Auto-Tune: A Comparison of Automated Hyperparameter Tuning for Neural Networks

Stanford CS361 Final Project Spring 2020

Noah Jacobson
Department of Computer Science
Stanford University
noahj08@stanford.edu

Daniela Gonzalez
Department of Symbolic Systems
Stanford University
danigonzalez@stanford.edu

Abstract

The process of tuning hyperparameters in order to optimize the training of a neural network is extremely time and resource consuming. We examine numerous automated strategies for hyperparameter tuning, including grid search, random search, hypergradient descent, Bayesian optimization, and population-based training (PBT). We implement each approach, comparing the amount of iterations each algorithm takes to converge with the train and test set accuracy.

Keywords: hyperparameter tuning, bayesian optimization, population based training, hypergradient descent, grid search, random search

1 Introduction

Neural networks are incredibly powerful due to their ability to automatically learn weights to solve a particular problem. That being said, the performance of neural networks is often dramatically affected by a few programmer-defined hyperparameters, including the learning rate, regularization strength, and momentum. Furthermore, it is often quite difficult to guess the correct hyperparameters. There is no clear-cut way of deriving the optimal values for these hyperparameters, and it is possible that the globally optimal hyperparameters for a network change during the course of training (i.e. with learning rate decay). Due to the performance advantages provided by hyperparameter tuning and the difficulty of finding the correct parameters, machine learning engineers end up devoting a significant portion of their time tuning hyperparameters for a project. Also, the computing resources required to perform hyperparameter tuning tend to be quite expensive. By enabling hyperparameters to be optimized automatically like all of the other parameters in a network, we would dramatically reduce the effort needed to create high-quality neural networks.
As a result, there is a significant research effort devoted to the automation and optimization of the hyperparameter tuning process. We focus our study on grid search, random search, hypergradient descent, Bayesian optimization, and population-based training (PBT). Details about each method are included in section 4.

2 Neural Networks

We test our optimization schemes on three different simple neural networks. We wanted to compare the performance of each optimization algorithm across different models in order to get a more accurate demonstration of each algorithm's performance on different applications. All networks map two inputs to one output between 0 and 1 (i.e. the probability of being from the '1' class). All networks use ReLU activations for hidden layers and a sigmoid activation function for the output layer.

2.1 Logistic Regression

The simplest model we used was a logistic regression model. Note that logistic regression is exactly the same as regression with a one-neuron neural network with sigmoid activation. Logistic regression can only represent linear decision boundaries. As a result, it overfits less than more complicated neural networks do. That being said, it is unable to represent more complex patterns. Logistic regression should be extremely quick to train due to the fact that the total number of parameters in our network will be 2 (two inputs to one output).

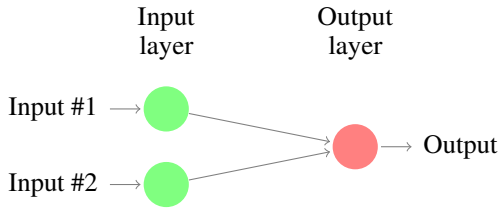

Figure 1: Architecture of 1-Layer Neural Network (Logistic Regression).

2.2 Simple Neural Network (2-3-1)

We call our next neural network the "simple" neural network. It has two inputs, one hidden layer with three neurons, and one output. This network can represent nonlinear patterns in data, but it cannot represent very complicated ones. It is not a very complicated neural network, so it will not be as prone to overfitting as others. The total number of parameters in the network is 5.

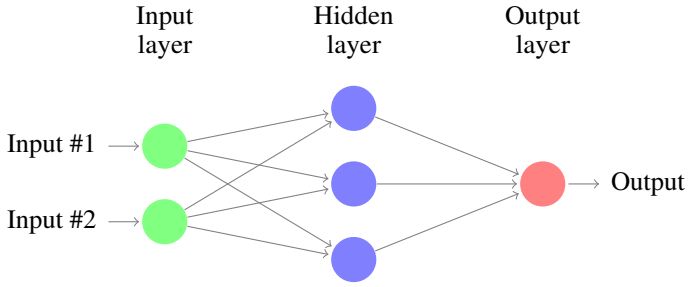

Figure 2: Architecture of 2-Layer Neural Network.

2.3 Three-Layer Neural Network (2-5-5-1)

Our next network has two hidden layers with five neurons each. This allows the network to represent even more complexities and nonlinearities in data. That being said, it also makes the network prone to overfitting. Furthermore, because the network is deeper, it should take slightly longer to train, although the difference in training time is insignificantly small. The total number of parameters in the network is 12.

3 Datasets

3.1 Dataset 1: Linear Decision Boundary

We generated our first dataset using sklearn's 'make classification' function. We use this function to generate 12000 samples of (x,y) pairs where $x \in \mathbf{R}^2$ and $y \in [0, 1]$. The 'make classification' function will generate these samples such that the $y = 0$ and $y = 1$ groups are clearly sepperable by a linear decision boundary. We split our 12000 samples into a training set (10000 samples) and a test set (2000) samples. We imagined that a single logistic regression unit would have an easy time achieving perfect accuracy, while the more complicated neural networks would overfit the data and perform slightly worse overall.

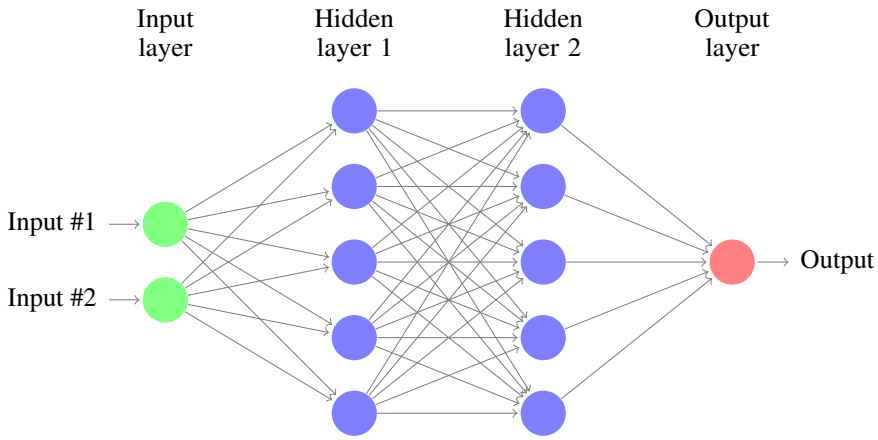

Figure 3: Architecture of 3-Layer Neural Network

3.2 Dataset 2: Perfectly Classifiable for Simple Neural Network

We wanted to create a dataset that our simple 2-layer (2-3-1) neural network could classify perfectly. To do this, we created a neural network with the same architecture as our simple neural net, and we initialized the weights to values sampled randomly using kaiming normal weight initialization. We then generated 12000 random inputs $x \in \mathbf{R}^2$ and ran them through the neural network to get the predictions y. Each of these (x, y) pairs represents a point in our dataset. We then split the samples into a training set (10000 samples) and a test set (2000) samples. We believe an optimal hyperparameter optimization scheme will enable the simple neural network to find the correct weights in a small number of epochs.

