Comparison of PSO vs. Gradient Descent in Optimization For Neural Networks

Hunter Price
EECS
University Of Tennessee
Knoxville, United States
hprice7@vols.utk.edu

***Abstract*—Neural Networks are becoming deeper with more layers and more weights to accommodate the more complex problems that are being solved. This brings to light the need for better and faster ways to optimize the weights of these networks. In this work, we implement a Neural Network that uses Particle Swarm Optimization (PSO) to adjust the weights of a network, and we compare this to the well known Adam Optimizer when training. We trained these networks on three datasets of increasing complexity: AND, XOR, and SYNTH. We use use the same number of layers and neurons for each model between the two models through each dataset and hyper-parameter tune their respective parameters. We found that Neural Network optimized by PSO performed well when training on the simpler AND and XOR data; however, when presented with the more complex SYNTH dataset the model failed to learn effectively. Conversely, it seemed that the model trained with the Adam Optimizer performed better on the more complex SYNTH data than the AND and XOR data in comparison to the PSO trained model.**

***Index Terms*—Neural Network, Particle Swarm Optimization, Adam Optimizer, Gradient Descent, Machine Learning, Deep Learning**

I. Introduction & Motivation

In today's day and age, deep learning is more prevalent than ever. Neural Networks are finding new and exciting applications, and are becoming deeper and deeper to account for the larger amount of information we are now able to collect and store. With this increase in size, there are more layers and tune-able weights; this forces us to spend more research on optimization techniques to speed up and increase the performance of training these networks. In this paper, we compare two different optimization techniques: Gradient Descent and particle swarm optimization (PSO).

The goal of this project was to find a way to formulate the problem of Neural Network optimization in such a way that the particle swarm optimization algorithm could be applied to train the weights. Additionally, we wanted to compare this to the more standard approach to the problem, Gradient Descent. We sook out to answer the questions of is it possible to use particle swarm optimization in the training of Neural Networks? And how does it perform against the widely used Adam Optimizer?

I chose this subject because I am extremely interested in Machine Learning and Deep Learning. I find the field to be new and exciting, and I want to be a part of it. Further, I wanted more hands on experience with exactly how a neural network works to experiment and find what works and what does not.

particle swarm optimization was used as it seemed that it was a very scalable efficient algorithm. It does not depend on any gradients of the problem, which has the possibility to give more flexibility to neural network activation functions: they would no longer have to be differentiable. Additionally, this algorithm is extremely parallelizable in contrast to gradient descent. This would allow a neural network to be more easily distributed throughout a cluster with a large amount of compute resources.

II. Related Work

There has been a significant amount of research focused specifically on optimizing Neural Networks. One of these first works was by Bell Labs [2]. They proposed Stochastic Gradient Descent. This was a vast improvement upon the previous plain gradient descent as computers at the time did not have enough memory and compute resources to compute all updates at the same time. Rather they derived a proof that shows one could feed one sample at a time to be updated, and this would eventually be equivalent to pure gradient descent. This work has been significantly improved upon over the years. One example of this is the Adam Optimizer [3]. They proposed an optimization technique that factored in the momentum of the gradient updates as well as a built in adaptive learning rate. This is the go to optimizer in the industry. This is the optimizer we will use in the gradient descent based neural network.

particle swarm optimization was originally proposed to simulate social behaviour; however, it has been adapted for a plethora of optimization problems for its simplicity and efficiency [1]. Researchers in the past have attempted to use particle swarm optimization as an optimizer for neural networks. One example of this Carvalho and Ludermir's attempt to use particle swarm optimization to optimize both the weights of a neural network as well as its architecture at the same time [5]. They concluded that their results showed that their particle swarm optimization approach was a valid alternative optimization technique rather than gradient descent. Another work, has previously compared particle swarm optimization against typical gradient descent back-propagation, and they concluded that particle swarm optimization converged faster that typical back-propagation [6].

Other work, focused on specific use cases for particle swarm optimization in neural networks. One example of this in a paper by Chunkai Zhang, Huihe Shao and Yu Li who attempted to train a neural network with particle swarm optimization to predict structural failure of multi-story reinforced concrete buildings [7]. In this work they achieved a high level of accuracy (90%) which outclassed a typical gradient descent neural network as well as a multi-layer perceptron feed-forward network classifier. Another work applied a particle swarm optimization neural network to predict seismic slope stability [8]. They showed that this model had superior results as compared to a gradient descent neural network.

