Exploring Gradient-Free Optimization Methods through "Gradient-Free Optimization With Constraint Algorithm" & "Federated Consensus Based Optimization Algorithm"

by

Deepansha Singh

Bachelor of Science

Mathematics Department Honors Thesis

University of California, San Diego (UCSD)

March 2024

ABSTRACT

Author: Deepansha Singh, Advisor: Dr. Yuhua Zhu

Bachelor of Science in Mathematics-Computer Science

Title: Exploring Gradient-Free Optimization Methods through "Gradient-Free Optimization With Constraint Algorithm" & "Federated Consensus Based Optimization Algorithm"

With the ongoing machine learning/deep learning technology boom, there have been many issues that are present with these gradient-based optimization methods. The "vanishing gradient" and "exploding gradient" problems along with the extreme complexity that comes with computing these gradients of high dimensional problems are just some of these issues. Thus, in my thesis, I explored non-gradient optimization methods further. Specifically, this thesis consists of two parts. The first component explores a "gradient-free optimization algorithm with constraints" that was proposed by [Car+20]. The second component of the thesis includes further exploring an algorithm is also originally proposed by my advisor and her research group called the "Federated Consensus-Based Optimization" (Fed CBO) algorithm [Car+23] For both parts, it is discussed what my experiments/methods/contributions are.

ACKNOWLEDGMENTS

This wouldn't be possible without the support of my thesis advisor Dr. Yuhua Zhu and members of her research group, Sixu Li and Aditya Akash. I would also like to give a huge thanks to all my mentors both in research/academia and industry. A huge thanks also goes out to my family and friends.

TABLE OF CONTENTS

Chapter Page

CHAPTER 1

INTRODUCTION

1.1 Machine learning / deep learning introduction

With the advent of AI services such as ChatGPT and Google's Gemini, the machine learning/deep learning area has become even more in the limelight. There are many buzzwords that people hear on a daily basis pertaining to this field. My honors thesis, advised by Dr. Yuhua Zhu, is in this machine learning/optimization space. Before delving deeper into the specifics of what I was working with for this project, I believe it is crucial to first give a good overview/background of various machine learning concepts at a high level.

As this field is very expansive and has countless sub-topics spanning from "natural language processing" to "reinforcement learning" I will only focus on the "deep learning" subfield as this pertains to the second portion of my thesis. As outlined in the abstract, this thesis consists of two research components - 1) a gradient-free constraint optimization research problem and 2) a federated learning research problem. The second portion of my thesis involves advanced machine learning concepts, so chapter 1 will cover these foundational concepts.

Of the machine learning problems that people explore, image classification is one that I would like to focus on. Using classical machine learning techniques, oftentimes people aim to classify images through a "supervised learning" approach where a classifier model is initially trained on some "trainset" and then we test this classifier's performance on a "testset" and see how well it truly performs. The original dataset is split into these two sets - a trainset and a testset - where the testset is data that the classifier can't see from before so it's a way of testing the classifier in the real world. Many times, people will also have a validation set when training, which sometimes helps with further improving the accuracy.

In classical machine learning problems, people need to go about feature extraction manually. Given some input and targets (labels associated with each class of images), people manually specify features that might distinguish one class of images from another class. For instance, if we are looking at some research problem of classifying fish from frogs, we might manually encode in our feature selection attributes such as fins or legs to distinguish between these two classes.

However, a more powerful method would be if we don't need to manually en-

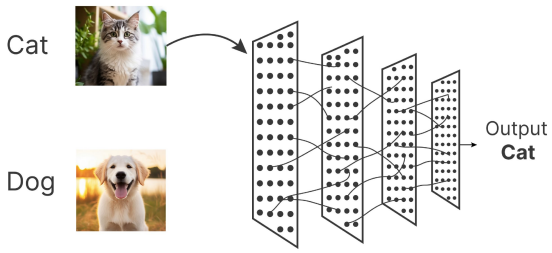

Figure 1.1. Cats vs. Dogs ML example, passing through neural network [Sha24]

code the features and instead the features are learned automatically. This is where deep learning plays a role.

ChatGPT and Gemini - and many of these other machine learning technologies - are all powered by some type of deep learning neural network architecture. "Large language models" (LLM's) in particular has been a very popular buzzword lately. Simply put, these LLM's have a special deep learning neural network architecture pattern. I won't be going in-depth regarding how large language models are constructed as they don't pertain to this thesis; however, there are many resources available to get more insight on how these transformer architecture patterns are built from concepts such as RNN's.

The specific kind of neural network that is relevant for this second portion of my thesis is "convolutional neural networks." However, before discussing this neural network architecture, I will give a deeper mathematical overview on how exactly neural networks work.