3.3 Dataset 3: Perfectly Classifiable for Three-Layer Neural Network

We wanted to create a dataset that our 3-layer (2-5-5-1) neural network could classify perfectly, similar to dataset 2. To do this, we created a neural network with the same architecture as our three-layer neural net, and we initialized the weights to values sampled randomly sampled randomly using kaiming normal weight initialization. We then generated 12000 random inputs $x \in \mathbf{R}^2$ and ran them through the neural network to get the predictions y. Each of these (x, y) pairs represents a point in our dataset. We then split the samples into a training set (10000 samples) and a test set (2000) samples.

4 Methods

4.1 Grid Search

In grid search, we decide on a few values that we want to try out for several different hyperparameters. We then train the network using all combinations of those values. Grid search takes a lot of time and computing resources, but it usually will thoroughly search the hyperparameter space.

4.2 Random Search

In random search, we select our hyperparameters at random from a distribution. The distribution can be any distribution, and a normal or uniform distribution is often used.

4.3 Hypergradient Descent

Hypergradient descent is so name because it perfroms gradient descent on hyperparametres. Specifically, it is an online method which tunes the learning rate by taking the gradient of the learning rate with respect to the loss function. Hypergradient descent can be used with existing optimizers including SGD and Adam [1].

4.4 Bayesian Optimization

Bayesian optimization maintains a surrogate probability model of the neural network to determine the most promising hyperparameters to run on the actual objective function. In this way it is able to improve on random and grid search by focusing on the most promising areas of the hyperparamter space based on the performance of hyperparameters in previous runs. A Gaussian process or Tree Parzen Estimator is typically used as the surrogate model.

4.5 Population Based Training

Population-based training was created by deepmind in 2019 [5]. Population-based training instantiates multiple copies of the neural network and trains them in parallel with different hyperparameter configurations randomly chosen for a fraction of an epoch. We then select a certain number of networks to be saved and trained further. Better performing networks are more likely to be chosen in this selection. In order to preserve diversity in hyperparameters, new models are also evaluated with random hyperparameters during later evaluation. This process of short, parallel evaluations coupled with fitness-based selection ultimately trains the neural network end-to-end, ultimately using a mixture of hyperparameter values during one training run. This algorithm leverages parallelism and fast evaluations to speedily find the best of many different configurations and train it.

5 Metrics

Time to convergence - number of epochs until the $|L_{i-1} - L_i| < \epsilon$

Training accuracy - correct classification/ total

6 Performance

7 Analysis

7.1 Grid Search

Grid search was very time consuming, but it was a reliable way of finding optimal hyperparameter values. The performance of grid search was very similar to that of random search, however the runtime was slower, so we did not include it in the paper.

7.2 Random Search

Random search was a reliable way to find the learning rate, as we expected. Although we had to try training the neural network with a lot of different hyperparameter configurations, which was time consuming, we always found a configuration that lead to fast convergence. We see that on these datasets, the two and three layer neural net are generally better able to fit the datasets than the logistic regression classifier. Here we see the benefit of more powerful neural networks. The impact of overfitting is not overstated.

7.3 Hypergradient Descent

Hypergradient descent worked really well and achieved perfect accuracy on all experiments (it performed well on the simple neural net and three layer neural net on all 3 data sets). We see that the learning rate is changing for about the time during which the loss is decreasing. Once the loss converges the learning rate also converges on a particular value. This method was likely very efficient because changing learning rate online allowed it to better find the optima.

7.4 Bayesian Optimization

We see that bayesian optimization is able to find a near perfect classifier for all 6 experiements. for both the simple neural network and three layer neural network found a nearly perfect option at the beginning and then quickly converged to a nearly perfect classifier on datasets 1 and 3 and a perfect

classifier on dataset 2. In all 6 of these experiments random search found hyperparameters that would nearly perfectly classify the data in 10 trials or less. The Sequence of Values Sampled Graph for each of the Bayesian Optimization experiments shows where values for the learning rate were sampled over the 50 trials. We can see that unlike random search the values are not distributed uniformly and are instead sampled.

7.5 PBT

Population-based training proved to be a somewhat robust approach to finding an accurate learning rate. We see that PBT with the simple neural network converges quickly to a nearly-perfect classifier on dataset one, and it converges even faster on dataset 2, with a surprising ly low accuracy, and dataset 3, with a surprisingly perfect accuracy. It is unclear why the model does not learn how to perfectly classify dataset 2, but it is possible that the learning rate never becomes small enough to allow the network to reach the perfect weights. This does not happen, however, for dataset 3, and the model achieves perfect accuracy despite the increased complexity of the process which generated the dataset. Then, we see that PBD with the three layer neural network converges immediately to a near optimum on dataset one, it takes longer but also reaches a near optimum on dataset two, and it works perfectly on dataset 3. With the three layer neural net, we really start to see the advantages of PBT for training neural networks. Even though the network's training got stuck in a rut on dataset 2, the algorithm was still eventually able to find a learning rate that could cause the model to generate a sufficient gradient again. In this demonstration, we clearly see how PBT can help with gradient-vanishing problems and saddle points in neural networks.

Method	Trials to Convergence	Training Accuracy	Parameters Optimized
Hypergradient Descent	<25 iterations	100	1
Bayesian Optimization	10 trials	99.9	1+
Population Based Training	1	100	1+
Random Search	<5 trials	99.9	1+

Figure 4: Table of results for 5 methods surveyed

8 Conclusion

We found that all of the techniques we implemented and ran trails for performed very well and were able to find hyperparameters to nearly perfectly classify the the datasets. On these fairly small neural networks with small datasets each method was able to find optimal hyperparameters with limited computation. We found that hypergradient descent was the most efficient method at perfectly classifying the datasets. It was able to perfectly classify all 6 experiments. Because it found the learning rate online the neural network only needed to be called once unlike all other methods where each network and dataset needed to be run for different trails. Thus, hypergradient descent was the most efficient and effective method on our neural networks and datsets. However, hypergradient descent is only able to tune the learning rate.

If we would like to tune other hyperparameters such as momentum the method we found worked best was PBT in terms of both speed and accuracy. All methods showed comparable performance in terms of accuracy, but PBT was the fastest. This explains why the industry standard is often random or grid search. Both of these methods provide optimal results without needing to set up another algorithm such as Bayesian optimization or PBT.