Some work has used particle swarm optimization to specifically optimize a neural networks architecture rather than the weights. One example of this is in work by Beatriz Garro and Roberto Vázquez who tuned the architecture of a neural network on 10 separate classification problems [9]. They concluded that their results were very promising. Another example is in work by Chunkai Zhang, Huihe Shao and Yu Li who used particle swarm optimization to evolve the structure of deep neural networks intended to classify images. This works results were promising as well [10].

III. Method

In this work, we implemented two networks using only fully connected layers. The first was a neural network using the gradient descent and specifically the Adam optimizer. This model was implemented with the Keras and Tensorflow libraries. The second was a neural network using particle swarm optimization. In order to adapt the neural network into something that could be optimized by particle swarm optimization, we need to adapt two core things about the algorithm. The first is a particles position or location. This was done by flattening the weights from each layer and appending them together into a large flat array. This represents the high dimensional position of the particle. The second thing we must adapt is the algorithms quality function. This was done by taking a particles location then reshaping and redistributing the weights to each layer. Then we feed forward the training data and calculate the loss of the prediction. This loss is what is returned by the quality function. In this work we used both mean squared error and binary cross entropy between different datasets for the loss.

There were 3 datasets used with increasing complexity. All datasets serve as simple 2 feature binary classification problems. The first is the logical AND dataset. This data is shown in Table I. It is a simple problem that only requires a simple linear classifier. The second dataset is the logical XOR dataset. This is shown in Table II. This problem provides slightly more complexity as it no longer can be perfectly separated by a linear classifier, rather there must be some for of non linearity within the classifier. The final dataset used in this work is the Synthetic (SYNTH) Dataset. This is a synthetically created dataset to test machine learning models by Ripley B. in Pattern Recognition and Neural Networks [4]. This is the most complex dataset out of the three as there is no perfect way to discriminate between the two classes.

In our experiments, we first trained different variations of the particle swarm optimization neural network to find a structure that would work. We then matched the same structure for the gradient descent neural network and hyper-parameter tuned both models one their respective parameters. This was done to be able to compare better compare what the outputted loss and accuracy curves between the two optimized models. Once we found an acceptable configuration for both, we trained both for 50 epochs and solved for the mean and standard deviation of the loss and accuracy curves.

TABLE I
AND Dataset

Features		Label
0	0	0
0	1	0
1	0	0
1	1	1

TABLE II
XOR Dataset

Features		Label
0	0	0
0	1	1
1	0	1
1	1	0

IV. Results

A. AND

For the AND dataset, we only used a single neuron for each model as it should be theoretically possible to classify this dataset with only a single neuron.

The particle swarm optimization neural network was composed of a single neuron with two inputs and a single output. This neuron had an activation function of relu. For the training process, we used 50 total particles, an inertia of 0.7, a cognition parameter of 0.7, a social parameter of 0.3, and a max velocity of 3. The gradient descent neural network was composed of a single neuron with two inputs and a single output. This neuron had an activation function of sigmoid. For the training process we used a learning rate of 0.1. Both models trained for 150 epochs and used a loss of mean squared error. Figure 1 shows the average loss and accuracy plots over 50 experiments. Particle swarm optimization is shown in blue and gradient descent is shown in orange. We can see from the loss plot that particle swarm optimization converged quickly to a value of approximately 0.30, and the gradient descent slowly converged throughout the entire training process. When we contrast this with the accuracy plots we see a curious result. The particle swarm optimization accuracy converged consistently converged to 100 percent accuracy within the first

