Three Methods of Hyper-Parameter Optimization

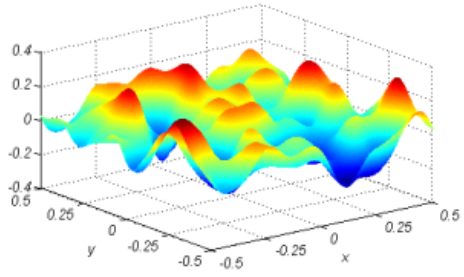
B Chase Babrich
University of Massachusetts, College of Information and Computer Sciences
CS682 Final Paper Submission
140 Governors Dr, Amherst, MA 01002
bbabrich@umass.edu

Abstract

In this study we consider three different methods of hyper-parameter optimization: completely random search, Poisson Disk Sampling, and a novel method we call Auto Manual Search, which we mean to mimic the way a human often tunes hyper-parameters. We constrain ourselves to searching for hyper-parameters for neural networks, and we perform our experiments on two simple networks using the MNIST basic data set. Paradoxically, we are forced to use several "meta" hyper-parameters, which include number of points to use in each search, hyper-parameter range, and a radius value for Poisson Sampling. We find that Poisson and Random methods perform similarly due to their analogous behavior although Poisson presents several more difficulties in its implementation, and that both seem to outperform Auto Manual when given enough points to search with.

1. Introduction

One of the biggest reasons hyper-parameter optimization is such a difficult problem is that the true relationship between hyper-parameters and a network's performace is unknown, and it may in fact be a very complex surface.



Screenshots taken from www.mysimlabs.com/surface_generation.html

Above, imagine the bottom two axes to be hyper-parameters and the z-axis to be the accuracy returned by a network. Were it the case that we could explore this space freely, the problem would not be so hard: we could simply start at any point and perform gradient ascent on the surface until the maximum accuracy was returned. Unfortunately, the z-coordinate of each point on the surface above can only be obtained after an entire neural network is trained and tested, and so we see that exploring hyper-parameter space is expensive.

There is also the issue of randomness which is inherent in each experiment. Each time the experiment is run, different parameters are set, and so the true function changes slightly each time. If we were to average all of the "true" functions for each setting of the parameters together, it would look even noiser than the plot above. For that reason, it is hard to confidently put forth a set of hyper-parameters as the "best" coordinate, because it may actually only be the best one for a surface which corresponded to a completely different set of parameters.

Intriguingly enough, the search for hyper-parameters forces us to choose yet more hyper-parameters, such as what range we would like to search for our hyper-parameters as well as how many points we are allowed to work with. Had we automated the choosing of these, we certainly would have run into questions such as "what is our maximum number of points we can work with?" which then produce more hyper-parameters. It seems it is at best inefficient and and at worst impossible to eleminate hyper-parameters entirely.

For all experiments in this study, we consider a neural network which has the following variable hyper-parameters: learning rate, hidden size, batch size, and number of epochs.

2. Related Works

The most naive way to go about searching for hyper-parameters is by simply computing a grid of points in the hyper-parameter space and choosing the one which corresponds to the highest accuracy returned by the network being tuned. While this is an easy method to implement, Bergrstra found that it cannot beat completely randomly selecting the hyper-parameter points in terms of statistical success in finding high-scoring points [1].

Since the true relationship between hyper-parameters and a network's performance is unknown, many currently developed methods use Bayesian methods in pursuit of learning enough about the surface to find the best point within reason, and these have been found to perform fairly well [5] [8]. Unfortunately, their performance is still reliant upon chance, and so others have conducted studies in which more creative methods of searching were used. Many of them are a twist on the idea of being able to stop a network before it is finished training.

For example, Li et. al developed what they called the "hyperband" strategy, which attempts to make random search more efficient by allowing for adaptive resource allocation, as well as the ability to stop a so-far unsucessful network before it wastes time training [6]. Similarly, looking ahead along a given network's learning curve to check whether or not continuing training is worth it has also been studied [4].

Even more complex methods have also been studied, such as fully integrating the hyperparameter selection into the backpropagation process [7] and implementing a unique meta data processing step [2]. However, when one also considers the fact that random selection is extremely easy to implement, it is not hard to see why it has risen to the top as the preffered method of hyper-parameter optimization.

