Gradient Boosting Regression Trees with Variable Learning Rate: Progress Report

Austin Barket

Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
amb6470@psu.edu

1 Proposed Work Summary

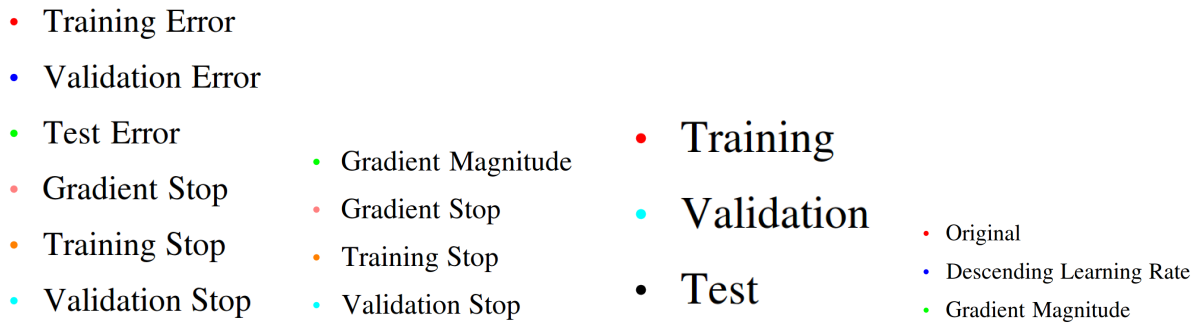


Fig. 1: Plot Legends for Error Curves (Left), Gradient Magnitude Curves (Middle Left), Learning Curves (Middle Right), and ParameterCurves (Right)

The goal of the this project is to explore the effect of variable learning rates on gradient boosting machines that utilize regression trees as the base learners. Until now all research and implementations of gradient boosting machines have used only constant learning rates. The conventional wisdom has been to use small learning rates of 0.01 or lower as this always seems to lead to high accuracy models with a low risk of overfitting. However this comes at the cost of increased computation time because more base learners must be trained [**?**].

We believe that variable learning rates provide an unexploited area of research and hypothesize that an intelligent method of adapting the learning rate can improve the convergence speed without compromising the model's resilience to overfitting. The use of regression trees as the base learners presents an interesting possibility of a simple, yet elegant adaptation method. Since the regression trees themselves can be seen as a summation of individual prediction terms, one for each leaf in the tree, a natural adaptation scheme is to apply a different learning rate to the predictions made by each leaf in the tree. Specifically, the lower the number of examples in a given leaf node, the lower its learning rate ought to be to discourage overfitting to the examples. On the contrary, a leaf node containing a large number of training examples poses less danger