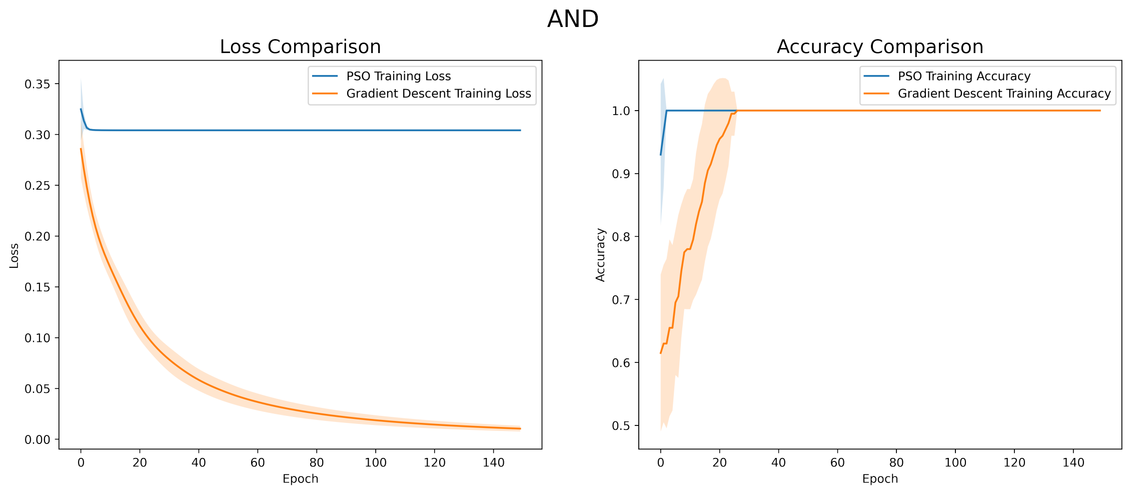

Fig. 1. Loss and Accuracy on the AND Dataset

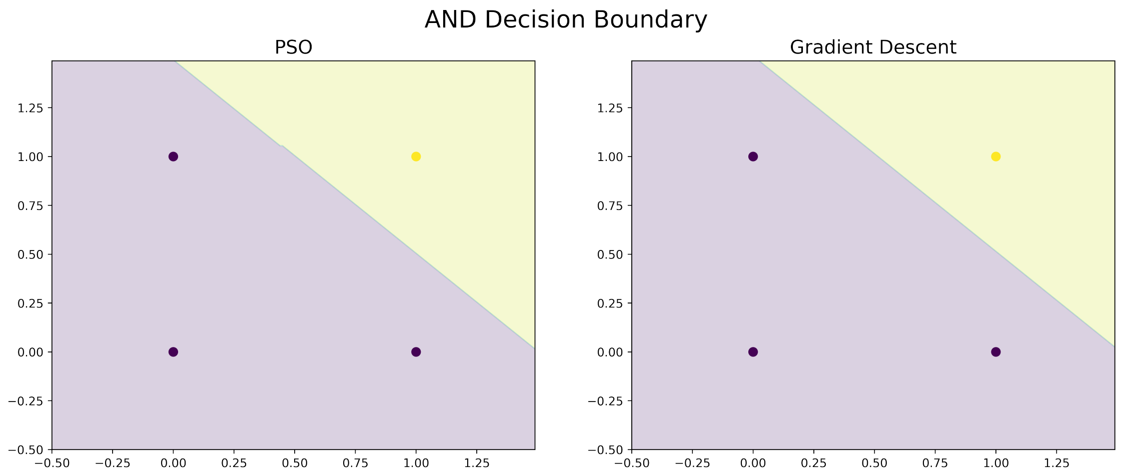

Fig. 2. AND Dataset Decision Boundary Comparisons

10 epochs, whereas gradient descent took around 25 epochs on average to reach 100 percent accuracy. This shows that the particle swarm optimization technique quickly converged to a position that worked and did not continue to improve where as the gradient descent technique had slow consistent improvement.

In Figure 2, we can see an example decision boundary for both particle swarm optimization and gradient descent on the AND data. They seem to converge to similar if not the same visual results.

B. XOR

For the XOR dataset, we could not use a single neuron as the problem is not longer linearly separable. Upon experimenting with the particle swarm optimization neural network, we found a structure of a single hidden layer with 2 neurons and an output layer with a single neuron to be sufficient.

The particle swarm optimization neural network was composed of a hidden layer with 2 neurons and an output layer with a single neuron. The hidden layer neurons used activation functions of relu and the output layer also used relu as an activation function. For the training process, we used 50 total particles, an inertia of 1.3, a cognition parameter of 0.7, a social parameter of 0.4, and a max velocity of 20.

