LR Reverse Engineering

In order to try and find a method by which I can determine a trained network's learning rate, I tried seeking for interesting patterns which I could utilize. But first, let us introduce the setup:

Setup

- Dataset:
  I looked for a dataset in which a network would be rather stable and not too sensitive to the choice of learning rate. CIFAR10 was slightly too hard to learn, and MNIST rather easy.
  Therefore, I created a "harder" version of MNIST, including random erasing, random noise etc. This version of MNIST is rather easily learned by a simple convolutional network, for a wide range of learning rates. However, it takes several epochs to do so. We'll later see why it's needed.
  Note that similar results to those below occur for CIFAR10 as well (even slightly better), but CIFAR is less representative, as many learning rates lead to failure to converge to a good solution.

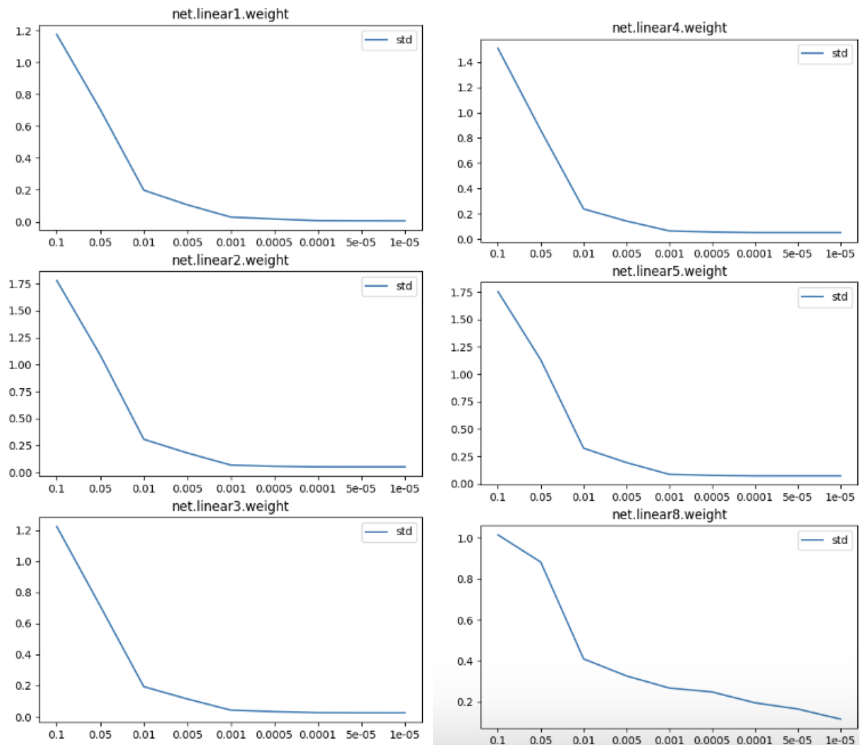
- Network:
  A rather standard CNN (two conv layers, multiple linear layers, ReLU activation, batch norm, etc).

- Optimizer:
  For easier convergence, the following results correspond to Adam optimizer, but I got similar results using SGD with momentum 0.9.

- Code can be found here.

Findings

- After some exploration, I found that when training the network using different learning rates, the weights of the network look rather different.
- Specifically, post training, there appears to be some positive correlation between the LR and the STD of each of the layers. Meaning high LR usually results in higher STD among the parameters of a layer.
  The graphs below show the standard deviation of each layer's parameters, as a function of the learning rate:



Those plots were produced using 9 trained networks, using the learning rates:
0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05.
Note the sharp changes in the STD are mostly notable in the higher learning rates.
As previously mentioned, all those networks got approximately the same accuracy, so are rather comparable.

- Another finding is that it does not depend much on whether the network finished converging or not. This behavior can be seen rather early in training.

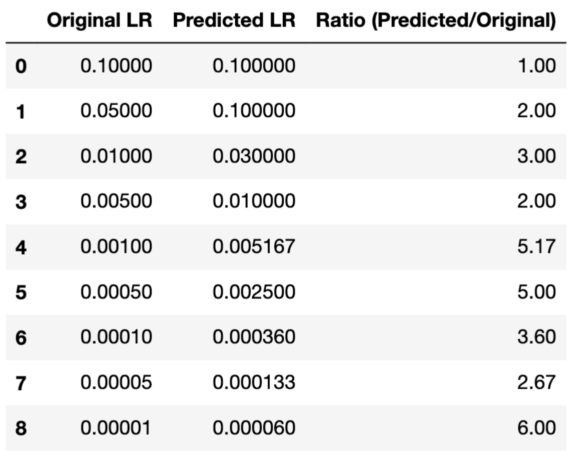
Reverse Engineering - Method

- From the above, we can consider the following method for reverse engineering the learning rate:
    a. Decide on a range of suspected learning rates.
       In our case, I chose the 9 previously mentioned LRs.
    b. For each LR:
        i. train a network for a short period of time (e.g., one epoch).
    c. For each layer k in the original network, using its STD, find the shortly-trained network with the closest STD in layer k.
       The LR of this network is a prediction for the original network's LR.
    d. We now have a prediction per layer. We can reduce the result to a single prediction (taking the mean, performing majority vote, etc).

  This is essentially a nearest neighbor/majority vote algorithm, based on the STDs of the network layers.

Reverse Engineering - Experiment

- For each LR, I trained a different network for 10 epochs (~90% train acc).
  Then, I tried predicting the LR for each of those networks:
- As discussed, for each of those networks, I shortly-trained several networks (1-epoch each), and using their STDs, predicted the learning rates.
- The results are as follows:



	Original LR	Predicted LR	Ratio (Predicted/Original)
0	0.10000	0.100000	1.00
1	0.05000	0.100000	2.00
2	0.01000	0.030000	3.00
3	0.00500	0.010000	2.00
4	0.00100	0.005167	5.17
5	0.00050	0.002500	5.00
6	0.00010	0.000360	3.60
7	0.00005	0.000133	2.67
8	0.00001	0.000060	6.00

- As expected from the graphs, the prediction became slightly less accurate from 0.001 onwards. However, it’s important to note that we usually choose learning rates by a factor of 10, so the predictions are pretty decent overall.

Thoughts and Future Steps:

- I currently don't have a theoretical explanation to why there's correlation between the variance and the learning rate. It might have to do with each descent step shooting the parameters far from one another when the LR is high. However, this is still a very lacking explanation.
- Note that the ratio between the prediction and the real LR is always greater than one (meaning the STD roughly decreases throughout epochs). This is an interesting phenomenon and can be used to further improve the prediction (for example, perform a weighted mean, favoring lower learning rates).
- The experiment was conducted only on two, simple, network architectures - on the described convolutional network, and on another simple FC network.
  Therefore, the experiment might be slightly biased. Further experimentation is needed.
- Both an advantage and disadvantage of this method, is the trade-off between the level of accuracy and the run-time. Using densely spaced LRs (i.e., more networks) can improve the prediction accuracy, but consume more time.