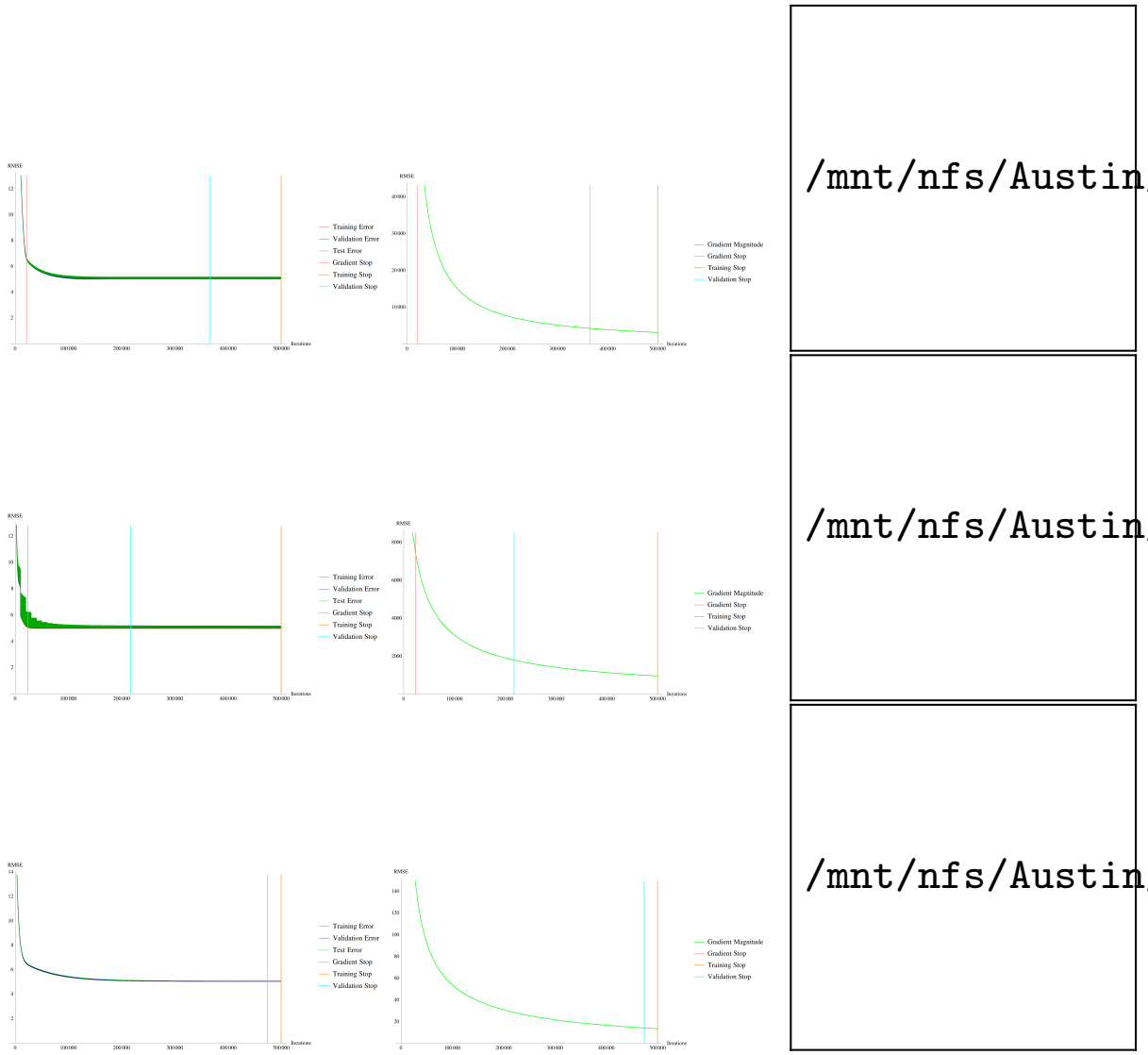


Fig. 2: Power Plant: Error, Gradient Magnitude, and Learning Curves for Best Original (Top), Descending Learning Rate (Middle), and Gradient Magnitude Scaling (Bottom)

to the generalization ability of the overall learner, and thus it should be safe to apply relatively high learning rates to these leaves.

Specifically, the model will take as input a maximum learning rate v_{max} instead of the constant learning rate v. The following equation will then be used to compute the learning rate for each of the J leaves in tree m.

