

MACHINE LEARNING GLASSES RESULTS LOG

KIRK SWANSON

1. 11/26/2017

- (1) Merged recent updates to GitHub
- (2) Downloaded images from a while ago. This first one is batch size 80, learning rate 0.001, beta 0.01, dropout 0.5, including the recent change that each batch is chosen from a newly randomly chuffed metadata:

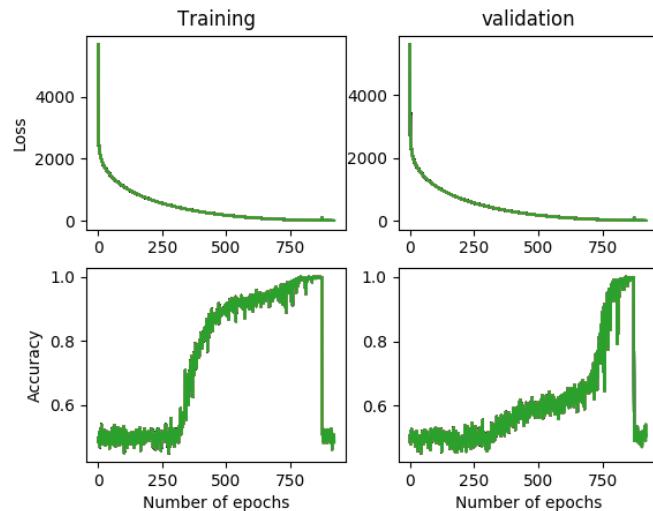


FIGURE 1. Batch Size = 80, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

- (3) This run had an optimal validation accuracy of 99.7635 and an optimal test accuracy of 99.7297. I also tested this multiple times, running the SAME set of hyperparameters, to see what is going on with that weird spike. Does it keep showing up? What is going on? Here are more examples, all run at the same hyperparameters as above:

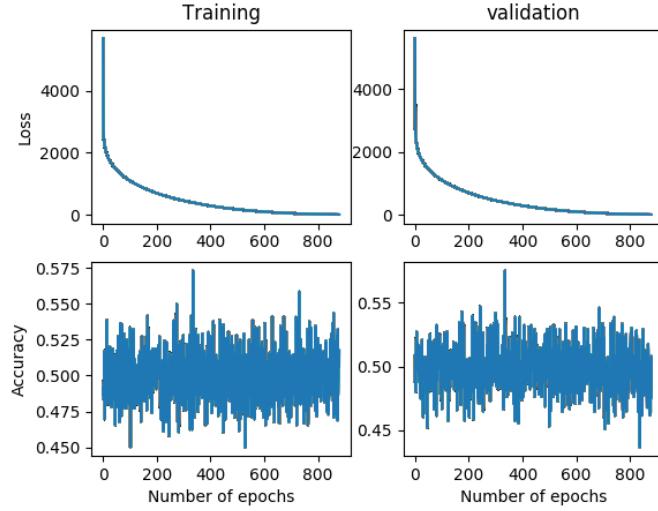


FIGURE 2. Batch Size = 80, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

- (4) This run had an optimal validation accuracy of 47.0608, optimal test accuracy of 47.6689, and final validation accuracy of 49.223, test accuracy of 50.0338

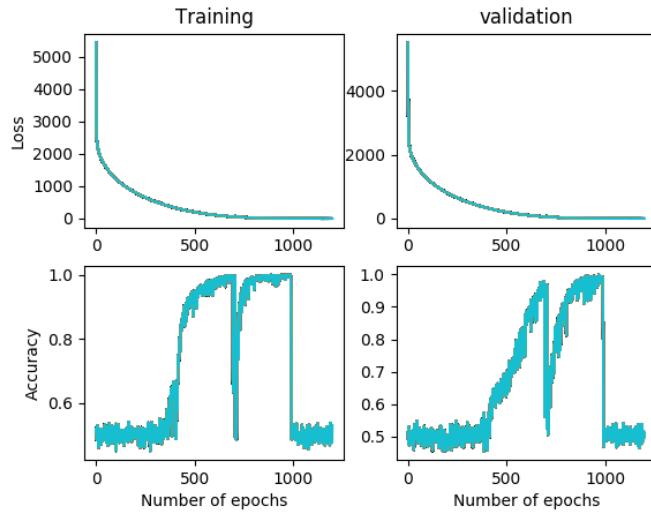


FIGURE 3. Batch Size = 80, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

- (5) This run had an optimal validation accuracy of 99.4595, optimal test accuracy of 99.3243, and final validation accuracy of 50.777, test accuracy of 49.9662.

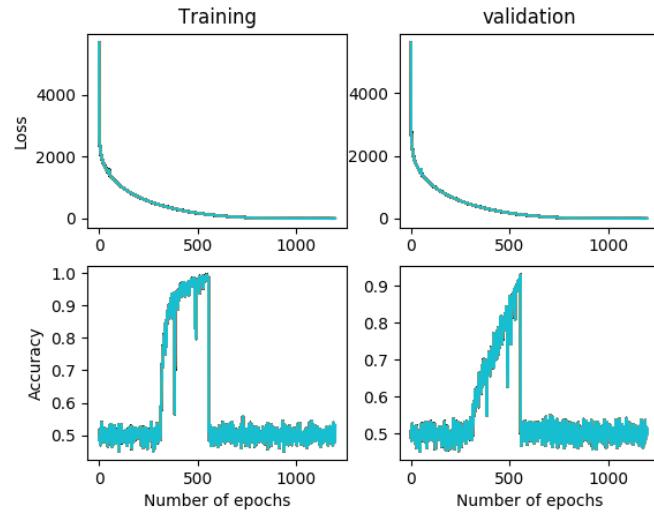


FIGURE 4. Batch Size = 80, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

- (6) This run had an optimal validation accuracy of 992.5, optimal test accuracy of 91.7568, and final validation accuracy of 49.223, test accuracy of 50.0338.

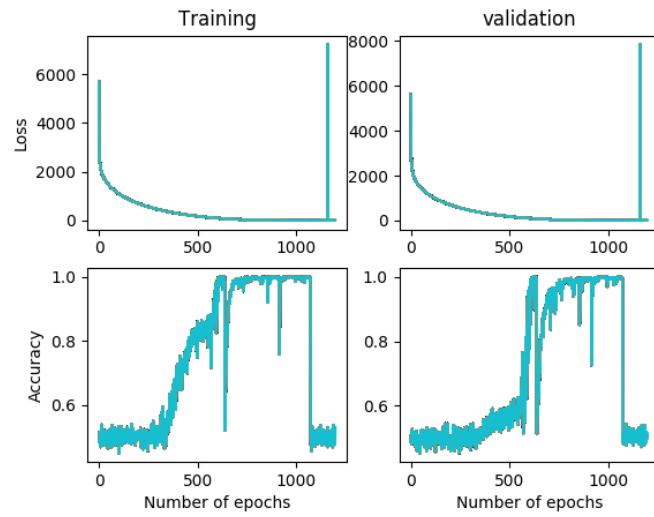


FIGURE 5. Batch Size = 80, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

- (7) This run had an optimal validation accuracy of 99.8987, optimal test accuracy of 99.8649, and final validation accuracy of 50.777, test accuracy of 49.9662.

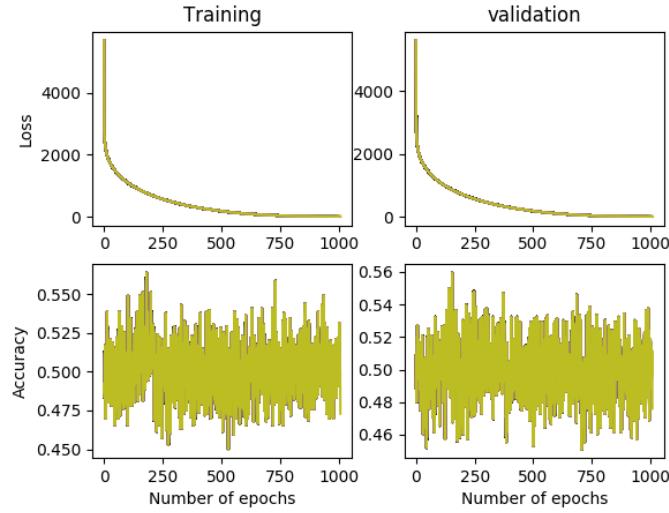


FIGURE 6. Batch Size = 80, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

- (8) This run had an optimal validation accuracy of 55.4054, optimal test accuracy of 54.223, and final validation accuracy of 49,223, test accuracy of 50.0338.
- (9) The following are images of runs on the dataset that consists of glasses and liquids that are each 12 time steps away from the assumed glass transition temperature of 0.21.

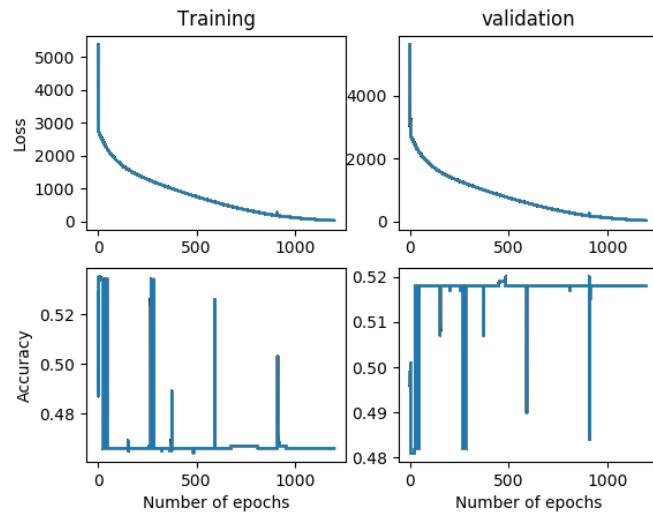


FIGURE 7. Batch Size = 100, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

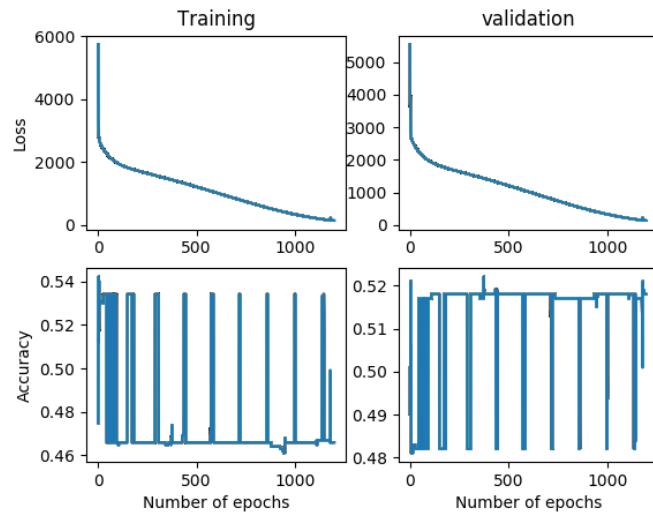


FIGURE 8. Batch Size = 10, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

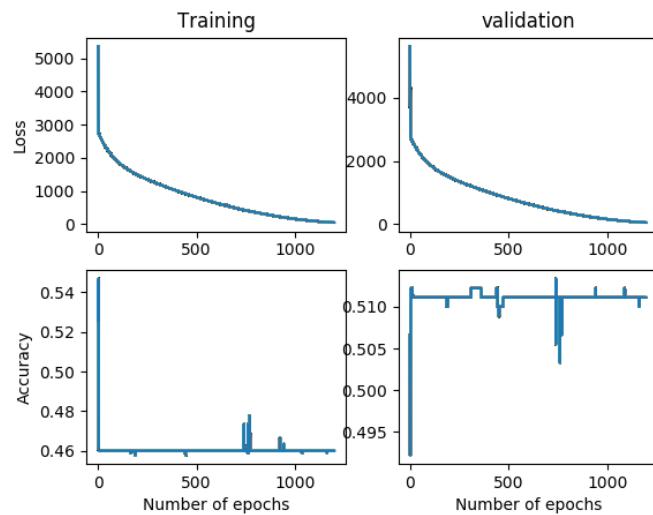


FIGURE 9. Batch Size = 150, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

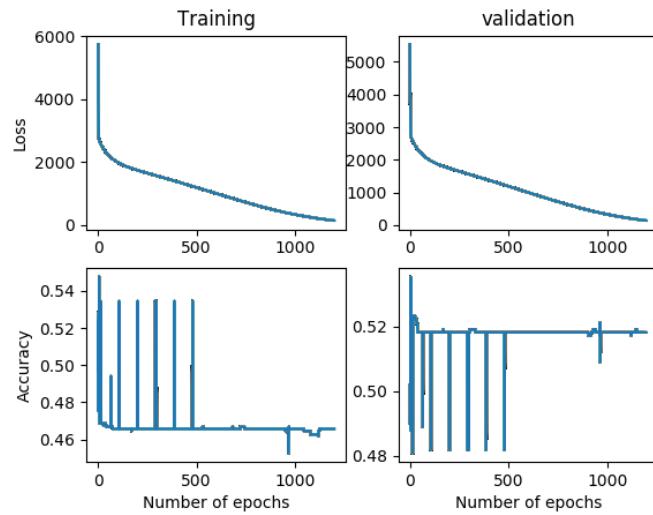


FIGURE 10. Batch Size = 15, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

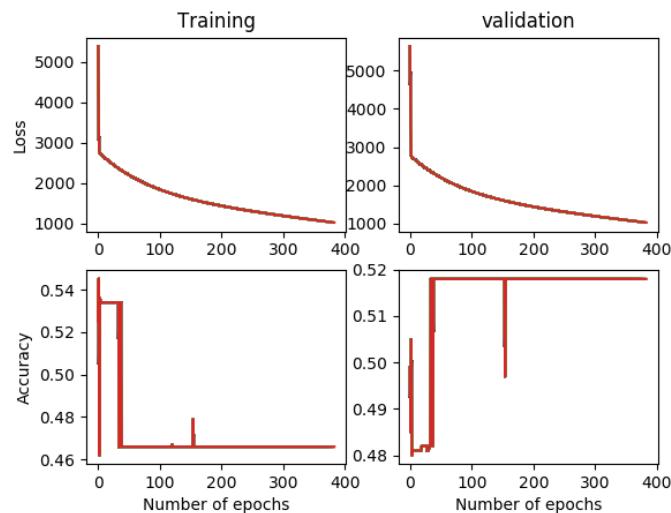


FIGURE 11. Batch Size = 200, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

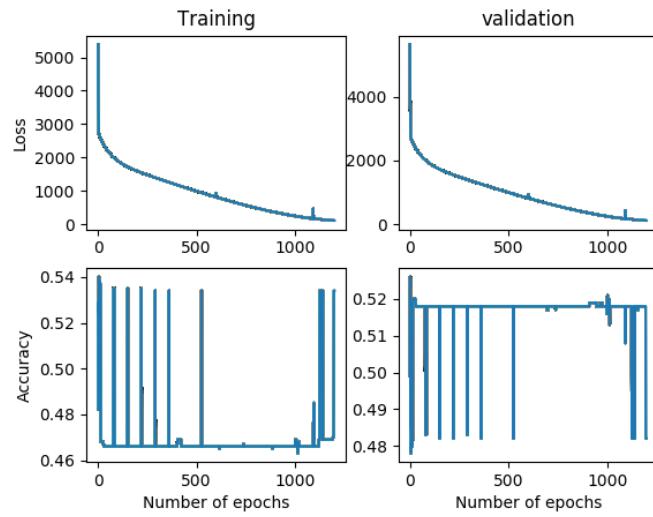


FIGURE 12. Batch Size = 20, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

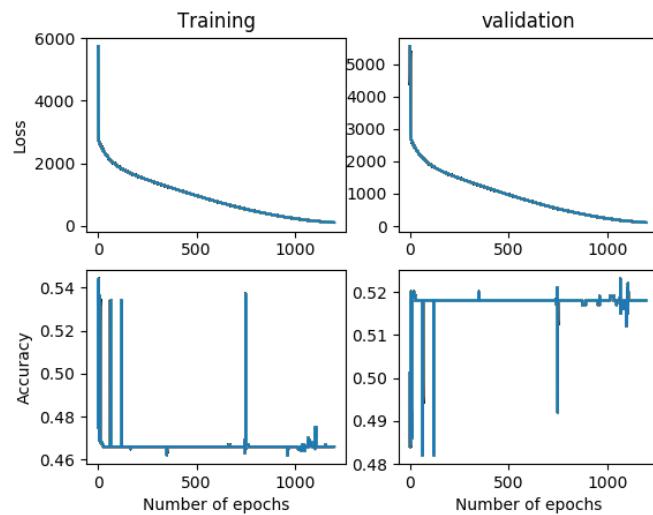


FIGURE 13. Batch Size = 25, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

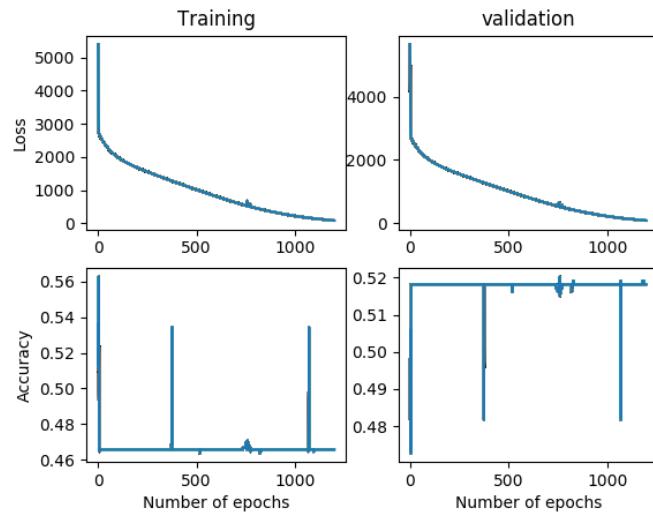


FIGURE 14. Batch Size = 30, Learning Rate = $1e-4$, Beta = 0.01, Dropout = 0.5

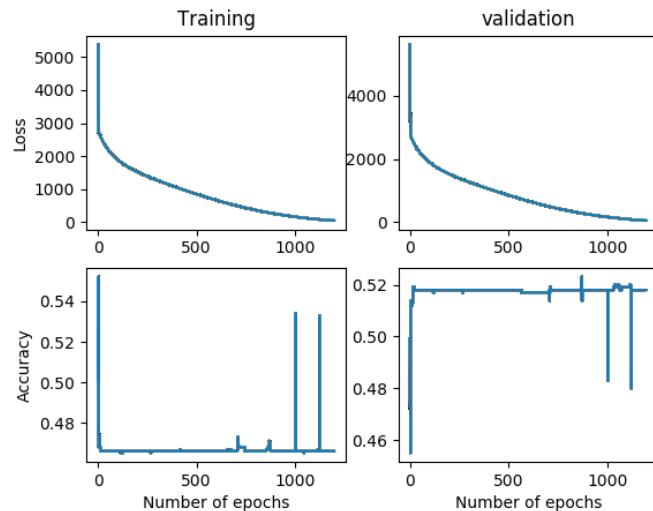


FIGURE 15. Batch Size = 50, Learning Rate = $1e-4$, Beta = 0.01, Dropout = 0.5

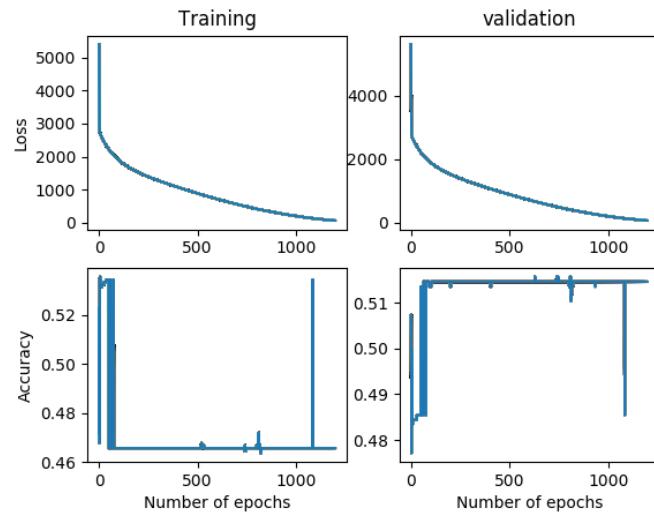


FIGURE 16. Batch Size = 80, Learning Rate = 1e-4, Beta = 0.01, Dropout = 0.5

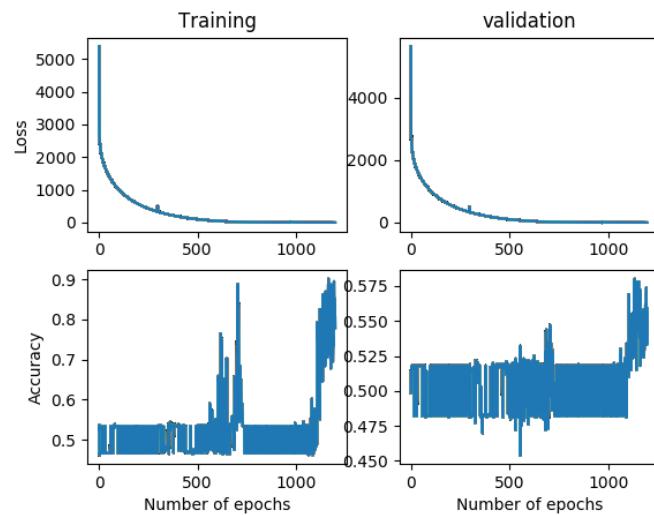


FIGURE 17. Batch Size = 100, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

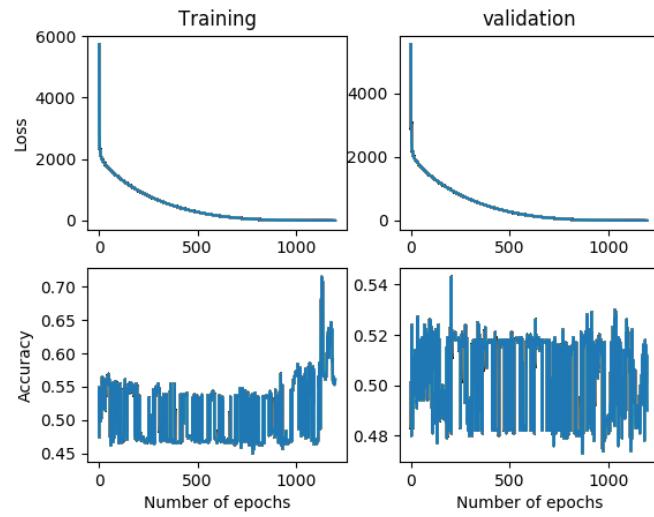


FIGURE 18. Batch Size = 10, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

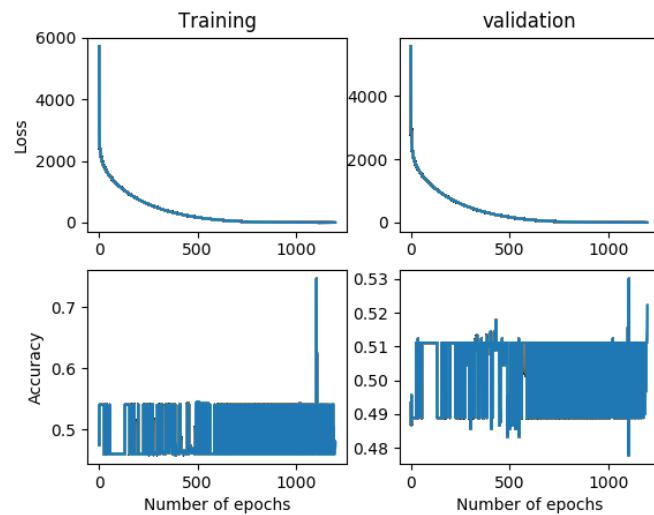


FIGURE 19. Batch Size = 150, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