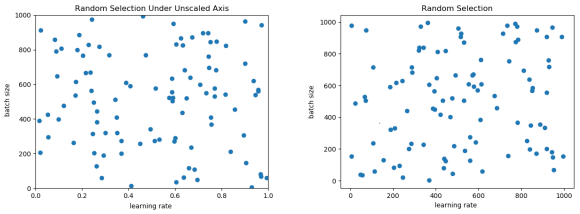
3. Approach

3.1. Random Sampling

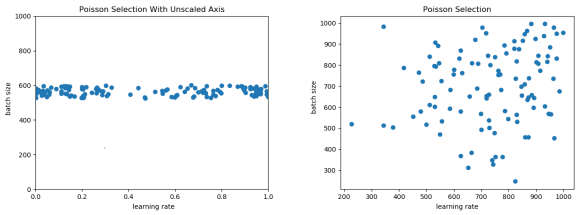
For each point sampled completely randomly, we simply choose a random integer value somewhere in the hyper-parameter range along each axis. Notably, this method is the fastest in terms of computation.

3.2. Poisson - Scaled V Unscaled Axis

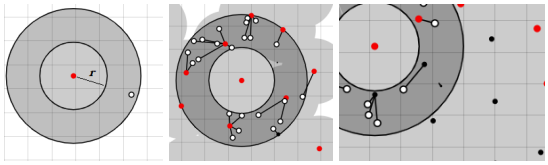
On top of being the easiest method to implement, random selection also has the added benefit that the points in space are spread somewhat evenly throughout the range of values being searched.



Further, this is true no matter which axis scale we choose; however, the same cannot be said for Poisson Disk Sampling.



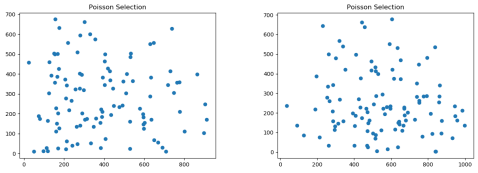
The linear clumping pattern we see in the first plot above is due to the fact that Poisson Disk Sampling operates using a fixed radius to search for new points in. If our range is anything greater than one, any new points chose will be rejected unless they lie almost perfectly along the same axis as the previous points.

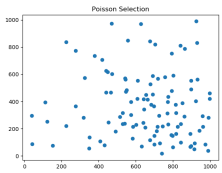


Screenshots taken from the animation at Mike Bostock's blog page https://bl.ocks.org/mbostock/dbb02448b0f93e4c82c3

For each new point chosen, we search for points in the area whish is at least distance r away from the previous point and at most distance $2 \cdot r$ away [3]. Depending on which r we choose, we limit ourselves to different size areas of the entire hyper-parameter space. Since the scale of horizontal axis on the second graph is so much smaller than that of the vertical axis, we are forced to use a radius which limits us to a small portion of the entire space we are searching.

Thus, we use the same scale for all axes and re-scale the hyper-parameters down once they are selected. Notice that the radius we choose is still a factor for Poisson in this case, hence the "bursting out" effect we see in these plots below.





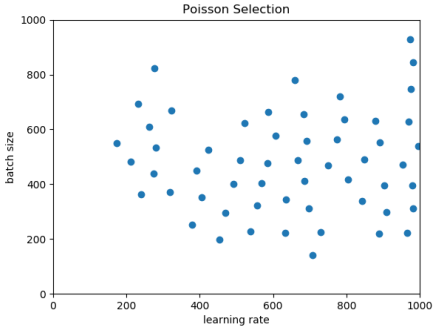
3.3. Poisson - Choosing a Radius

Clearly, there is a relationship between our hyper-parameter range and the radius we choose for Poisson. In order to prevent the radius from becoming yet another manifested hyper-parameter which we have to manually enter ourselves, we implement a method which guesses a random radius and then only returns a batch of points if the size of the batch is close to the desired number of points is too big:

Poisson Sampling

1. *Choose random radius r*
2. *sample points using r :*
   3. *While sampling, if number of points exceeds desired number of points, return and go back to 1.*
   4. *else entire range has been explored; return sampled points*

This adds even more computation time to this Disk Sampling method, and it still does not lead to a point distribution which covers the entire space evenly even if a radius is found which leads to an even distribution. We also note that this method adds even more randomness to the entire study, which in turn adds more noise to the data and uncertainty to each individual result.