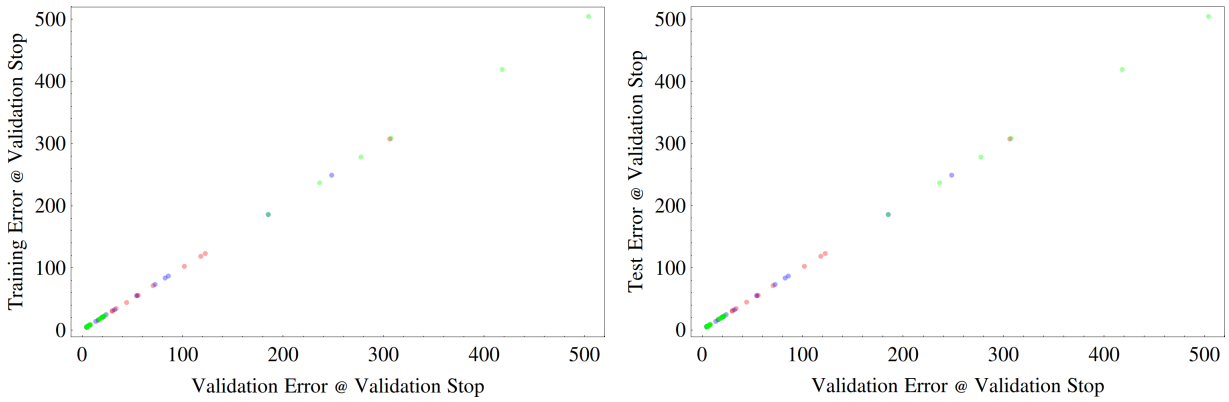


Fig. 3: Validation Error vs. Training and Test Error

$$v_{m,j} = v_{max} \cdot \frac{|R_{m,j}|}{|S_m|} \tag{1}$$

2 Completed Work Summary

I have completed the core of the implementation and am now working on designing and running my experiments and collecting data for analysis. As mentioned in the proposal, my original intention was to extend the existing production quality gbm package in R to support my variable learning rate scheme. After a week of struggling to get up to speed with their implementation, I decided the best way forward would be to work with a simpler implementation I found written in Java, known as JSGBM. However, after beginning to use JSGBM, I became dissatisfied with its performance and limitation of only supporting numerical attributes in its regression trees. Thus, I set out to implement my own regression tree, focusing on an efficient implementation of the algorithm to find the next optimal split and adding support for splitting on categorical attributes. I've also implemented a cross validation based method of finding the optimal number of trees to use, similar to that found in the gbm R package.

So far I have done some extensive testing on a wide range of parameter combinations, in an effort to reduce the number of parameter combinations I'll need to look at for my final tests. Note that in all the experiments described below, I first select randomly without replacement 20% of the dataset to set aside as a test set, this same partitioning is used for all combinations of parameters tested. In the descriptions below, training a single model on a given combination of parameters involves performing 4 fold cross validation on the remaining 80% of the dataset. E.g. that 80% is shuffled again, then 4 different gradient boosting machines are constructed, each using a different 20% of the original dataset as the validation fold, and remaining 60% as the training set. Training stops once the average RMSE across the 4 models stops descending and begins to increase; or when the maximum number of trees is reached. The optimal number of trees is defined as the number of trees corresponding to the minimum average RMSE across the cross validation folds. Alongside the cross validated models, a model is trained using all 80% of the training data. The optimal number of trees found via cross validation is associated with the all training data model, which is then evaluated against the held out test set.

Gradient boosting machines with regression tree base learners are defined by 5 parameters. The maximum number of trees to grow, learning rate, bag fraction, minimum number of examples in a leaf node, and the maximum number of splits in each tree.

After some initial testing on the power plant dataset [**?**], I determined that it should be safe to hold the minimum examples in each node parameter constant in future tests. My process was to hold the minimum examples in each node constant at 1, 10, 100, and 1000 in turn. For each value, I then varied all the other parameters through a variety of configurations, using 500,000 as the maximum number of trees. I found that approximately the same minimum RMSE was achieved for the values of 1, 10. Minimums of 100 and 1000 resulted in increasingly higher minimum error values, both for constant and variable learning rates. This seems to make sense in that the interactions between the maximum number of splits and learning rate should be able to sufficiently regularize the complexity of the regression trees on their own. Using a high minimum on the examples in each leaf node forces the algorithm to make sub optimal splits, which lead to higher error rates. With this in mind, I decided to simplify future testing by holding the minimum number of examples in each node constant at 1.