The goal of the machine learning/deep learning subspace is to mimic the way humans learn to learn new concepts naturally in our world given a set of observations and patterns that can be deduced, instead of hard-coding a fixed if-else algorithm to make conclusions. As the name suggests, neural networks are composed of neurons. Just like the human brain, these neurons take in a set of inputs and then produce some output by taking a linear combination of the inputs. Each layer in the neural network consists of different hidden layers, a given input layer, and also an output layer. In terms of the connections between these neurons, we can assign "weights" on each of these connections, where the weights are randomly initialized in the beginning, and then finetuned as time progresses. These weights help

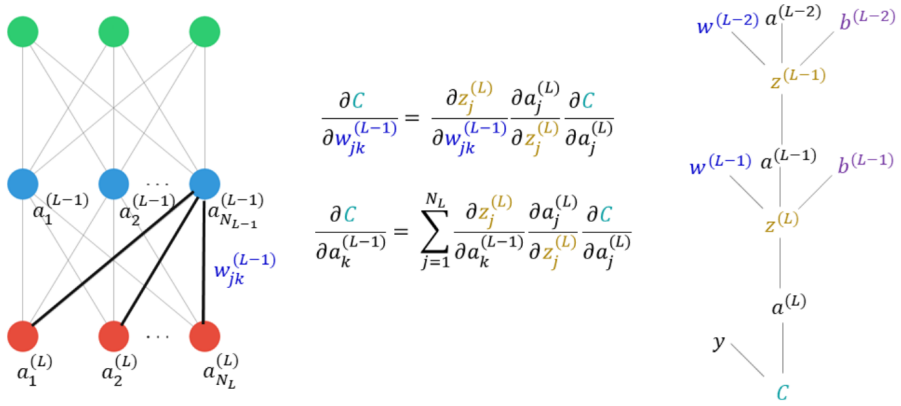

Figure 1.2. Backpropagation [Sil20]

determine how important each input is towards determining the final output. However, if all we are doing is a linear combination between these weights and inputs for each of these hidden layers, it is natural to wonder why instead can't we just have one larger layer consisting of the final weights to assign to each of the images. This is because we assign an element of "non-linearity" to the neural network by introducing an activation function. This non-linearity is what prevents us from having just one layer that is accumulating all the weights for the inputs.

This process of starting from a given set of inputs and propagating towards the final output in the output layer is called "forward propagation" in deep learning. Simply put, the neural network model is arriving at some output with some weights and inputs and then we can compare its output with the actual true target/label output. From here, we can see the error that is present between the true and the observed outputs and then go about a process called "backpropagation." As the name suggests, backpropagation is the process of updating our weights according to the error that we get. This is where the concept of partial derivatives and computing gradients comes into the picture. Given some error "C" as denoted in the figure, we can take the partial derivative with respect to each of the weights for the different layers, essentially starting from the end and back-propagating our way back to the very first layer.

Once these gradient terms are computed, we can then update the parameters using an optimizer such as gradient descent. The "alpha" term specifically in "Figure 1.5" denotes a hyperparameter called "learning rate." Learning rate is what enables us to update the parameters/weights of our neural network at some specific

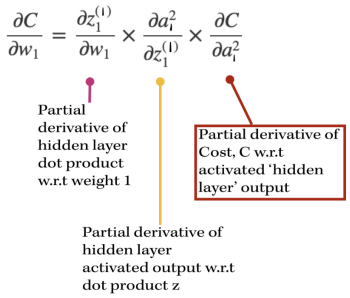

Figure 1.3. Backpropagation, labeled [Ram20]

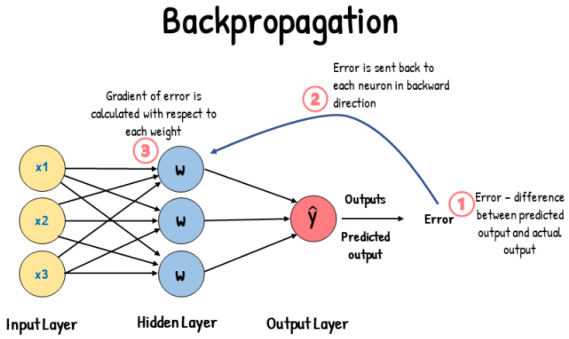

Figure 1.4. Backpropagation, Another Image Showing the Process [Wah24]

rate. We can either have a fixed rate that doesn't change or keep it variable. There are many interesting optimizers that have been develoepd through past extensive research efforts such as "Adagrad" and "Adam" optimizers that adjust the learning rate in very clever ways. PyTorch, specifically, is a popular machine learning/deep learning library in which people write deep learning programs and use these different concepts such as optimizers; the documentation explains each of these algorithms very well and also links previous research papers in which these findings were made.