We would consider furthering this research by applying these techniques to larger neural networks and larger data sets that are not perfectly classifiable. This would provide a more accurate picture of how different hyperparameter tuning methods will perform on real world neural networks.

9 Acknowledgements

We want to thank Mykel Kochenderfer for being the best professor ever (the BOMBdp!)!

10 Contributions

Daniela worked on the hypergradient descent and bayesian optimization Methods.
Noah created the datasets and worked on population-based training.

11 Code

You can find the code for this project at: https://github.com/noahj08/autotune

References

[1] Atilim Gunes Baydin, Robert Cornish, David Martínez-Rubio, Mark Schmidt, and Frank D. Wood. Online learning rate adaptation with hypergradient descent. *CoRR*, abs/1703.04782, 2017.

[2] Ted Moskovitz, Rui Wang, Janice Lan, Sanyam Kapoor, Thomas Miconi, Jason Yosinski, and Aditya Rawal. First-order preconditioning via hypergradient descent, 2019.

[3] Matthias Feurer and Frank Hutter. *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, Cham, 2019.

[4] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

[5] Mykel J Kochenderfer and Tim A Wheeler. *Algorithms for Optimization*. MIT Press, 2019.

[6] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. 2011.

12 Appendix

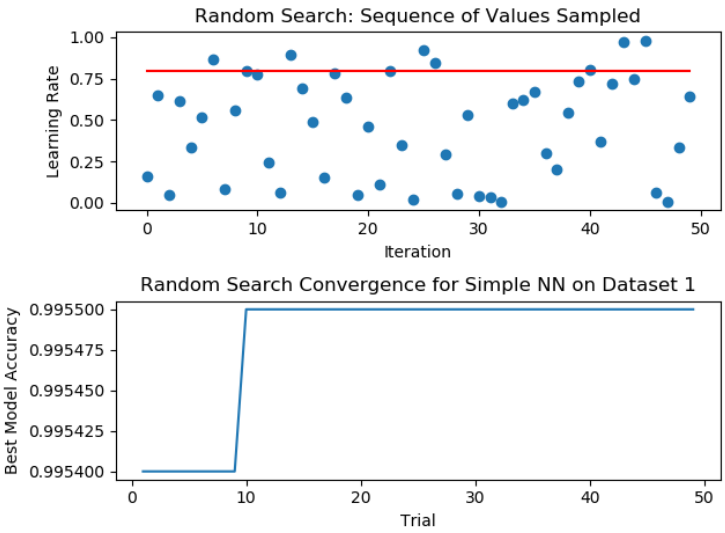

Figure 5: Random Search: Simple Neural Net on Dataset 1

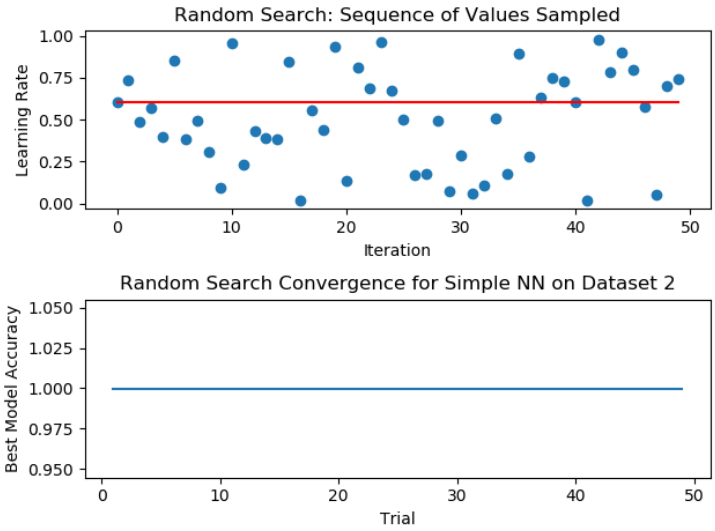

Figure 6: Random Search: Simple Neural Net on Dataset 2

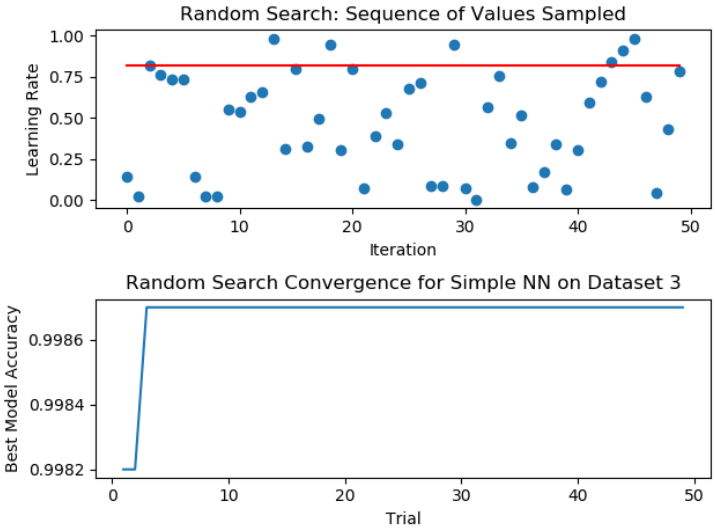

Figure 7: Random Search: Simple Neural Net on Dataset 3

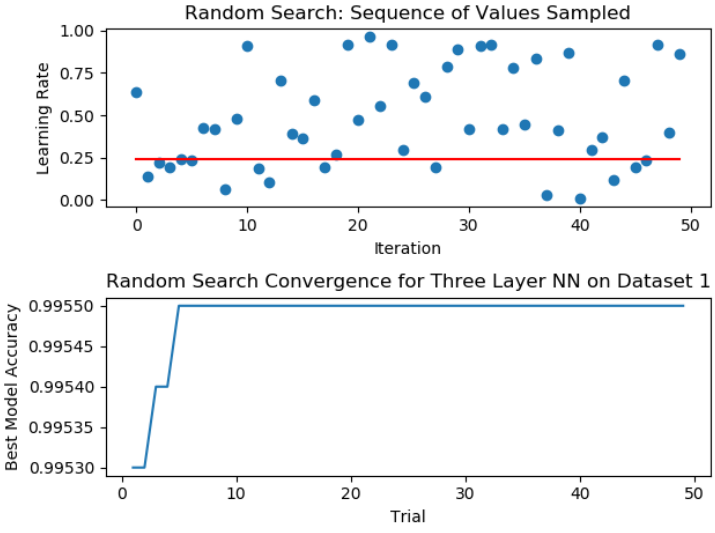