This week I've been running similar parameter tuning tests on 3 datasets which can be found in the following references [**?**] [**?**] [**?**]. The maximum number of iterations was set to 250,000. The learning rate was allowed to vary between 2.5 and .00122, dividing by 2 each iteration. Bag fractions of 1, 0.75, and 0.5 were evaluated. The maximum number of splits varied from 20 down to 4, in increments of 4. Each parameter combination was tested using both the constant and variable learning rate schemes, except for the learning rates of 2.5 and 1.25 which were evaluated using variable learning rates only.

I'm still in the process of evaluating these results. However my preliminary analysis has yielded the following realizations. First, the initially proposed learning rate scheme does not generalize well across datasets with varying numbers of examples. The minimum learning rate possible using equation 1 with a given max learning rate decreases as the number of examples increases. This of course is not ideal. To avoid this, I plan to introduce a minimum learning rate and change the adaptation rule to use a direct linear mapping from the range $[1, |S_m|]$ (possible number of examples in each leaf), to the range $[v_{min}, v_{max}]$ (possible learning rates). The new rule is shown in equation 2 below. In this way we will ensure that regardless of the number of examples in the dataset, a node with X% of the training examples in dataset A will have the same learning rate applied as a node with X% of the training examples in dataset B, making cross dataset comparisons of model performance more meaningful.

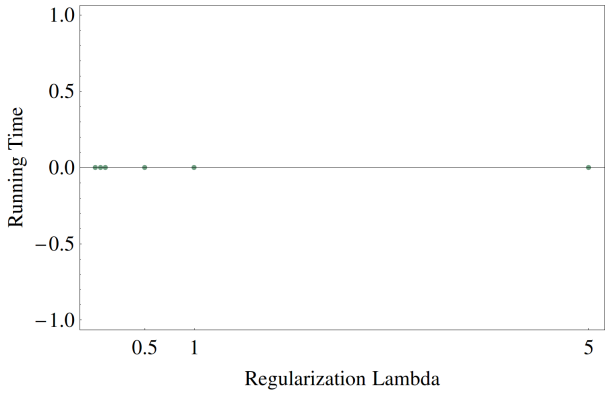
$$v_{m,j} = (|R_{m,j}| - 1)\frac{v_{max} - v_{min}}{|S_m| - 1} + v_{min} \tag{2}$$

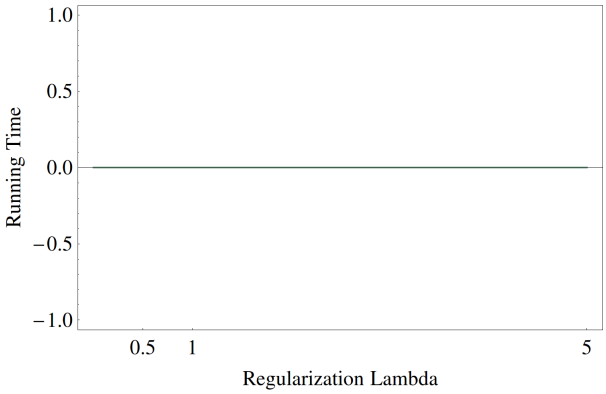
Furthermore, the data seems to show that the choice of bag fraction does not significantly impact the minimum error that can be achieved using combinations of the other parameters. Basically, by graphing a each combination of parameters on a Test set RMSE vs Bag Fraction graph, the results show that roughly the same minimum RMSE is achieved regardless of if the bag fraction is set to 0.5, 0.75, or 1.0. Thus for future tests it should be safe to fix the bag fraction at 0.5, take advantage of a slight training time speed up as a result and focus our analysis on the two parameters that do show definite correlation with the error, the learning rate and maximum number of splits.