Regarding gradient descent (Figure 1.5), the concept is given some non-convex function we can compute the gradient from backpropagation and then traverse in the direction that is opposite of the gradient direction. Specifically, the concept of learning rate discussed in the previous paragraph, connects with gradient descent as we can decide how much in this negative gradient direction we would like to traverse. A key thing to note is an issue that can rise with having both global and local minimas for a non-convex function. Ideally, when given the optimization

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Figure 1.5. Gradient Descent [Sar22]

problem of minimizing a given function, we would like to output the global minima point once the algorithm concludes. However, it is also possible for it to be stuck in a local minima point if we choose the initial starting point to be closer to the local minima instead of the global minima. This is a classic ML problem, which is not too relevant for this thesis but is interesting to explore; there have been different interesting strategies that people employ to ensure that we don't get stuck with this issue.

Reverting back to the activation functions for a quick moment, I just wanted to quickly mention that ReLu is a very commonly used activation function in a wide array of machine learning problems. Activation functions, as mentioned earlier, serve a critical purpose after we get some weighted linear combination with inputs and weights/parameters.

For this thesis (second portion), the specific type of neural network that is relevant is called "CNN" or convolutional neural networks. These networks are known for classifying images. To summarize, the convolution layers can just be thought of as taking some kind of filter and sliding it across the input image as some sliding window to yield some intermediate output in our neural networks. Pooling layers, as the name suggests, aggregate a given set of pixels and generally will look at either the "max" or the "average" values over a specific subsection of the image. These layers, along with some linear layer(s) at the very end form convolutional neural networks.

In Figure 1.6, an example is shown with an MNIST image. MNIST, along with CIFAR, are popular image datasets used in the Machine learning community to run experiments and test hypotheses. MNIST, in particular, is a set composed of handwritten images from 0-9 and CIFAR is a set of images also with 10 classes consisting of images with labels of common things such as "ship," "plane," etc.

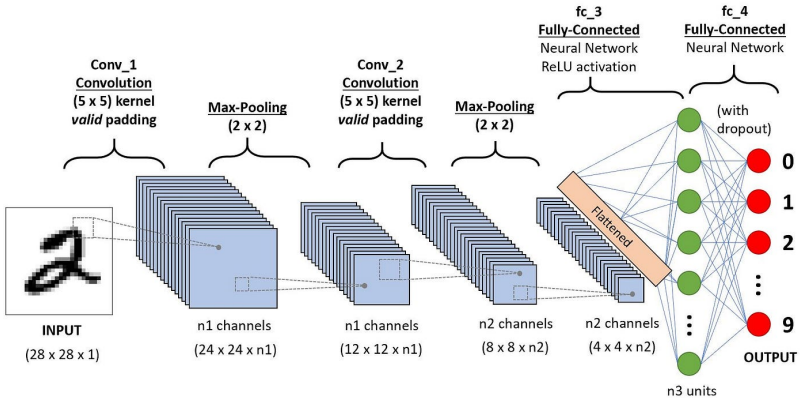

Figure 1.6. Convolutional neural networks [Sah18]

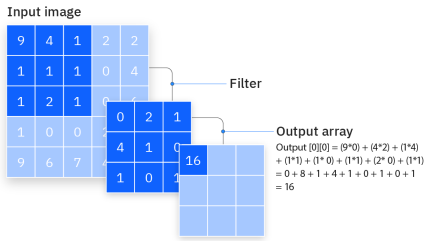

Figure 1.7. Convolution Layer in CNNs [Per24]

(https://www.cs.toronto.edu/ kriz/cifar.html).

The CNN that is used for the experiments in portion 2 of the thesis is not very complex, as in it doesn't have a lot of layers, but still is complex enough to help us with CIFAR dataset classification.

1.2 Federated learning introduction

Now that an overview of important machine learning concepts has been covered, we can discuss the concept of "federated learning" in particular. Federated learning was a concept initially proposed by Google AI. This is a privacy-preserving machine learning approach which has a central server and different clients that communicate and have different updates with this central server.

Oftentimes, certain applications of machine learning research problems involve the need to preserve the client's privacy. For example, dealing with sensitive pa-

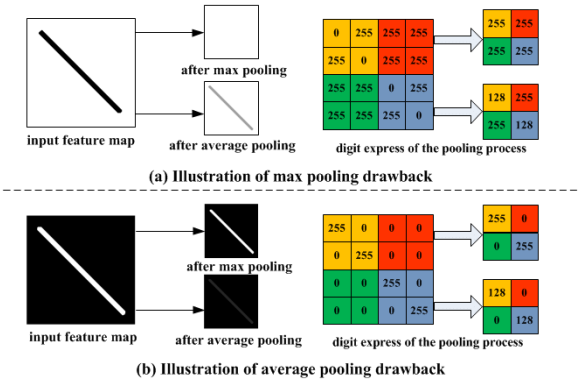

Figure 1.8. Pooling Layer in CNNs [AR20]