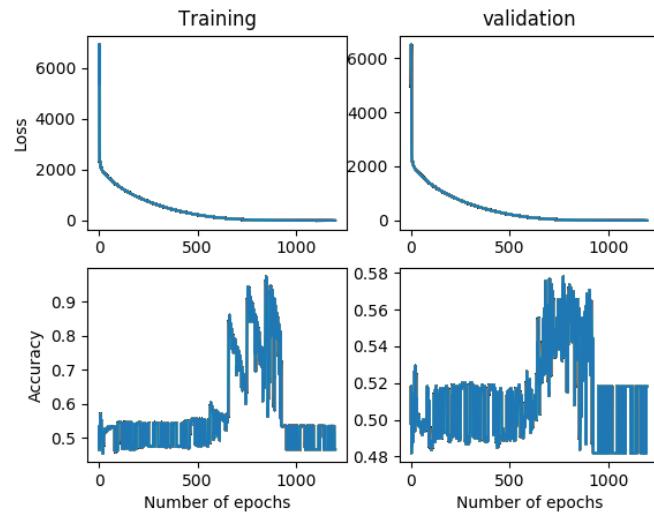


FIGURE 20. Batch Size = 15, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

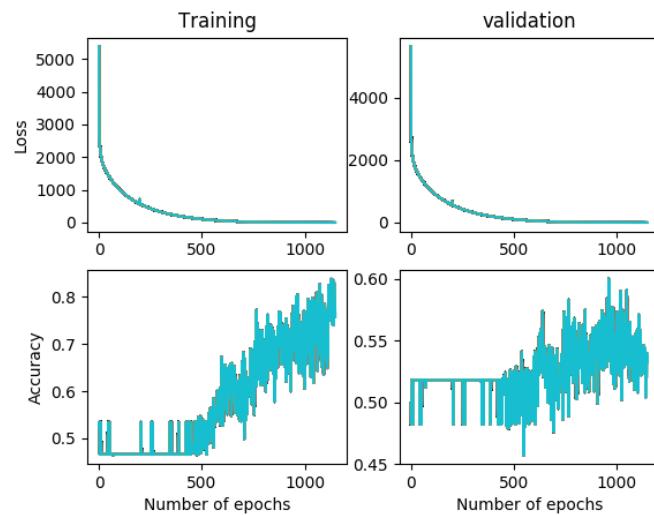


FIGURE 21. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.1

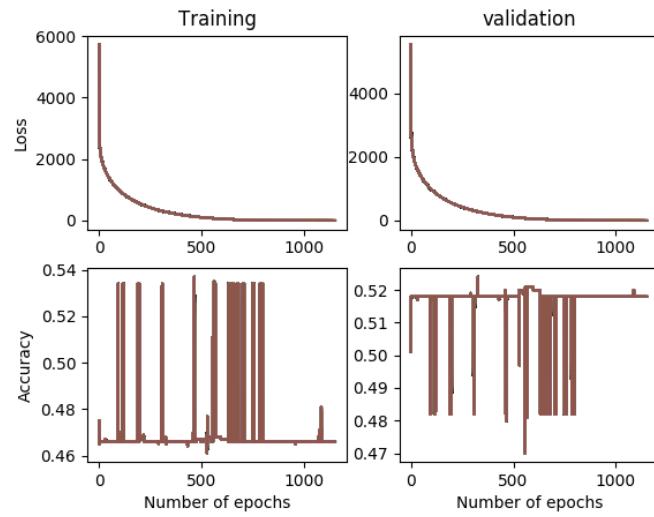


FIGURE 22. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.2

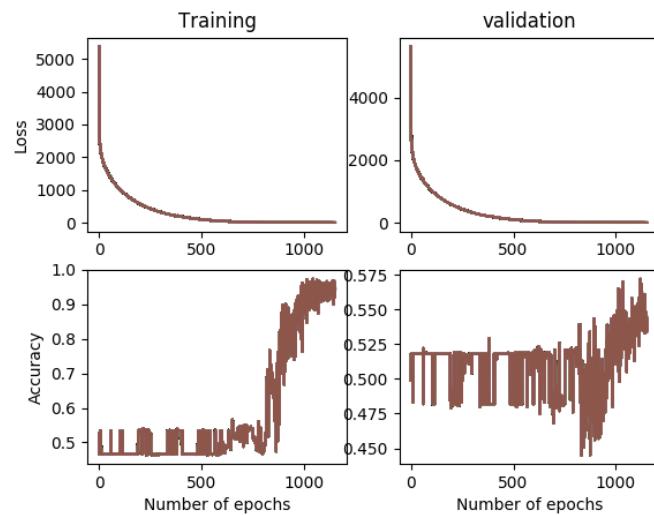


FIGURE 23. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.3

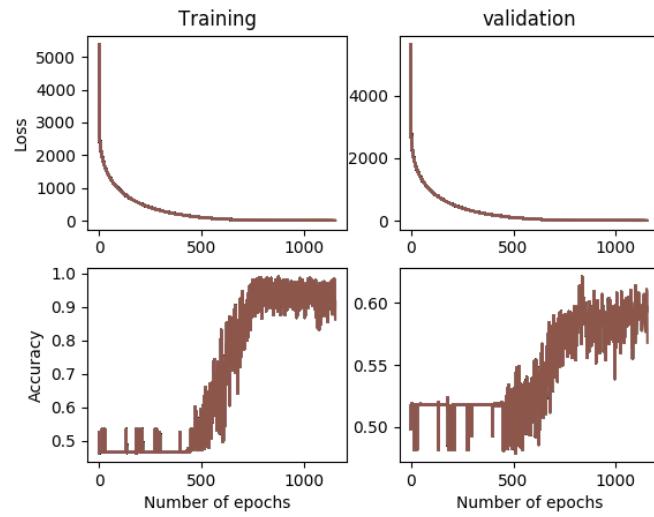


FIGURE 24. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.4

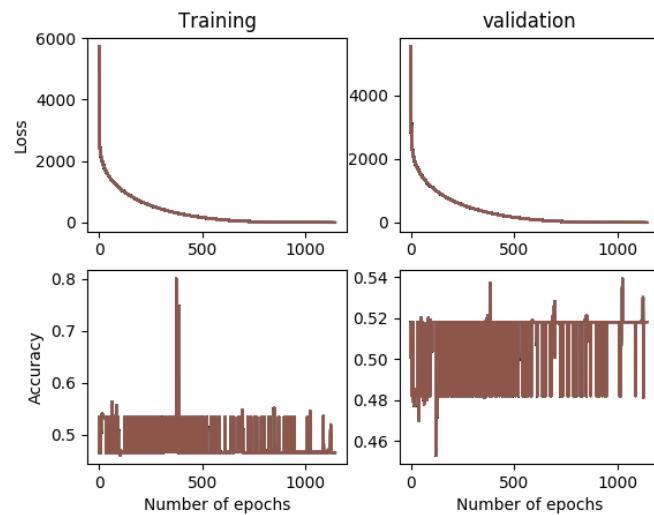


FIGURE 25. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

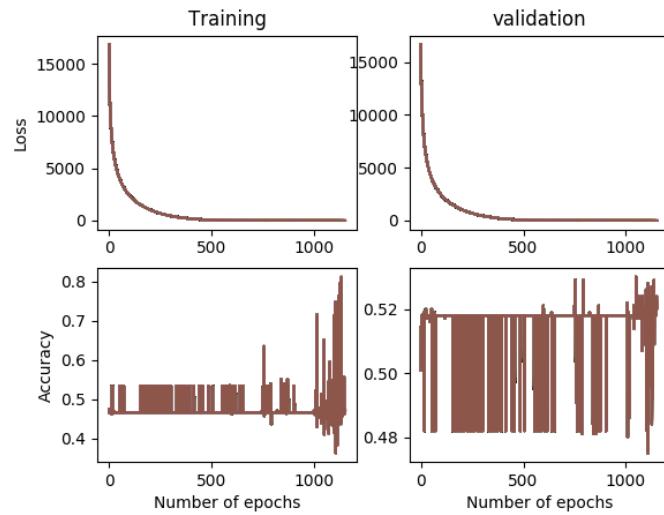


FIGURE 26. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.3

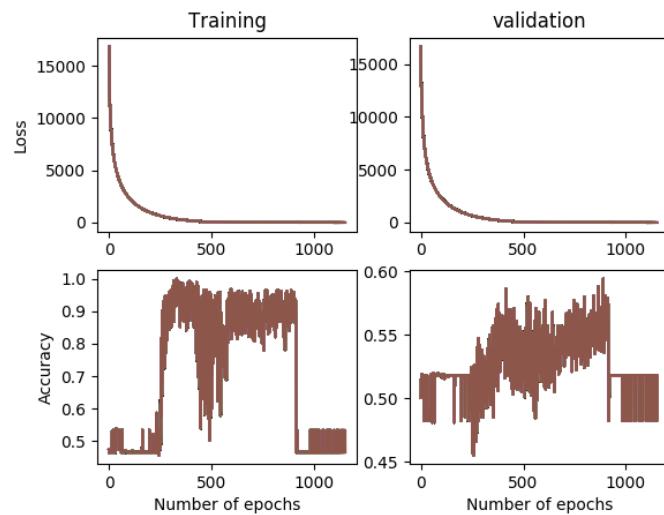


FIGURE 27. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.4

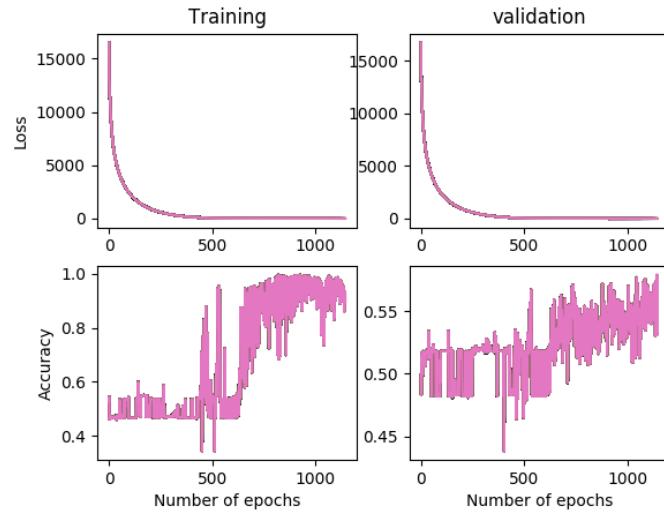


FIGURE 28. Batch Size = 200, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.4

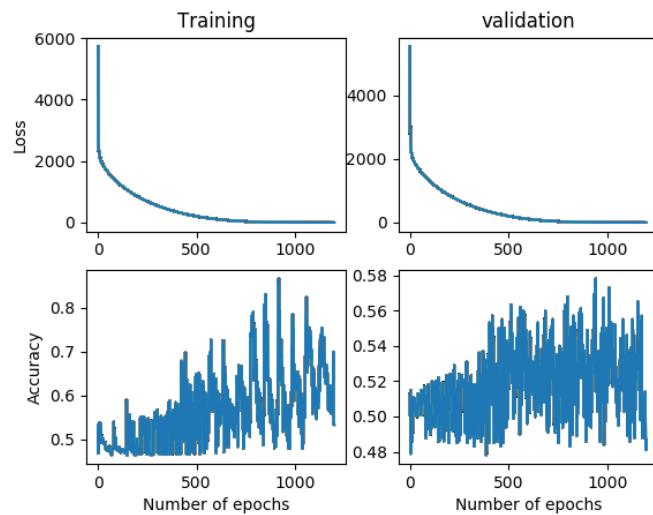


FIGURE 29. Batch Size = 20, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

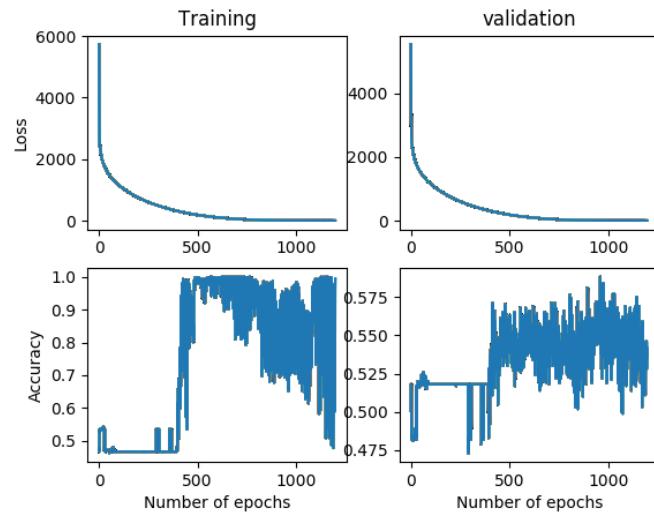


FIGURE 30. Batch Size = 250, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

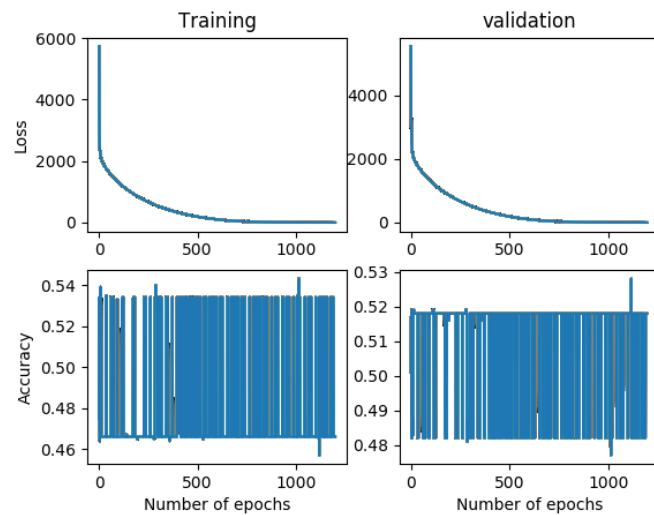


FIGURE 31. Batch Size = 25, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

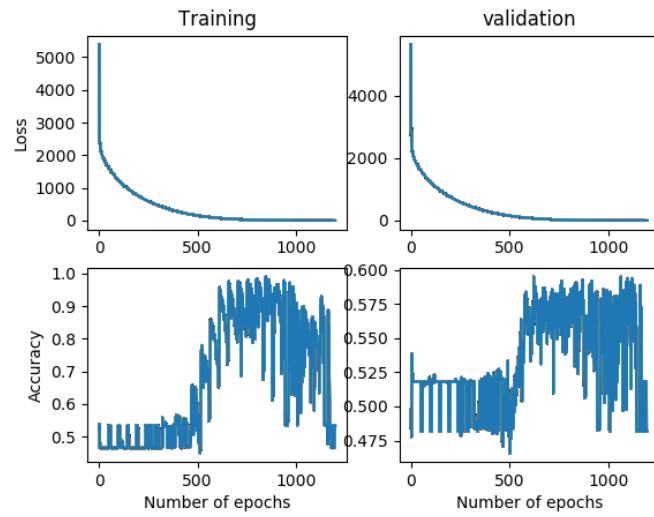


FIGURE 32. Batch Size = 30, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

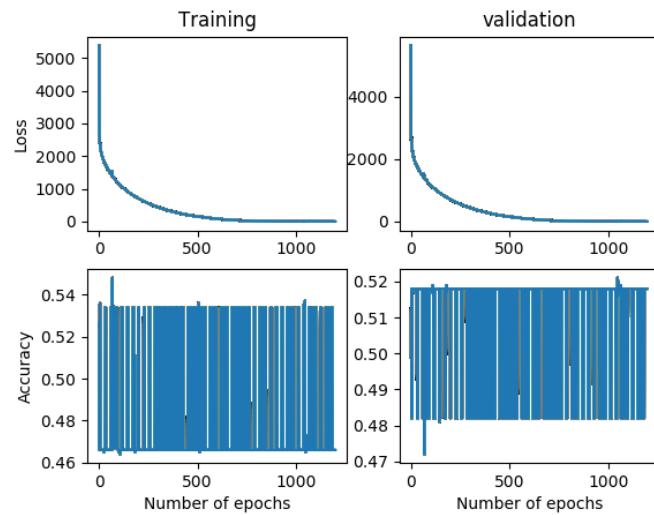


FIGURE 33. Batch Size = 50, Learning Rate = 1e-3, Beta = 0.01, Dropout = 0.5

(10) Visualizations.

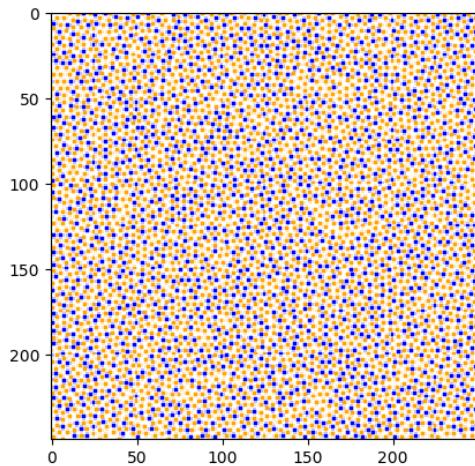


FIGURE 34. Glass original

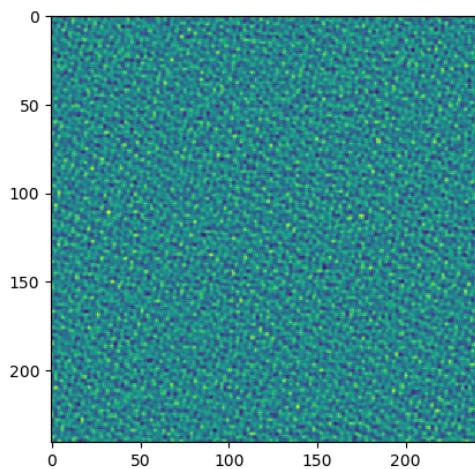


FIGURE 35. Glass channel 1

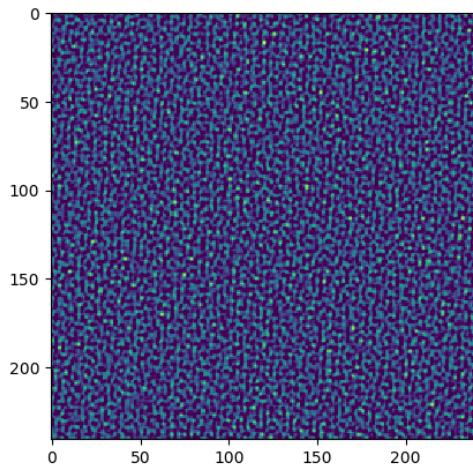


FIGURE 36. Glass channel 2

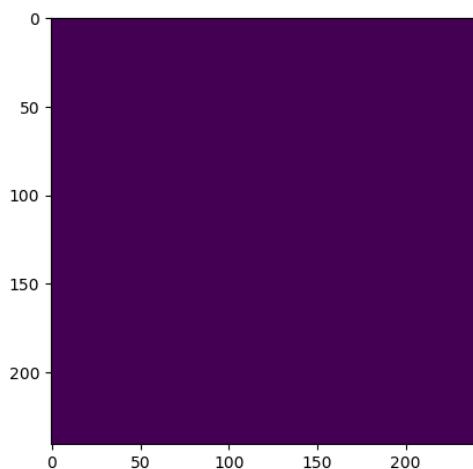


FIGURE 37. Glass channel 3

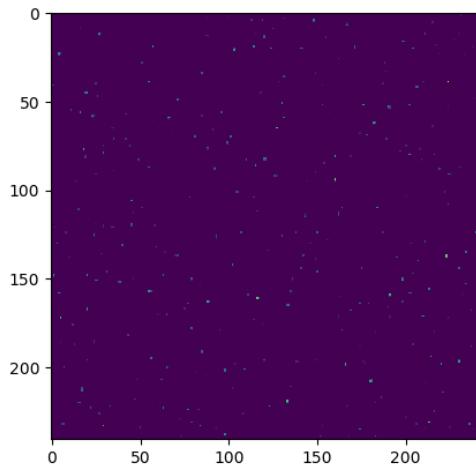


FIGURE 38. Glass channel 4

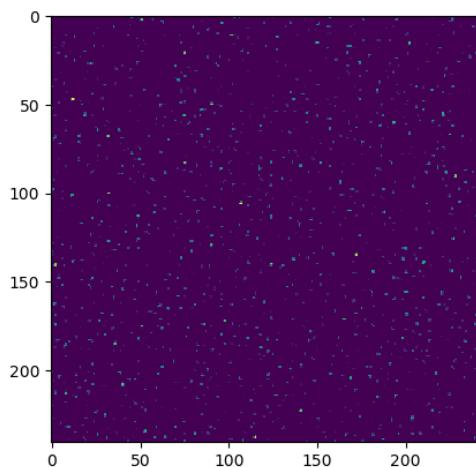


FIGURE 39. Glass channel 5

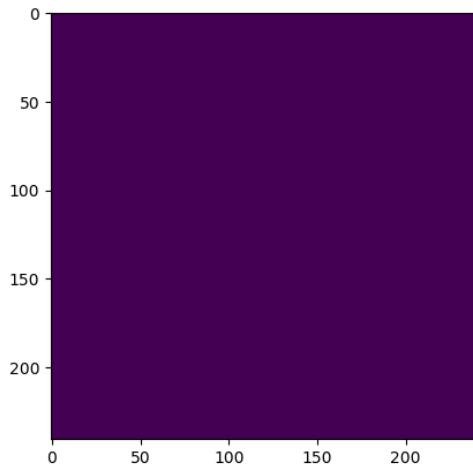


FIGURE 40. Glass channel 6

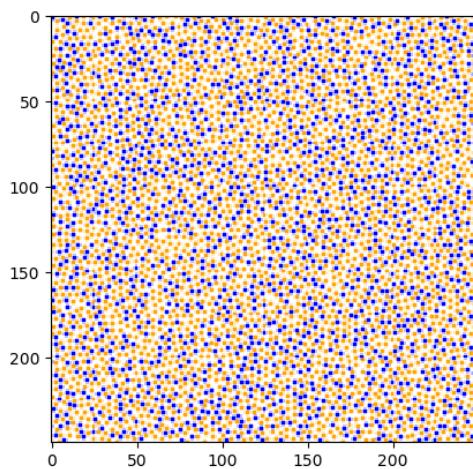


FIGURE 41. Liquid original

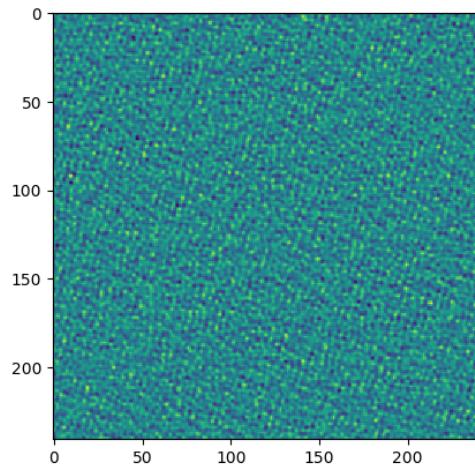


FIGURE 42. Liquid channel 1

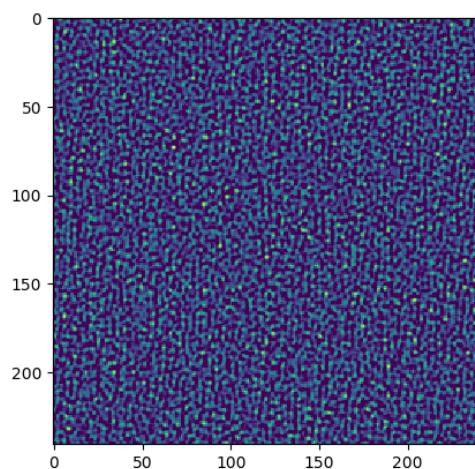


FIGURE 43. Liquid channel 2

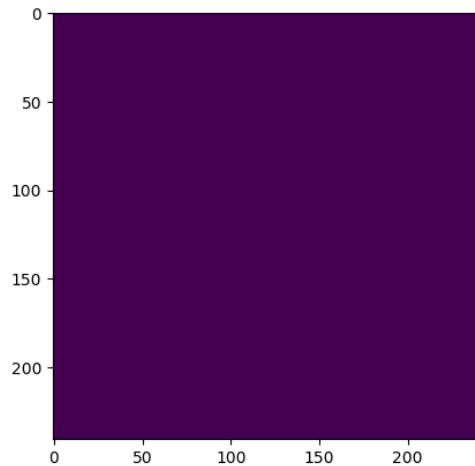


FIGURE 44. Liquid channel 3

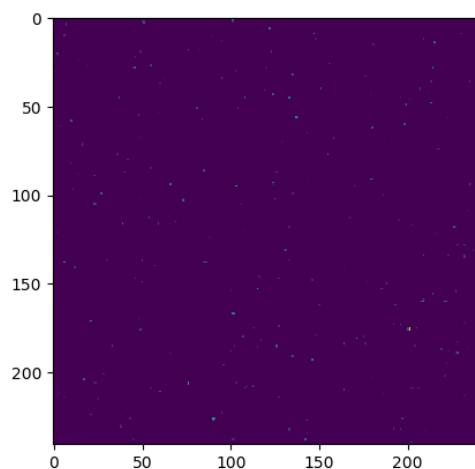


FIGURE 45. Liquid channel 4

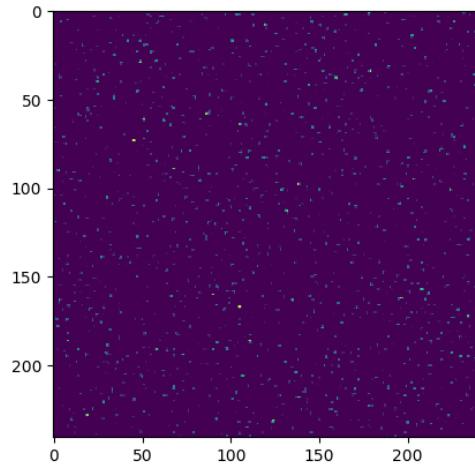


FIGURE 46. Liquid channel 5

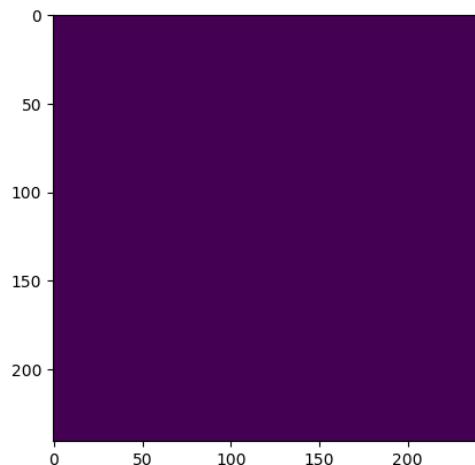


FIGURE 47. Liquid channel 6

2. 12/13/2017

- (1) Updated python scripts and merged with GitHub - the recent additions include streamlining the learning rate schedule and data augmentation options. The following is for a standard run on the original endpoint data:

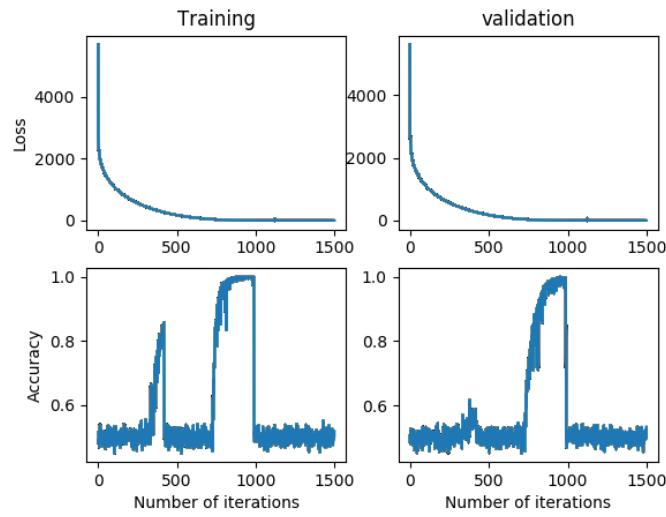


FIGURE 48. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-3, Schedule Length = 550 iterations, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

(2) I then, twice, set the final learning rate to be smaller, at 1e-4:

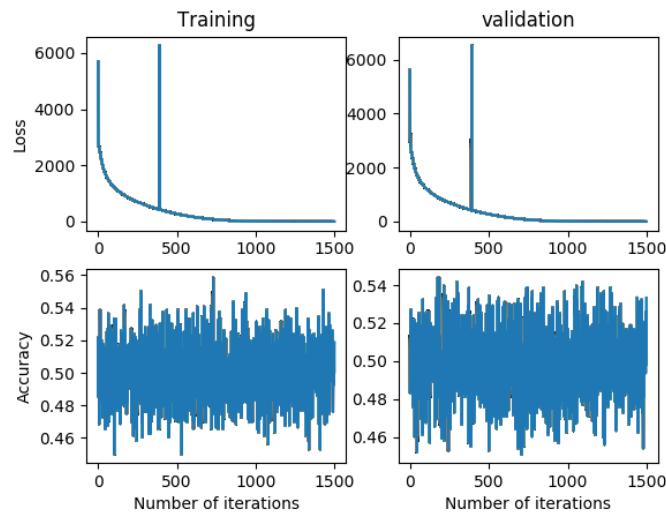


FIGURE 49. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = 550 iterations, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

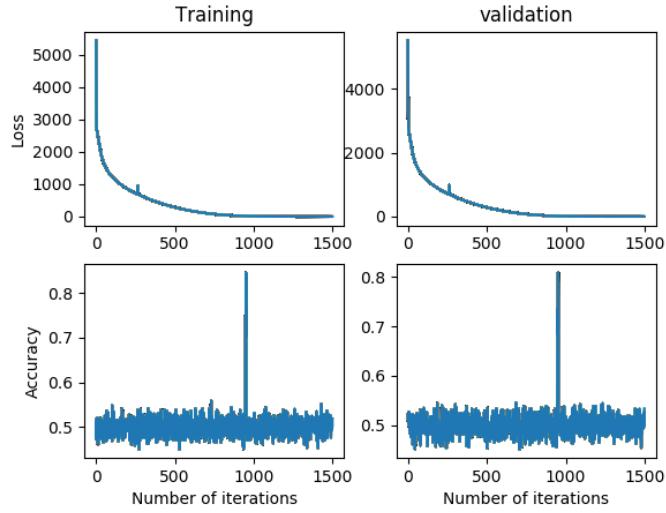


FIGURE 50. Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-3$, Schedule Length = 550 iterations, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

- (3) What is going on here? It seems as though setting the learning rate a magnitude lower really screws things up...
- (4) Note: one iteration goes through 10 batches, so if the batch size is 80, then one iteration goes through 800 examples. The training set is 70 percent of the total 20,000 images, so the training set is 14,000 images. Thus, one epoch is 17.5 iterations, or approximately 18 iterations. That means that 1500 iterations is about 83 epochs. Question: how many epochs is typically needed for a binary classification problem? And how long does this typically take?
- (5) I also tested the usual hyperparameters on the original endpoint data and turned on the data augmentation, to see if there would be any issues. I got the following training runs:

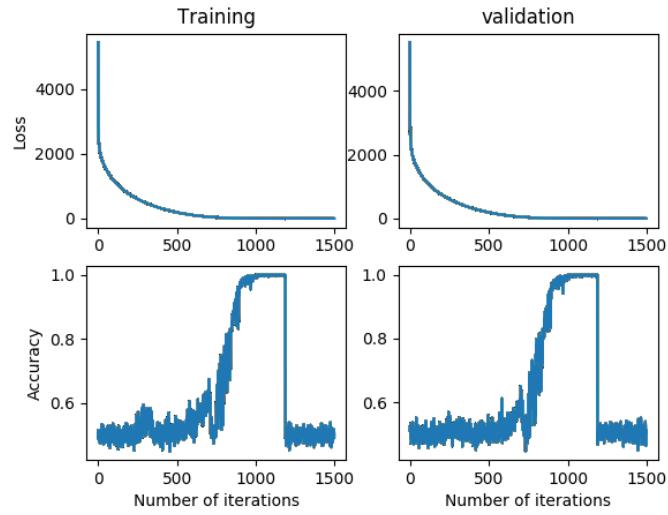


FIGURE 51. Initial Learning Rate = NA, Final Learning Rate = 1e-3, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

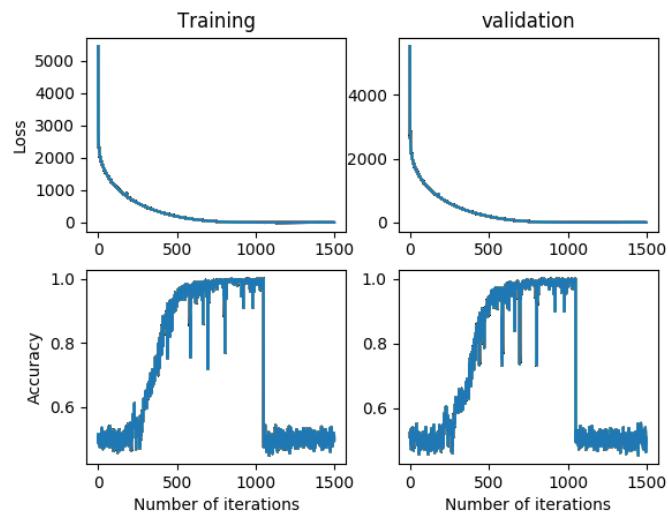


FIGURE 52. Initial Learning Rate = NA, Final Learning Rate = 1e-3, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

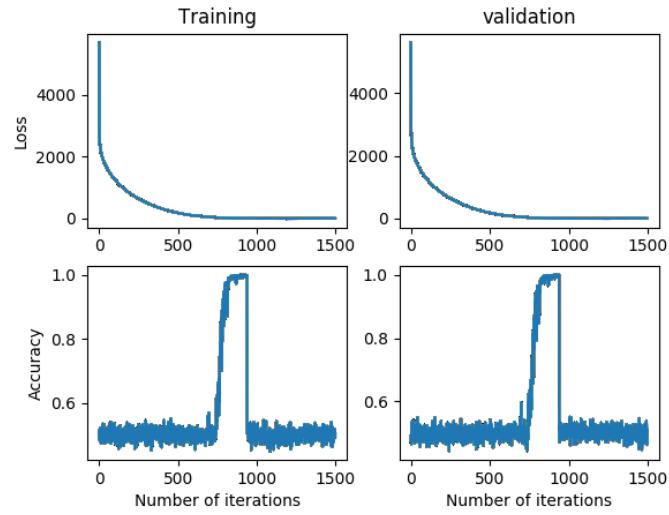


FIGURE 53. Initial Learning Rate = NA, Final Learning Rate = 1e-3, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

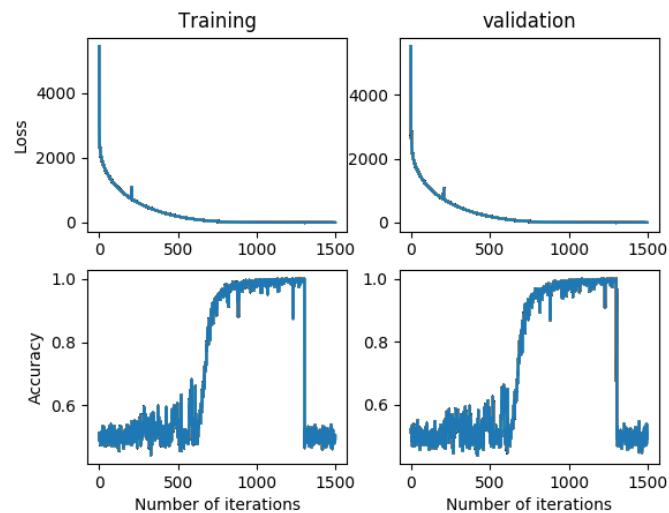


FIGURE 54. Initial Learning Rate = NA, Final Learning Rate = 1e-3, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

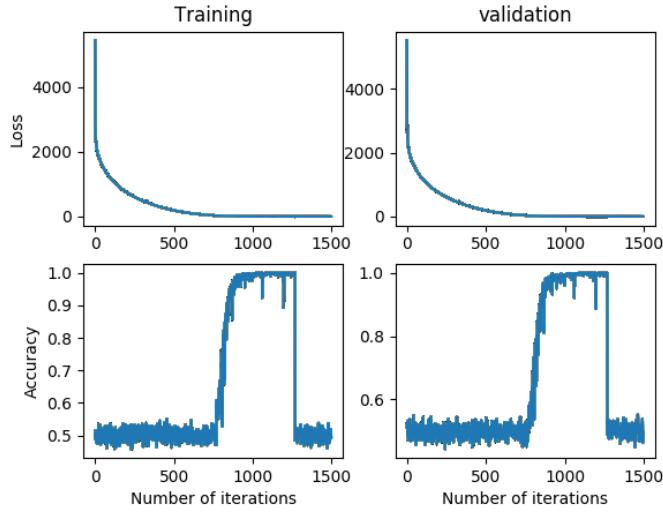


FIGURE 55. Initial Learning Rate = NA, Final Learning Rate = 1e-3, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

(6) I should double check that the dataset augmentation is actually working.