The gradient descent neural network was composed of a hidden layer with 2 neurons and an output layer with a single neuron. The hidden layer neurons used activation functions of sigmoid and the output layer also used sigmoid as an activation function. This model had difficulties consistently learning the XOR pattern, so we resulted to using a learning rate scheduler with an initial learning rate, decay steps of 50, a decay rate of 0.2, and staircase set to true.

Both models trained for 300 epochs and used a loss of mean squared error. We used a larger number of epochs here to allow

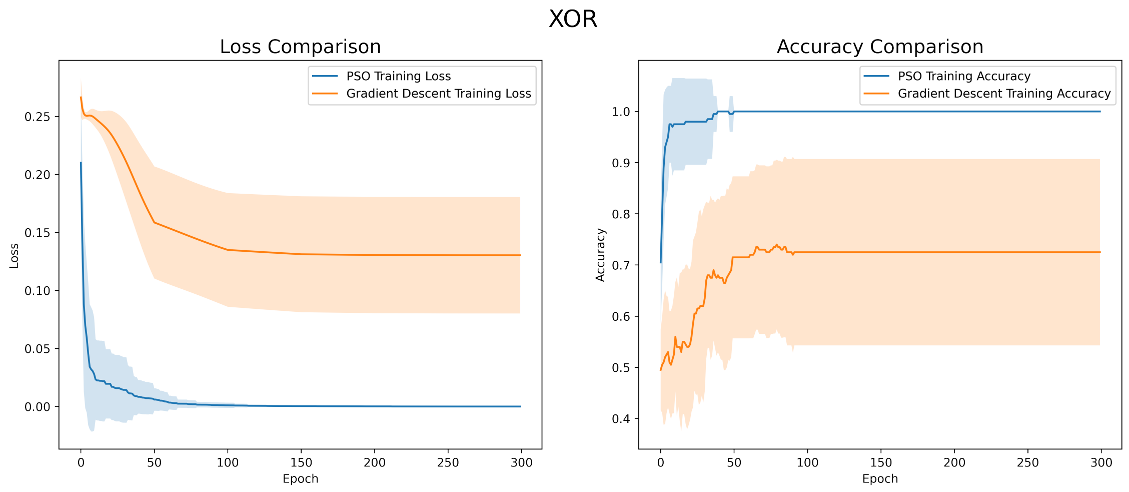

Fig. 3. Loss and Accuracy on the XOR Dataset

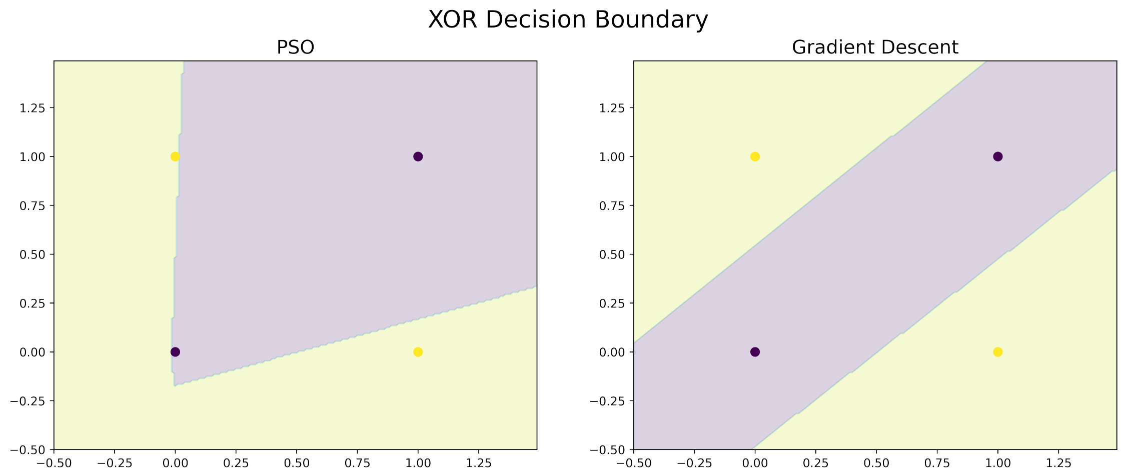

Fig. 4. XOR Dataset Decision Boundary Comparisons

the gradient descent model to have time to learn.

