f, η, τ, M, L);
    j ← j + n;
    if (E_j − E_{j−n}) > −𝓕 then
        x^{j+1}, E_{j+1} ← PACD(x^j, f, η, τ, r, T, L);
        j ← j + 1;
        if (E_j − E_{j−1}) > −𝓕 then
            break;
        end
    end
end
return x^j
```

Approach

- Each of these algorithms (including GD and PGD) will be implemented from scratch in Python using the NumPy software
- Later implementation (for validation) may also be done within PyTorch in Python

Validation Methods

I plan on testing my code on the following three non-convex problems:

- Matrix Sensing
- Matrix Completion
- Tensor Decomposition

$$\min_{\mathbf{M}\in\mathbb{R}^{d_1\times d_2},\mathrm{rank}(\mathbf{M})=r} f(\mathbf{M}) = \frac{1}{2m}\sum_{i=1}^{m}(\langle \mathbf{M}, \mathbf{A}_i\rangle - b_i)^2$$

$$\min_{\mathbf{M}\in\mathbb{R}^{d_1\times d_2},\mathrm{rank}(\mathbf{M})=r} \frac{1}{2p}\|\mathbf{M} - \mathbf{M}^\star\|_\Omega^2$$

$$\min_{\forall i,\|u_i\|^2=1} \sum_{i\neq j} T(u_i, u_i, u_j, u_j)$$

- I will first reproduce the results from these problems in papers [4] and [5] using PGD before testing SEACD
- I plan on using a synthetic database for testing
  - The data is arbitrarily complex

Deliverables

For this semester, I aim to build from scratch the following algorithms:

- Gradient Descent (GD)
- Perturbed Gradient Descent (PGD)
- Single Worker Asynchronous Coordinate Descent (SWACD)
- Global Asynchronous Coordinate Descent (GACD)
- Perturbed Asynchronous Coordinate Descent (PACD)
- Saddle Escaping Asynchronous Coordinate Descent (SEACD)

Milestones and Timeline

My major milestones are implementing and testing each one of the algorithms described on the previous slide

Rough Timeline:

- October-November: Implement and validate results on one of the test problems for PGD and GD
- November-January: Implement and validate results from each test problem for SEACD, optimize hyper-parameters, and analyze convergence

References

1. How to Escape Saddle Points Efficiently, Jin et al. https://arxiv.org/pdf/1703.00887.pdf
2. On Nonconvex Optimization for Machine Learning: Gradients, Stochasticity, and Saddle Points, Jin et al. https://arxiv.org/pdf/1902.04811.pdf
3. Asynchronous Coordinate Descent under More Realistic Assumptions, Sun et al. https://arxiv.org/pdf/1705.08494.pdf
4. Escaping From Saddle Points – Online Stochastic Gradient for Tensor Decomposition, Ge et al. https://arxiv.org/pdf/1503.02101.pdf
5. No Spurious Local Minima in Nonconvex Low Rank Problems: A Unified Geometric Analysis, Ge et al. https://arxiv.org/pdf/1704.00708.pdf
6. Ji Liu, Stephen J. Wright, Christopher Re, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. 16(1):285-322, 2015.
7. F. Niu, B. Recht, C. Re, and S. J. Wright, Hogwild: A lock-free approach to parallelizing stochastic gradient descent, Advances in Neural Information Processing Systems, 24 (2011), pp. 693–701.
8. Kfir Y Levy. The power of normalization: Faster evasion of saddle points. arXiv:1611.04831 2016
9. Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In COLT, 2015.