3. 12/22/2017

(1) I am now going to run a series of training runs that will explore the relationship between the learning rate schedule and accuracy. First, let's try the threshold idea. Then, we will try the standard exponential decay idea. Let's try the following. When the accuracy gets to 90 percent, let's decrease the learning rate by an order of magnitude. Let's just run that simple test, say, 8 times, and see what happens!

(2) Initial learning rate = , Final learning rate = ,

4. 12/24/2017

(1) We ran a series of training runs that used a learning rate threshold. The learning rate starts at the initial learning rate. The learning rate changes to the final learning rate whenever the validation accuracy is above 90 percent. This allows the learning rate to go back to the initial learning rate if the accuracy drops below 90 percent. Here are plots showing six tests. These plots demonstrate that the sharp drops in accuracy were, indeed, the result of a learning rate that was too large towards the end of training, since there are no sharp drops in accuracy here:

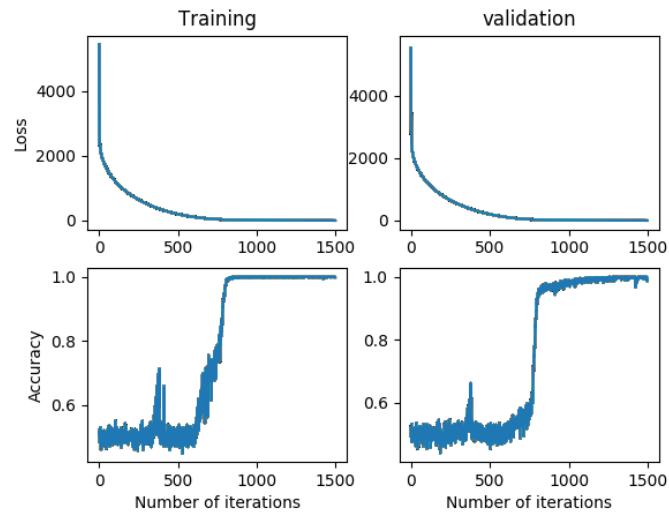


FIGURE 56. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

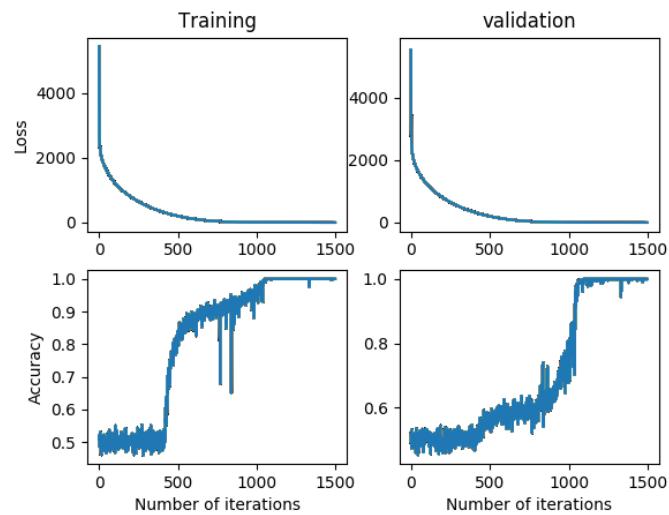


FIGURE 57. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

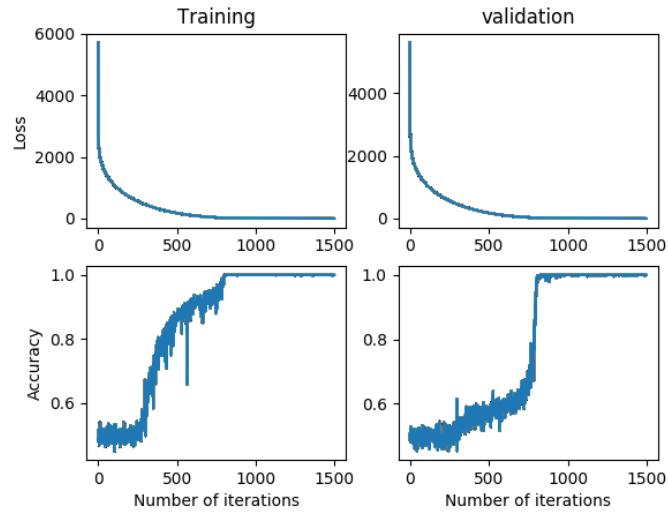


FIGURE 58. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

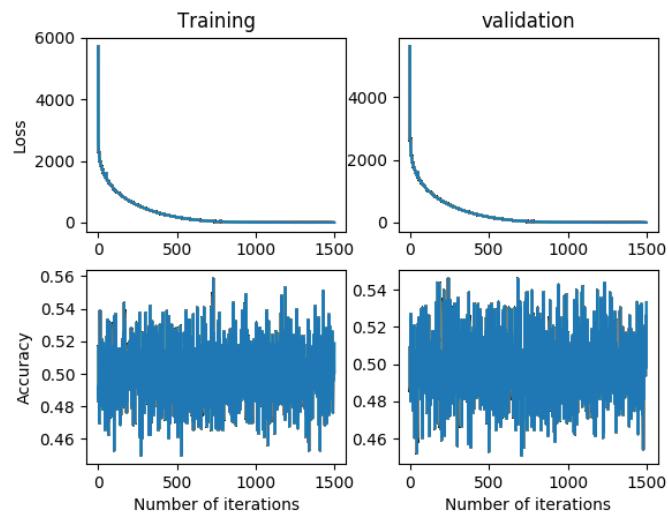


FIGURE 59. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

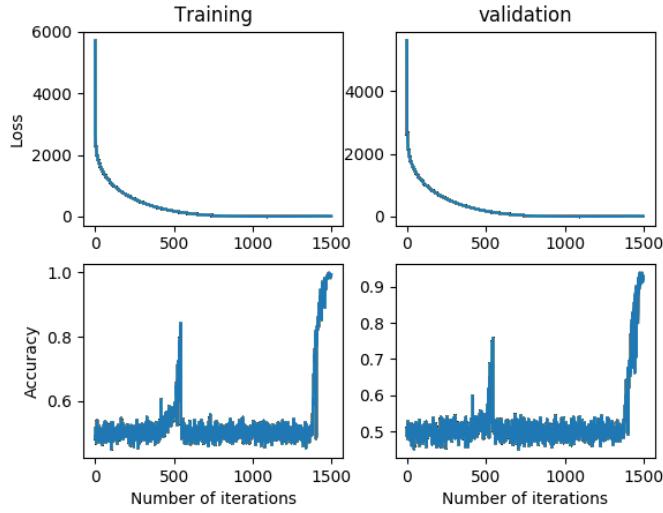


FIGURE 60. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

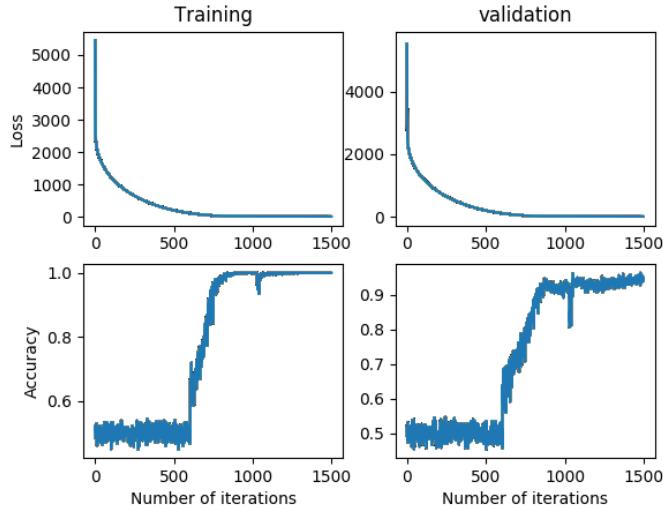


FIGURE 61. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

4.1. 12/29/2017.

- (1) Ran another series of runs that used a similar learning rate threshold just to check that my GitHub had been updated appropriately. This time used only 1e-3 to 1e-4. In one example, the accuracy crashed. Perhaps evidences that smaller learning rates are needed closer to the minimum. For the sake of completeness, these extra runs are reported below:

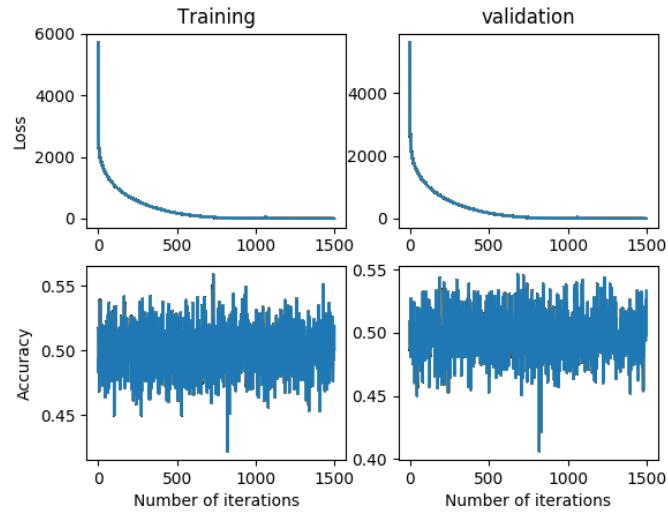


FIGURE 62. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

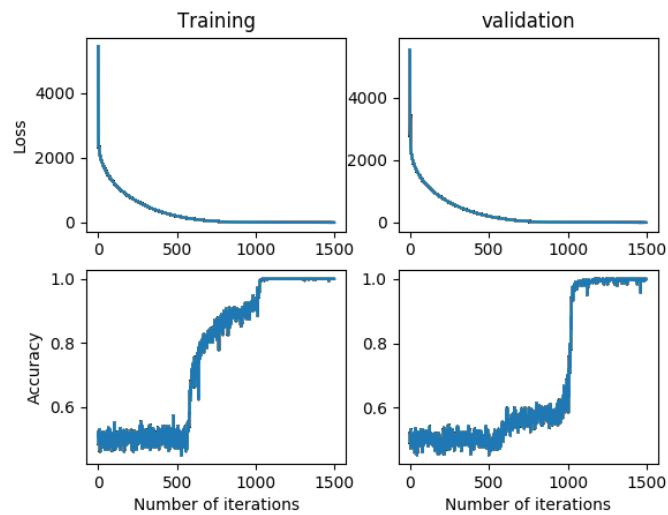


FIGURE 63. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

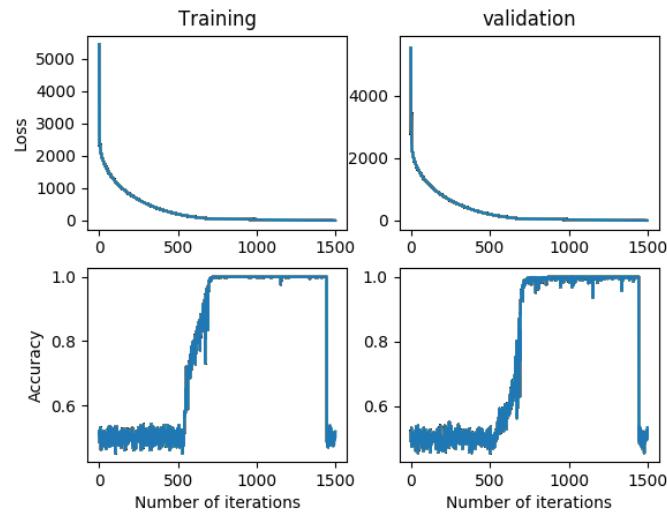


FIGURE 64. Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

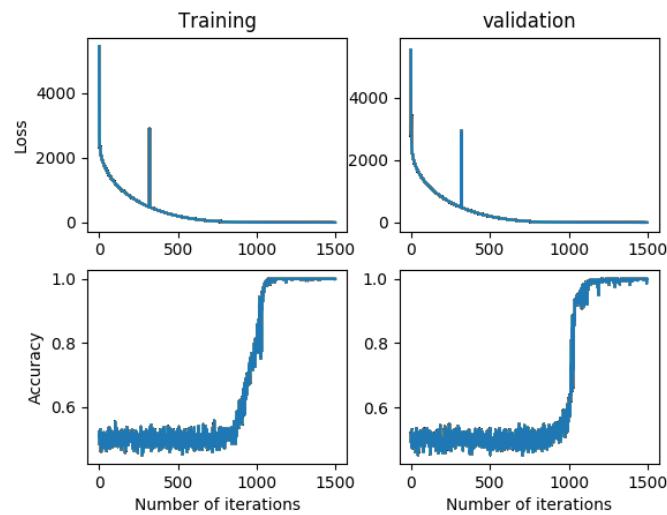


FIGURE 65. Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

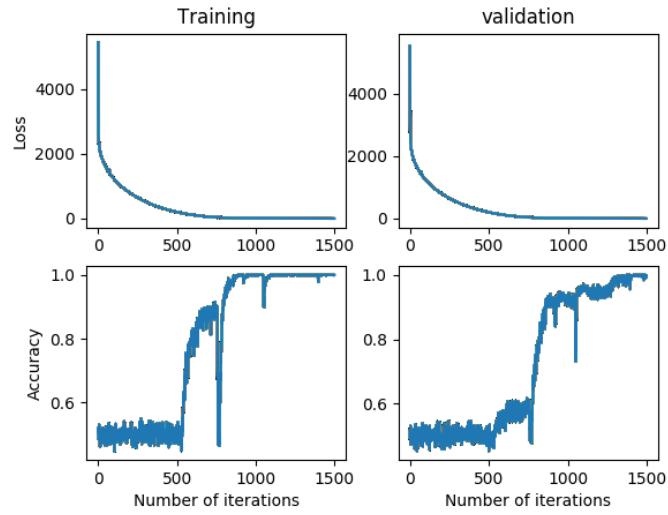


FIGURE 66. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

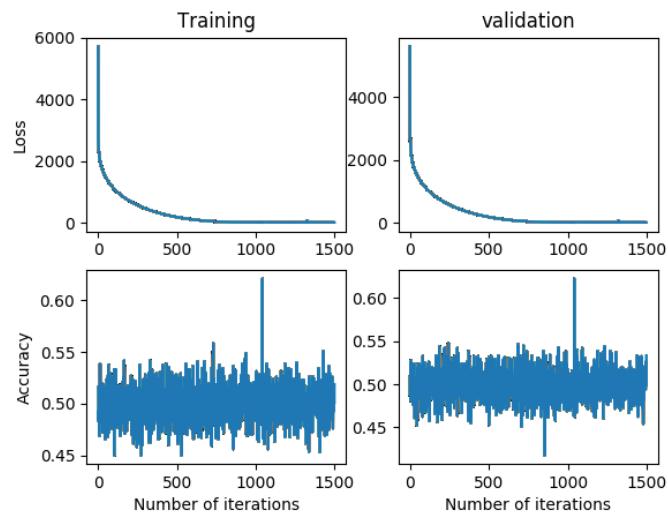


FIGURE 67. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

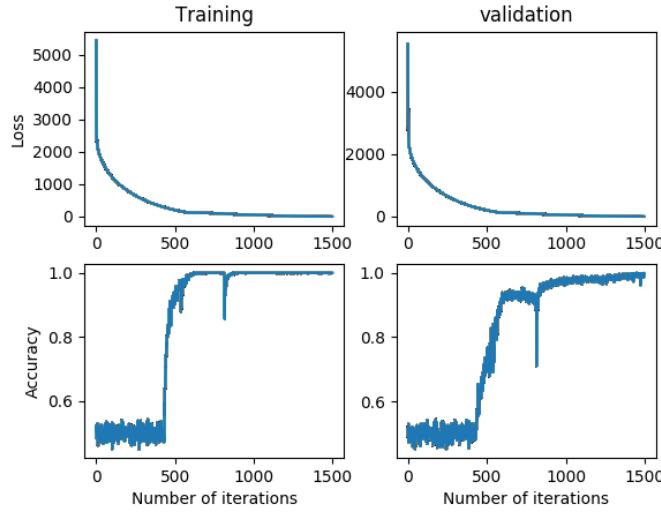


FIGURE 68. Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5

4.2. 12/30/2017.

- (1) Tested the data augmentation scheme manually. Printed out a series of images and their respective augmented versions that are given when generator.next is invoked, respectively with the data aug option set to false and with the data aug option set to true.

4.3. 1/1/2018.

- (1) Added main_glassliquid_restart.py, a file that can run training starting from a previous saved model. Tested it. Now, let's try to train again on the dataset that consists of glasses and liquids that are each 12 time steps away from the assumed glass transition temperature of 0.21. Previously, the best parameters were batch size 200, learning rate 1e-3, beta 0.01, dropout 0.4. Now, we are going to do a series of simulations using the same parameters, except, first, we are going to include dataset augmentation, and second, we are going to run the simulation at least twice as long. Then, once we get a sense for how these train, we will add in a learning rate threshold step, or even multiple threshold steps if necessary!

4.4. 1/4/2018.

- (1) Realized that I had run on the original endpoint data. So, copied over the folder called "data_12steps" into the current machine-learning-glasses directory, removed the current metadata folder, and created a new metadata folder with this data. Then, set up new training runs again.

5. 1/9/2018

- (1) Collected data from the above-mentioned simulations. The training curves are as follows:

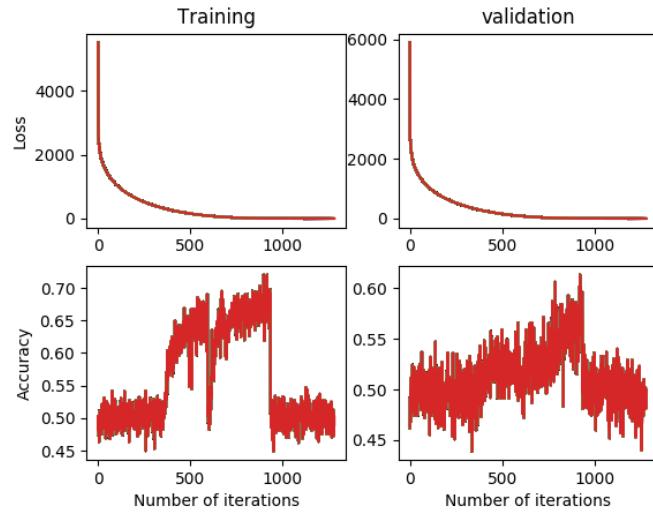


FIGURE 69. Version 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

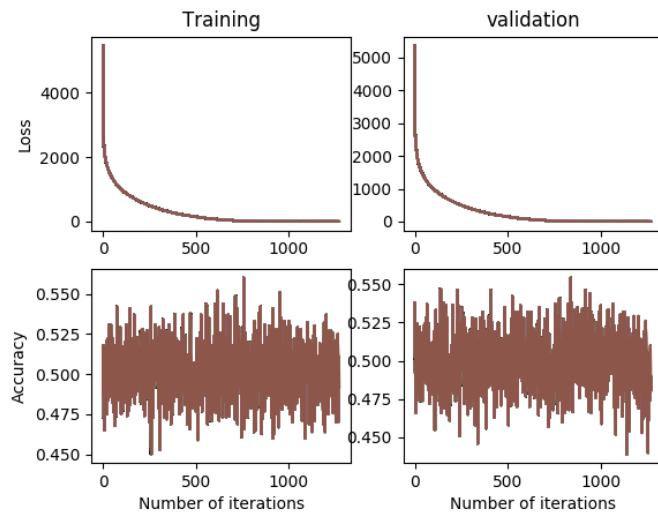


FIGURE 70. Version 2: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

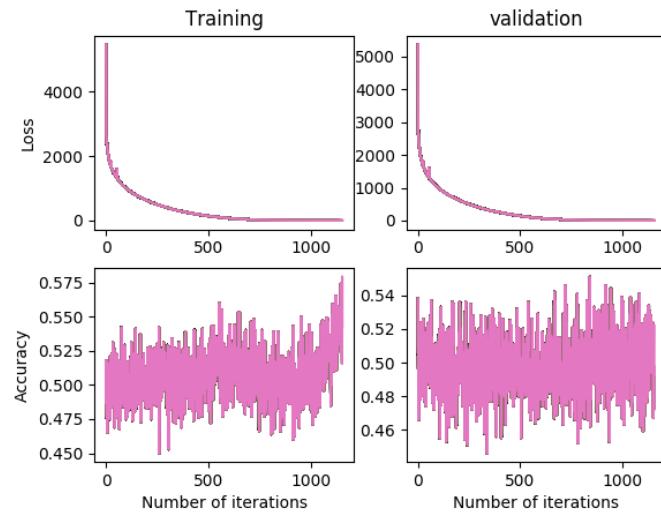


FIGURE 71. Version 3: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

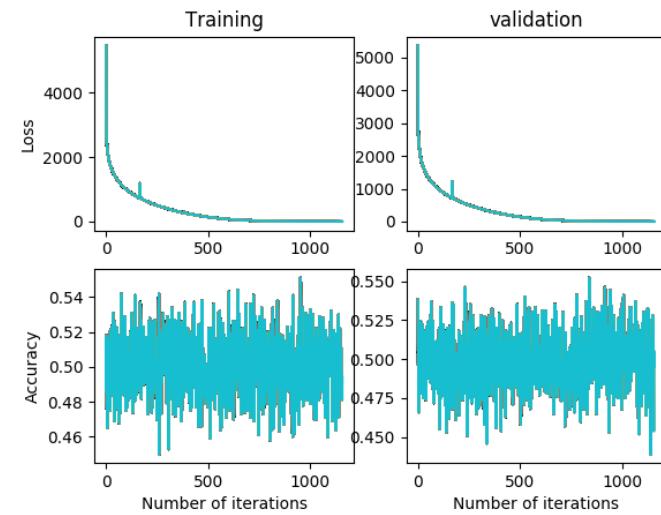


FIGURE 72. Version 4: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

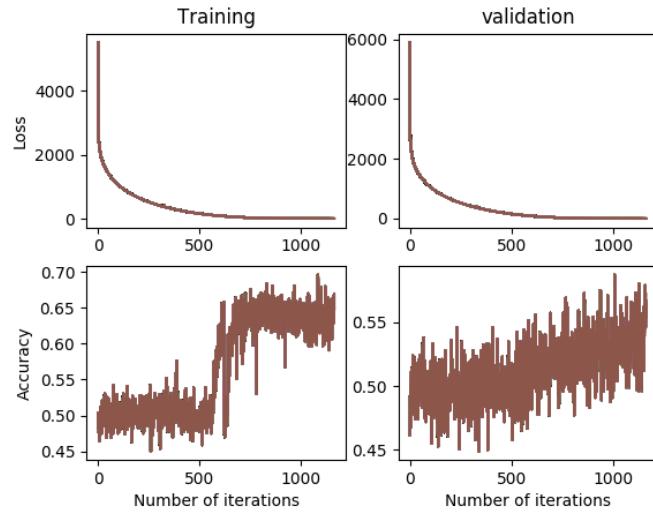


FIGURE 73. Version 5: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

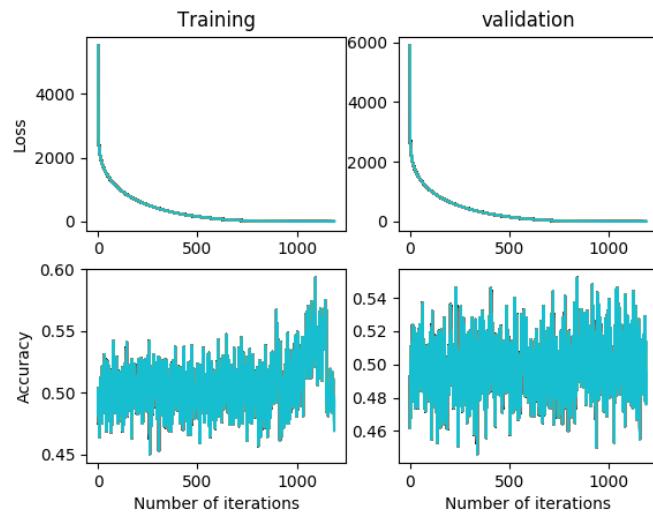


FIGURE 74. Version 6: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

- (2) The problem is that my allotted time (36 hours) was not enough, so the simulations terminated early. My program, however, was only set to save the current model at the end of the iterations. So, the current model was not saved, only the best model was saved. So, I went back into the main code and changed it so that the current model is updated and saved at every iteration, in addition to the best

model being updated when necessary. I also included restart capability into the main code (using parsing options), so that there is not a separate restart file.

- (3) I am now running 7 new simulations. The first is just a re-do of version1 step 0. The others take version5 step 0 and start from the saved best model as a starting point, using the new restart capabilities of the main code. The first two simulations from this starting point (version5 step2 and version5 step2 trial2) use the original parameters. The next two (version5 step2 trial3 and version5 step2 trial4) use $\eta_{\text{initial}}=1e-4$ and $\eta_{\text{final}}=1e-5$, and the final two (version5 step2 trial5 and version5 step2 trial6) use $\eta_{\text{initial}}=1e-5$ and $\eta_{\text{final}}=1e-6$. We will see how these 7 new runs turn out.

5.1. 1/14/2018.

- (1) Here is the result for the continuations from Version 5 step 0, i.e. Version 5 step 1 trials 1 through 6.