Figure 8: Random Search: Three Layer Net on Dataset 1

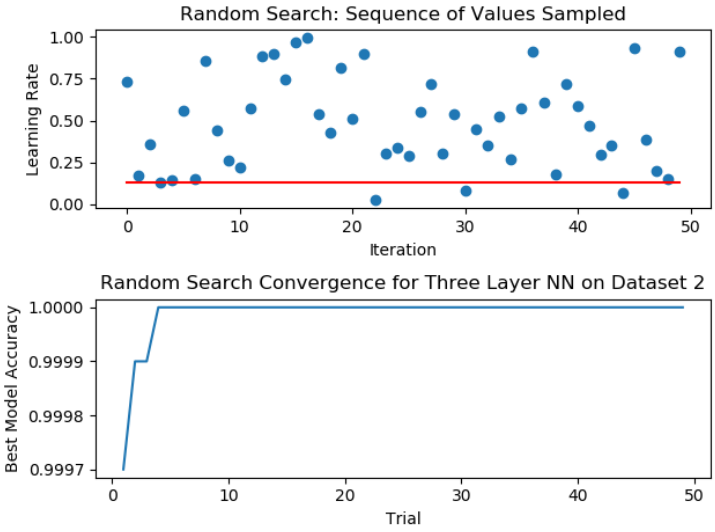

Figure 9: Random Search: Three Layer Net on Dataset 2

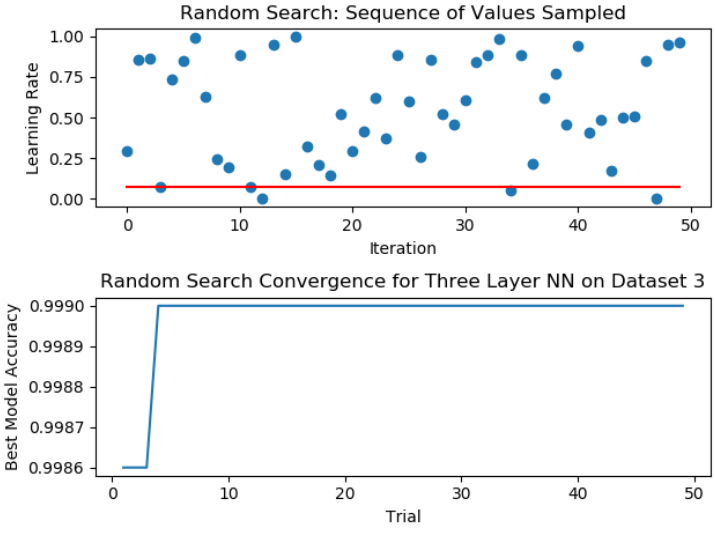

Figure 10: Random Search: Three Layer Net on Dataset 3

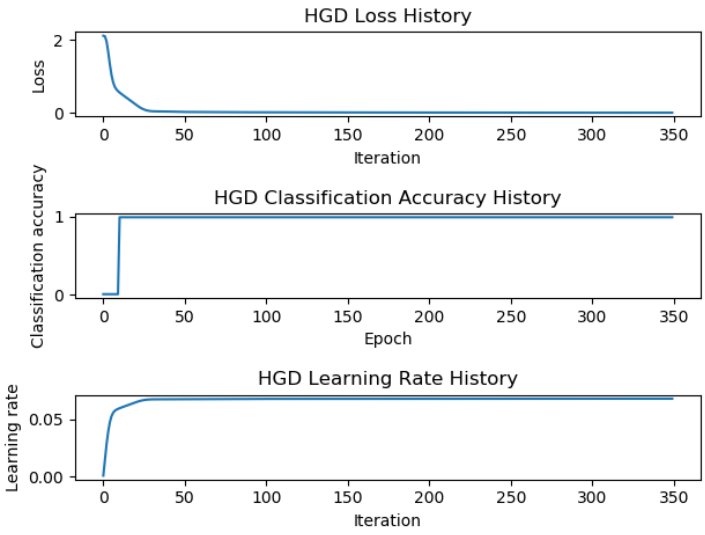

Figure 11: Hypergradient Descent: Simple Neural Net on Dataset 1

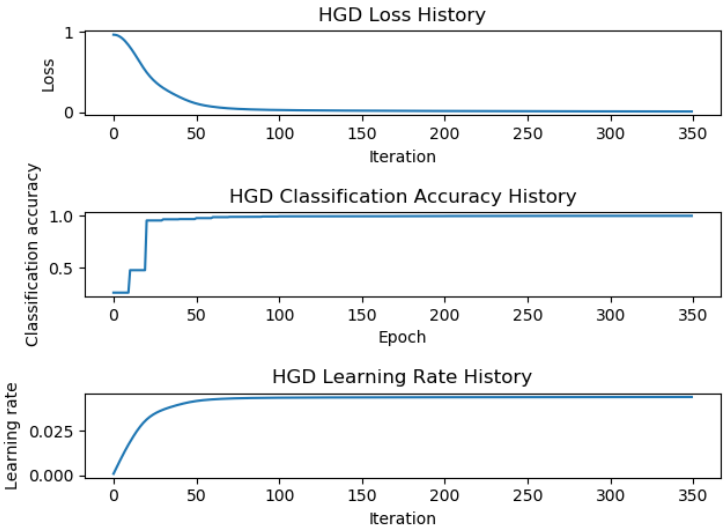

Figure 12: Hypergradient Descent: Simple Neural Net on Dataset 2

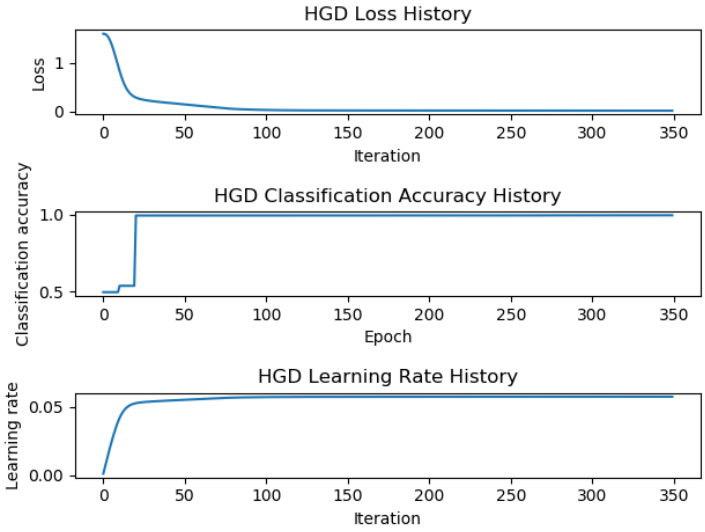

Figure 13: Hypergradient Descent: Simple Neural Net on Dataset 3

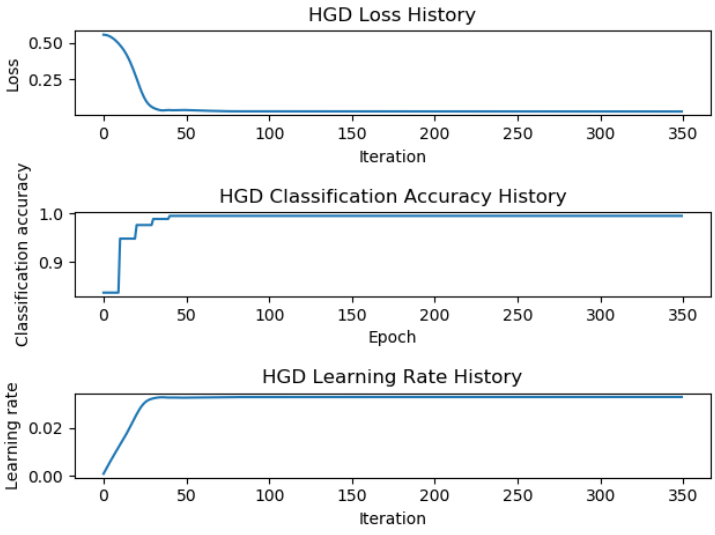

Figure 14: Hypergradient Descent: Three Layer Net on Dataset 1

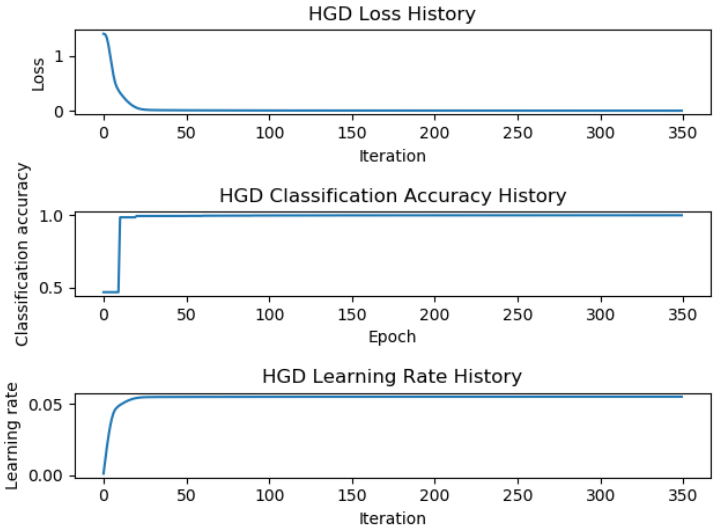

Figure 15: Hypergradient Descent: Three Layer Net on Dataset 2

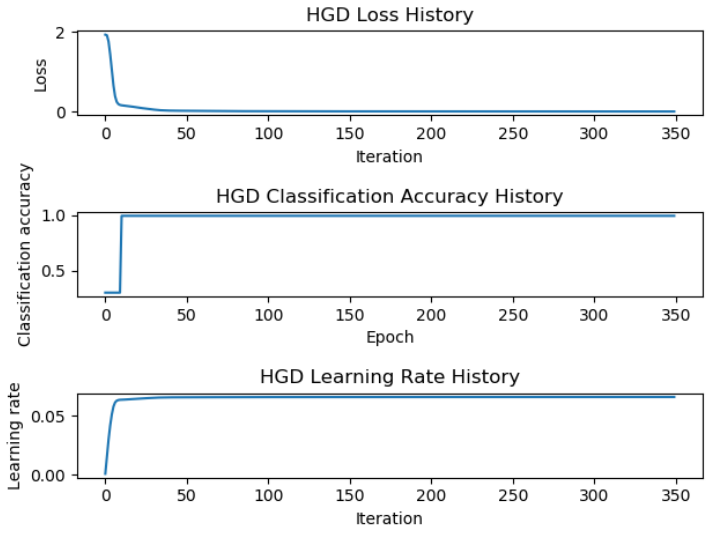