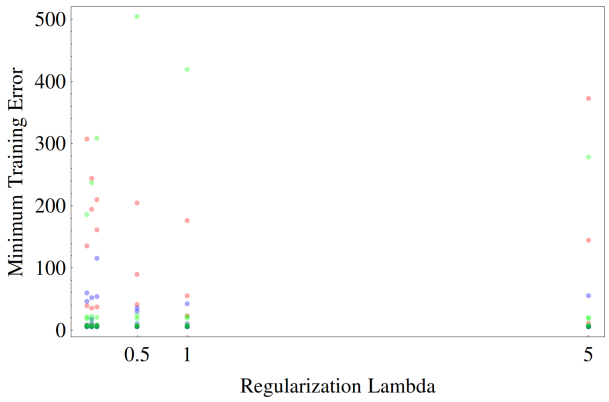
The plan now is to implement the revised learning rate adaptation described above. Fix the bag fraction at 0.5, continue to use 1 as the minimum examples in each leaf, and runs tests across a variety of combinations of max learning rate, min learning rate, and number of splits using the variable learning rate scheme. The goal being to come up rules of thumb as to what parameter combinations tend to work the best and which ones seem to be the worst. Once I have this data I'd like to examine the models built with these parameters

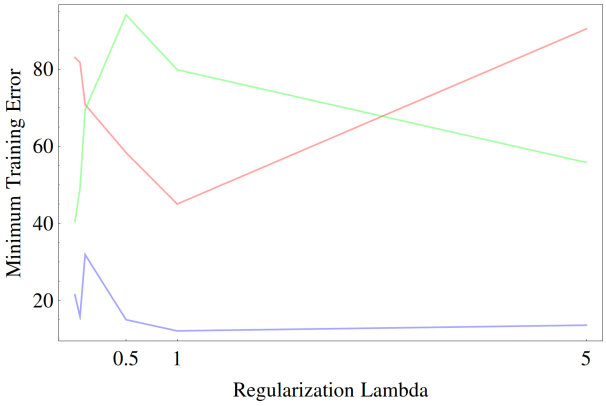
in more depth, and see if I can explain from a machine learning point of view why the model performs the way it does.

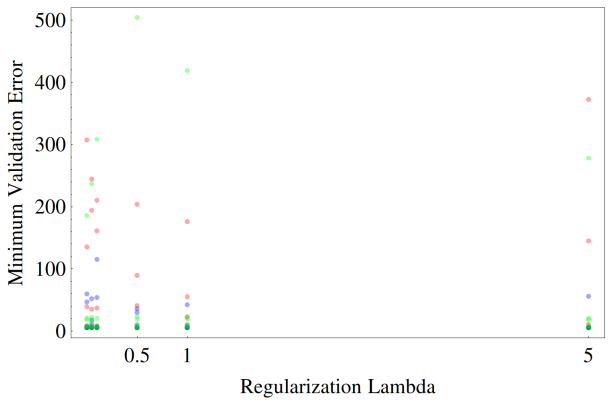
The good news is that the variable learning rate scheme, even as is without updating it to work more consistently across datasets, seems to be performing quite well. On the power plant and air foil datasets, the best combination of parameters using variable learning rates comes within tenths of the minimum RMSE, which is still achieved using a constant learning rate. However on the bike sharing dataset, which has significantly more noise and is a much more difficult regression problem than the other too, variable learning rates are a clear winner. Eighty sets of parameters using variable learning rates precede the first constant learning rate entry in the list of parameters sorted by their error on the test set. For future tests, the plan is to track down more noisy datasets as it seems this may be the error where variable learning rates will show their value. This performance increase makes sense because the regression trees should split the noisy examples out into their own leaf nodes, which will have very few examples in them and thus be heavily penalized by the variable learning rate scheme. The non noisy, more "average" looking examples on the other hand will find their way into large leaf nodes and will be able to contribute a relatively large amount to the function approximation as a whole.