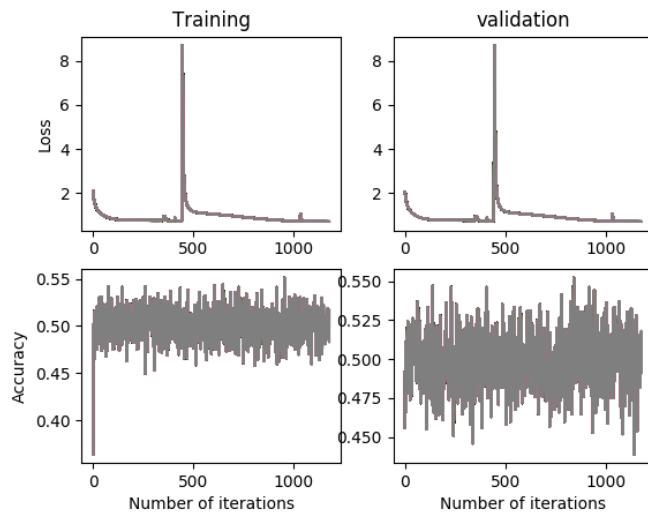


FIGURE 75. Version 5 step 1 trial 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

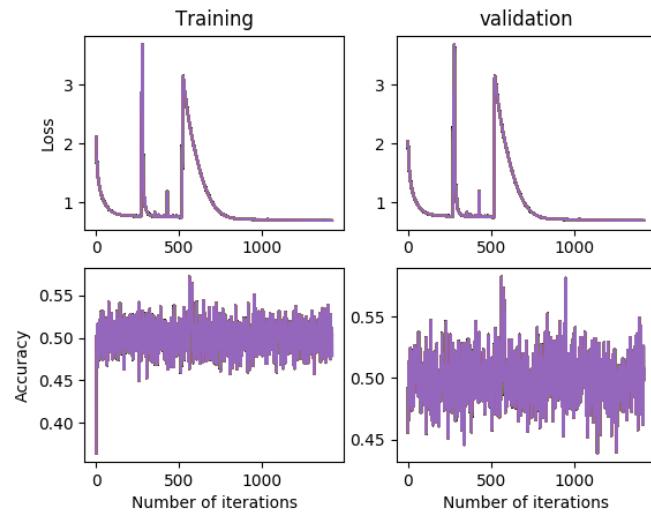


FIGURE 76. Version 5 step 1 trial 2: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

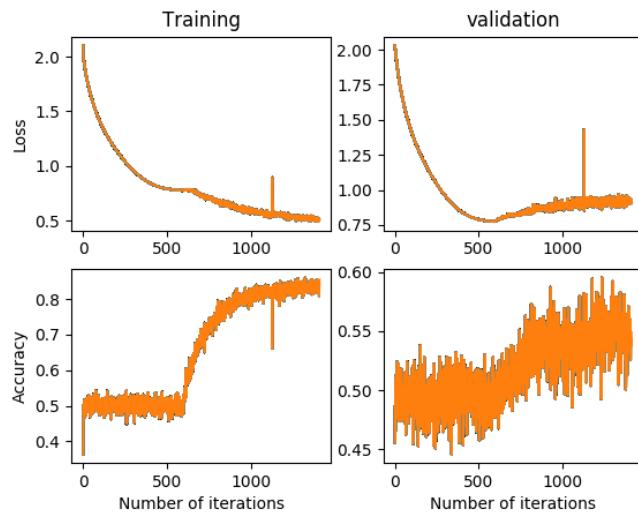


FIGURE 77. Version 5 step 1 trial 3: Initial Learning Rate = 1e-4, Final Learning Rate = 1e-5, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

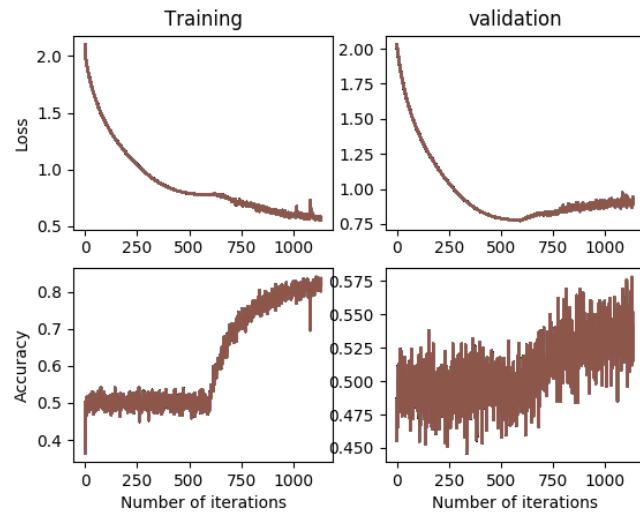


FIGURE 78. Version 5 step 1 trial 4: Initial Learning Rate = 1e-4, Final Learning Rate = 1e-5, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

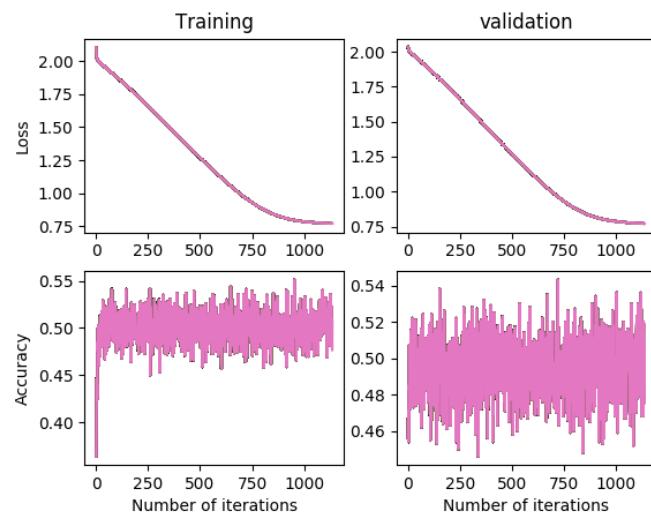


FIGURE 79. Version 5 step 1 trial 5: Initial Learning Rate = 1e-5, Final Learning Rate = 1e-6, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

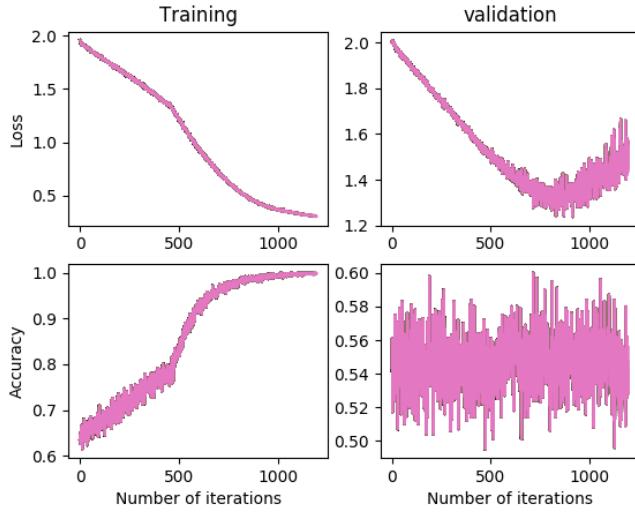


FIGURE 80. Version 5 step 1 trial 6: Initial Learning Rate = 1e-5, Final Learning Rate = 1e-6, Schedule Length = NA, Batch Size = 80, L2 beta = 0.01, Dropout = 0.4, data aug = true

(2) I am now going to generate new data that is farther away from the assumed glass transition temperature of $T_g = 0.21$. Based on cooling.dat file in the folder N2e7/0, which is the first of 10,000 simulations, this temperature is at the 184th line of the file (0.21575), which is the 183rd time listed. There are 200 times listed in cooling.dat, each one separated by a time length of 100,000 time steps. This is because the total simulation is $2e7$, or 20 million. In the folder /project/depablo/kswanson/2017, I open the file make_images_kirk.sh and change the liquid generation number to 500000 and keep the glass generation number at 20000000.

5.2. 1/15/2018.

- (1) After generating the new datapoint, I create a folder called data_liquid5 and transfer it to the git repo. Now, I delete the current metadata folder and create a new metadata folder that contains the data_liquid5 in it.
- (2) V1-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (3) V2-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (4) V3-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (5) V4-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true.

- (6) V5-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (7) V6-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (8) V7-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (9) V8-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (10) V9-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (11) V10-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true.

5.3. 1/19/2018.

- (1) Using timestep 5 (500000) for the liquid and timestep 200 (20000000) for the glass, here are training results:

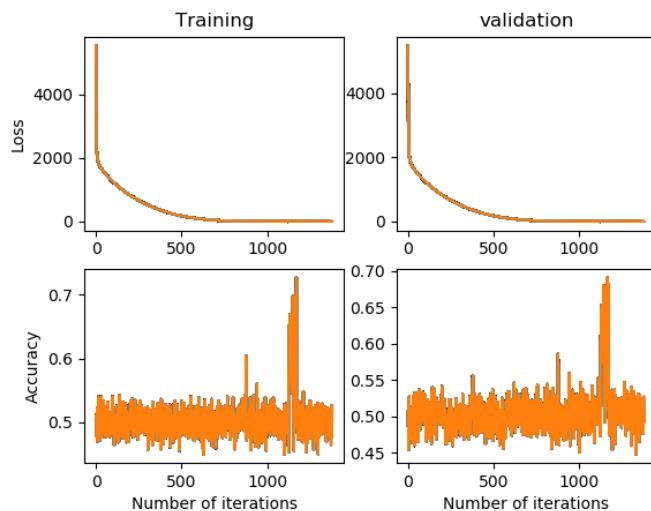


FIGURE 81. Version 1 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

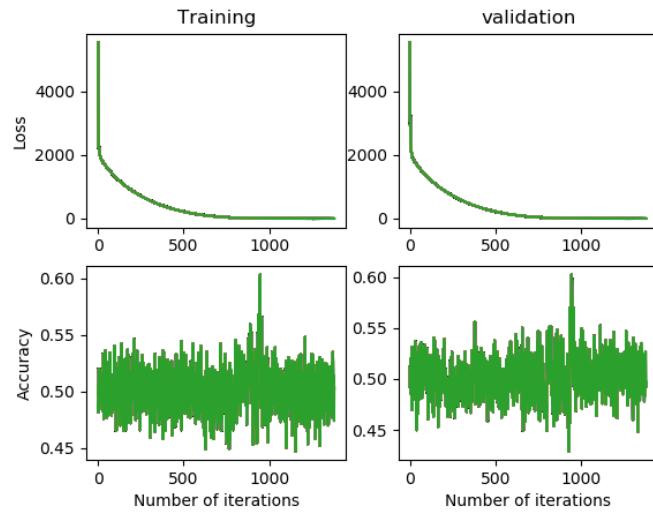


FIGURE 82. Version 2 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

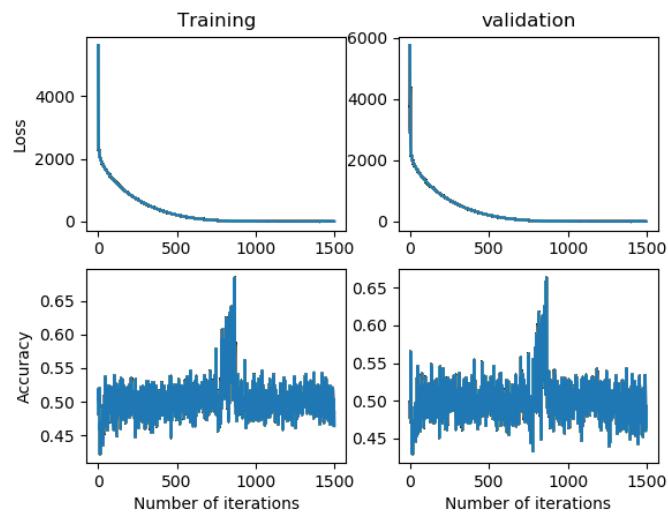


FIGURE 83. Version 3 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

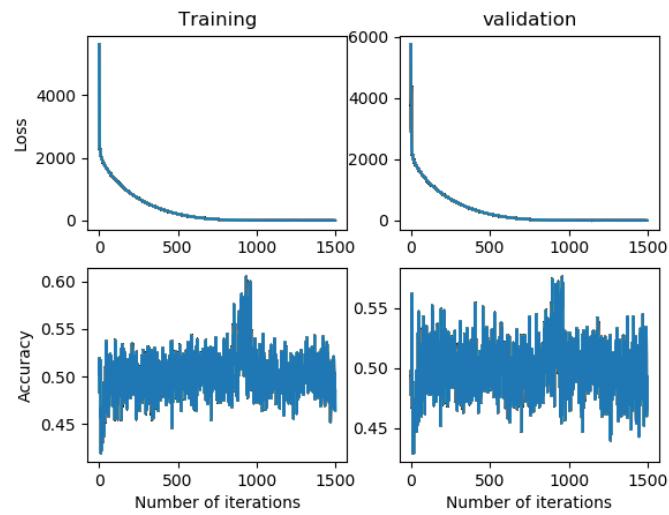


FIGURE 84. Version 4 step 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

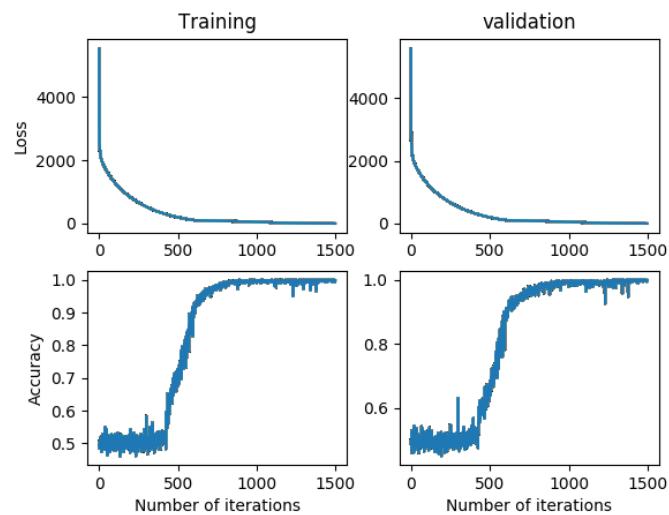


FIGURE 85. Version 5 step 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

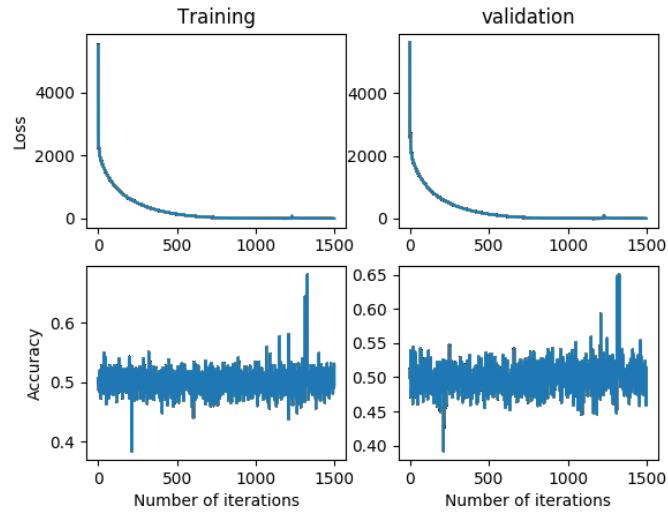


FIGURE 86. Version 6 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

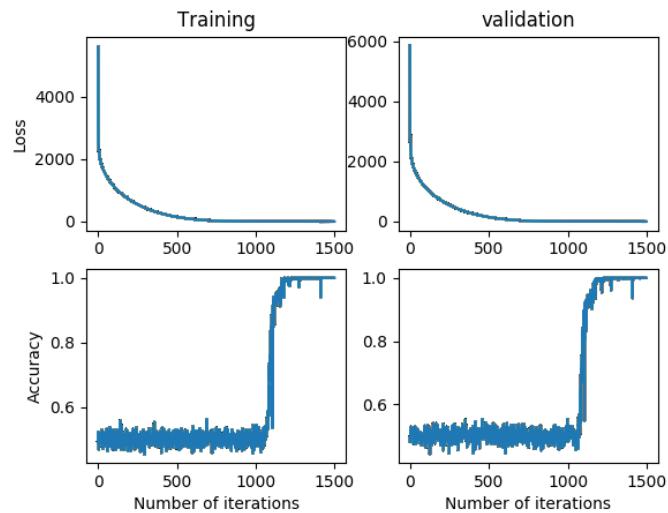


FIGURE 87. Version 7 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

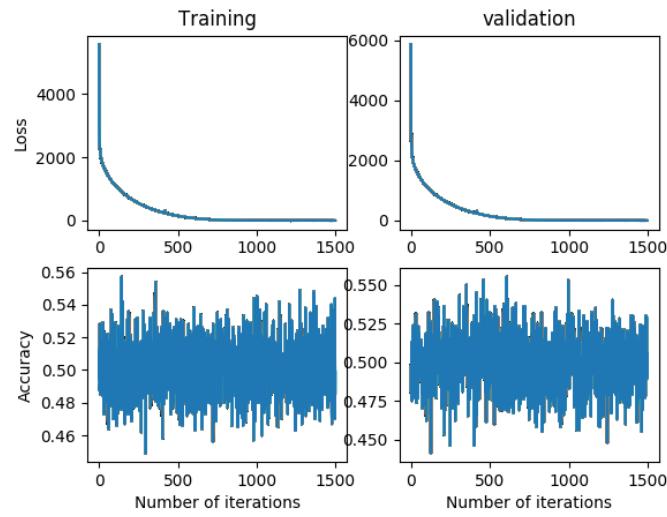


FIGURE 88. Version 8 step 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

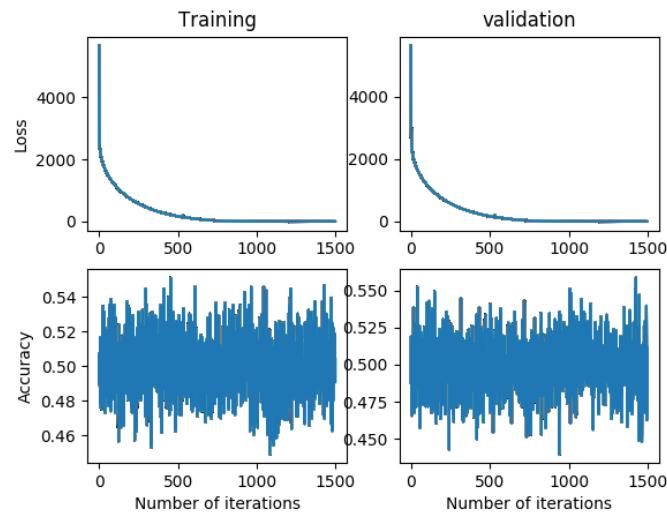


FIGURE 89. Version 9 step 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

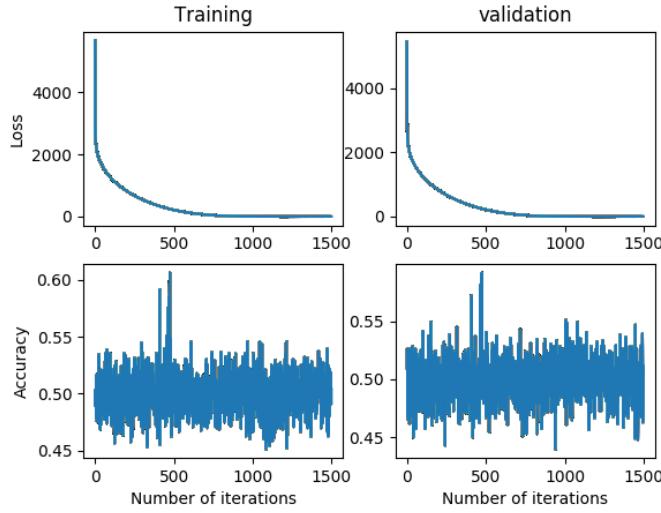


FIGURE 90. Version 10 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

- (2) Version 5 Step 1 had a final validation accuracy of 99.7% and a final test accuracy of 99.7333%. Added changes to main_glassliquid.py so that by setting -restart=True and -evaluation=True in the options, and setting -restart_file to the desired filename, one can check the final validation and test accuracy of a given model.
- (3) Version 7 Step 1 had a final validation accuracy of 99.8% and a final test accuracy of 99.8333%.
- (4) Now I am generating images for liquid timestep 10 (1000000).

5.4. 1/20/2018.

- (1) I am now switching metadata to be from data_liquid10, our new current dataset.
- (2) V1-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (3) V2-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (4) V3-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (5) V4-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (6) V5-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true.

- (7) V6-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (8) V7-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (9) V8-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (10) V9-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (11) V10-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (12) V11-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (13) V12-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (14) V13-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 130, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (15) V14-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 130, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (16) V15-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 150, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (17) V16-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 150, L2 beta = 0.01, Dropout = 0.5, data aug = true.

5.5. 1/23/2018.

- (1) Collected results from training runs on the data_liquid10 data. Here they are:

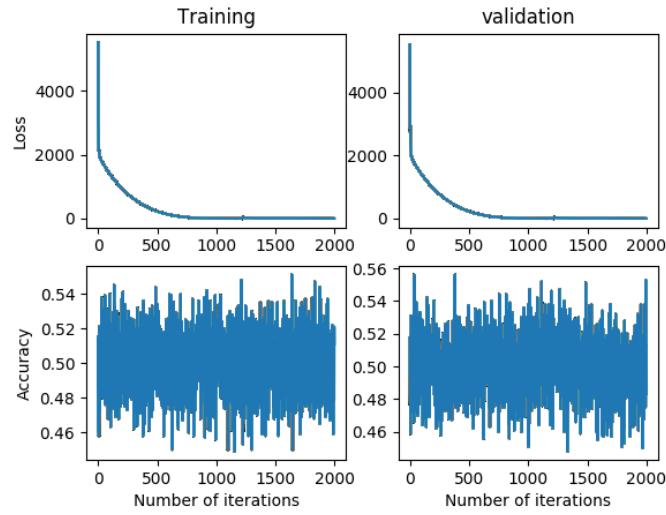


FIGURE 91. Version 1 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

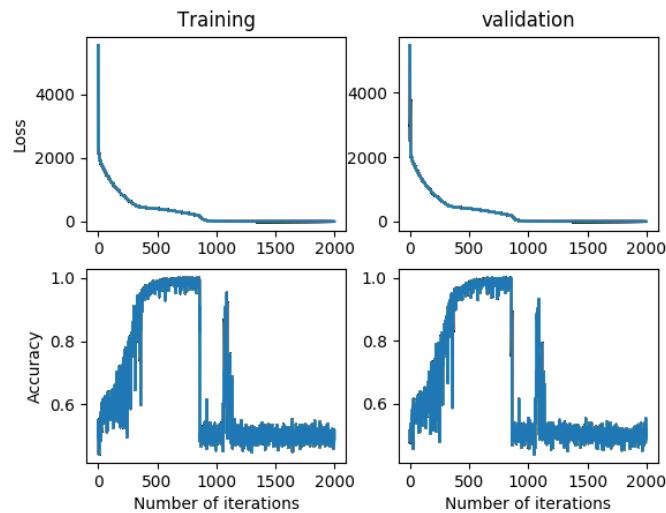


FIGURE 92. Version 2 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

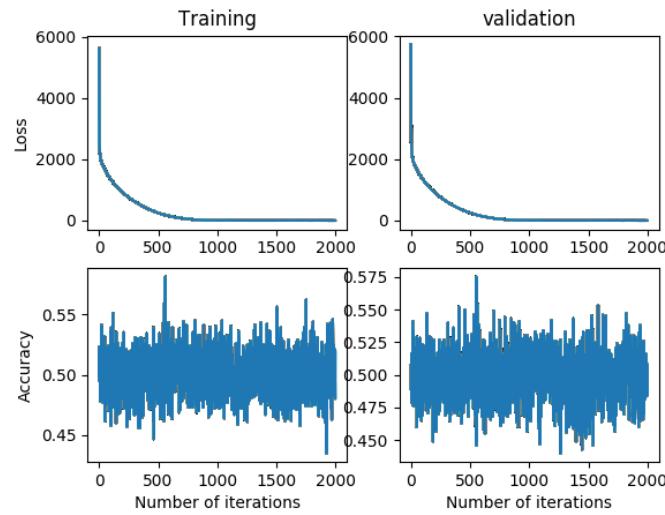


FIGURE 93. Version 3 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

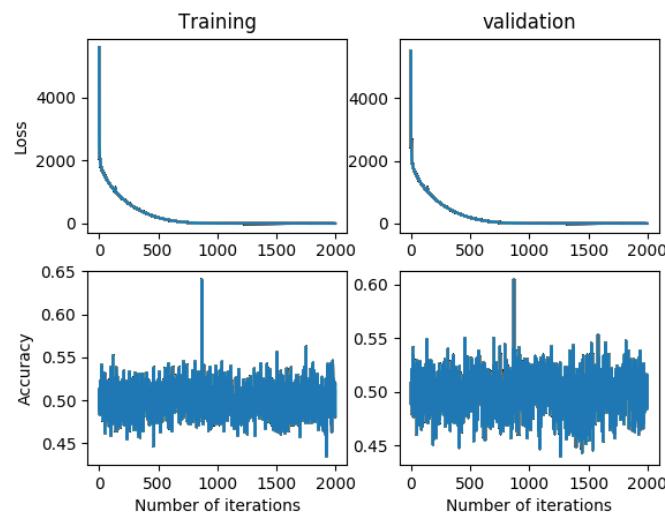


FIGURE 94. Version 4 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

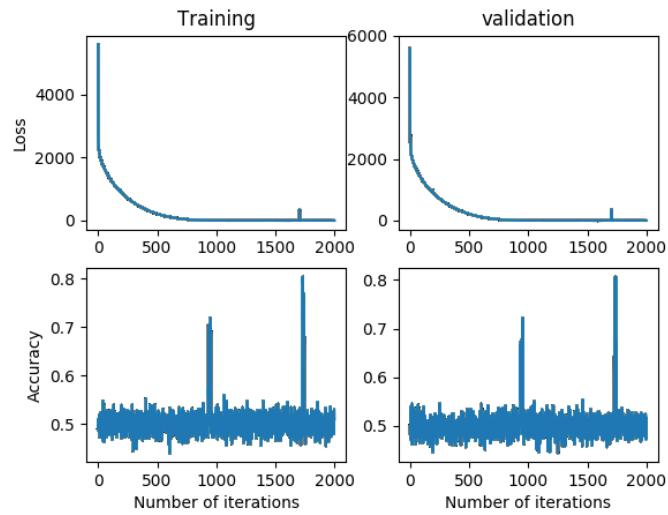


FIGURE 95. Version 5 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

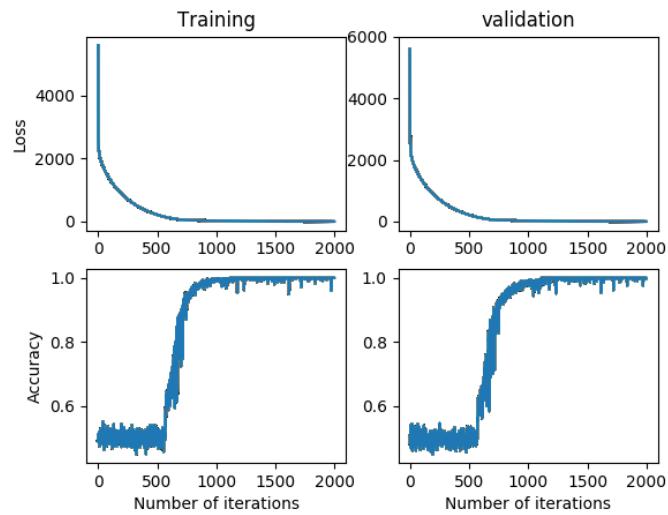


FIGURE 96. Version 6 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

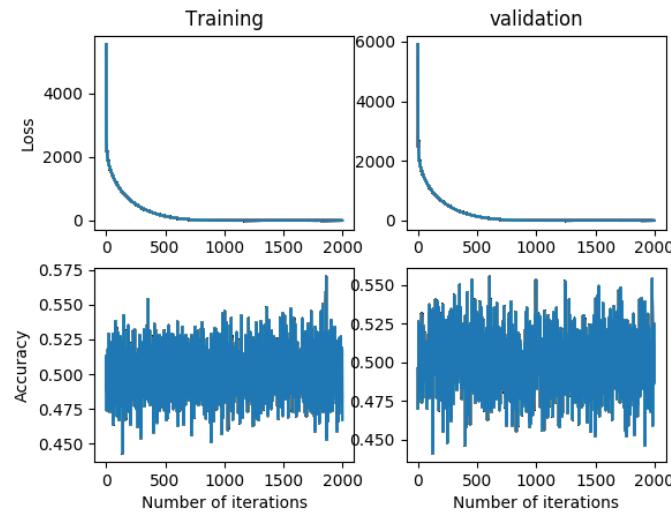


FIGURE 97. Version 7 step 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

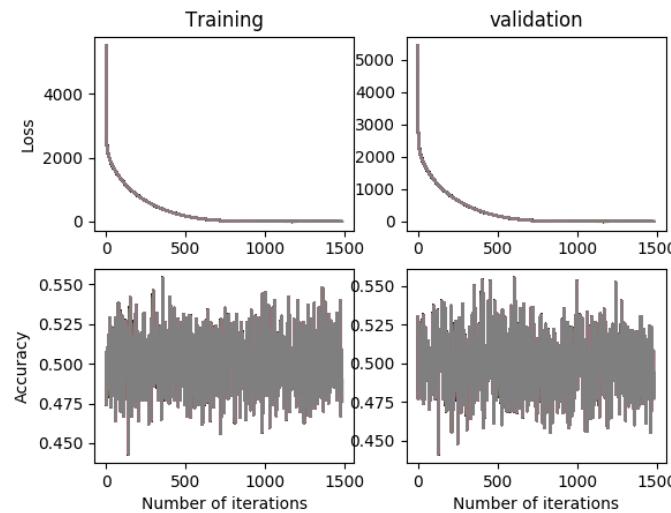


FIGURE 98. Version 8 step 1: Initial Learning Rate = $1e-3$, Final Learning Rate = $1e-4$, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