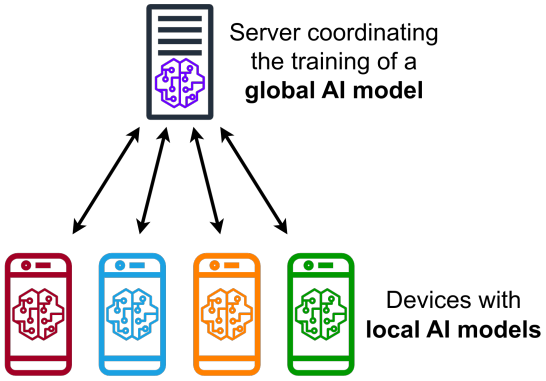

Figure 1.9. Federated Learning [Yan23]

tient data in the healthcare system is an application where it's highly critical to preserve privacy. Simply put, the objective of federated learning is such that there is some kind of collaboration between these different client clusters where they train a common neural network, which is called a global model, but the local updates are done locally for each client on their own respective datasets. There have been many algorithms that are already present such as FedAvg that further illustrate this concept. However, the common theme of this overall thesis - for both parts - is to have a focus on non-gradient based approaches as well. There are many issues that arise from gradient-based methods and if we take inspiration from other works we can focus on non-gradient based algorithmic methods to boost performance and also not run into any issues that are generally present with gradient-based approaches. More information on this is in the next section and next chapter

as well.

1.3 Non-gradient based methods

In the previous section, the key topic of discussion was how neural networks work mathematically and how it's essential to do gradient computations by taking partial derivatives with respect to each of the layer's weights/parameters. However, as the number of layers increase, there's a chance for neural network model running into the "vanishing gradient" problem. This essentially just means that the gradient value keeps on diminishing as we keep on taking partial derivatives when the number of layers is a very huge number that the gradient somewhat loses its meaning. On the other hand, there's also the "exploding gradient" problem. These are just two of the many problems that are present when it comes to dealing with gradients and computing them for complex, high-dimensional machine learning problems. Hence, this prompts for non-gradient based method exploration. My advisor, Dr. Yuhua Zhu and her research group, had previously developed an algorithm called "Gradient-free optimization with constraint" (built off of ideas from [Car+20]) that demonstrates this idea of optimizing a certain function along a constraint without using any gradients. I discuss my experiments, contributions, and results in the next section. For the second portion of the thesis, I worked with another algorithm that is in the "federated learning" space also initially developed in this paper [Car+23]. While this algorithm isn't entirely gradient-free, a great bulk of it is, and I discuss more in-depth about the algorithm and experiments, and my contributions for this portion of the thesis as well.

CHAPTER 2

METHODS/EXPERIMENTS

2.1 "Exploring Gradient-Free Optimization Methods Through Gradient-Free Optimization with Constraint Algorithm"

The setup of the algorithm is like so: there are N agents where the goal is to minimize a certain function L(x) and we are given a constraint g(x) = 0. The motivation behind this algorithm draws from particle physics and modeling this problem as a system of particles such that each one is updated accordingly based on looking at values like the gradient, hessian, and function value, etc. During each iteration of the algorithm there is a consensus point that is generated which is essentially the point that yields in the minimum possible L(x) value. Then using this, all the other agents are updated like so in figure 2.1.

My contributions for this portion of the thesis have been coding up the experiment in PyTorch, verifying the results seem to be expected. I mainly coded up four such experiments. For most of these experiments, I used the algorithm in Figure 2.1 with datasets from the following paper [For+21]. To give some additional context, this paper is also a consensus-based optimization approach and the goal was to see if we get some comparable results with this paper. Before going in this direction, I first ran a quick initial experiment, highlighted in the next section.

Small example

For this portion of the experiments/methods, L(x) = $x_1^2 + x_2^2$.
$g(x) = (x_1 + 1)^2/2 + ((x_2)^2 - 1)$. The particles were initialized to be uniformly distributed in the beginning and then over time they converged towards the minimum. Figures 2.2 and 2.3, in specific, show this overall convergence.

A key thing to note is that I also plotted the particles and how they progressed over time. This visualization also helped with debugging. Figure 2.4 depicts this. Furthermore, we can see how the particles are all forming this overall circular shape that we're given. Over time, the consensus point and the overall minimum are supposed to converge (the green and red points).

I also tried running this experiment for more iterations, such as 20 iterations, and got the following results, shown in figures 2.5 and 2.6.

3 Algorithm

We use implicit numerical scheme to discretize the model (2.2).

$$X_{k+1}^j = X_k^j - \lambda\gamma(X_k^j - \bar{x}^*) - \frac{\gamma}{\epsilon}\left(g(X_k^j)\nabla g(X_k^j) + J(X_k^j)[X_{k+1}^j - X_k^j]\right) + \sigma\sqrt{\gamma}\sum_{i=1}(X_k^j - \bar{x}^*)_i\vec{e}_i(z_i)_k,$$

where $J(x) = \nabla^2(g^2)$ is the Hessian of g^2, $(z_i)_k \sim \mathcal{N}(0,1)$. The reason why we use implicit scheme is because we require ϵ small in the model, implicit algorithm can guarantee the unconditional stable of the algorithm, while simple explicit algorithm is unstable for reasonable step size when ϵ is close to zero. Therefore the algorithm is the following.