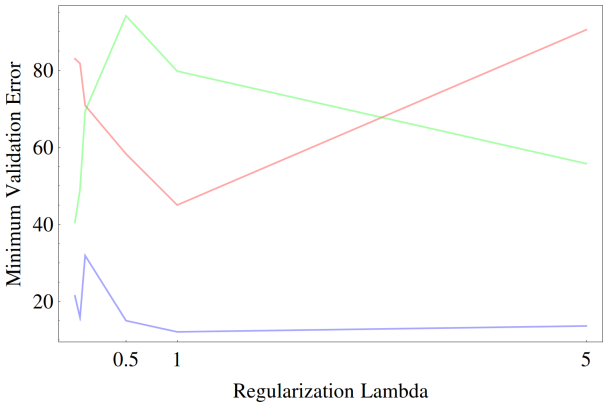

Running Time
1.0
0.5
0.0
−0.5
−1.0
0.5
1
5
Regularization Lambda

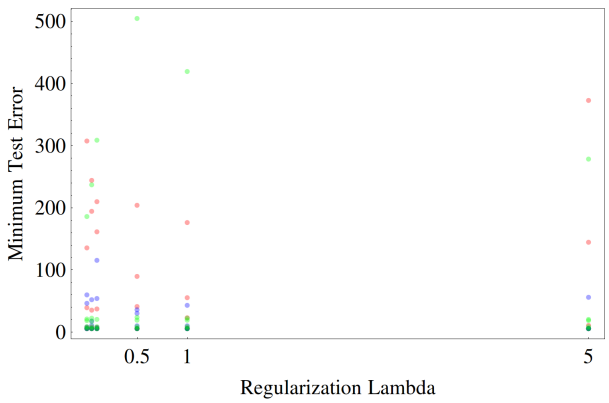

Running Time
1.0
0.5
0.0
−0.5
−1.0
0.5
1
5
Regularization Lambda

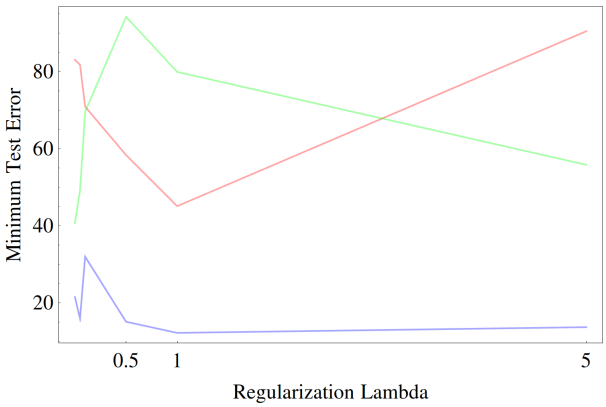

Minimum Training Error
500
400
300
200
100
0
0.5
1
5
Regularization Lambda

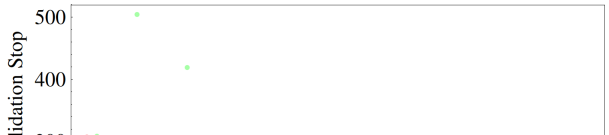

Minimum Training Error
80
60
40
20
0.5
1
5
Regularization Lambda

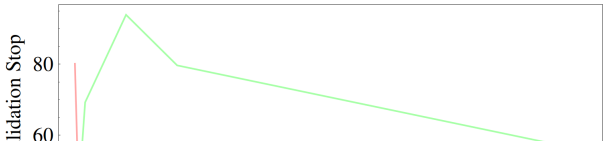

Minimum Validation Error
500
400
300
200
100
0
0.5
1
5
Regularization Lambda

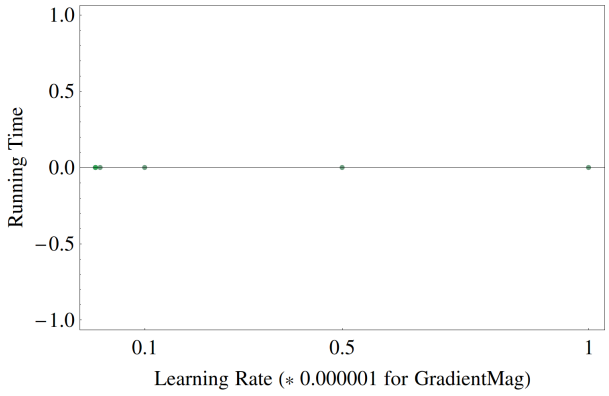

Minimum Validation Error
80
60
40
20
0.5
1
5
Regularization Lambda