Furthermore, it still leaves a need for a specified range to sample the random radius value from, a subtle yet unfortunately present hyper-parameter.

An Auto Manual Method

In the Auto Manual Method, we attempt to mimic what a human might do when adjusting the hyper-parameters themselves, which is to fix all other axis constant and tweak one at a time to achieve the highest accuracy.

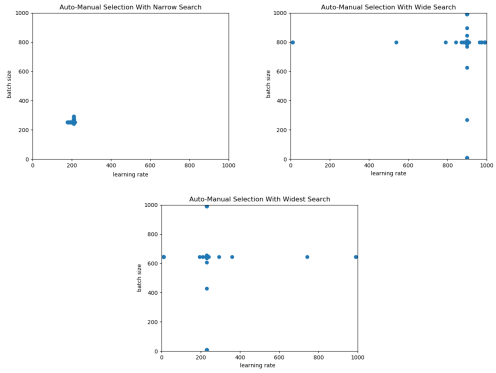
We implement this method by first picking a random point in space and doing just this. In order to emualte "tweaking", we feed our current axis value h into one of the following three equations,

$$h \cdot k \tag{1}$$

$$h^{int[k^{log(k)}]} \tag{2}$$

$$h^k \tag{3}$$

where k is the number of failed consecutive trials. Note that that they are are listed from narrowest to largest in terms of how much they tweak each time, and so they correspond to the following three graphs:



It should be also noted that this method picks which axis it travels along in random order to save the user from as much work as possible.

4. Experiment

4.1. Data

Inspired by Bergrstra's original paper, we used the MNIST Basic data set for our experiments.

Some classic MNIST Basic Figs https://conx.readthedocs.io/en/latest/MNIST.html

Therefore, the task of each neural network was to correctly label each image in the either the validation or test set once it had finished training using our supplied hyper-parameters.

4.2. Methods

Two experiments were performed, one done in two dimensions and one done in four. We shall call them "experiment A" and "experiment B" respectively.

In experiment A, each method was used to sample 50 points. These 50 points were then run on a simple fully connected network (below) using a traditional train/val/test split, and the point which was found to perform the best on the validation set was used to train a final network before returning its accuracy. This was done 5 times, thus, overall, $3 \cdot 50 \cdot 5 = 750$ points were tested in experiment A. The two dimension sampled were learning rate and batch size.

Affine: *Linear, ReLU, Linear (one hidden layer)*

Experiment B was similar to experiment A with a few adjustments. First, hidden layer size and number of epochs were added as dimensions. Second, 25 points were sampled each time. Third, these points were tested on the affine network described above as well as a convolutional network with the following structure:

Convolutional: *Conv2d, Conv2d, Linear.*

Experiment B ended up testing the exact same number of points as experiment A: 2 *networks* $\cdot 5$ *trials* $\cdot 3$ *sampling methods* $\cdot 25 = 750$.

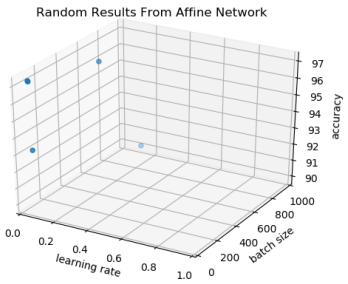
It should also be noted that the radius selection for Poisson, discussed above, was only used in experiment B, while a hand-tuned radius was used for experiment A.

4.3. Experiment A Results

The following averages were returned from the experiment A:

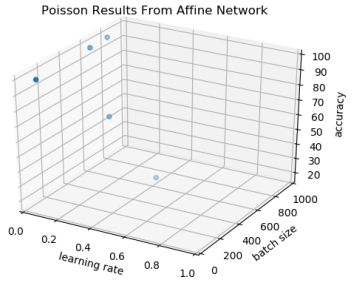
	Affine
Random	93.4%
Poisson	73.4%
AutoMan	55%

Clearly, the random sampling method fared the best out of the three, with a range of 7 and a minimum accuracy of 90%:

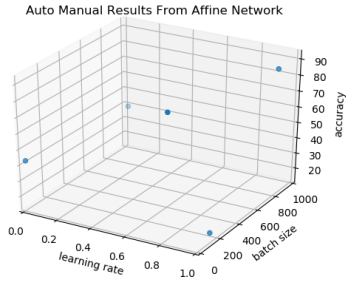


Curiously, only four points are visible due to the fact that the points (0.009, 94, 97) and (0.002, 101,97) were so close together