Algorithm 3.1. Initialize N particles X^j following the same distribution.

Step 1 Calculate $\bar{x}^*$,

$$\bar{x}^* = \underset{X^j}{\operatorname{argmin}}\, L(X^j).$$

Step 2 Update all $\{X^j\}_{j=1}^N$ according to $\bar{x}^*$

$$X^j \leftarrow X^j - \left[I + \frac{\gamma}{\epsilon}J(X^j)\right]^{-1}\left(\lambda\gamma(X^j - \bar{x}^*) + \frac{\gamma}{\epsilon}g(X^j)\nabla g(X^j) + \sigma\sqrt{\gamma}\sum_{i=1}(X^j - \bar{x}^*)_i\vec{e}_i z_i\right),$$

where $z_k \sim \mathcal{N}(0,1)$.

Step 3 Check the stopping criteria,

$$\|\Delta\bar{x}^*\|_2^2/d \le \epsilon_{\text{stop}},$$

where $\Delta\bar{x}^*$ is the difference between the most recent two $\bar{x}^*$. If the above condition is satisfied, we stop the algorithm, otherwise, we go back to Step 1.

For multiple constrains optimization,

$$\min_x L(x)$$
$$\text{with} \quad g_1(x) = 0, \cdots, g_k(x) = 0$$

Similar to Algorithm 3.1, one changes Step 2 to

$$X^j \leftarrow X^j - \left[I + \frac{\gamma}{\epsilon}\sum_{i=1}^k J_i(X^j)\right]^{-1}\left(\lambda\gamma(X^j - \bar{x}^*) + \frac{\gamma}{\epsilon}\sum_{i=1}^k g_i(X^j)\nabla g_i(X^j) + \sigma\sqrt{\gamma}\sum_{i=1}(X^j - \bar{x}^*)_i\vec{e}_i z_i\right),$$

where $J_i(x) = \nabla^2 g_i(x)$ is the hessian of the i-th constrain.

Figure 2.1. Gradient-Free Constraint-Based Optimization Algorithm

Ackley function, small dimension example

Then, we took an example application from another paper and implemented the algorithm for this application to see the results [For+21]. This paper was centered around optimizing functions with sphere constraint, as can be seen in figure 2.7, such that the L(x) was the ackley function and the g(x) is this sphere.

A key thing to note for both Ackley function experiments - this current subsection and the next one - is that when programming this upper hemisphere constraint of radius 1, I ran into some issues. In particular, for the gradient computations, it was sometimes an undefined or complex number value of the square room term in

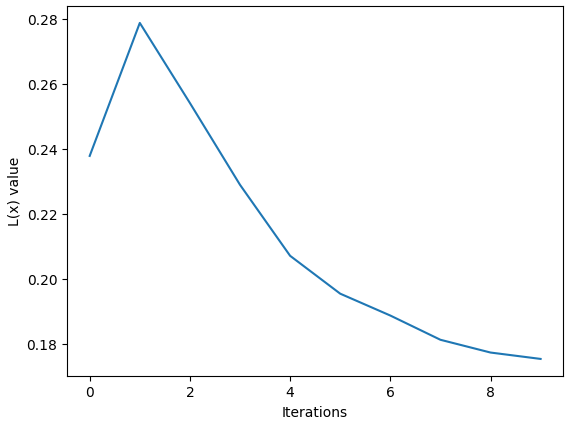

Figure 2.2. L(x) values for the small first example experiment.

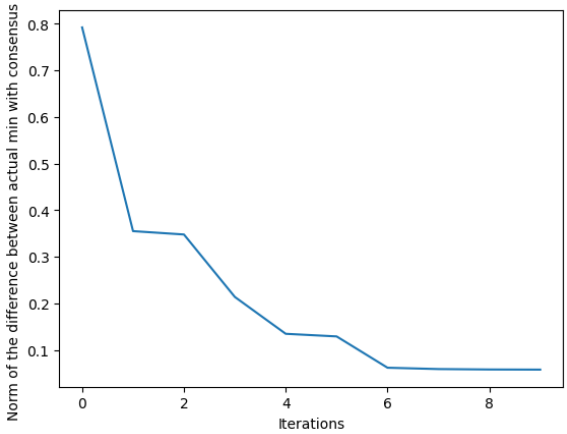

Figure 2.3. Norm of the difference between the consensus value point and the actual min

the denominator of the gradient, needing the number to be positive for that square root. However, this wasn't always the case, and after different attempts - such as adding some epsilon noise value to offset these undefined values in the square root term - I ended up keeping the constraint as a regular sphere. This would prevent this issue, and then kept the particles to still be initialized as distributed across the hypersphere of radius 1 and positive-z values. Thus, coding the constraint part isn't entirely accurate but still yielded some decent results