Figure 16: Hypergradient Descent: Three Layer Net on Dataset 3

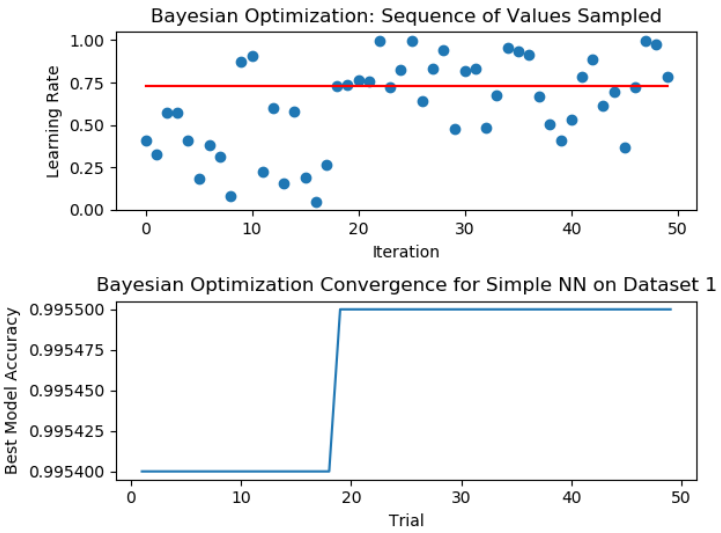

Figure 17: Bayesian Optimization: Simple Neural Net on Dataset 1

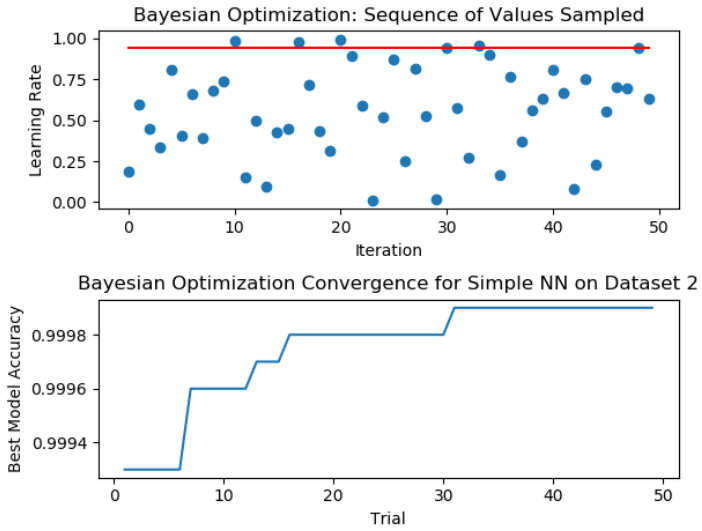

Figure 18: Bayesian Optimization: Simple Neural Net on Dataset 2

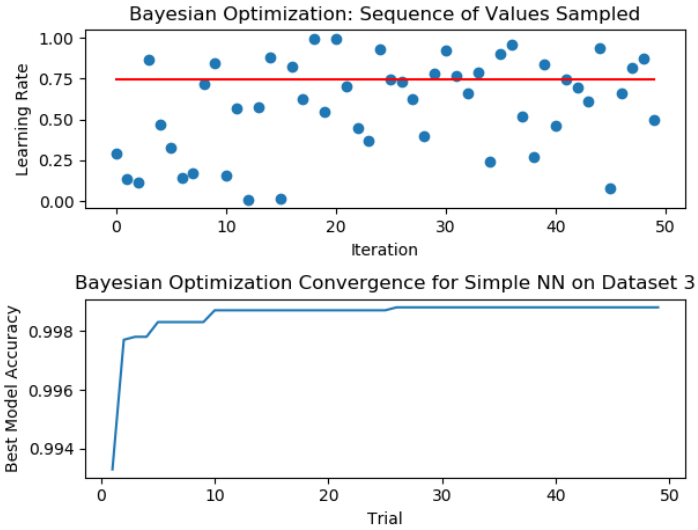

Figure 19: Bayesian Optimization: Simple Neural Net on Dataset 3

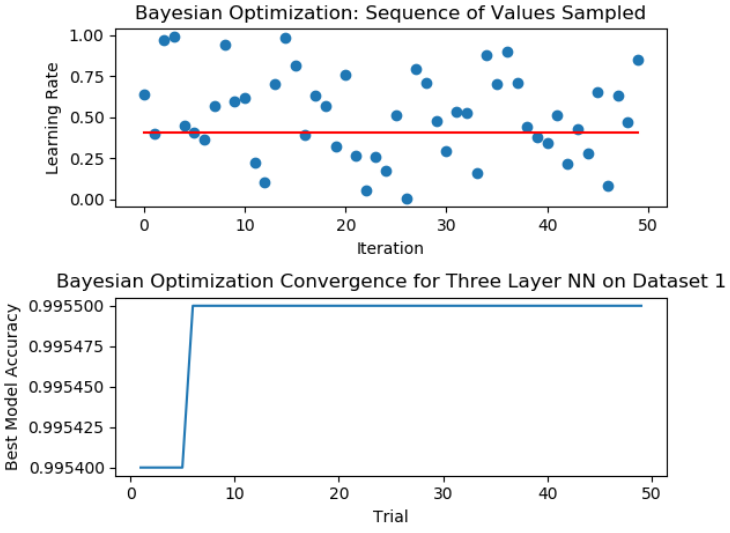

Figure 20: Bayesian Optimization: Three Layer Net on Dataset 1

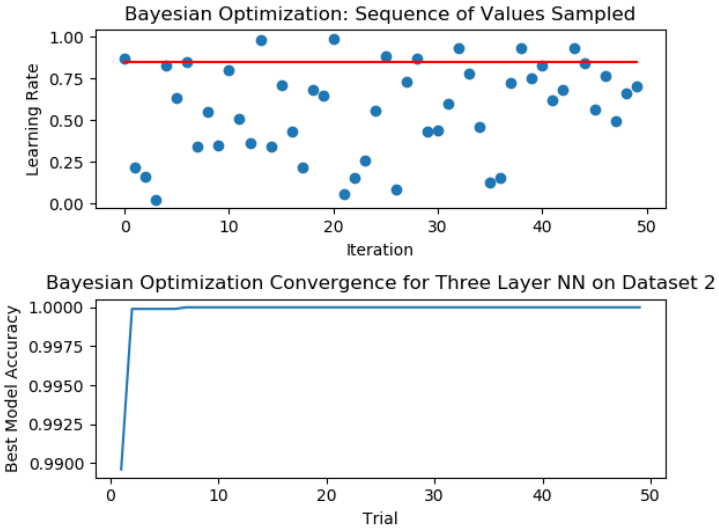

Figure 21: Bayesian Optimization: Three Layer Net on Dataset 2

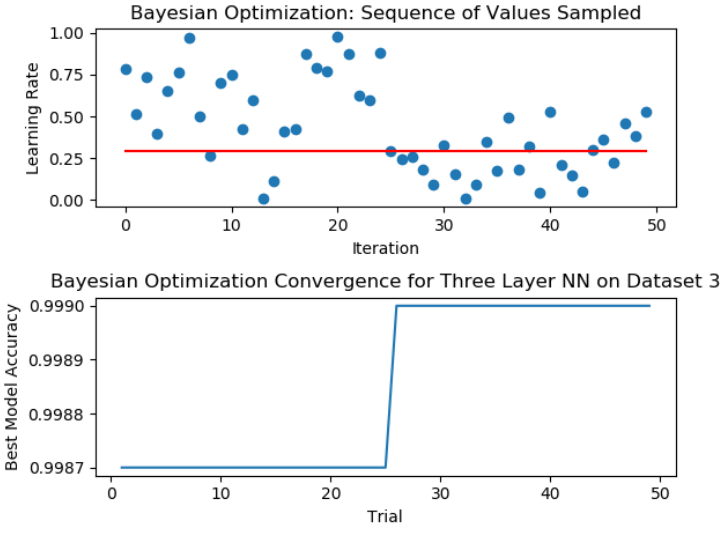

Figure 22: Bayesian Optimization: Three Layer Net on Dataset 3

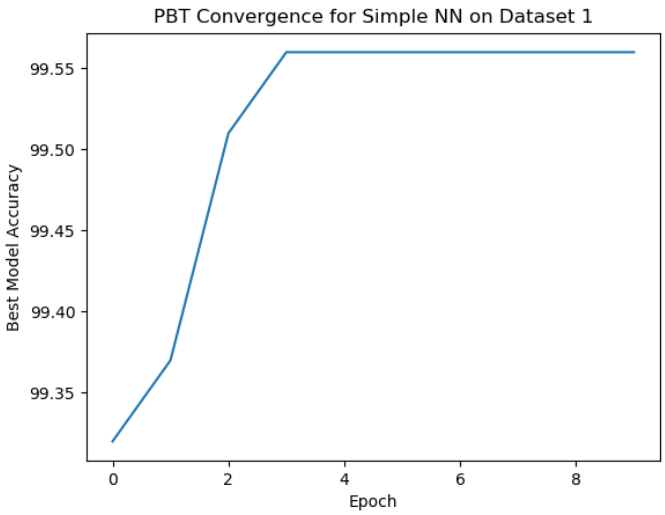

Figure 23: Population Based Training: Epoch vs Best Model Accuracy for Simple Neural Net on Dataset 1