Figure 3 shows the average loss and accuracy plots over 50 experiments. Particle swarm optimization is shown in blue and gradient descent is shown in orange. We can see from the loss curve that particle swarm optimization quickly converged to an optimal loss within 50 epochs; however, the gradient descent curve rarely converged to a loss below 0.10. This correlation is reflected in the accuracy plots where the particle swarm optimization quickly converged to 100 percent accuracy within 50 epochs and the gradient descent model had an average accuracy of 75 percent.

In Figure 4, we can see an example decision boundary for particle swarm optimization and gradient descent models on the XOR data. Gradient descent seemed to learn a very clear pattern down the diagonal. However, the particle swarm model seems to have overfit to the data as it is just barely correctly classifying the labels correctly.

C. SYNTH

For the SYNTH dataset, the complexity was far higher than both the previous datasets. Due to this we used a model with more layers and more neurons. We decided upon a network configuration of 2 hidden layers with 5 neurons each and an output layer with a single neuron.

The particle swarm optimization neural network was composed of 2 hidden layers with 5 neurons each and an output layer with a single neuron. All hidden layer neurons used activation functions of sigmoid and the output layer also used sigmoid as an activation function. For the training process, we used 50 total particles, an inertia of 0.7, a cognition parameter of 0.7, a social parameter of 0.3, and a max velocity of 20.

The gradient descent neural network was composed of a hidden layer with 22 hidden layers with 5 neurons each and an output layer with a single neuron. All hidden layer neurons used activation functions of sigmoid and the output layer also