Something to note is that for a plot of theirs [For+21] they looked at the average norm value of the difference between the consensus value point and each of the agents. As the iterations increased, we can see that the value decreased - and in

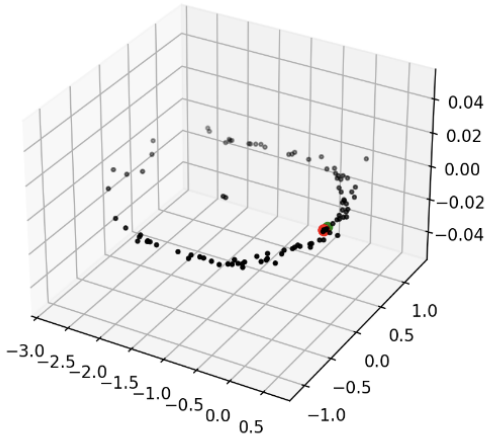

Figure 2.4. Visualization of the particles for the "small experiment"

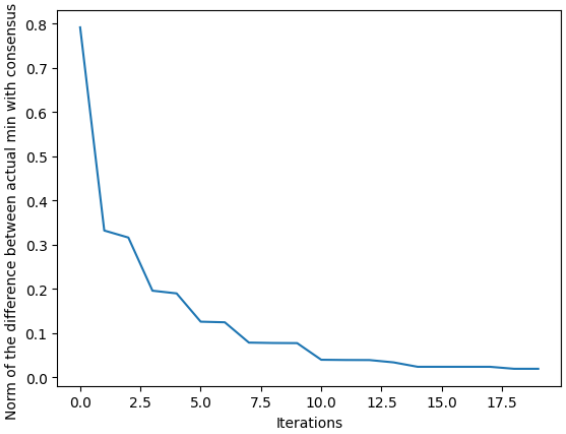

Figure 2.5. Norm of the difference between actual min with consensus, for more iterations

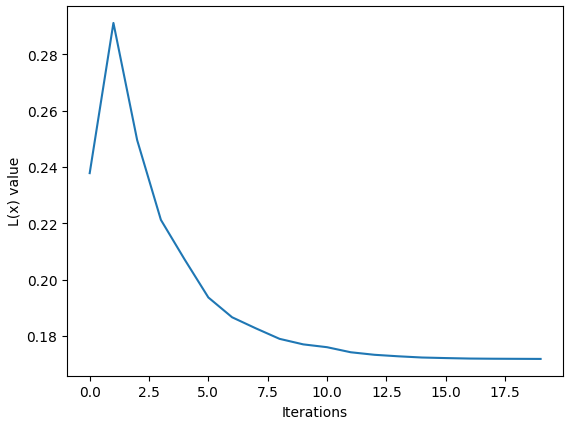

Figure 2.6. L(x) values for small example for large number of iterations

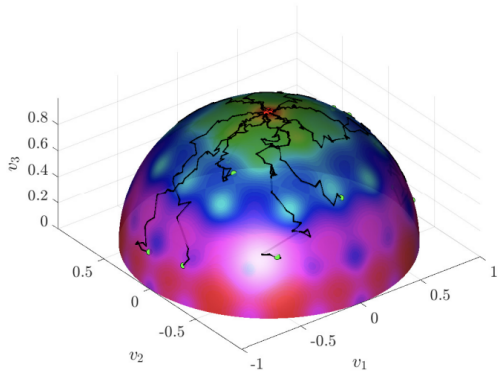

Figure 2.7. Sphere constraint [For+21]

fact it seems like for their experiment it reaches a value of somewhere around .6 to .7 but for us it reaches a value of around .4, seeming to reach convergence faster.

Ackley function, large dimension example

Similar to [For+21], I also tried with a larger dimension of dimension 6 and tried obtaining results in this case as well, which seemed to be pretty similar to the previous case as well; however, they worked with dimension 20 for the larger dimension case whereas I tried dimension 6.
I also tried getting results for the overall difference between the consensus value and the actual minimum per iteration to see if it converged, but didn't see the pat-