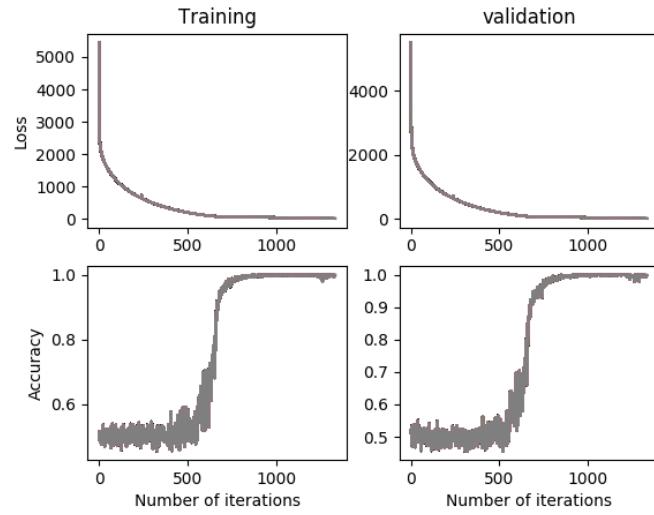


FIGURE 99. Version 9 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

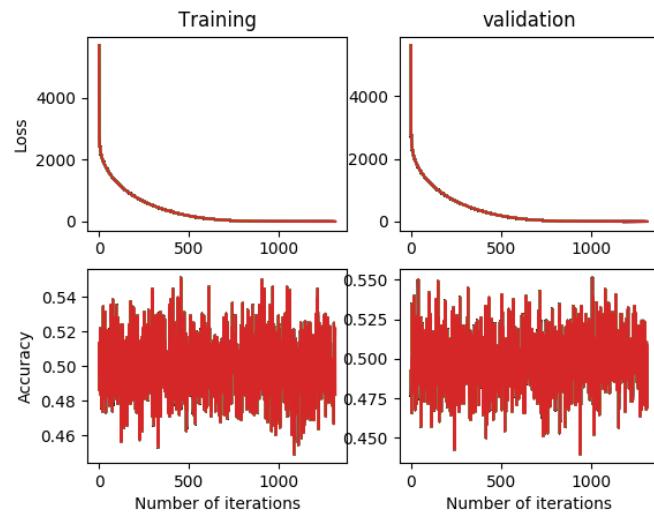


FIGURE 100. Version 10 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

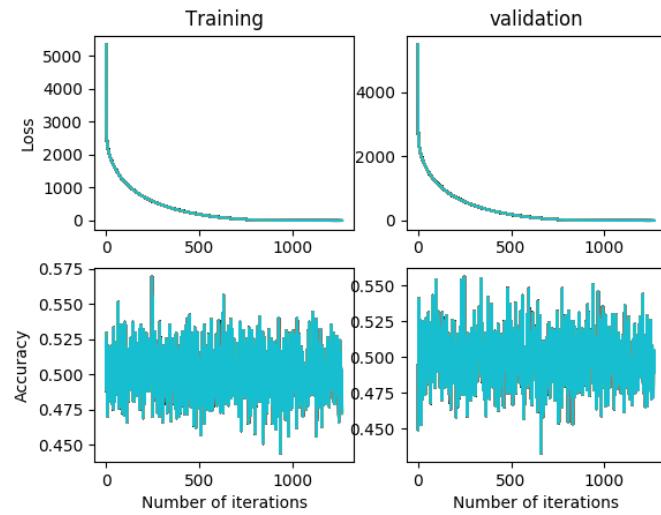


FIGURE 101. Version 11 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

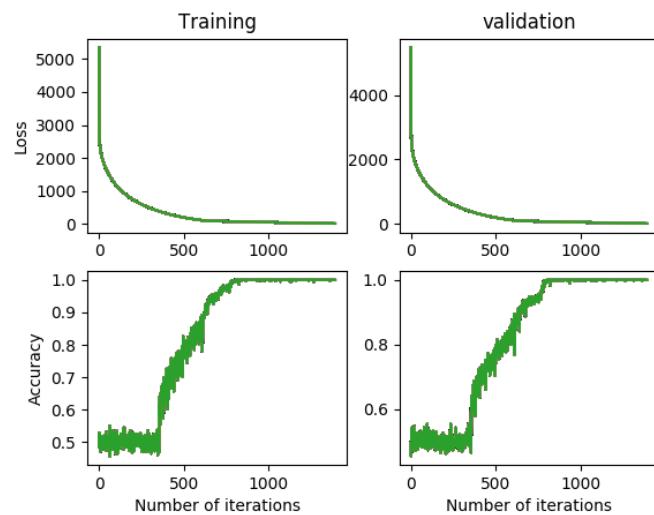


FIGURE 102. Version 12 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

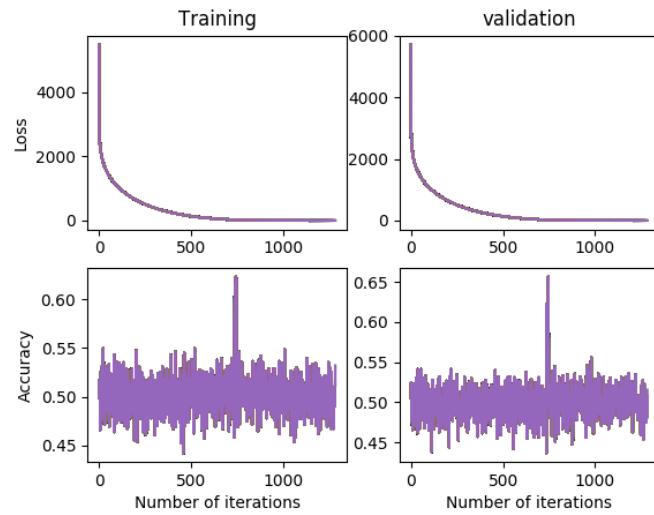


FIGURE 103. Version 13 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 130, L2 beta = 0.01, Dropout = 0.5, data aug = true

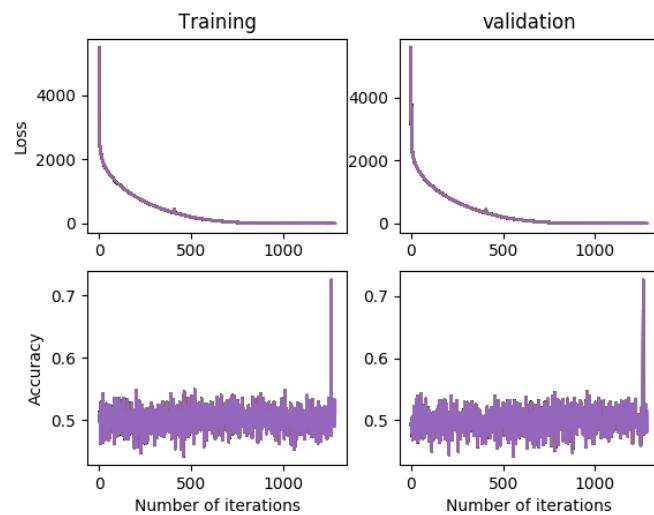


FIGURE 104. Version 14 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 130, L2 beta = 0.01, Dropout = 0.5, data aug = true

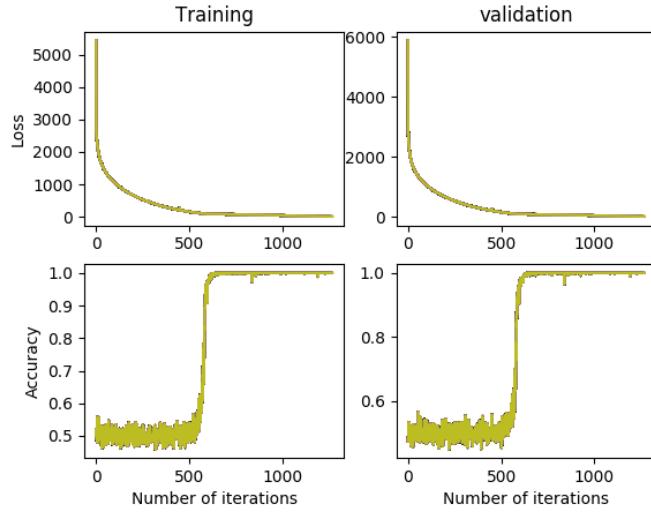


FIGURE 105. Version 15 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 150, L2 beta = 0.01, Dropout = 0.5, data aug = true

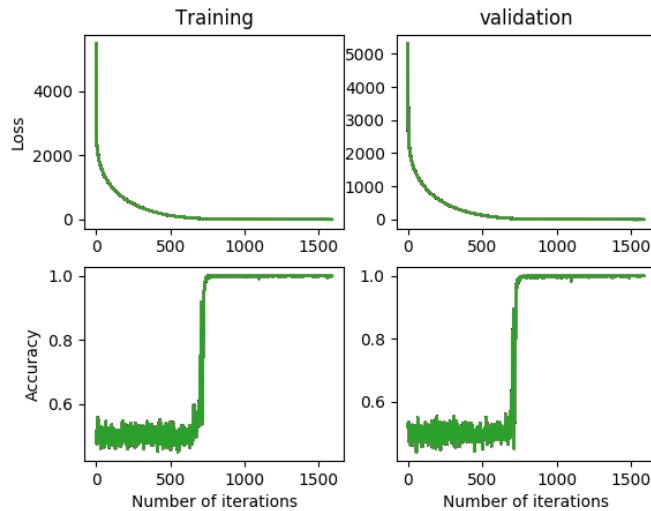


FIGURE 106. Version 16 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 150, L2 beta = 0.01, Dropout = 0.5, data aug = true

- (2) Version 16 Step 1 best model had a final validation accuracy of 99.9% and a final test accuracy of 99.9333%.
- (3) Now I am going to save this model, and delete the others.
- (4) Moved the data_liquid15 file to the working directory. Deleting metadata and creating a new metadata for this dataset.

- (5) V1-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (6) V2-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (7) V3-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (8) V4-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (9) V5-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (10) V6-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (11) V7-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (12) V8-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (13) V9-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true.
- (14) V10-S1 has hyperparameters: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true.

5.6. 1/23/2018.

- (1) Collected results from training runs on the data_liquid15 data. Here they are:

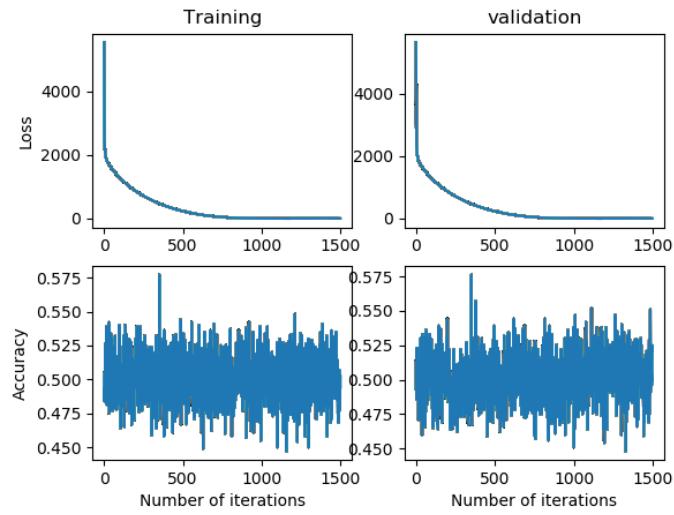


FIGURE 107. Version 1 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

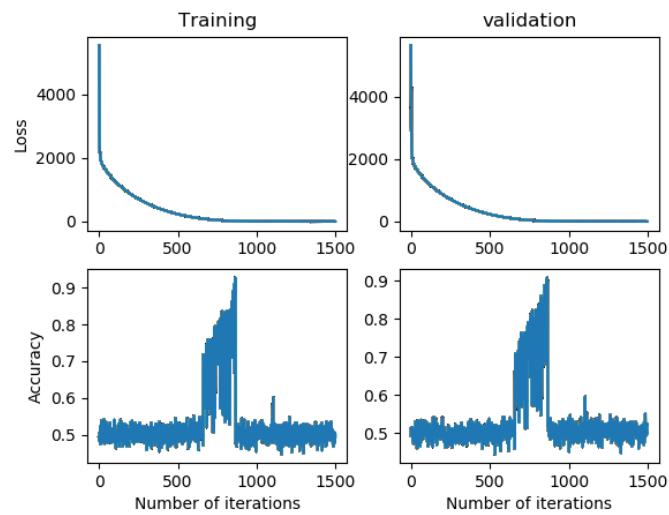


FIGURE 108. Version 2 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

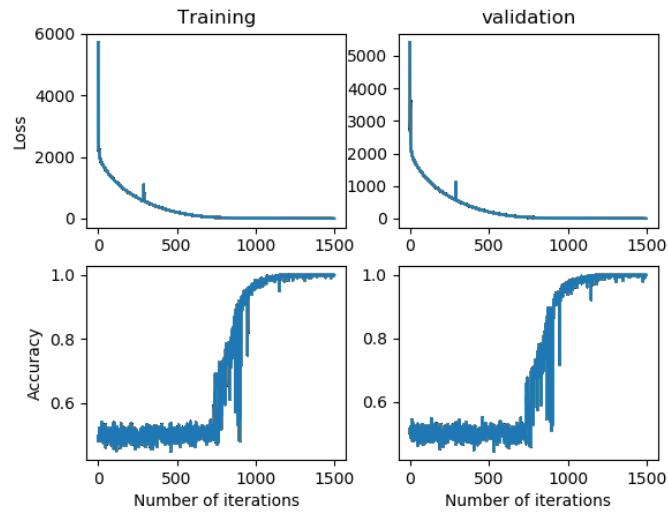


FIGURE 109. Version 3 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

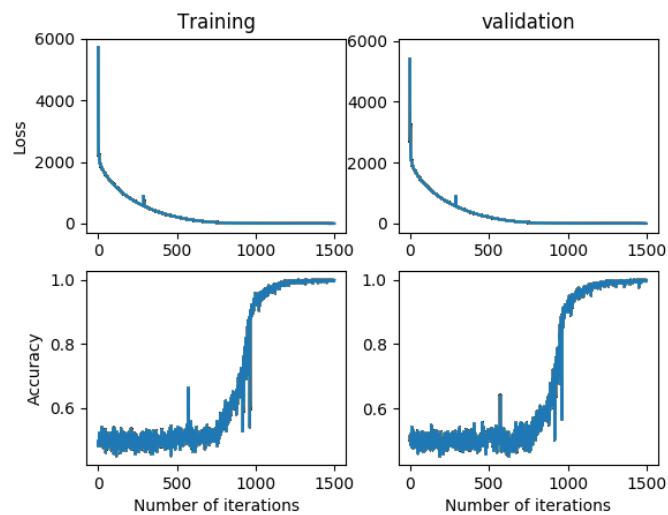


FIGURE 110. Version 4 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

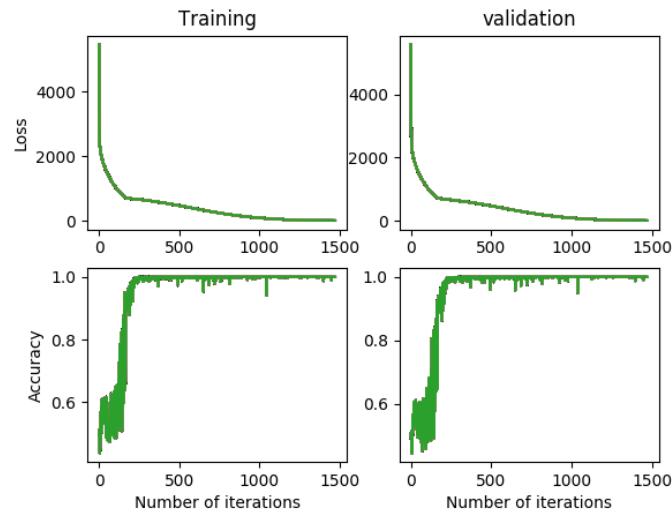


FIGURE 111. Version 5 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

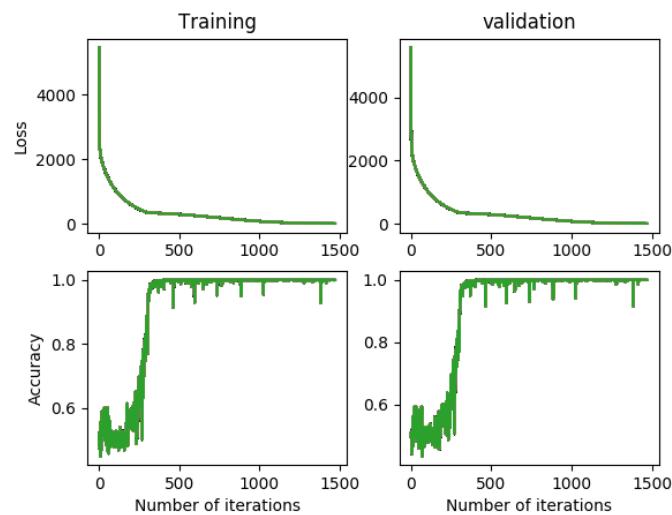


FIGURE 112. Version 6 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

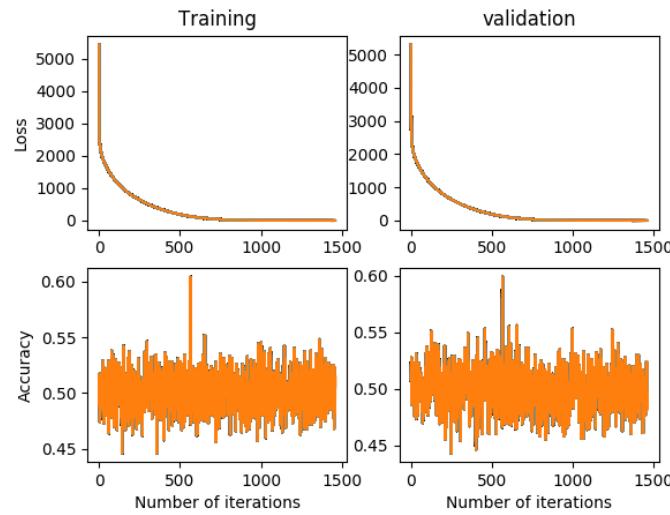


FIGURE 113. Version 7 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

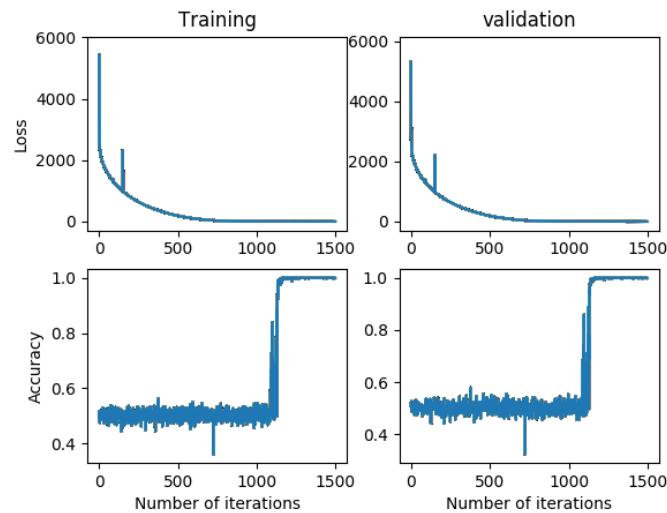


FIGURE 114. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

- (2) Version 4 Step 1 had a best model final validation accuracy of 99.7667% and final test accuracy of 99.9333%.
- (3) Saving this model, deleting others.
- (4) Switching metadata to data_liquid25.
- (5) Running same set of versions as before.

5.7. 2/2/2018.

- (1) Collected results from training runs on the data_liquid25 data. Here they are:

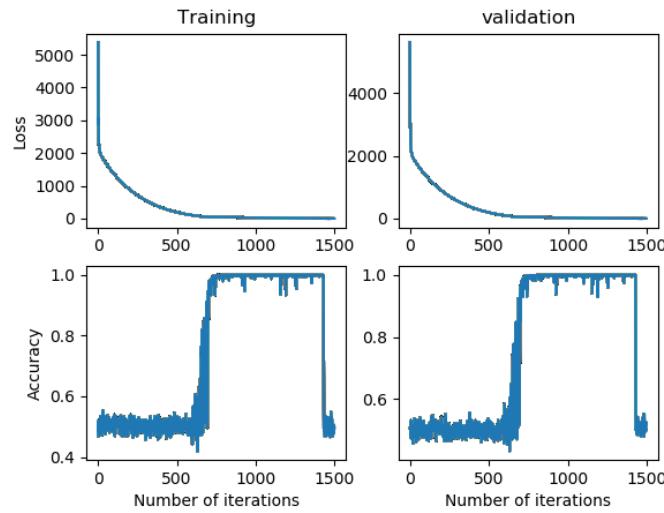


FIGURE 115. Version 1 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

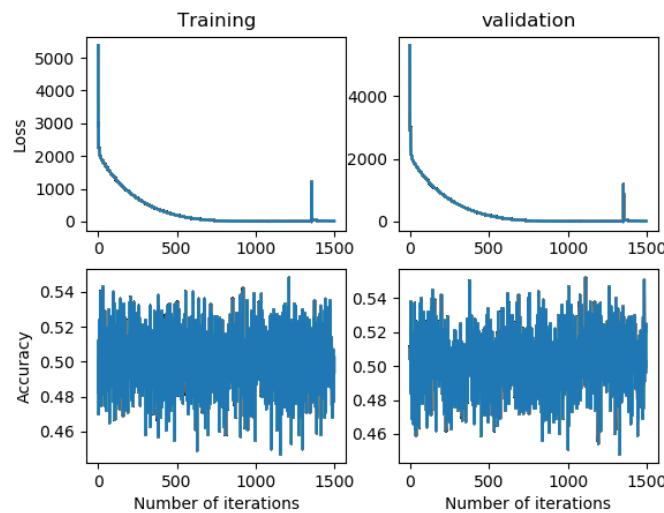


FIGURE 116. Version 2 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

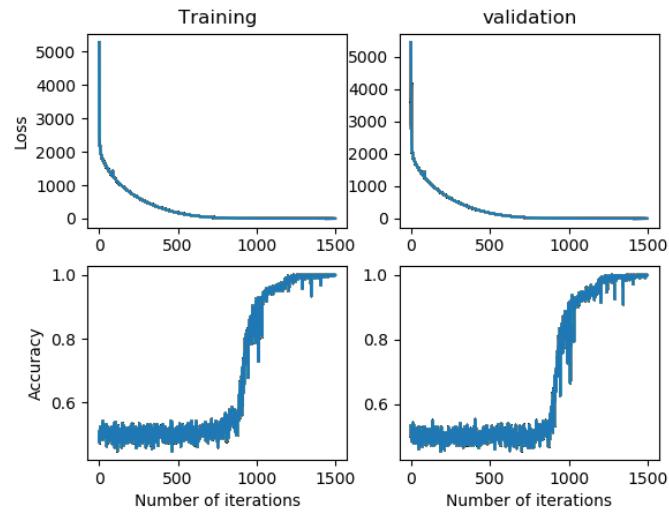


FIGURE 117. Version 3 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

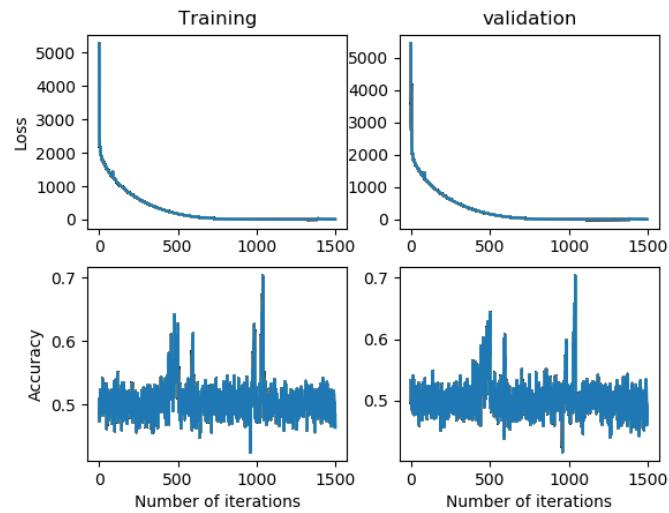


FIGURE 118. Version 4 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

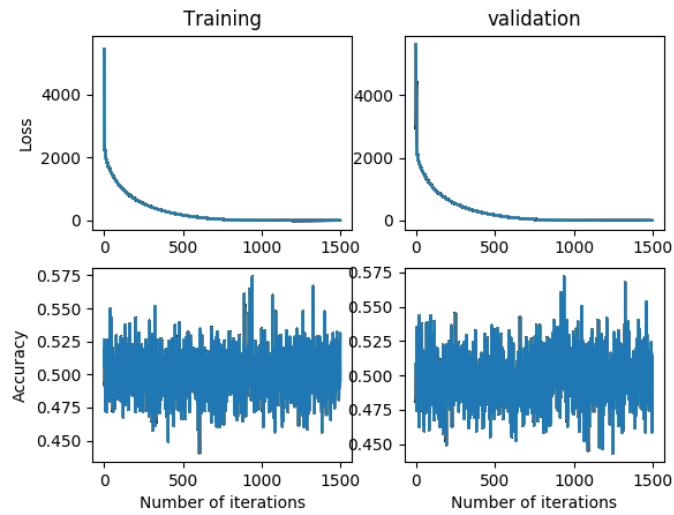


FIGURE 119. Version 5 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

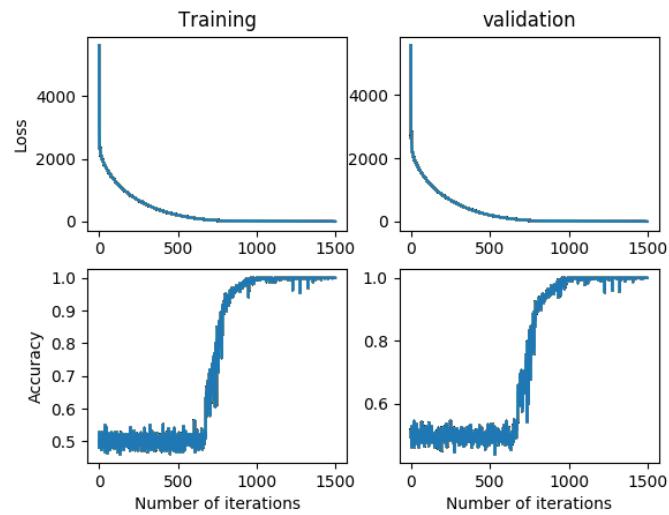


FIGURE 120. Version 6 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

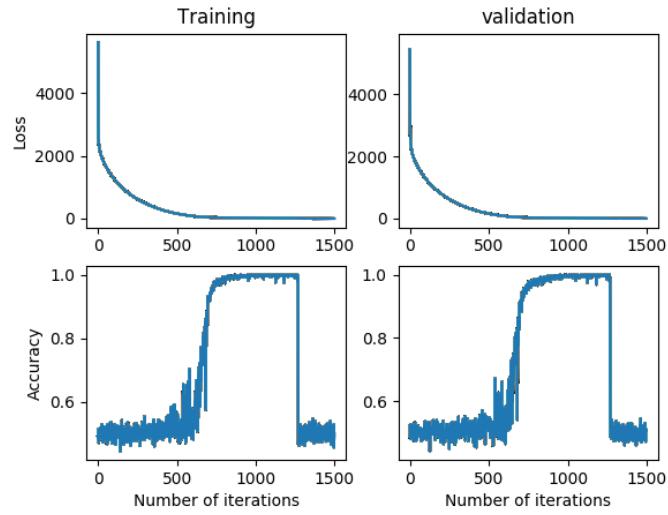


FIGURE 121. Version 7 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

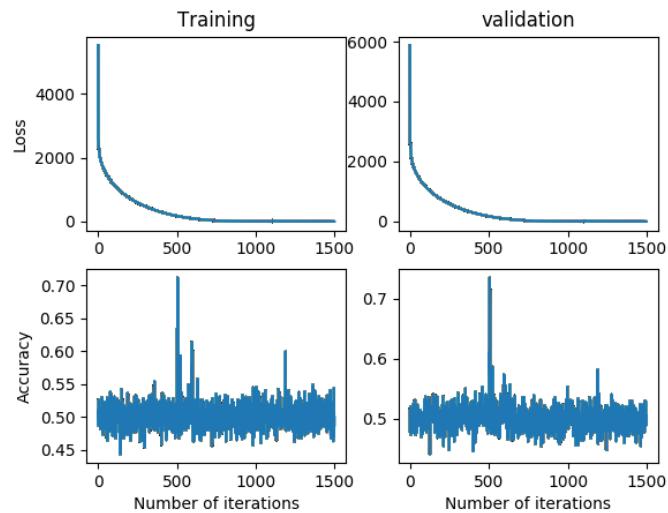


FIGURE 122. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

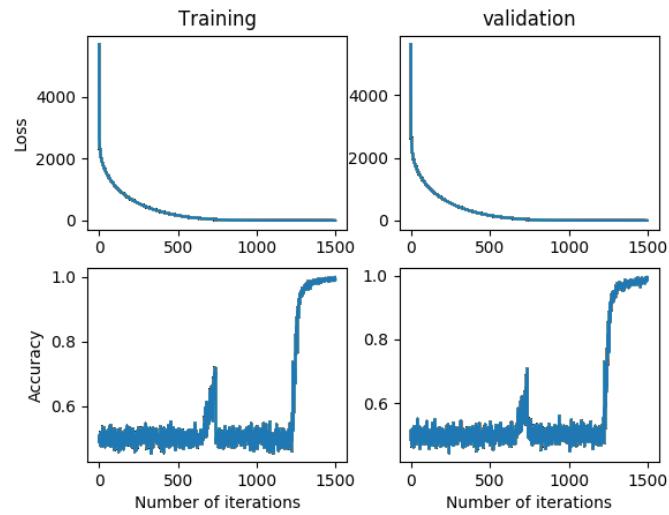


FIGURE 123. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

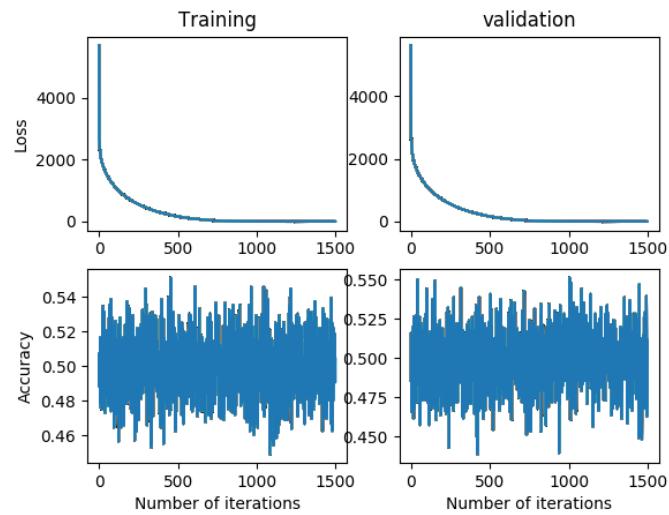


FIGURE 124. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

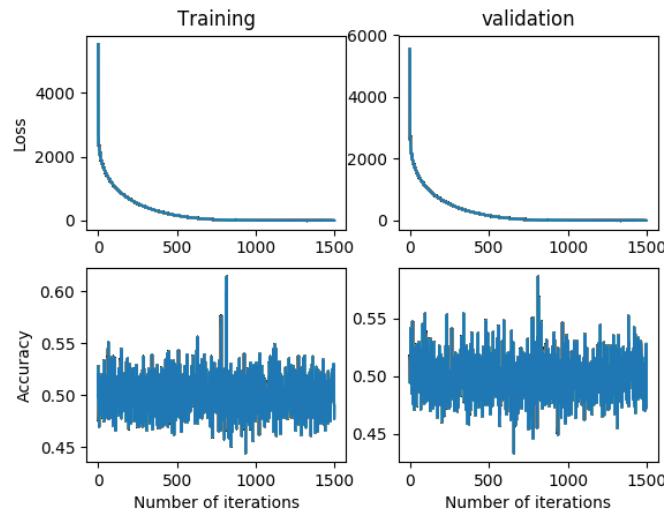


FIGURE 125. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

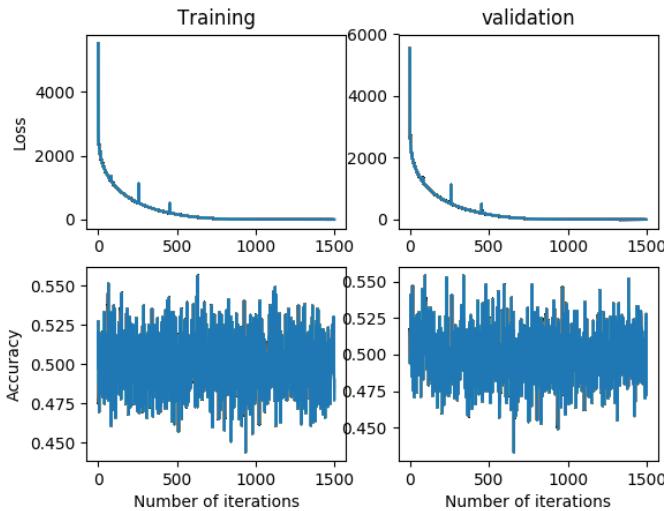


FIGURE 126. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

- (2) Version 6 Step 1 has a final validation accuracy of 99.7% and test accuracy of 99.6667%.
- (3) Saving Version 6 model.
- (4) Just ran the saved `data_liquid5_version7_step1_best_model.ckpt` on the metadata from `data_liquid25`. It had a final validation accuracy of 99.8557% and final test accuracy of 99.6333%. Wow! So how about for the time being, we

just keep running this model on closer and closer liquid data sets to see how well it does...

- (5) Changing metadata to be from data_liquid35.
- (6) Running the data_liquid5 model as above...this time gets 99.5667% validation and 99.7333% test.
- (7) Creating data now for data_liquid45 and data_liquid60.
- (8) Ok, now changed the metadata to be from data_12steps. This yields a validation accuracy of 48.2667% and test accuracy of 48.5%. Interesting...so where does it break down? Let's try to find this.
- (9) Removing all other models.

5.8. 2/3/2018.

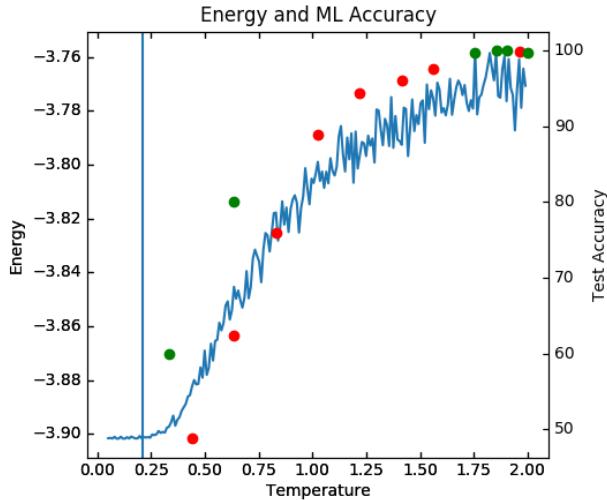
- (1) Switching metadata to be from data_liquid45.
- (2) Running data_liquid5_version7_step1_best_model.ckpt gives 96.4755% validation accuracy and 97.5466% test accuracy on data_liquid45.
- (3) Switching to metadata data_liquid60.
- (4) Running data_liquid5_version7_step1_best_model.ckpt gives 96.4446% validation accuracy and 96.0663% test accuracy on data_liquid60.
- (5) Switching to metadata data_liquid80.
- (6) Running data_liquid5_version7_step1_best_model.ckpt gives 92.8547% validation accuracy and 94.408% test accuracy on data_liquid80.
- (7) Switching to metadata data_liquid100.
- (8) Running data_liquid5_version7_step1_best_model.ckpt gives 90.0206% validation accuracy and 88.8201% test accuracy on data_liquid100.
- (9) Switching to metadata data_liquid120.
- (10) Running data_liquid5_version7_step1_best_model.ckpt gives 76.8569% validation accuracy and 75.9367% test accuracy on data_liquid120.
- (11) Switching to metadata data_liquid140.
- (12) Running data_liquid5_version7_step1_best_model.ckpt gives 60.5473% validation accuracy and 62.3961% test accuracy on data_liquid140.
- (13) Switching to metadata data_liquid160.
- (14) Running data_liquid5_version7_step1_best_model.ckpt gives 47.7124% validation accuracy and 48.7964% test accuracy on data_liquid160.
- (15) Switching metadata to data_liquid140 and running training again, using the same stuff as before.

5.9. 2/19/2018.

- (1) New idea on MPNN. Modified Josh's code, "plot_config.py", and renamed it "coord_config.py". This new file eliminates the plotting function, and instead outputs a text file with the raw (x, y, z) atomic coordinates of a given configuration. Added this file to GitHub.
- (2) Currently running the (x, y, z) coords program on Midway to generate these files for the endpoints of the simulation (i.e. at 20,000,000 and 100,000).

5.10. 2/22/2018.

- (1) Here is a plot showing progress to date:



- (2) What I sent to Josh and Shubhendu in email: I have a few new results. Attached to this email is a plot I put together a few minutes ago. The blue curve is the value of the inherent structure energy as a function of temperature (values on left axis) taken from one example simulation. The vertical blue line is the assumed glass transition temperature for the 2D Kob-B-Andersen glass, which is at 0.21 (from Dan's vapor-deposited paper). The dots show various machine learning accuracies I have achieved (values on the right axis). I'll explain the red dots first, then the green dots.

Red dots: I created a data set where the glass images were derived from timestep 20,000,000, or temperature = 0.05 and the liquid images were derived from timestep 500,000, or temperature = 1.95125. Temperature = 0.05 is the farthest we can go into the glass regime with our data. I then trained the CNN on this data set and achieved a test accuracy of 99.8%. I saved this model. Then, I used this model to predict whether images were glass or liquid on seven other data sets. Each of these data sets uses the SAME glass images, i.e. images from temperature = 0.05, but uses liquid images from temperatures that are increasingly close to the glass transition. The results are shown as red dots. So, the red dot at temperature = 0.635 and accuracy = 62.3961%, for example, means that this model achieves approximately 62.3961% accuracy when tested on liquid images derived from temperature = 0.635.

Green dots: These dots represent CNNs that have been trained exactly at that data point. So, the green dot at temperature = 0.33275, for example, represents the test accuracy of a CNN that has been trained on liquid images taken from that exact temperature, 0.33275, with glass images again taken from temperature = 0.05.

It looks nice, but what if we're just learning a much more complicated way of calculating the inherent structure energy?

- (3) After speaking to Josh, these are the steps I decided that I need to take, using CNNs:
- (4) Classify glass vs liquid:

- Improve model accuracy using data across the liquid regime
- Calculate T_g rigorously
- Calculate the potential energy instead of the inherent structure energy
- NEW classify between two liquids? (Added 3/6/2018)
- (5) NEW idea: identify the phase transition temperature by stepping across into the glass regime until the classification accuracy gets really low. (Added 3/8/2018)
 - Generate supercooled liquid data at the same energy as the glass
 - Generate artificial 5-fold patterns
 - Generate 3D data for classification
- (6) Classify liquid-cooled versus vapor-deposited glasses:
 - Generate data
- (7) Other tasks?
 - Classify low density vs high density water
- (8) We also have an answer as to why the CNN learning to classify liquid versus glass, irrespective of what it is learning (i.e. potential energy, inherent energy, geometry, or anything else). From the viewpoint of the CNN, we know nothing about the material that is being given to the CNN other than the geometric configuration of particles. We don't know whether interactions are pairwise or more complicated, we don't know how the material operates, other than that we have been able to train the CNN on labelled materials, where the labelling could be based on some macroscopic rather than microscopic property. And yet, the CNN is still able to take just this raw structure information, no dynamics, and still tell the difference between a glass and a liquid. So it is able to classify these materials with high accuracy based on nothing but structural atomic configurations. This is interesting.
- (9) So, based on these above steps, I will first start by running more training to improve green dot accuracy.

5.11. 2/27/2018.

- (1) Logging results now that should have been logged a while ago. This is the training results for data_liquid140.

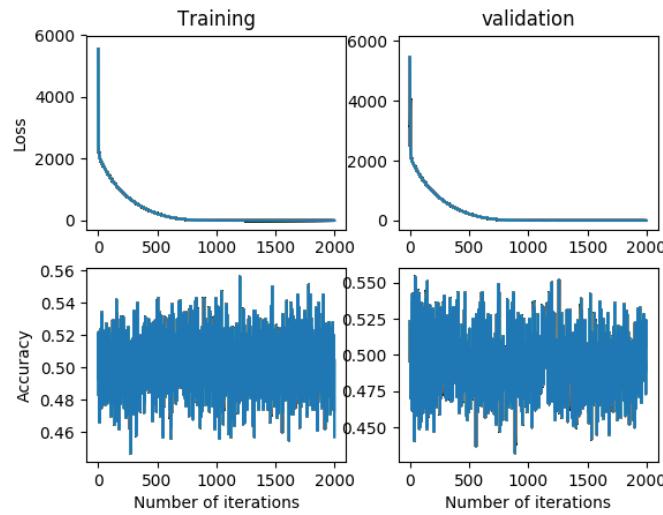


FIGURE 127. Version 1 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

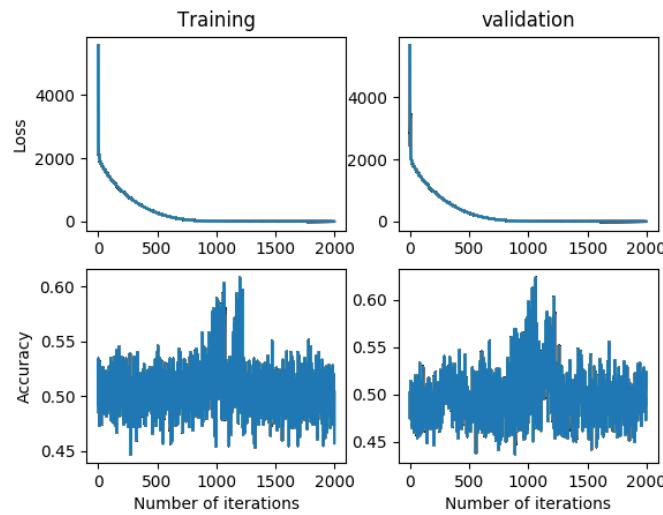


FIGURE 128. Version 2 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

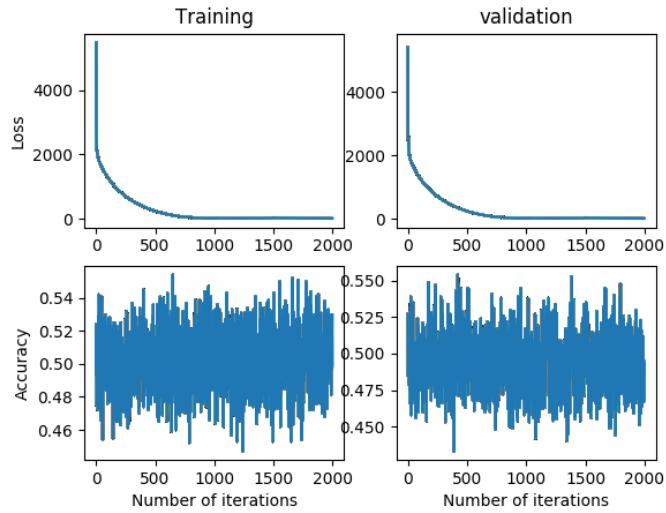


FIGURE 129. Version 3 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

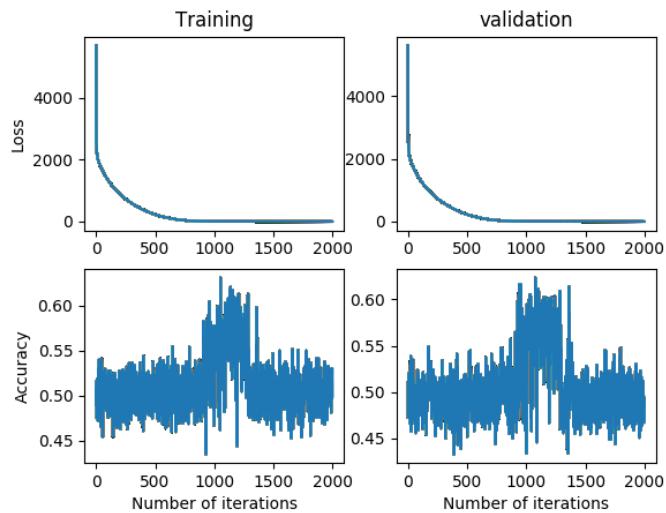


FIGURE 130. Version 4 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

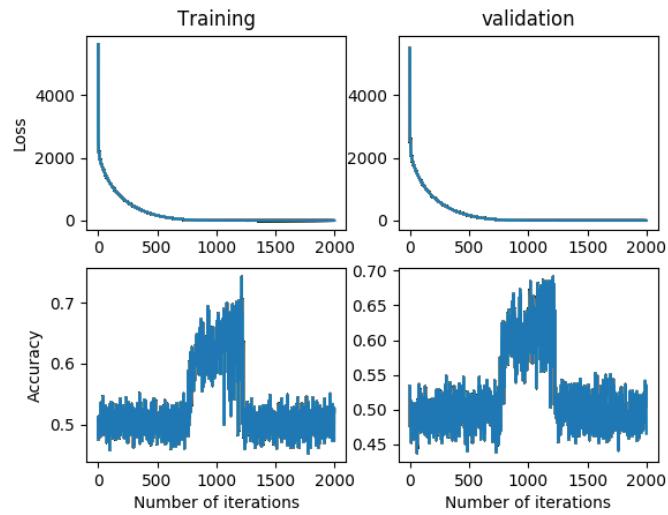


FIGURE 131. Version 5 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

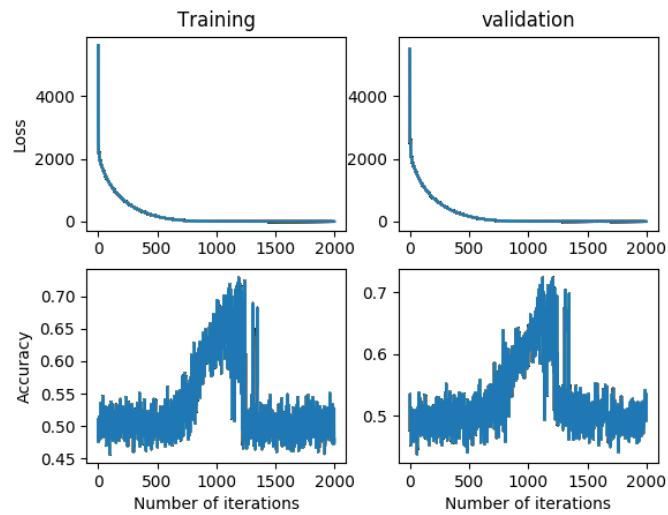


FIGURE 132. Version 6 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

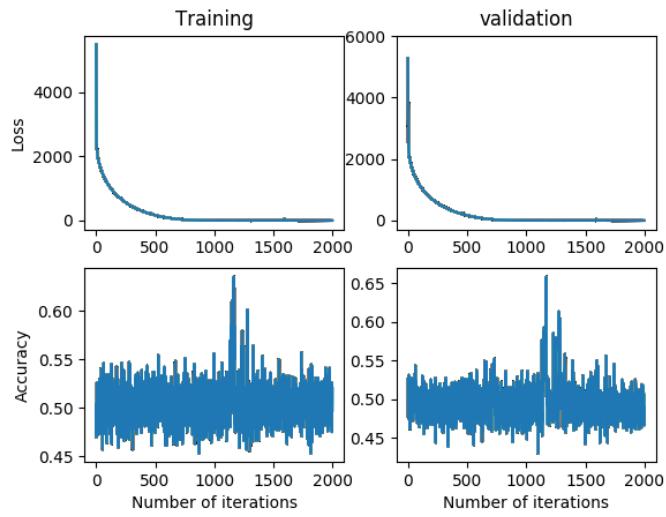


FIGURE 133. Version 7 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

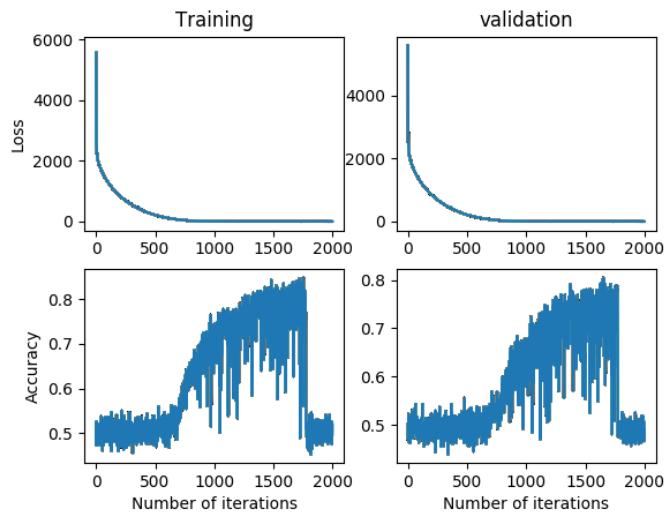


FIGURE 134. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

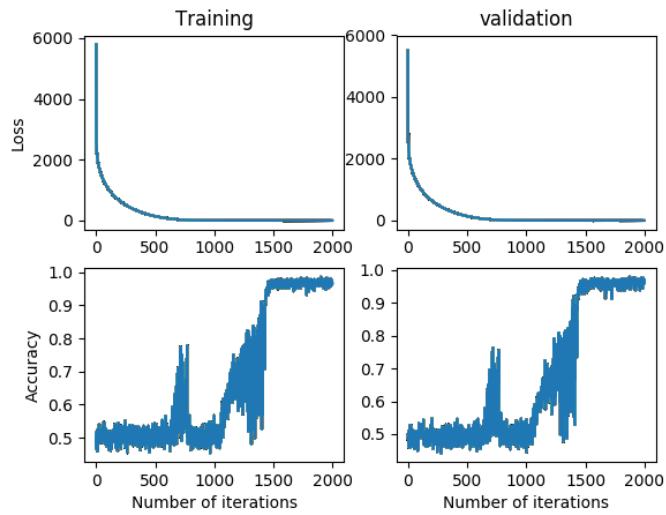


FIGURE 135. Version 9 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

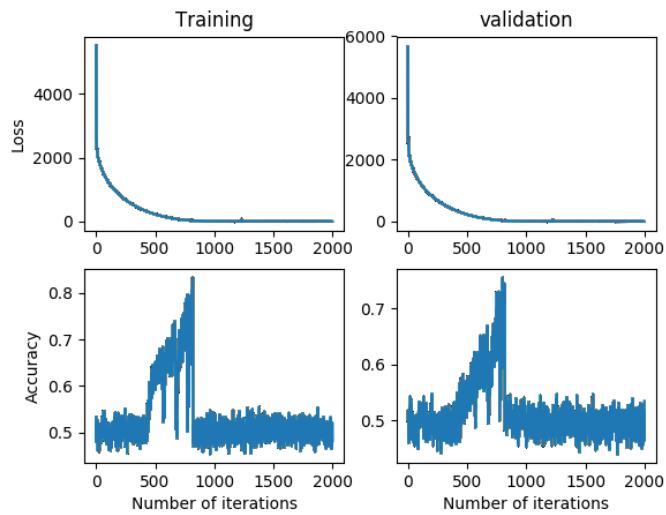


FIGURE 136. Version 10 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

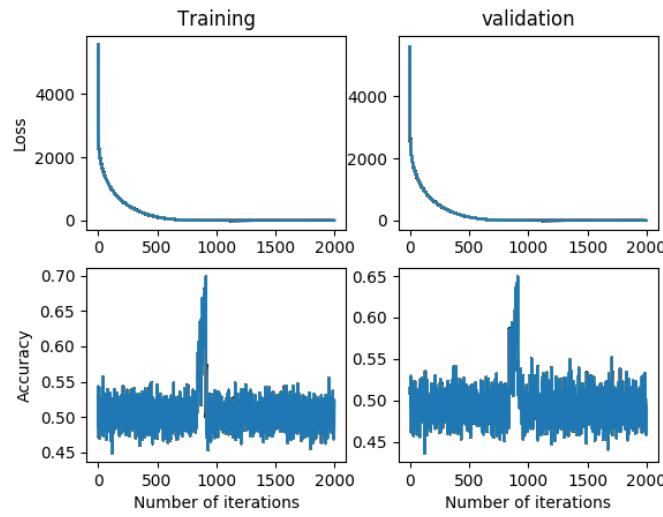


FIGURE 137. Version 11 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

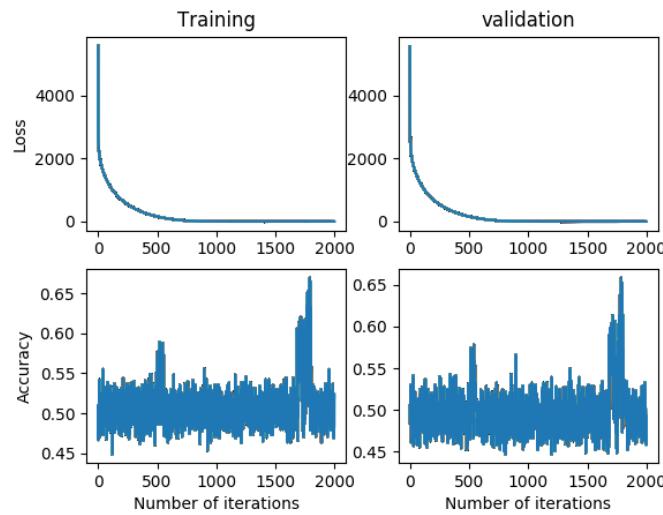


FIGURE 138. Version 12 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

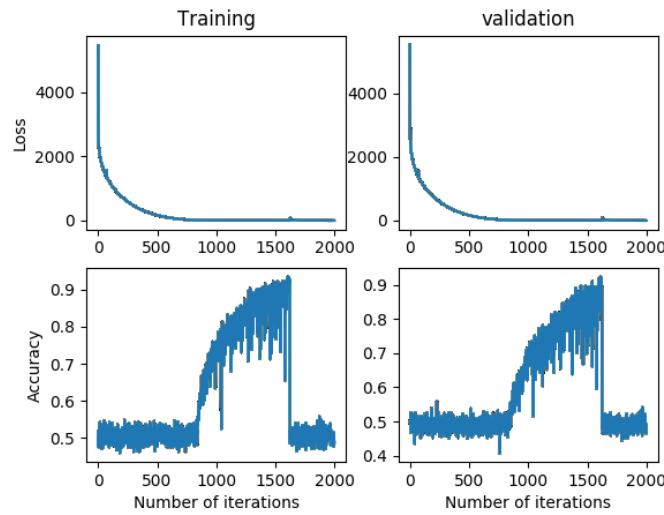


FIGURE 139. Version 13 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 130, L2 beta = 0.01, Dropout = 0.5, data aug = true

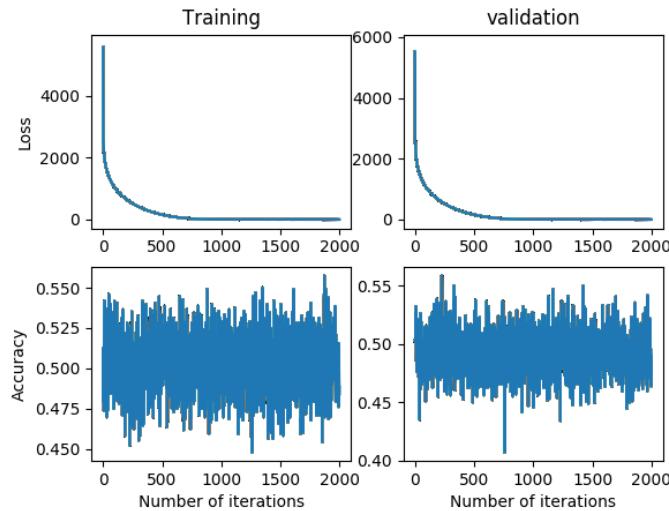


FIGURE 140. Version 14 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 130, L2 beta = 0.01, Dropout = 0.5, data aug = true

- (2) Version 9 Step 1 has a final validation accuracy of 96.2847% and test accuracy of 96.3194%.
- (3) Saving Version 9 Step 1 model. Deleting other information.
- (4) Now I want to find the true average glass transition temperature for the simulation data that we have. What I will do is construct an average cooling.dat plot, i.e. a plot that averages the data from all 10,000 cooling.dat files that we have.

Then, I will estimate by eye (or using some slightly more quantitative method for sake of argument) what the true average glass transition temperature is. We anticipate that it is higher than 0.21.

5.12. 3/6/2018.