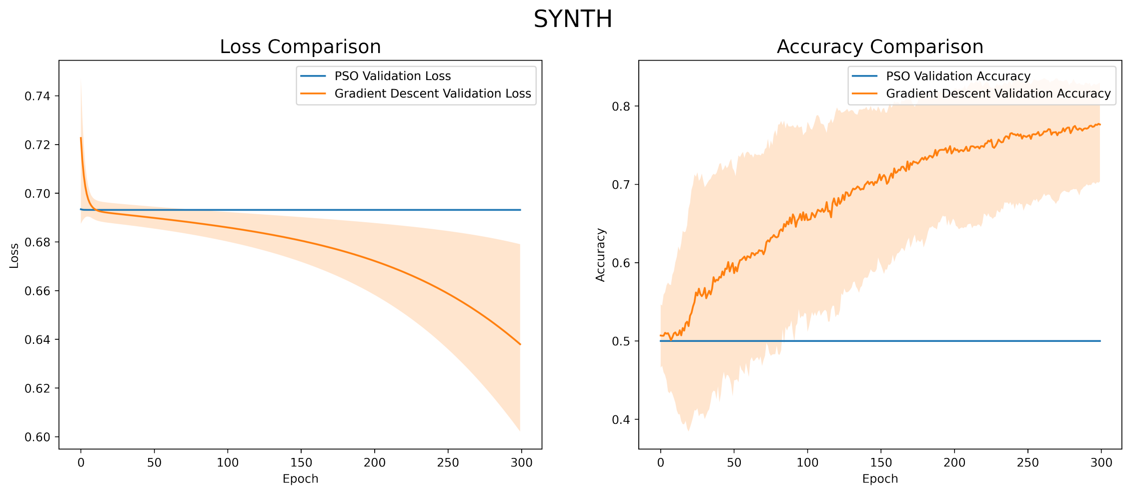

Fig. 5. Loss and Accuracy on the SYNTH Dataset

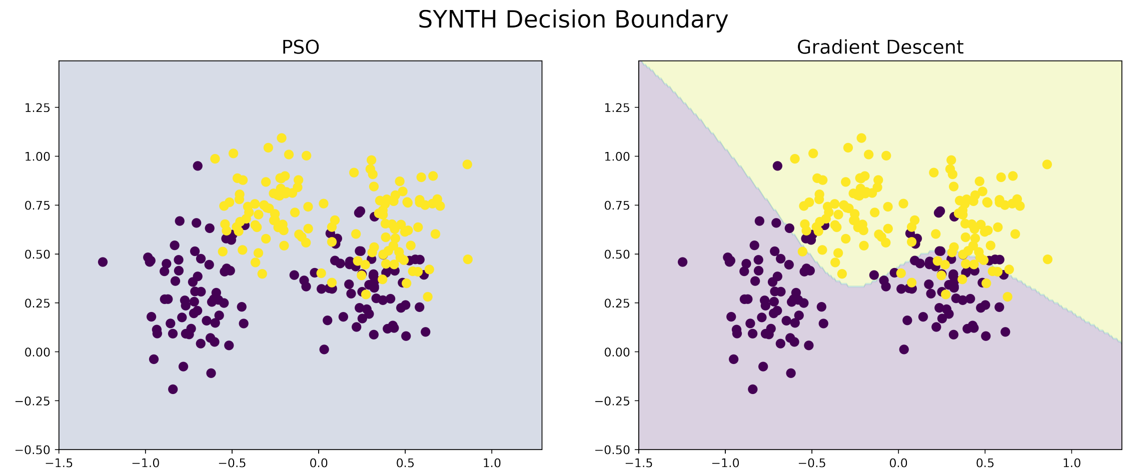

Fig. 6. SYNTH Dataset Decision Boundary Comparisons

used sigmoid as an activation function. For training the model used a learning rate of 0.03.

Both models trained for 300 epochs and used a loss of binary cross entropy.

Figure 5 shows the average loss and accuracy plots over 50 experiments. Particle swarm optimization is shown in blue and gradient descent is shown in orange. We can see from the loss curve that gradient descent began slowly converging over the 300 epochs and was still decreasing by the end of training on average; however, the particle swarm optimization immediately flat lined. Upon inspection of the outputted loss values, the curve was still decreasing, but at the order of magnitude of 1e-5. The accuracy plot reflects the same results. The gradient descent model had a constantly increasing accuracy and was still increasing at the end of training, whereas the particle swarm optimization model had a flat accuracy at 50 percent.

In Figure 6, we can see an example decision boundary for particle swarm optimization and gradient descent models on the SYNTH data. It is immediately obvious why the particle swarm optimization had a value of 50 percent for its accuracy as it classified all points as a single class. Because the value is 50 percent we can assume that there was an equal distribution of class 0 and class 1. The gradient descent model performed far better as we can see a very good decision boundary weaving through the clusters.

V. Discussion

After having looked through the results we can begin to find some correlations between the various datasets. Firstly, the results for the particle swarm optimization between the AND and XOR data are very consistent. They both converge very quickly to a result that gives 100 percent accuracy. The AND data was difficult to overfit as the model could only output a single decision boundary. This could be why we do not see similar results as we see in the XOR results. Here we see a decision boundary that just barely classifies the points. This could be because it quickly finds a solution with a small loss and sticks to that. It may not generalize well with small amount of data. That said, with these small simple datasets particle swarm optimization significantly outperformed the Adam optimizer. The gradient descent model is very slow to converge; however, its results for the AND and XOR data do not seem nearly as overfit as the particle swarm optimization model. It was also interesting that gradient descent did not consistently converge to the correct solution on the XOR data. This could be because it was not given much data and was not given enough time to learn with a small enough learning rate. Our goal was to train the models within the same number of epochs to compare the speed of convergence of the two models. This may not have been fair to the gradient descent model.

Next, we look at the SYNTH data versus the other simpler AND and XOR datasets. We can see a vast difference between how the particle swarm optimization model performed here versus the other results. It made little if any progress throughout the 300 epochs in training for all 50 experiments. This is most certainly due to the large increase in amount of training data as well as the increase in complexity between this problem and the others. Another reason for this is that we used a batch size equal to the size of the training data. Because we are not propagating gradient backwards through the layers with respect to the loss, and are instead simply applying the average gradient of the training data to all weights at once, it could be extremely difficult for this algorithm to find a minima. This is in stark contrast to the gradient descent model which performed extremely well over all of its iterations with the increased complexity of the data. It continuously learned over all epochs. In fact, the gradient descent model seemed to perform better on this dataset than on the simpler AND and XOR datasets.