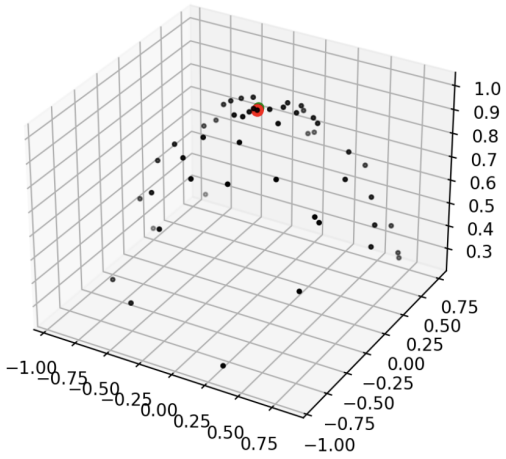

Figure 2.8. Ackley function, smaller dimension case

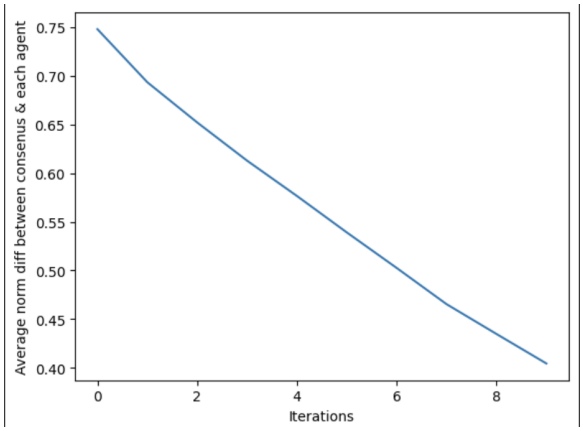

Figure 2.9. Average norm diff between consenus value point & each of the agents

tern that I was looking for exactly, and was close to constant value, so this seemed to be some bug which was needed to be addressed.

2.2 Federated Constraint Based Optimization

Federated Constrained Based Optimization [Car+23] is a "federated learning" algorithm which draws from ideas from particle physics. Specifically, the system is modeled as a system of particles reaching some convergence point that is described in more detail in the paper. Figures 2.11 and 2.12 give an outline of this FedCBO algorithm and the steps that are present starting from the initialization phase to

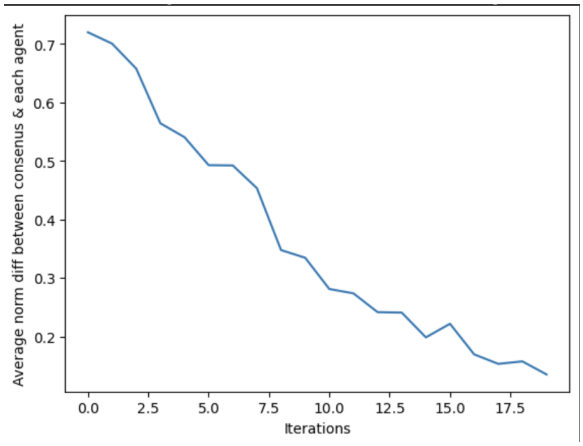

Figure 2.10. Ackley function, large dimension example

the local update and aggregation steps for each of these agents.

To give a concise picture of the algorithm, there are initially some agents/clients initialized with some neural network model in the beginning and first enter the "local update" phase. This step involves training the neural network for a few steps, updating the parameters. Then, during the local aggregation phase, a certain subset of agents is sampled from all of the agents and look at their respective losses by taking each of these model's computation output for the current inputted agent's dataset, and then use this with computing the consensus point. Using this consensus point, then the agents are updated along with the sampling likelihood.
Just like the algorithm discussed in the first portion of this thesis, this algorithm's "local aggregation phase" is also based on this very similar idea of looking at a system of points and updating them based on some central consensus point that is calculated.

My contributions for this portion of the thesis mainly involved extending the work to testing with the CIFAR dataset along with making code contributions towards making the algorithm runtime faster. To make the algorithm runtime faster, this involved making changes centered around the kernel/operating system level such that different processes/threads could run concurrently. With regards to extending the current work to CIFAR, I needed to learn more deep learning concepts - specifically related to dataset creation/loading concepts - and how to properly code these techniques in PyTorch.
Regarding the experiment, a small convolutional neural network was used and I

Algorithm 1 FedCBO

Input: Initialized model $\theta_0^j \in \mathbb{R}^d, j \in [N]$; Number of iterations T; Number of local gradient steps τ; Number of models downloaded M; CBO system hyperparameters $\lambda_1, \lambda_2, \alpha$; Discretization step size γ; Initialized sampling likelihood $P_0 \in \mathbb{R}^{N\times(N-1)}$;

for $n = 0, \cdots, T-1$ **do**
  $G_n \leftarrow$ random subset of agents (participating devices);
  LocalUpdate$(\theta_n^j, \tau, \lambda_2, \gamma)$ for $j \in G_n$;
  LocalAggregation(agent j) for $j \in G_n$;
end for

Output: θ_T^j for $j \in [N]$.