Poisson did not perform as consistently, returning only 3 points with accuracies above 90%:



Auto Manual performed the worst, barely managing to return one point with an accuracy of 90%:

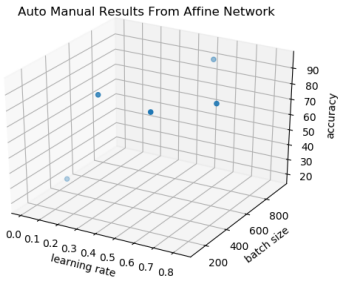


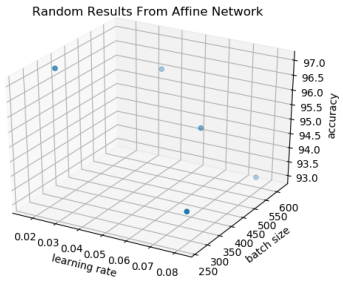
4.4. Experiment B Results

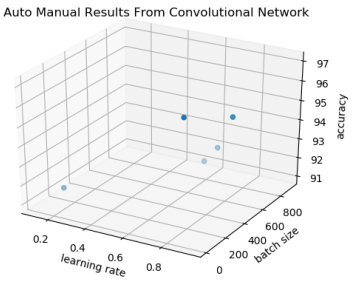
The following averages were returned from the experiment B:

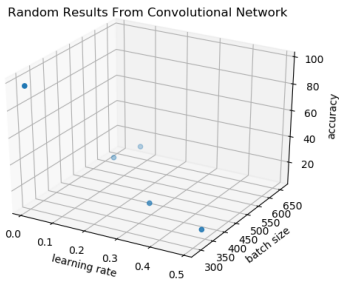
	Affine	**Conv**
Random	95%	47.4%
Poisson	72.2%	27.2%
AutoMan	77.6%	93.4%

Perhaps surprisingly, the Auto Manual method seems to have been the more consistently succesful of the three, achieving the highest accuracy in the case of the convolutional network and being the only method to achieve higher than 75% on both networks. We can also see that the method seemd to find points which are clustered together:

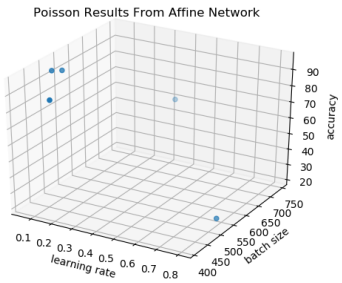


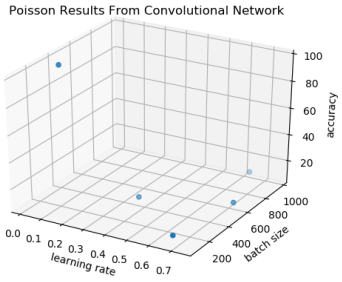






We can also see that most effective hyper-parameter settings for this data set live on the lesser side of the learning rate axis and the greater side of the batch size axis.





These patterns are somewhat repeated when we plot our random results.

5. Analysis

5.1. Random Sampling - The Most Effective Method

In both experiments, random sampling managed to find a point which returned above a 90% accuracy on the affine network. Thus, our research seems to support the claim that random sampling is the most economical method of sampling. Furthermore, as mentioned above, random sampling seems to hint at some sort of structure in the function between hyper-parameters and accuracies, since it consistently returned points with low learning rates and high batch sizes. We even run into the curiosity of two points repeating almost exactly.

However, random sampling performed very poorly in the case of the convolutional network in experiment B. This implies that the function between hyper-parameters and accuracies is much noisier in this case, and so it was much less probable to land on a spot where a high peak existed. In general, it could be argued that random sampling should be used on networks with relatively simple architectures.

5.2. Poisson - Dependent Upon Its Radius

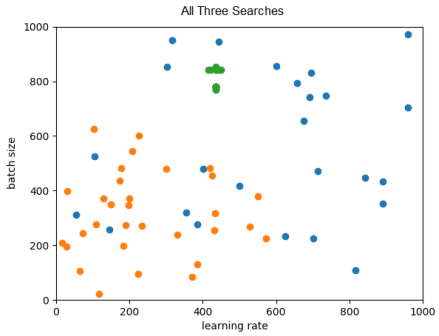
It was not entirely unexpected that Poisson should perform so similarly to random, since it is meant to be random's more uniformly-spread cousin. The reason that it did not perform equally as well or even better than random is most likely due to its success being so

dependent upon which radius we use when sampling points.