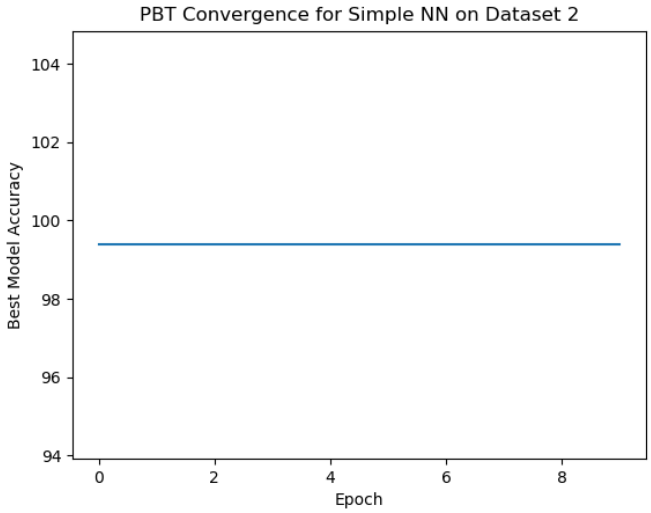

Figure 24: Population Based Training: Epoch vs Best Model Accuracy for Simple Neural Net on Dataset 2

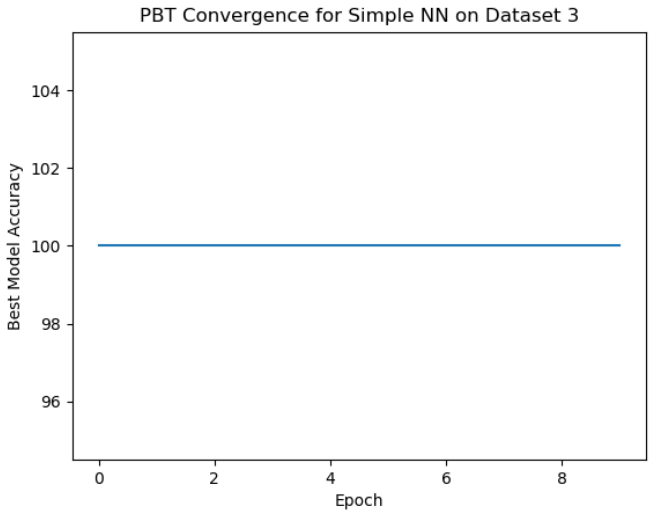

Figure 25: Population Based Training: Epoch vs Best Model Accuracy for Simple Neural Net on Dataset 3

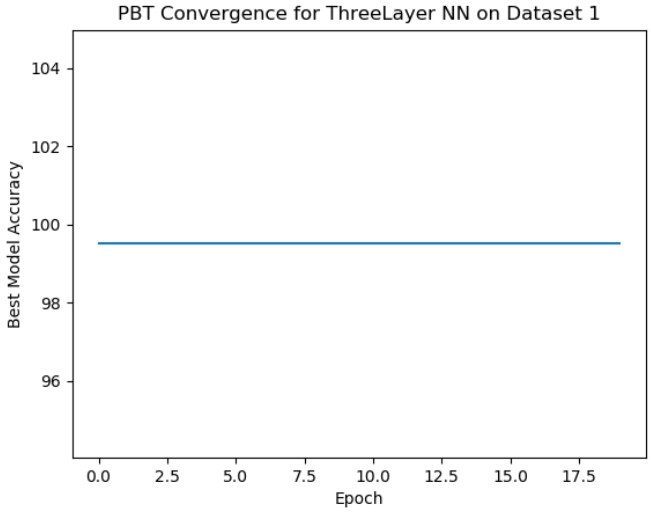

Figure 26: Population Based Training: Epoch vs Best Model Accuracy for Three Layer Neural Net on Dataset 1

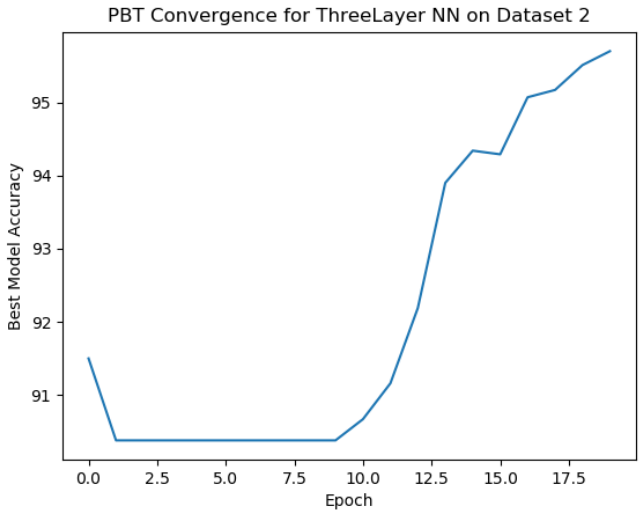

Figure 27: Population Based Training: Epoch vs Best Model Accuracy for Three Layer Neural Net on Dataset 2

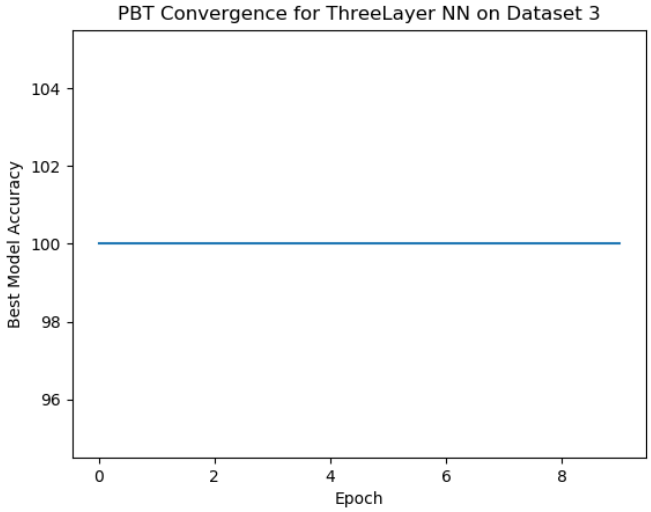

Figure 28: Population Based Training: Epoch vs Best Model Accuracy for Three Layer Neural Net on Dataset 3

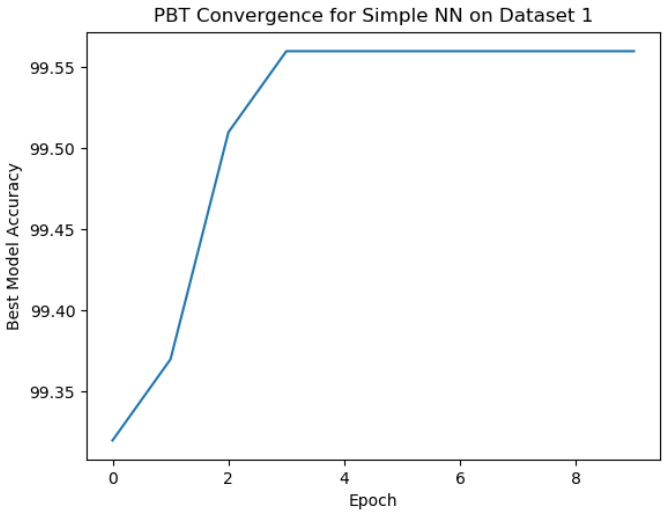

Figure 29: Population Based Training: Epoch vs Best Model Accuracy for Simple Neural Net on Dataset 1