LocalUpdate$(\widehat{\theta}_0, \tau, \lambda_2, \gamma)$ at j-th agent

for $q = 0, \cdots, \tau-1$ **do**
  (stochastic) gradient descent $\widehat{\theta}_{q+1} \leftarrow \widehat{\theta}_q - \lambda_2\gamma\nabla L_j(\widehat{\theta}_q)$;
end for
return $\widehat{\theta}_\tau$;

Figure 2.11. FedCBO Algorithm [Car+23]

Algorithm 2 LocalAggregation(agent j)

Input: Agent j's model $\theta_n^j \in \mathbb{R}^d$; Participating devices at n iteration G_n; Sampling likelihood $P_n^j \in \mathbb{R}^{N-1}$; CBO system hyperparameters λ_1, α; Discretization step size γ; Random sample proportion $\varepsilon \in (0,1)$; Number of models downloaded M;

$A_n \leftarrow$ **ε-greedySampling**(P_n^j, G_n, M);
Agent j downloads models θ_n^i for $i \in A_n$;
Evaluate models θ_n^i on agent j's data set respectively and denote the corresponding loss as L_j^i;
Calculate consensus point m_j by

$$m_j \leftarrow \frac{1}{\sum_{i\in A_n}\mu_j^i}\sum_{i\in A_n}\theta_n^i\mu_j^i, \qquad \text{with } \mu_j^i = \exp(-\alpha L_j^i) \tag{19}$$

Update agent j's model by

$$\theta_{n+1}^j \leftarrow \theta_n^j - \lambda_1\gamma(\theta_n^j - m_j), \tag{20}$$

Update sampling likelihood P_n^j by

$$P_{n+1}^{j,i} \leftarrow P_n^{j,i} + (L_j^j - L_j^i), \qquad \text{for } i \in A_n \tag{21}$$

Output: $\theta_{n+1}^j, P_{n+1}^j$

ε-greedySampling(P_n^j, G_n, M)

Randomly sample $\varepsilon * M$ number of agents from G_n, denoted as A_n^1;
Select $(1-\varepsilon) * M$ numbers of agents in $G_n \backslash A_n^1$ with top value $P_j^{j,i}, i \in G_n \backslash A_n^1$, denoted as A_n^2;
return $A_n = A_n^1 \cup A_n^2$

10

Figure 2.12. FedCBO, Local Aggregation Portion [Car+23]

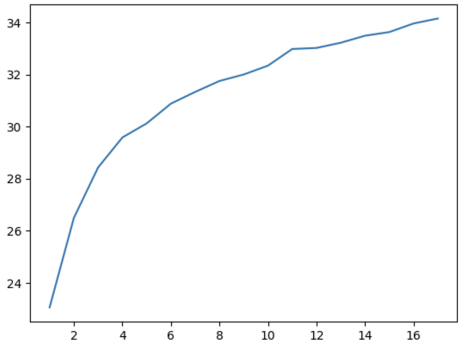

Figure 2.13. FedCBO Experiment, with accuracy shown across the iterations of the algorithm

tried to follow a similar setup to the following paper [Li+22] with some modifications, such as the number of iterations for the algorithm. Please find the results that were obtained in figure 2.13.

CHAPTER 3

KEY LEARNINGS

I learned a lot during this project as there were many challenges and bugs I had to fix along the way and many new concepts I had to ramp up on, which I hadn't worked with before - I am very grateful for all the support I received during this thesis and learned a lot about the overall research process for running numerical experiments and testing hypotheses and just writing good code to test these mathematical claims. I plan on continuing some extension work of the project and in the long term will be going to graduate school so have been very grateful for the key learnings obtained throughout this project and during all the research work in my undergradute career in general!

REFERENCES CITED

[AR20] N. Akhtar and U. Ragavendran, *Interpretation of intelligence in CNN-pooling processes: a methodological survey*, Neural Computing and Applications **32** (2020), DOI: 10.1007/s00521-019-04296-5.

[Car+20] J. A. Carrillo et al., *A consensus-based global optimization method for high dimensional machine learning problems*, arXiv: 1909.09249 [math.OC], 2020.

[Car+23] J. A. Carrillo et al., *FedCBO: Reaching Group Consensus in Clustered Federated Learning through Consensus-based Optimization*, arXiv: 2305.02894 [cs.LG], 2023.

[For+21] M. Fornasier et al., *Consensus-based optimization on the sphere: Convergence to global minimizers and machine learning*, Journal of Machine Learning Research **22** (2021), no. 237, 1–55.

[Li+22] Z. Li et al., *Towards effective clustered federated learning: A peer-to-peer framework with adaptive neighbor matching*, IEEE Transactions on Big Data (2022).

[Per24] S. Perkins, *What is a vision model?*, 2024.

[Ram20] A. Ramesh, *The Journey of Backpropagation in Neural Networks* (2020).

[Sah18] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* (2018).

[Sar22] A. Saravanakumar, *AI and Calculus: The Vanishing Gradient* (2022).

[Sha24] P. Sharma, *Build Your First Image Classification Model in Just 10 Minutes!* (2024).

[Sil20] S. D. Silva, *The Maths behind Back Propagation* (2020).

[Wah24] B. Wahlberg, *Mastering Machine Learning: How Backpropagation Powers Neural Network Training* (2024).

[Yan23] M. Yan, *Responsible AI: Balancing CO2 Emissions and Energy Efficiency for a Sustainable Future* (2023).