In experiment A, we hand-picked a radius for Poisson, and it was only able to return an accuracy which was 20% less than random. Perhaps the space which Poisson did not explore in this case held that last 20%. In experiment B, we attempted to allow the program to choose a radius for us in order to close the gap, but our efforts were not met with success; around the same discrepancy in highest accuracies returned exists for each network.

5.3. Auto Manual - The Most Economical Method

In the case of experiment B, which had such a small budget of points to work with, it makes sense that the auto manual method would fare the best, since it sacrifices space explored in order to gain reliability. The Random and Poisson methods simply do not have enough points to work with in order for their strengths to shine. Had the experiment been run using 50 or 100 points the average test accuracies of these two methods most likely would have caught up.



A snapshot of the points explored in experiment B, which is an example of a poorly chosen Poisson radius value, as well as a lucky first Auto Manual point

We can see this phenomenon occuring in the above graph, in which Poisson points, random points, and auto manual points are colored as orange, blue, and green accordingly. Clearly, the original Poisson point was chosen somewhere in the bottom left corner and the original auto manual point somwhere up toward the top of the graph.

Of course, much of Auto Manual's success depends upon where the first point it chooses lands. It may be the case that this first point is near a local maximum which the method will then latch onto, and other maxima which may be hold even higher accuracies will be left unexplored. While Auto Manual is not guaranteed to find the highest accuracy within a given range, it will return at least a somewhat acceptable point most of the time.

6. Conclusion

To a computer scientist, the necessity to manually alter values in code is a jarring and often frustrating one. Currently, the relationship between hyper-paramters and a neural network's performance are not well understood enough to yield an objective function which tells us exactly which values to use.

This, coupled with the noise added by randomized aspects of a network's initialization, have meant that simply guessing the values has turned out to be the most economical method of optimization. We have compared this method against Poisson Disk Sampling and what we call Auto Manual Selection.

Poisson behaves analogously to random selection in that it somewhat covers the search space uniformly and grows in effectiveness with the number of points being tested. However, its ease of implementation is hindered by the relationship between which radius it uses and the ranges of each hyper-parameter desired. It struggles to explore the entire space being searched as a result.

Auto Manual search's strength seems to lie in its ability to find at least a somewhat acceptable point with a small number of test points to work with due to the fact that its method is not based on randomness at all. However, as the number of points being tested grows, probability of success shifts over to random and Poisson's side since they explore more of the space at once.

The most philosophically interesting result from our study was the discovery that hyper-parameters themselves seem unavoidable. Although our original goal was to completely automate the process of hyper-parameter selection, it turns out that when one hyper-parameter is automated, another is manifested in its place. We observed this phenomenon in our use of the desired number of points and hyper-parameter range variables. Further, we also ran into this issue when implementing Poisson and trying to find the correct radius to use.

We have several recommendations for anyone wishing to build upon this work. Several more tests should be performed using small sampling sizes in order to determine if the success of Auto Manual in experiment B was a fluke or not. Also, different implementations of Auto Manual should be studied. Perhaps its performance could be improved if it switched between axis each time an adjustment was made, thus exploring the space in more than just one dimension at a time. It would also greatly benefit from a more effective method of choosing a starting point then complete randomness so it could better avoid gettings stuck on local maxima.

Furthermore, a large part of the failer of Poisson could

reasonably be attributed to how its radius was chosen. While we attempted to create a method which would choose this radius for us, it ultimately did not seem to help. Ideally, some sort of formula using the desired number of points and hyper-parameter range would be implemented. Less specifically, these same tests should be performed on different data sets and with different and perhaps more complex network architectures.

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization, 2012.

[2] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, 2013.

[3] R. Bridson. Fast poisson disk sampling in arbitrary dimensions.

[4] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, 2015.

[5] A. Klein, S. Falkner, S. Bartels, Philipp, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets, 2017.

[6] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talkwaker. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[7] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning, 2015.

[8] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

Please feel free to view the code used to generate this report at https://github.com/bcbabrich/CS682FinalProject