- (1) Wrote a program called plot_average_cooling data, which I moved to the 2017/N2e7 directory. It collects the data from all 10,000 cooling.dat files and averages them across each temperature, so that we obtain an average cooling curve that is representative of the simulations:

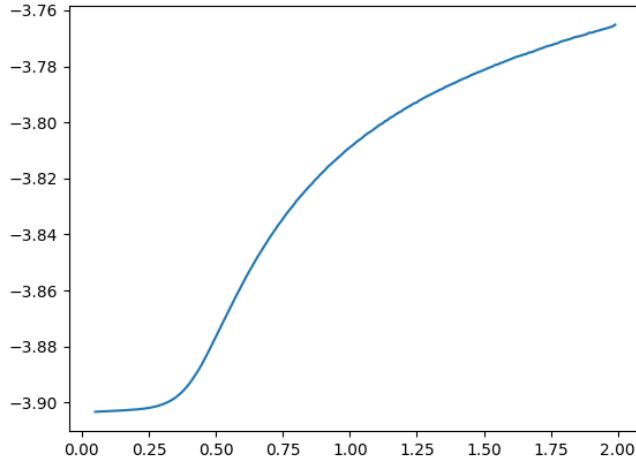


FIGURE 141. Averaged cooling.dat data

- (2) Estimated the kink in inherent structure energy by finding the intersection of two lines, using points (0.05, -3.9033) and (0.245, -3.902) for the glass data and points (0.44, -3.8876) and (0.557, -3.8654) for the liquid regime. The result is a glass transition at around 0.37. Removing the data_12steps result, as this is actually stepping over the new glass transition, here is a new plot of our results:

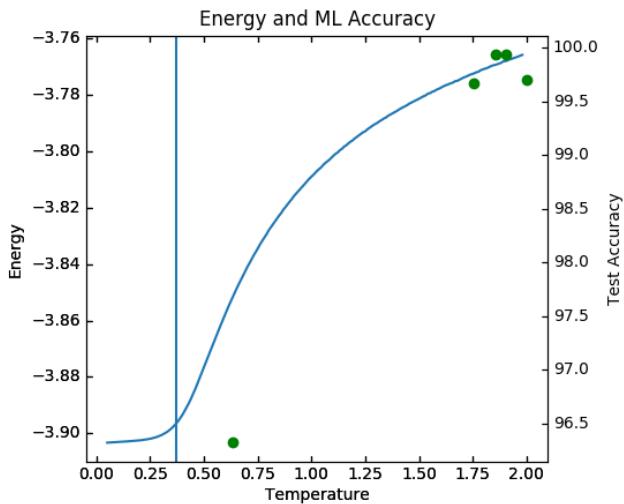


FIGURE 142. Updated results plot

- (3) Switching to data from data_liquid155, which corresponds to $T = 0.479$.
- (4) Now running the same versions as before.
- (5) PROBLEM: How is this any different from distinguishing between two liquids at different energies/temperatures? SO we should try to classify between two liquids!!! Added to the to do list above.

5.13. 3/7/2018.

- (1) Read sections of papers (now on Box) on MPNN, meeting with Shubhendu tomorrow.
- (2) Still running training on 155.
- (3) Looked into simulating new liquids that will be at the same inherent structure energy as the glass, but ran into issues with the batch scripts as written.

5.14. 3/8/2017.

- (1) New idea: identify the phase transition temperature by stepping across into the glass regime until the classification accuracy gets really low.
- (2) Reading <https://arxiv.org/pdf/1802.09876.pdf>. Suggests that networks with one layer are not only sufficient, but more interpretable.
- (3) Analysis of activation of neurons. Previously, for Ising model, only 1 layer with 3 neurons needed. Plotting weight matrices of each neuron. Look at activations as a function of five-fold symmetry??

5.15. 3/9/2018.

- (1) We will be implementing MPNN using: https://github.com/priba/nmp_qc
- (2) First instruction is to install requirements using pip install -r requirements.txt
- (3) On Midway 2, /home/swansonk1 directory, error in installing torch: "PyTorch does not currently provide packages for PyPI"
- (4) For the time being, I will try this on my laptop instead. Actually, my desktop, which has cuda installed.

- (5) Again, pip install -r requirements.txt failed, so for torch, i.e. PyTorch, I used the command conda install pytorch torchvision -c pytorch from pytorch.org.
- (6) Successfully installed all packages in the requirements.txt file.
- (7) Downloaded the qm9 dataset.
- (8) Following Sec. 6 on <https://arxiv.org/pdf/1704.01212.pdf>, the input feature for our particles will be a one-hot encoding of either particle A or particle B. The adjacency matrix will be a raw distance feature matrix, where the entries are one-dimensional and indicate the euclidean distance between the pair of atoms.
- (9) Tried running python main.py, but saying no module rdkit.
- (10) So, went to <http://www.rdkit.org/docs/Install.html> and conda installed. Now I can create an environment as described. However, torch no longer works inside the environment. Need to investigate how environments work, then.
- (11) Opened the rdkit environment. Then re-installed PyTorch as above.
- (12) No module named networkx. So, pip install networkx within the rdkit environment. Same thing for joblib and tensorboard_logger.
- (13) Tried running main.py. Got error, 'Graph' method has no attribute 'nodes_iter'. Found this solution: <https://stackoverflow.com/questions/33734836/graph-object-has-no-attribute-nodes-iter-in-networkx-module-python>. Trying it now by replacing with nodes() in datasets/utils.py, specifically on line 28 in the qm9_nodes function.
- (14) Also removing edges_iter on line 54. Issue seems to be that the new version of networkx has mismatches.
- (15) Having some more issues with GPU compatibility. PyTorch says my GPU is too old. Also, there are more inscrutable errors. So, I am going to uninstall networkx and install the version that should be compatible with this model, networkx version 1.11. Used pip install networkx==1.11.
- (16) New error with .expand in line 174 of m_mpnn. Stack suggests that this has to do with bug in new version of PyTorch: <https://github.com/pytorch/pytorch/issues/2491>. So, like networkx, I am going to find the versin of PyTorch that was latest when the last commit for this program was made.
- (17) Found that the most recent version of PyTorch prior to the last commit on MPNN May 29, 2017, was the version v0.1.12: <https://github.com/pytorch/pytorch/releases>, on May 2, 2017. So, I am installing this version using conda install pytorch=0.1.12 -c soumith. This lets me download without cuda: <http://pytorch.org/previous-versions/>. Hopefully this will get rid of the GPU problem? If not, we'll turn to Midway.
- (18) Ok, this works! I was just able to run, in the demos folder, demo_qm9_mpnn.py. Now, I am going to go through this and try to understand what's going on. If necessary, I will add comments to this code file.
- (19) The data is transformed into special data objects via the Qm9 class in the datasets folder. Each object of this class has four main attributes: the graph, the nodes, the edges, and the targets. The graph is the adjacency matrix that describes connections between atoms in the molecule. It is real symmetric and is a numpy matrix. The nodes are given as a list of features for each particle. The edges are described by a dictionary, where (1, 2): [distance, one-hot enconding of bonding] is the format. Finally, the target properties are given as a list as well.

- (20) We would probably choose our adjacency graph to take into account only a certain number of nearest neighbors as "bonded." Then, we would just need to make sure that the graph is **connected**. That is the only requirement of the scheme, which can then later be optimized.

5.16. 3/13/2018.

- (1) Modified the coords_config.py file so that it outputs atom type, x-coordinate, and y-coordinate instead of x, y, z.
- (2) Produced a sample file from the file N2e7/0 at time 100,000
- (3) Wrote a script adjacency_matrix.py to compute the adjacency graph and relevant information for MPNN input. It works generally, but the loops appear to be inefficient. Next time, I will have to look into using matrices, as well as ensuring that the matrix is **connected**.

5.17. 3/14/2018.

- (1) Added some functionality to adjacency_matrix. Found functions that could be used for matrix manipulations, namely euclidean_distances from sklearn and arr.argsort() for numpy arrays. Also found that you can load txt into numpy array using np.loadtxt().

5.18. 3/15/2018.

- (1) Created graph_reader.py, which returns graph information from an input glass/liquid data file in less than 7 seconds. Used matrix operations rather than only loops to increase computational efficiency.
- (2) Tomorrow, I need to go to RCC to check how to make the MPNN code work on Midway, and I also need to implement a check for connected graphs.
- (3) Downloading results from data_liquid155.

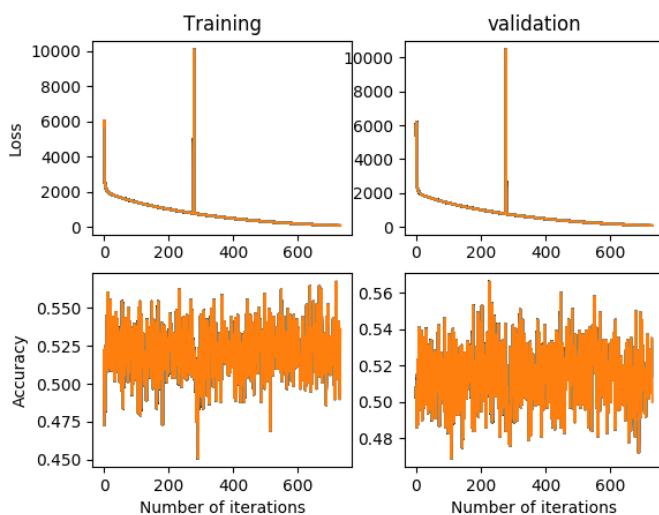


FIGURE 143. Version 1 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

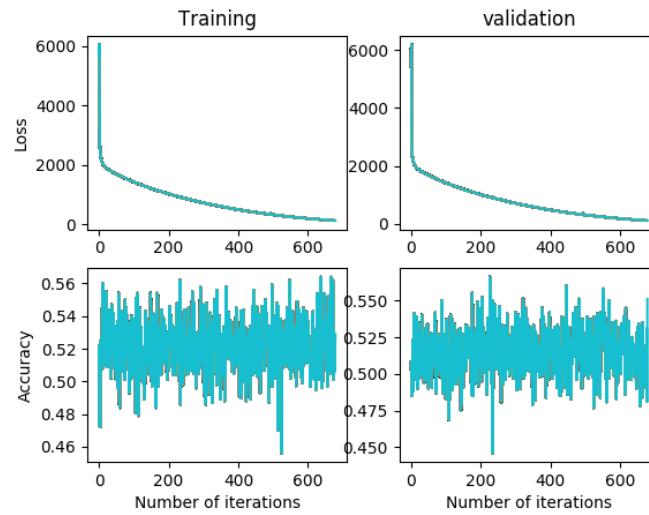


FIGURE 144. Version 2 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 10, L2 beta = 0.01, Dropout = 0.5, data aug = true

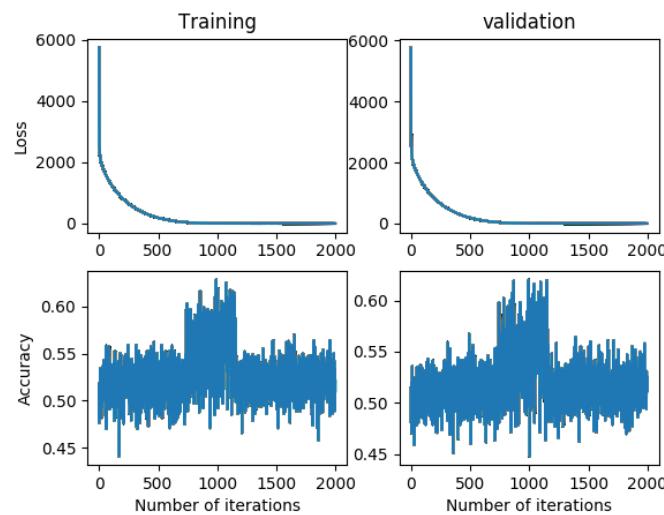


FIGURE 145. Version 3 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

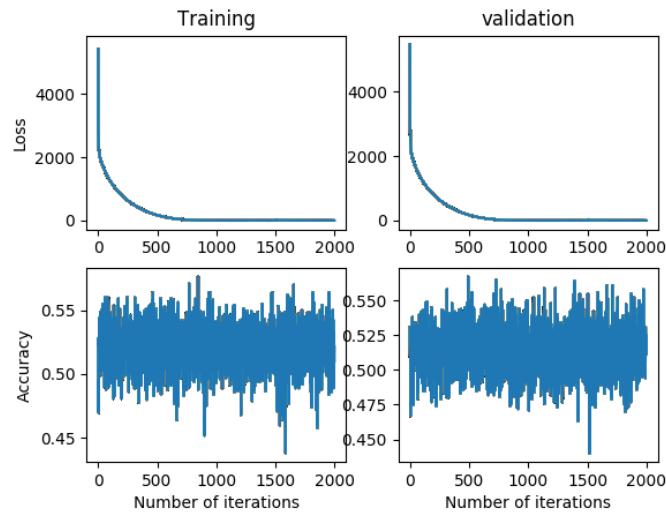


FIGURE 146. Version 4 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 20, L2 beta = 0.01, Dropout = 0.5, data aug = true

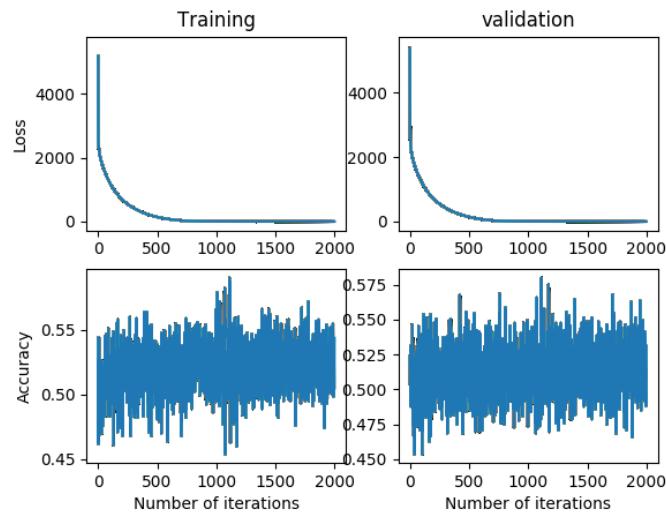


FIGURE 147. Version 5 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

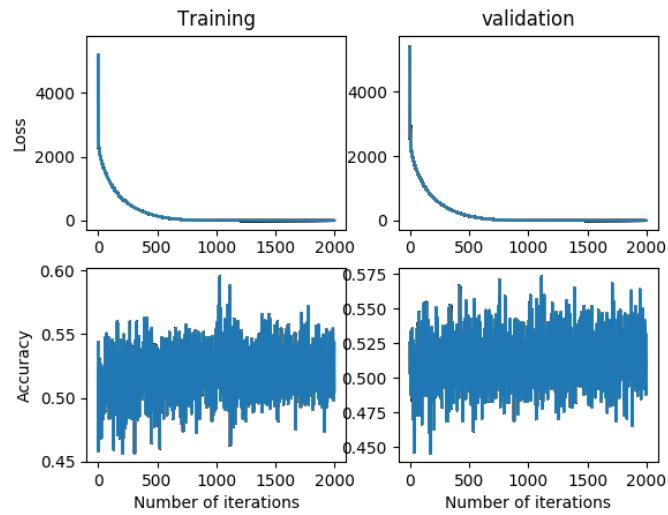


FIGURE 148. Version 6 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 30, L2 beta = 0.01, Dropout = 0.5, data aug = true

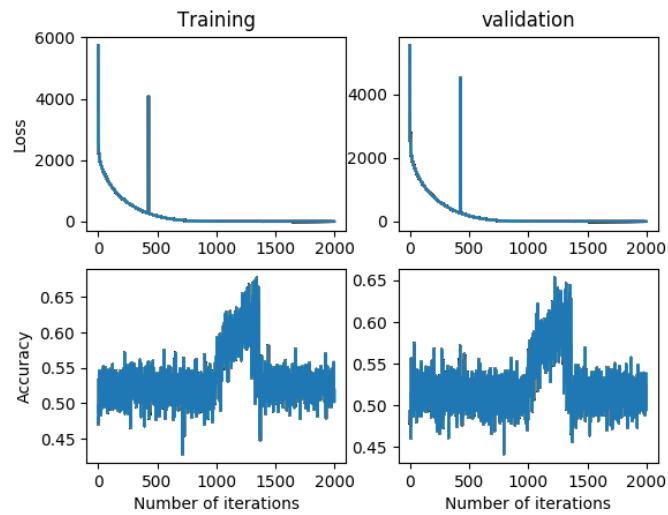


FIGURE 149. Version 7 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

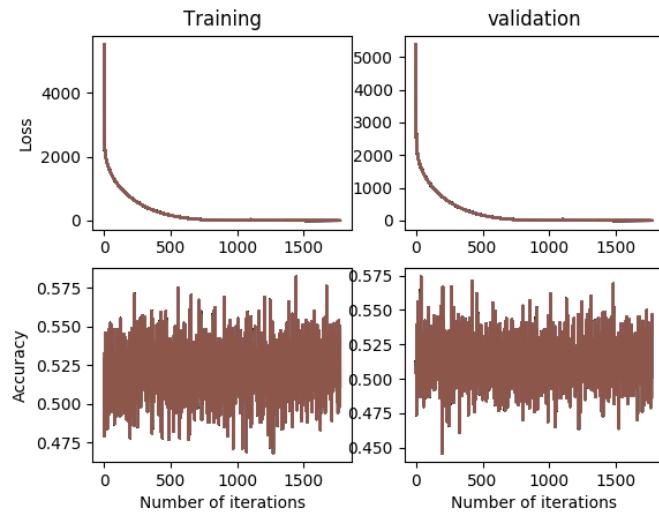


FIGURE 150. Version 8 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 50, L2 beta = 0.01, Dropout = 0.5, data aug = true

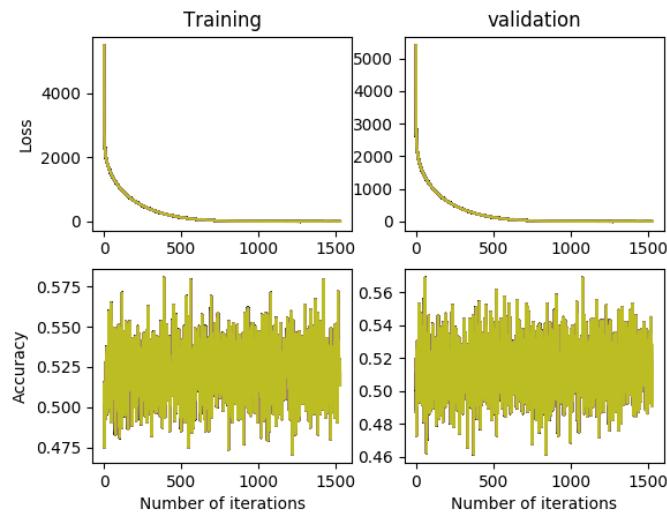


FIGURE 151. Version 9 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

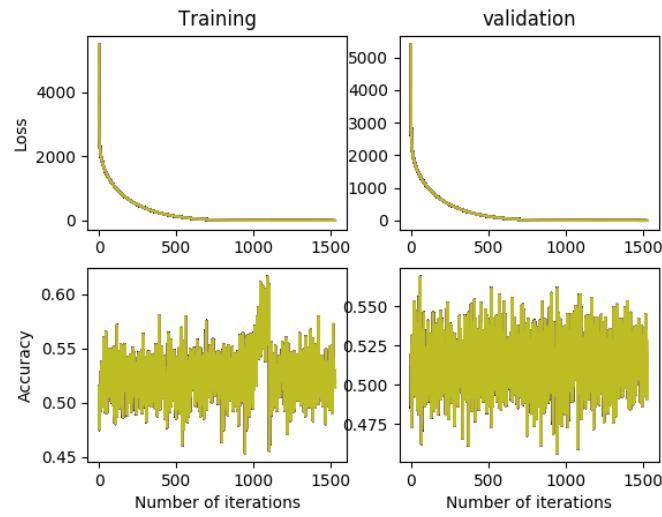


FIGURE 152. Version 10 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 80, L2 beta = 0.01, Dropout = 0.5, data aug = true

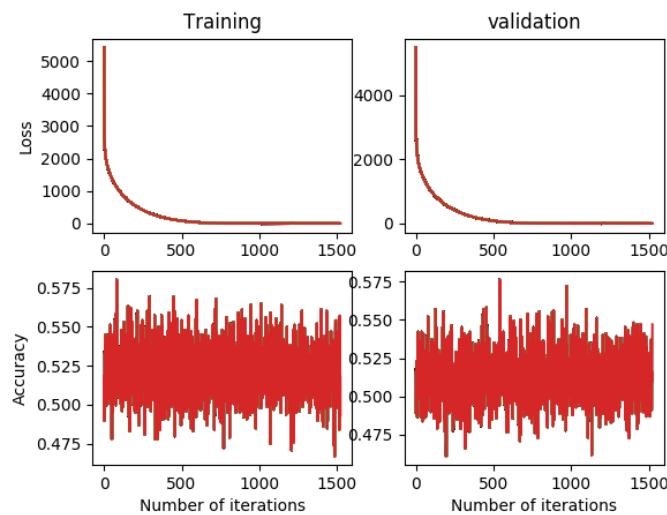


FIGURE 153. Version 11 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

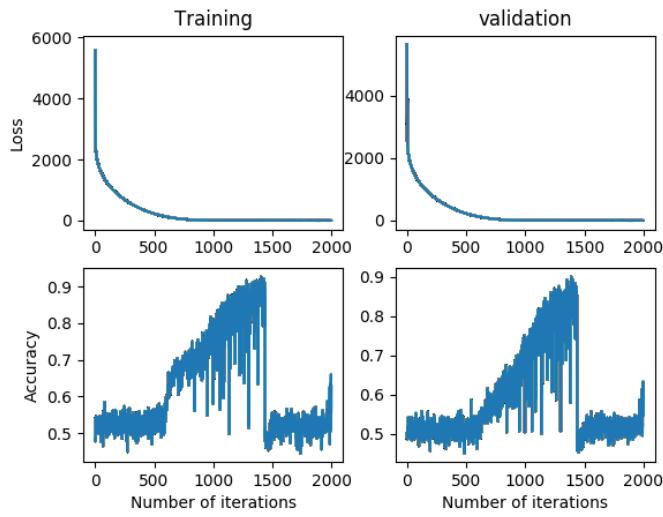
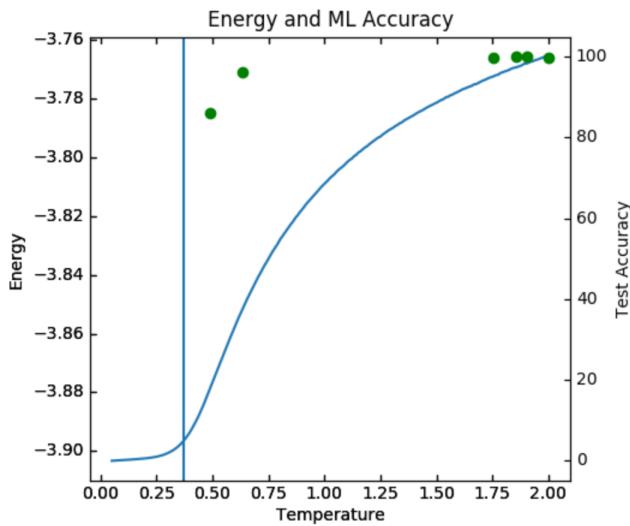


FIGURE 154. Version 12 step 1: Initial Learning Rate = 1e-3, Final Learning Rate = 1e-4, Eta threshold = 0.90, Batch Size = 100, L2 beta = 0.01, Dropout = 0.5, data aug = true

- (4) Version 12 Step 1 has a final validation accuracy of 86.9408% and test accuracy of 86.1111%. These could have probably been improved using a lower learning rate threshold.
- (5) Saving Version 12 Step 1 model. Deleting other information.
- (6) Updated accuracies:



5.19. 3/21/2018.

- (1) The functions that manipulate graph data in `demo_qm9_mpnn.py`, which is the demo I am currently working off of, use classes and objects. So, I wrote `graph_reader_class.py` based on `graph_reader.py`, which defines a GlassyData

class that can be used to construct the graph information (adjacency matrix etc.) for a text file with glass/liquid coordinate information.

- (2) I saved this file on the workstation in the nmp_qc folder. I then added a line in demo_qm9_mpnn.py that says from graph_reader_class import GlassyData.
- (3) I saved a single file example in a folder called glass, within the data folder, whose filename is glass.00000.txt.
- (4) I then used train_ids = [files[i] for i in idx] and data_train = GlassyData(root, train_ids, 7) to try printing data_train[0][0][0] etc..
- (5) In order for this to work, I had to pip install sklearn within the rdkit environment.
- (6) Then, it worked!
- (7) Next, I need to move the data to the local machine, then write a demo script that is strictly for glass/liquid data.

5.20. 3/26/2018.

- (1) Added "len" attribute to graph_reader_class.py. Moved all data to local workstation, and wrote demo script that is strictly for glass/liquid data.
- (2) No issues yet on local machine.

5.21. 3/27/2018.

- (1) Added more functionality to demo_glasses_mpnn. Need to reduce dataset size to 500 in each class and move this crap to Midway so we can get better computing power/GPUs!

5.22. 3/28/2018.

- (1) Installed pytorch version 0.1.12 onto Midway 2 home directory. Found in the data file, data_coords_endpoints that there is an empty file called "loading".
- (2) Copied data_coords_endpoints to home, removed "loading" file, and truncated it so that there are 600 glass files and 600 liquid files.
- (3) There appears to be an accuracy function in datasets/utils.py
- (4) module load Anaconda3, then source activate my-rdkit-env
- (5) Went into project directory. Did the usual rdkit command. Then, removed networkx from its location based on teh error message that was being given. Then pip installed using –user flag networkx version 1.11 as well as joblib and tensorboard_logger. Also downloaded appropriate version of PyTorch as usual. Seems to work.
- (6) Had to pip install sklearn –user in /project/depablo/kswanson
- (7) At last, it appears to be working!
- (8) Problem on workstation: once I add the line, output = model(g, h, e), the process gets killed. So, perhaps I've reached the limit for what my local workstation can do?
- (9) Turning to Midway, here are the following outcomes: if I go onto depablo-gpu, then within the train function, I get an error that starts like THCCudaCheck FAIL file=/py/conda-bld/pytorch_1493672080115/work/torch/lib/THC/THCCachingHostAllocator.cpp line 258 error=11 : invalid
- (10) When I try to run on the log-in node, I get Assertion Error: Found no NVIDIA driver on your system. Please check that you have an NVIDIA GPU and installed a driver from <http://www.nvidia.com/Download/index.aspx>.

- (11) I am hoping that if I can find the appropriate way to use torch/PyTorch on Midway, then I will be able to solve this. So, I need to go to RCC next time I work on this to resolve this problem.
- (12) ALSO, next time, I OUGHT TO WORK ON MAKING NEW LIQUID SIMULATIONS!!

5.23. 3/30/2018.

- (1) Finally fixed the issue with new liquid simulations. Can't start seq 0 10, must start seq 1 10, because otherwise the program just sleeps. So, I am now running a cooling rate of N2e6, 10 examples. Once they are done, I will look at their average cooling inherent structure energy and determine if I have achieved a lower transition temperature, etc., from which I can determine exactly what data I want to generate and collect.

6. 4/2/2018

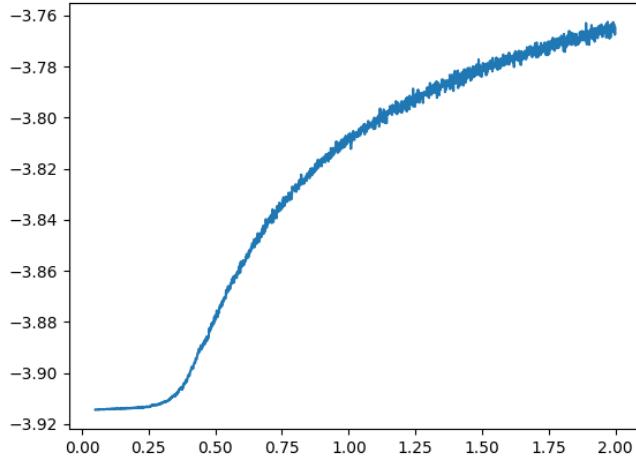
- (1) Spoke to Igor at RCC, PyTorch not available on Midway. Sent him an email to work on it.