Another interesting observation throughout our experiments was that the partical swarm optimization performed very poorly when using the sigmoid activation function in any layer. The resulting predictions would consistently be around the range of 1e-5 to 1e-9. We did not come to a conclusion as to why this was happening. However, when we did switch to using only the relu activation function we saw an immediate improvement. For the SYNTH dataset we tried both the sigmoid and relu activation functions and combinations and permutations of the two, but neither improved the results.

VI. Conclusion

Through our extensive experiments and analysis of the results we have to a few conclusions. The first of these are that the particle swarm optimization technique when applied to a neural network performs excellent on simple datasets with little complexity and small model sizes. This is because there are fewer weights to train, so the particles can more quickly converge to a results within a smaller search space. Secondly, when complexity is added to a problem the particle swarm optimization neural network will very rapidly decline in performance. As mentioned before, with more complexity there comes more weights to tune; particle swarm optimization does not seem to handle this increase well. Next, when compared to particle swarm optimization, gradient descent (specifically the Adam optimizer) performs poorly for the small simple datasets. But, with more complexity the gradient descent model significantly outperforms the particle swarm optimization model.

VII. Future Work

In the future, we plan to experiment on more datasets to be able to find a clean cutoff of where particle swarm optimization is better for the simple datasets and where gradient descent is better for the complex datasets. Secondly, we plan to spend more time hyperparemeter tuning the models to achieve better results. Due to the time constraint we could not finish an entire grid search of parameters and network structure. Next, we would like to find the exact reason why relu tended to outperform the sigmoid activation function for the particle swarm optimization models. Next, we beleive it may benefitial to use a combination of particle swarm optimzation and gradient descent. Because PSO converges faster, one could start with this as warm up epochs, then once converged switch to gradient descent and fine tune the model. Next we want to try to train a PSO-NN model with only one layer and many neurons as the non linearity in the activation functions over many layers may make it difficult for PSO to optimize as it will not account for this. Lastly, we want to implement a working stochastic/mini-batch particle swarm optimization technique to potentially prove our results making it such that the average loss applied is for a smaller subset of the data, and the incremental improvements will improve the final converged loss value.

References

[1] J. Kennedy and R. Eberhart, ”Particle swarm optimization,” Proceedings of ICNN’95 - International Conference on Neural Networks, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[2] Bottou, Léon. ”Stochastic gradient learning in neural networks.” Proceedings of Neuro-Nımes 91.8 (1991): 12.

[3] Kingma, Diederik P., and Jimmy Ba. ”Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980 (2014).

[4] Ripley, B. (1996). Pattern Recognition and Neural Networks. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511812651

[5] M. Carvalho and T. B. Ludermir, ”Particle Swarm Optimization of Neural Network Architectures and Weights,” 7th International Conference on Hybrid Intelligent Systems (HIS 2007), 2007, pp. 336-339, doi: 10.1109/HIS.2007.45.

[6] V. G. Gudise and G. K. Venayagamoorthy, ”Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks,” Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS’03 (Cat. No.03EX706), 2003, pp. 110-117, doi: 10.1109/SIS.2003.1202255.

[7] Chatterjee, S., Sarkar, S., Hore, S. et al. Particle swarm optimization trained neural network for structural failure prediction of multistoried RC buildings. Neural Comput & Applic 28, 2005–2016 (2017). https://doi.org/10.1007/s00521-016-2190-2

[8] Gordan, B., Jahed Armaghani, D., Hajihassani, M. et al. Prediction of seismic slope stability through combination of particle swarm optimization and neural network. Engineering with Computers 32, 85–97 (2016). https://doi.org/10.1007/s00366-015-0400-7

[9] Beatriz A. Garro and Roberto A. Vázquez. 2015. Designing Artificial Neural Networks using particle swarm optimization algorithms. Intell. Neuroscience 2015, Article 61 (January 2015), 1 pages. https://doi.org/10.1155/2015/369298

[10] Chunkai Zhang, Huihe Shao and Yu Li, ”Particle swarm optimisation for evolving artificial neural network,” Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. ’cybernetics evolving to systems, humans, organizations, and their complex interactions’ (cat. no.0, 2000, pp. 2487-2490 vol.4, doi: 10.1109/ICSMC.2000.884366.