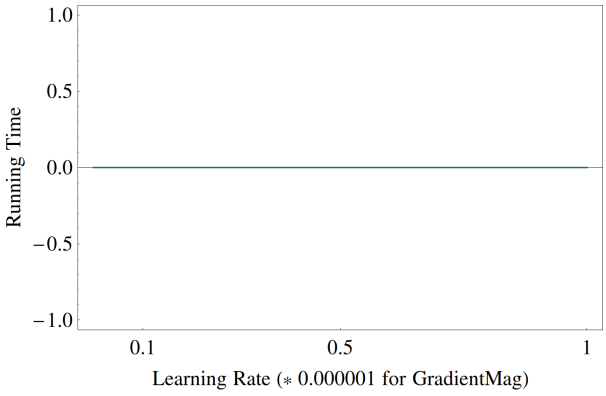

Minimum Test Error
500
400
300
200
100
0
0.5
1
5
Regularization Lambda

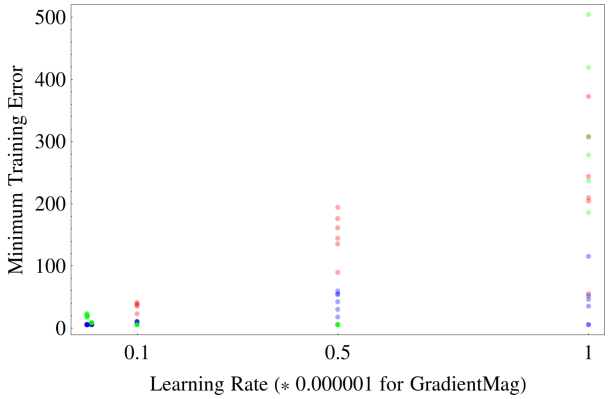

Minimum Test Error
80
60
40
20
0.5
1
5
Regularization Lambda

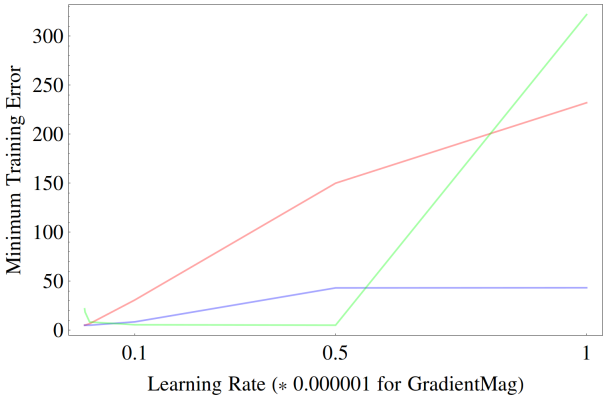

500
400

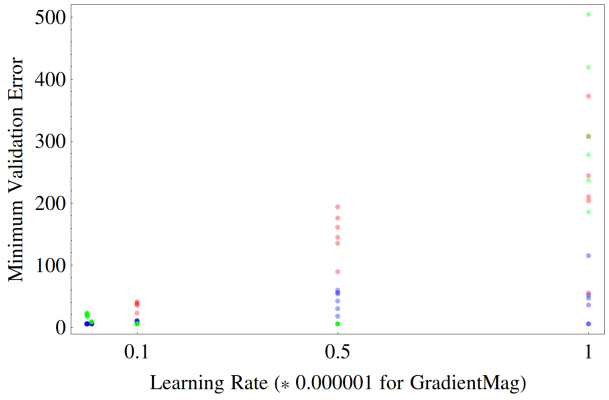

80
60

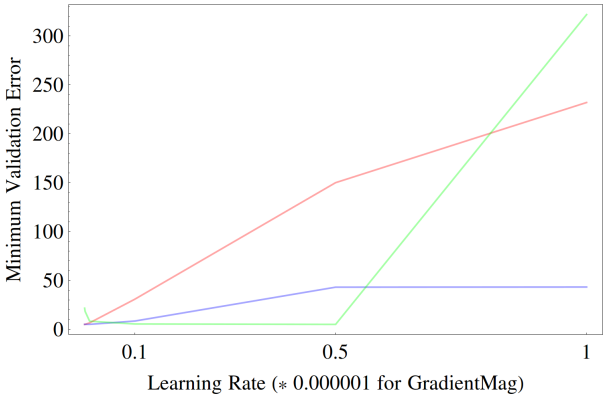

Running Time
1.0
0.5
0.0
−0.5
−1.0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)

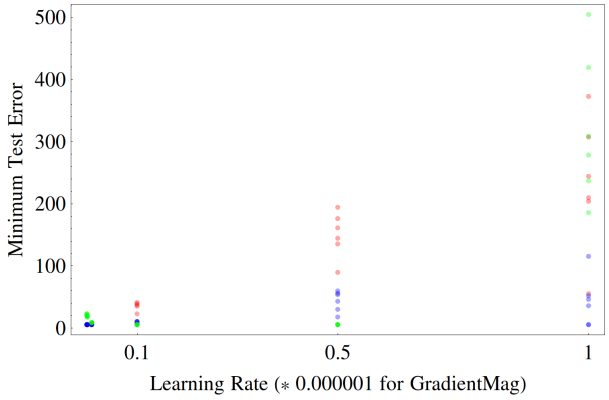

Running Time
1.0
0.5
0.0
−0.5
−1.0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)

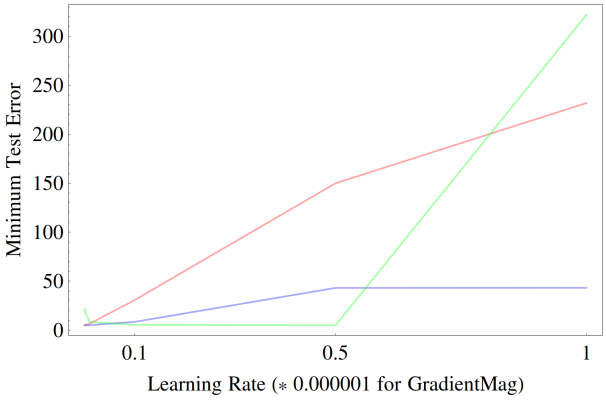

Minimum Training Error
500
400
300
200
100
0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)

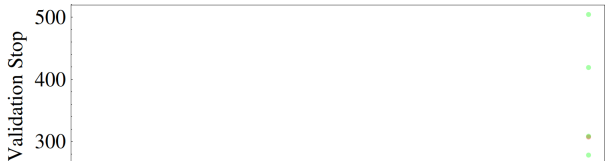

Minimum Training Error
300
250
200
150
100
50
0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)

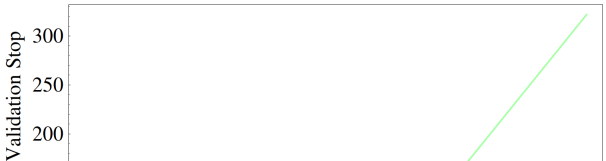

Minimum Validation Error
500
400
300
200
100
0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)


Minimum Validation Error
300
250
200
150
100
50
0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)


Minimum Test Error
500
400
300
200
100
0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)


Minimum Test Error
300
250
200
150
100
50
0
0.1
0.5
1
Learning Rate (∗ 0.000001 for GradientMag)


Validation Stop
500
400
300


Validation Stop
300
250
200