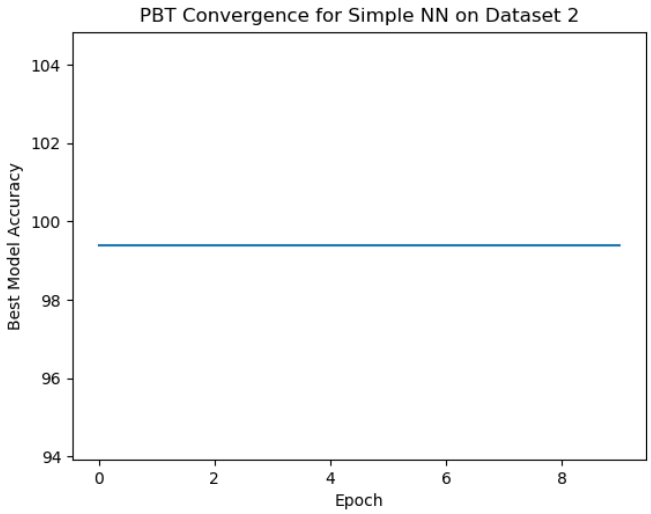

Figure 30: Population Based Training: Epoch vs Best Model Accuracy for Simple Neural Net on Dataset 2

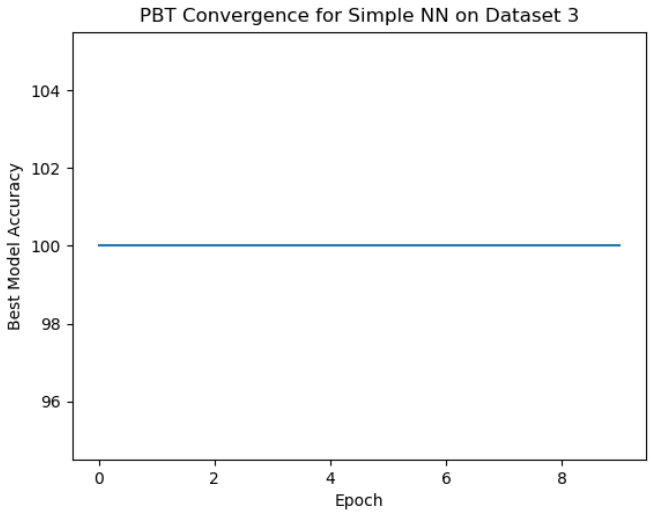

Figure 31: Population Based Training: Epoch vs Best Model Accuracy for Simple Neural Net on Dataset 3

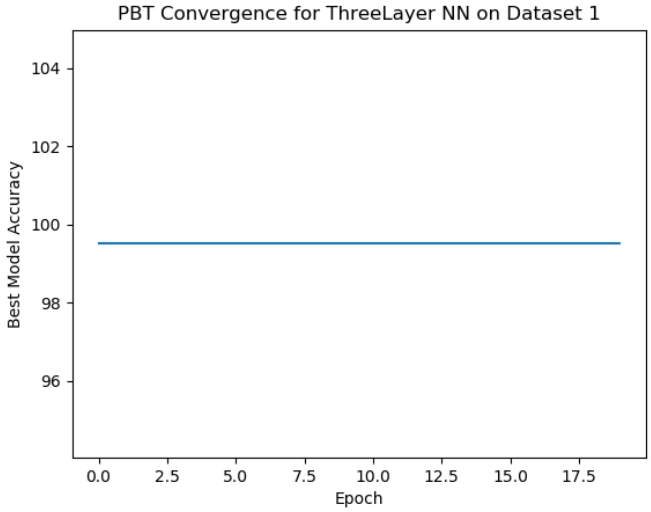

Figure 32: Population Based Training: Epoch vs Best Model Accuracy for Three Layer Neural Net on Dataset 1

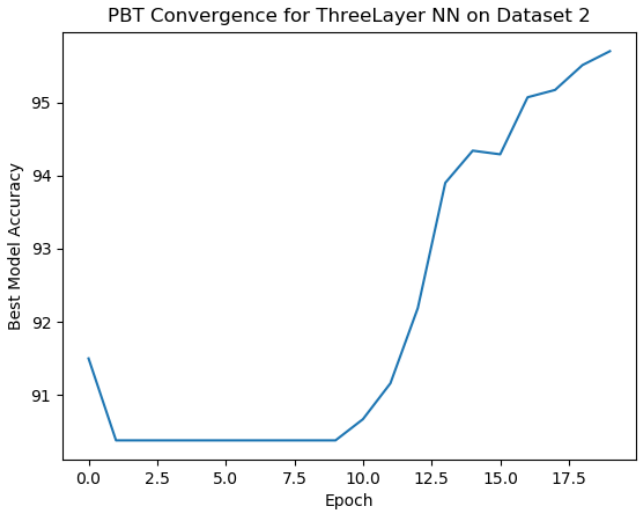

Figure 33: Population Based Training: Epoch vs Best Model Accuracy for Three Layer Neural Net on Dataset 2

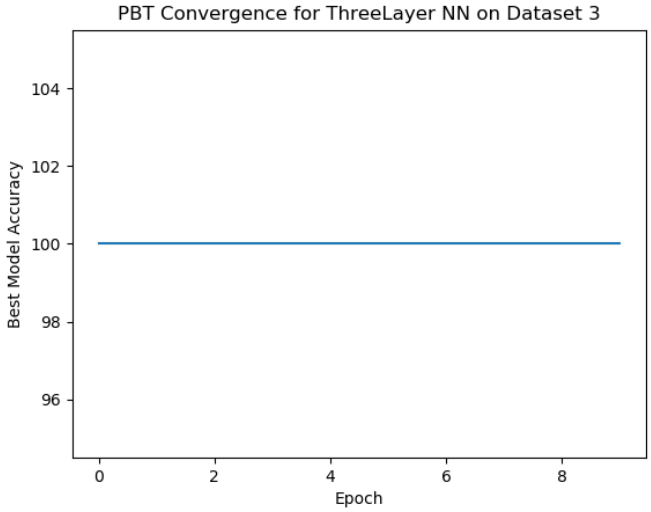

Figure 34: Population Based Training: Epoch vs Best Model Accuracy for Three Layer Neural Net on Dataset 3