7. 4/3/2018

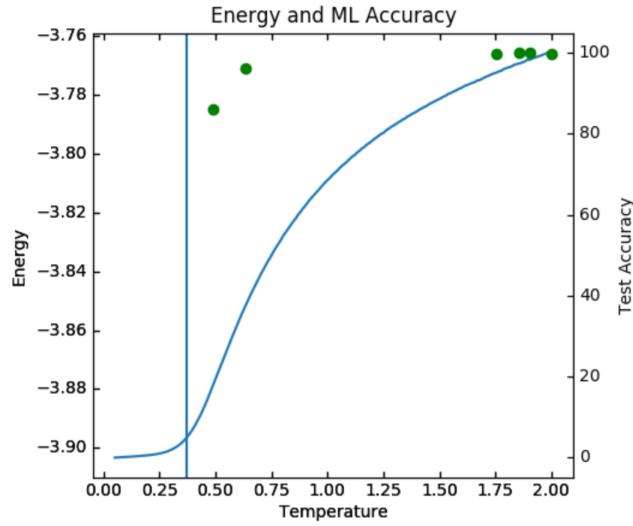
- (1) A decrease of the cooling rate leads to a decrease of the glass transition T_g . 2e8 corresponds to a lower cooling rate, 2e6 to a higher one. So, 2e8 must be related to the number of timesteps necessary to get to $T = 0.05$, hence more steps, lower rate. So, I am now running 20 examples each of rate 2e8, 2e9, and 2e10 to check whether this measurably changes the cooling rate to be below the glass transition.
- (2) Now I'm going back to MPNN to see if I can get this to work on Midway.
- (3) In an email, Igor from RCC says: try sinteractive -p gpu2 -gres=gpu:1, module load cuda/8.0, module load Anaconda/5.0.1, check "which python" and it should show /software/Anaconda3-5.0.1-e17-x86_64/bin/python. If not, module purge, repeat second two steps.
- (4) Logged on. Ran demo_glass_mpnn.py, and it gets inside the train function, prints one or two things, then screen connection is terminated.
- (5) I am going to test this with the qm9 dataset, just to see if it can do normal training not on my glass/liquid shit. So, let's try that.

8. 4/9/2018

- (1) The N2e8 data finished running. It computed 20 examples, and using the average cooling data python script, I computed the average cooling curve of these 20 simulations:



Compared to the N2e7 cooling curve,



this curve does not look markedly different. In fact, using a rough approximation by looking at the intersection of linear approximations to the glassy and liquid regimes, the glass transition is around 0.4, barely different from 0.37.

- (2) So, instead I will try a faster cooling rate, such as N2e5, and see if I can shift the curve to the right. There will probably be issues with the cooling.dat file, but we will first just try to run the simulation to see what happens and then diagnose the error.
- (3) Try to distinguish between two glasses in the glass regime, try to train on aggregated liquid data....

- (4) Now, working on MPNN on Midway. Was able to go into the data folder in nmp_qc and download the qm9 dataset using the command "python download.py qm9". Now, let's try one of the standard demos on this dataset to see if that at least works on Midway.

8.1. 4/10/2018.

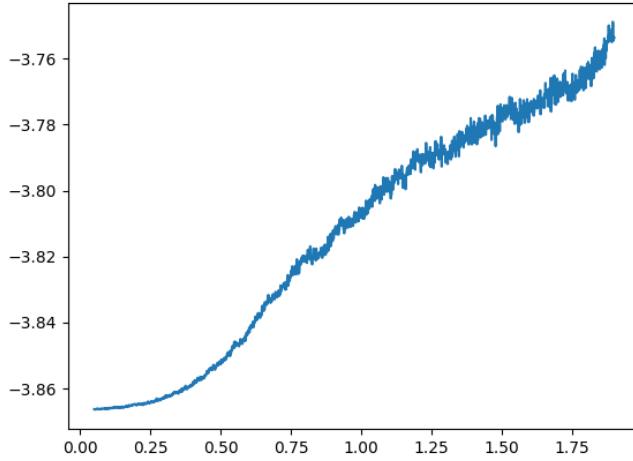
- (1) Downloaded qm9 dataset successfully. Had to pip install wget in environment. Also, now able to do "python demo_qm9_mpnn.py" and it appears to be running completely fine.
- (2) Running this program takes about 4775MiB/11439MiB on midway2-gpu01
- (3) Error occurs when running the enumerate(train loader) line...
- (4) Checking capabilities of demo_glass_mpnn.py versus demo_qm9_mpnn.py. I want to make sure that they are the same at least up until data_train etc. by checking exactly what data is being stored and showing that the data types and structures are the same.
- (5) For a single datapoint, data_qm9_mpnn.py prints a 'numpy.matrixlib.defmatrix.matrix', 'list', 'dict', and 'list'.
- (6) For a single datapoint, data_glasses_mpnn.py prints a 'numpy.ndarray', 'list', 'dict', and 'numpy.ndarray'.
- (7) Ok, well here is a point of dissimilarity, so we should make sure to enforce perfect similarity at this point before moving on.
- (8) Went into the file graph_reader_class.py and added np.matrix for the adjacency matrix and list() for the target. Now, we also get 'numpy.matrixlib.defmatrix.matrix', 'list', 'dict', and 'list' for the output. Dissimilarity resolved.
- (9) Screen still closes when I try to enumerate(). Fuck.

8.2. 4/11/2018.

- (1) Making a new directory, 2e5-working-directory, to run N2e5 simulations. They require having a seed with different variables set so enough cooling.dat information is printed, i.e. in in.analyze-glass and in.make glass set variable freq equal some number less than variable tend.

8.3. 4/12/2018.

- (1) Here is the cooling curve for the 2e5 data:



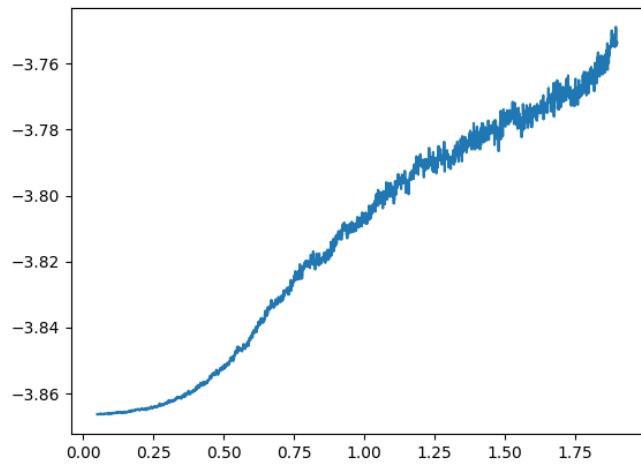
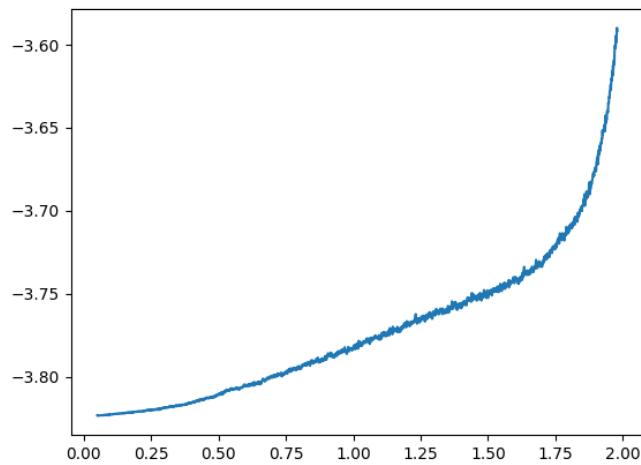
- (2) Something looks weird, so let's run examples for 2e6 and also for 2e7 just to have a complete comparison with what I'm doing with my current files. Then, we can play around to get things better.
- (3) Created 2e6 folder and set the freq variable to 1000. Running 20 simulations.
- (4) Created something similar for 2e7 folder and set freq variable to 10,000. Running 10 simulations.
- (5) Finished these, need to download the images. Change plot so it cuts off initial drop to make plot more reasonable. Running 2e4. Looks like kink is less distinct.

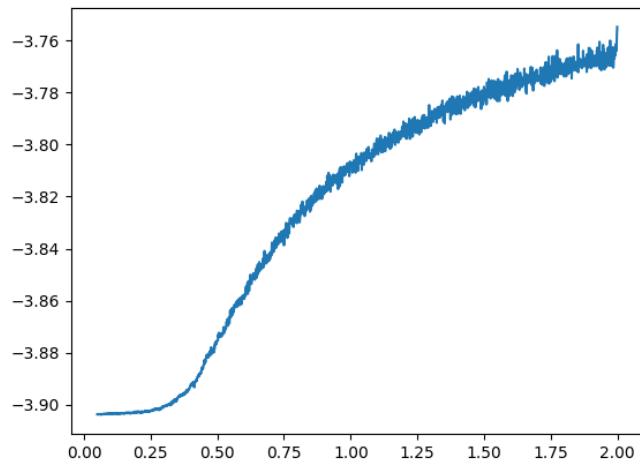
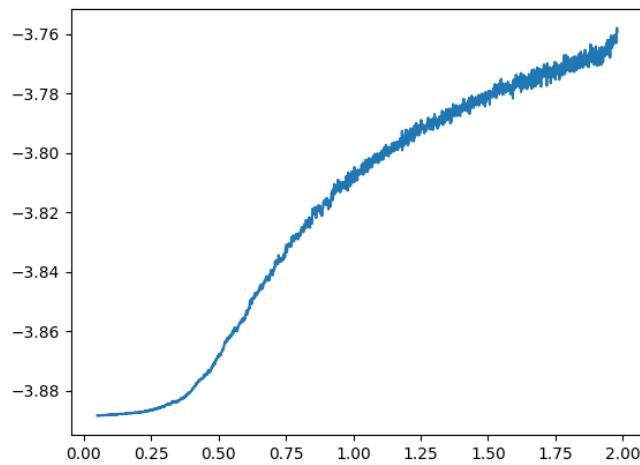
8.4. 4/13/2018.

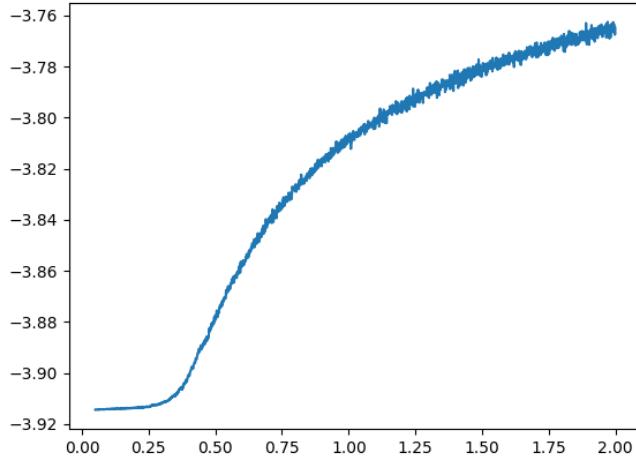
- (1) Beginning collection of images for glass data_164 $T = 0.401$, data_174 $T = 0.3035$, data_184 $T = 0.206$, data_194 $T = 0.1085$. We will use the best model from data_155 to see how that model does as we cross the glass transition temperature. In this sense, it's kind of an artificial dataset, since the liquids are not actually liquids but are glasses. WE SHOULD ALSO try training the model on the ENTIRE liquid curve. Maybe that's what I can set up on Monday.

8.5. 4/16/2018.

- (1) We now have representative cooling curves for 2e4, 2e5, 2e6, 2e7, and 2e8, respectively:







- (2) Let's throw out 2e4, since the curve looks pretty different from the others, and start with 2e5.
- (3) Wrote a python script, Tg_calculation.py, which I used to compute approximate Tg values:

$$\begin{aligned}
 2e5 & \quad T_g \approx 0.385279666197 \\
 2e6 & \quad T_g \approx 0.374750431427 \\
 2e7 & \quad T_g \approx 0.340983034936 \\
 2e8 & \quad T_g \approx 0.290198100183
 \end{aligned} \tag{8.1}$$

For the glass points that determine a line in the glass regime, I used temperatures 0.05 and close to 0.20015, and for the points in the liquid regime I used temperatures close to 0.50045 and 0.80075. Because 2e5 and 2e8 appear to have a different enough T_g , I will now generate 10,000 simulations for both 2e5 and 2e8. I will re-compute the glass transition temperatures after this, to confirm their values. I will then take the midpoint temperature, around $T = 0.338$, and take configurations from 2e5 that will correspond to glass and configurations from 2e8 that will correspond to liquid. I will then train the network to see if I can distinguish between them.

- (4) Created folders 2e8-folder and 2e5-folder. For 2e8, changing the in.make-glass and in.analyze-glass in seed to have variable freq equal 1000000. For 2e5, changing to 1000. Running 500 simulations for each. Also, removing the sleep command, as it is unnecessary automation.
- (5) Tried running main_glassliquid.sbatch to test the performance of the best model from data_liquid155 on other datapoints, but getting error, "No such file or directory." Not sure what is going on. Emailed Igor to figure it out.
- (6) Tested the best version from data_liquid155 on other datasets. Accuracy at least 90% for anything lower than 155, then gradually decreases down to 80, 70, 60, finally 45% for data_liquid194. Not a sharp cutoff though.

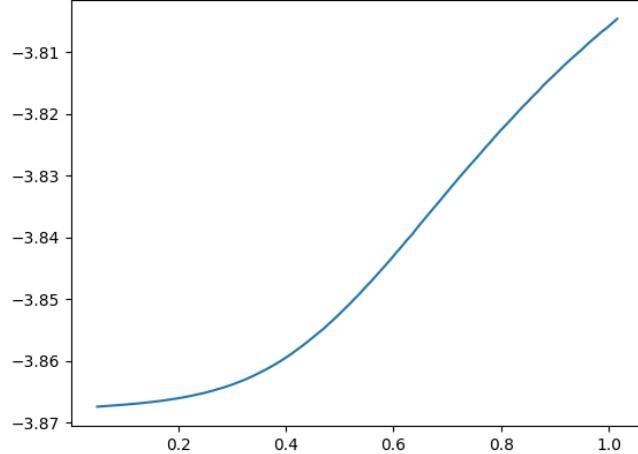
- (7) Inherent structure energy of -3.66 will be a glass for 2e5 but a liquid for 2e8. That is what we want to compare. We will start first by evaluating the data_liquid155 model on this. We would also try training across the curve. And then training directly on 2e5 versus 2e8.

9. 4/18/2018

- (1) Evaluate current model on a different cooling curve, say 2e5 or 2e6.
- (2) Construct artificial geometries, tuning separation between particles, to feed into trained network.
- (3) Talk to Daniel about his configurations.
- (4) Compare 2e5 and 2e8 when it's done.
- (5) Choose a smaller subsection of the 4320 to do MPNN with....LOL WHY DIDN'T I DO THIS BEFORE AHHAHA

10. 4/23/2018

- (1) Looking at the cooling curves again, the glass at energy -3.866 for cooling rate 2e5 is still a liquid for cooling rate 2e7, for which we already have data. So, we already have FULL data to do this, haha great.
- (2) First, let's construct a complete average cooling curve for cooling rate 2e5:



- (3) Now, let's recalculate the estimated glass transition temperature. We get:

$$2e5 \quad T_g \approx 0.380974669843 \quad (10.1)$$

which is similar to previous value.

- (4) Now, we will evaluate the current model on 2e5. We will construct a dataset that is composed of glass around $T = 0.05$ and liquid around $T = 0.8$. Specifically, we will choose $T = 0.05$, which corresponds to energy -3.867438353 and $T = 0.80075$, which corresponds to energy -3.82253291198. Generation of 10,000 images of each is currently running.

- (5) We will also need data from 2e7 that corresponds to energy -3.867438353. Based on the average cooling curve for 2e7 data, $T = 0.54725$ (i.e. simulation step 14800000) has an average energy of -3.86698027516. So, we will compute 10,000 images at this temperature. This is running now.

11. 4/24/2018

- (1) The image processing for 2e5 is complete. We now have 10,000 images of glass and liquid, where glass is at $T = 0.05$ and liquid is at $T = 0.80075$. We will now evaluate the version 12 best model at 155 on this and see what accuracy it obtains.
- (2) Moving the folder containing 2e5 images, called data_liquid_2e5123, to /project/depablo/kswanson/ml-learning-glasses.
- (3) Changing metadata to be from that folder.
- (4) Evaluating, we get validation accuracy of 89.3311% and test accuracy of 89.1304%. This is good news!
- (5) Now, let's move on to glass versus supercooled liquid.
- (6) Constructed a folder called data_supercooled, which contains the image data from $T = 0.05$ for cooling rate 2e5, labelled glass, and the image data from $T = 0.54725$ for cooling rate 2e7, labelled liquid.
- (7) Switching metadata to be from data_supercooled.
- (8) Able to classify with 87.2148% validation and 85.2013% test! Excellent.

12. 4/26/2018

- (1) We are now going to return to MPNN to crush it. Start by reducing our dataset so that each configuration file only consists of 20 atoms. Let's see if our algorithm can at least process that.
- (2) Copy coords_config.py to coords_config_restricted.py. Change line 60 so that we only range over 20 atoms. Let's try to use this now to construct a new dataset. Modifying make_coords_kirk.sh on Midway 2 in the 2017 folder so that it executes the new restricted coords file.
- (3) Generated this dataset with about 200 total glass + liquid samples, each with 20 atoms. Let's try our MPNN code on this shit.
- (4) Ok, gpu2 is busy. In the meantime...
- (5) Removed a lot of files from Download folder. Followed instructions to get my-rdkit-env and associated packages on my laptop (desktop is, like, broken or something).
- (6) Downloaded the 20 atom data file from Midway. Modifying graph_reader_class.py.
- (7) Doesn't work, because no NVIDIA driver.

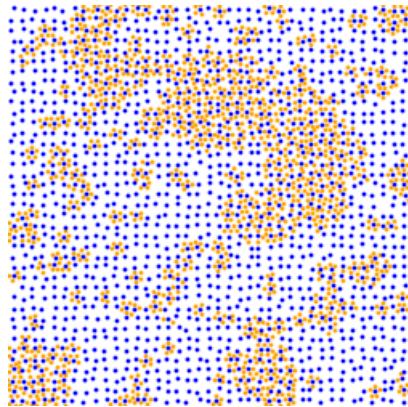
13. 4/30/2018

- (1) Took demo_glass_mpnn_working.py and graph_reader_class_classification.py and moved to Midway. On midway2, gpu2, do module load cuda/8.0, module load Anaconda3/5.0.1, then python filename –no-cuda. It works! Time to see how large of a graph we can process on Midway.

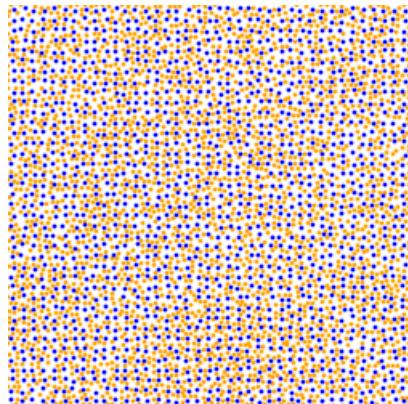
- (2) So, moving data_coords_endpoints_restricted100 to the data/glass folder. Have to change the graph reader class to support 100 sized data instead of 20.
- (3) Ok, getting 'bus error' or 'screen terminated'. Let's try backing it off to 25, and incrementally move here to find the breaking point. Then we will check with RCC to see what the issue is, and move on from there.
- (4) System I. On Midway 2 gpu 2, we do: sinteractive –partition=gpu2 –gres=gpu:1. Then, we do module load cuda/8.0, module load Anaconda3/5.0.1, source activate my-rdkit-env. Then, we do python demo_glasses_mpnn_classification.py –batch-size=. So, we are using cuda. We are experimenting with different datasets. Each dataset has approximately 100 glass and 100 liquid configurations. The difference between the datasets is the number of atoms in each configuration. Also, we are using the change g, h, e, target = next(iter(train_loader)) instead of enumerate(train_loader), which appears to work better. So far, we have gotten up to: size 20 batch size 100, size 25 batch size 100, (batch sizes not really representative), 100 batch size 20, (we'll do batch size 20 from now on), 200 with batch size 20.
- (5) Noticed that 10624 MiB / 11438 MiB was the memory usage in this process. So, my guess is that a similar problem is occurring as before with TensorFlow. We need a command that will restrict memory usage on the GPU to allow growth. This is probably why I am getting kicked off/it is complaining about memory problems.
- (6) From the developer: i think measuring GPU memory usage via nvidia-smi might be misleading (and might be the issue). We use a caching allocator and a few tricks, so we can't use nvidia-smi's reporting. <https://discuss.pytorch.org/t/using-volatile-true-for-prediction-uses-more-gpu-memory-than-volatile-false/8330>
- (7) Let's test the volatile thingy?
- (8) Use multiple GPU nodes in parallel?

14. 5/15/2018

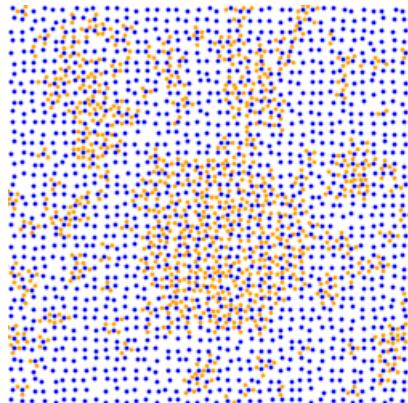
- (1) Was not writing stuff in the log for a while. So there will be some kind of discontinuity here.
- (2) The first thing we want to do is to generate 100 examples of the new, filled out, artificial five-fold symmetry images. The python script we are working from is called artificial-geometries.py. This is running now.
- (3) We are also running something similar, but for three-fold symmetry. Once these are done, we will render the images and test.
- (4) Finally, to test another hypothesis we have, we will construct images that have no symmetry or clustering at all. We will simply leave that part out. We will create a new python file called artificial-geometries-liquid.py for this task. This is running now as well on Midway.
- (5) This is what the clusters of 5-fold symmetry look like:

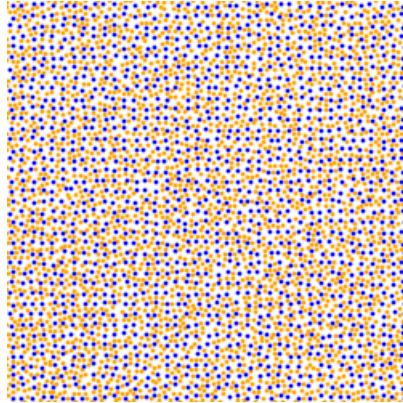


(6) This is what the filled out picture might look like:

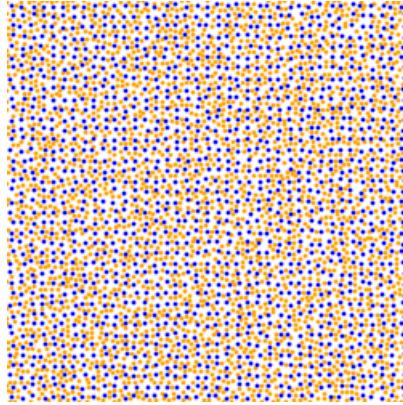


(7) This is what the clusters of 3-fold symmetry look like:





- (8) With zero symmetry, i.e. random placement only, we have:



- (9) POTENTIAL IDEAS: Don't fill in the cluster area with any more extra particles. Save that for the other regions. For the artificial liquid, i.e. zero symmetry, adjust the blue particles more so that they clump a bit.

15. 5/16/2018

- (1) Potential problem: Midway 2 gpu2 is not giving expected accuracies. On data_supercooled, for example, it's showing the inverse of 86% accuracy. Wondering if there is some kind of memory issue going on here. Tried using depablo-gpu but getting the following error: kernel version 384.69.0 does not match DSO version 367.44.0 – cannot find working devices in this configuration. Emailed Igor and awaiting information.
- (2) Also, turned the 0 symmetry, 3 symmetry, and 5 symmetry files into images.

16. 6/11/2018

- (1) We are going to start by cleaning up the code for generating artificial images. We will then move on to trying Josh's idea. We could also train a CNN on small subsets in parallel with MPNN on small subsets. See the smallest subset that MPNN can train on with a batch size of 20. We should also pitch the current results to Prof. Skinner to see what he thinks about publishing without interpretation.

17. 6/22/2018

- (1) From a while ago. We discovered that labeling from the function `create_one_hot_mapping` was inconsistent. So, when checking accuracy, one must make sure to manually assign the correct labeling after checking what the correct labeling is. For the `data_liquid155_version12_step1_best_model.ckpt`, the correct labeling is liquid as [1, 0] and glass as [0, 1].
- (2) So for now, we are going to do a manual assignment. Updated this on `main_glassliquid.py` and on GitHub.
- (3) Now, we have an issue where the program is getting killed. Tried `./filename` for saver, but still it just gets killed. What is going on with the saver? Restoration seems to go ok, but then the accuracy evaluation just fails.

18. 6/25/2018

(1) Current To Do List:

Get the singularity container to work on depablo-gpu nodes, since this is the only tool we currently have left

Overlay the glass cooling curves to show when the depart each other, as Skinner mentioned

Train on glass versus supercooled liquid at the same energy, or glass versus glass at the same energy, in order to get a better percentage

Deal with the question of whether rendering the plots using the blue and orange dots screws with the data

Train on PVD versus liquid-cooled data in order to show a complete story

Try Josh's Monte Carlo scheme

Use TensorBoard, Try Keras

Implement MPNN on subsets, train CNN on subsets

Implement MPNN + pooling to analyze the entire image

Read de Pablo + Ediger papers

Understand why PVD versus liquid cooling structure analysis was important for Dan's paper

Coordinate system independent necessary? Ask Alexei

Analyze spherical harmonics

Analyze RDFs.....

Check out Nick's DeepChem suggestion (MPNN implemented there?)

Run full 2e8 data in order to have that full dataset

Issues to consider for PVD: need 4320 atoms exactly, need 80:20 stoichiometry exactly,

Understand Tg and other issues/metrics for PVD and LC glasses...

Read papers on machine learning!!!

Try both MPNN as well as Geometric Deep Learning github

- (2) Just ran the following on midway-13b-18 (i.e. depablo-gpu) from the `/project/depablo/kswanson/mac-learning-glasses` directory, and it printed final validation accuracy 0.862784 and final test accuracy 0.863454 on `data_supercooled`:
`sinteractive --partition=depablo-gpu --gres=gpu:1 module load singularity singularity exec -B /project:/project -nv /software/src/singularity_images/tf_1.8.0.simg`

```
python main_glassliquid.py -eta_initial=1e-3 -eta_final=1e-4 -eta_threshold=0.90
-batch_size=1 -iterations=1500 -beta=0.01 -keep_probability=0.5 -filename="artificial-
eval" -training_data_aug=False -restart=True -evaluation=True -restart_file="data_liquid155_version
```

- (3) Now, we will go back to MPNN and see how large we can go in terms of number of particles. Goal is to get at least 100, perhaps up to 300, with batch size 20.
- (4) Problem. We are trying the following command, from Igor's email a while ago:

```
module load singularity singularity shell -nv /software/src/singularity_images/PyTorch_cuda_9.0.simg
python demo_glasses_mpnn_classification.py -batch-size=5 from /home/swansonk1/nmp-
qc/demos. It says no module named rdkit. So, tried source activate my-rdkit-
env, but this is not possible. Only works after module load Anaconda3. But then after source activate, I try the module load singularity stuff and I still get the rdkit error. I'm not sure how I got this stuff to work before. But perhaps I need to load this environment in singularity itself, in other words, after I do module load singularity, I THEN need to install the rdkit environment. So, we will try this tomorrow morning. If this does not work, and we have exhausted all other combinations, we will email Igor.
```

19. 6/26/2018

- (1) Starting where we left off yesterday.
- (2) In /home/swansonk1, tried module load Anaconda3, then conda create -c rdkit -n my-rdkit-env-test rdkit. It appeared to download and install all relevant packages. Source activate my-rdkit-env-test, then pip list shows that the packages are indeed installed, apparently.
- (3) Tried module load Anaconda3, module load singularity, source activate my-rdkit-env-test, singularity shell -nv /software/src/singularity_images/PyTorch/cuda_9.0.simg python demo_glasses_mpnn_classification.py -batch-size=5, but got the error /usr/local/Anaconda3-5.1.0/bin/python: /usr/local/Anaconda3-5.1.0/bin/python: cannot execute binary file. Then tried module load Anaconda3, module load singularity, source activate my-rdkit-env-test, singularity exec -nv /software/src/singularity_images/PyTorch/cuda_9.0.simg python demo_glasses_mpnn_classification.py -batch-size=5 and got the error ModuleNotFoundError: No module named 'rdkit', even though the virtual environment definitely contains rdkit.
- (4) Tried simply downloading using conda install the PyTorch version I did on local machine a long time ago, only using module load Anaconda3 and no singularity. Then just ran python demo_glasses_mpnn_classification.py -batch-size=10, but the program crashed after "Check cuda."

20. 7/3/2018

- (1) Beginning to seriously decipher Brian's code. Making folder in /project/depablo/kswanson/machine-learning-glasses called brian_pvd. We will deposit essential files in there to figure out how to run the 3D code.
- (2) We start with master_pvd.py. This script begins by importing functions from write_pvd and write_restart_sbatches, which we will look at in a moment. First, set timeLimit, which is given as 48 hours in seconds (172800). film_index is a list of two, which are indices for different films (i.e. simulations?) will be

run for a given set of conditions, I believe. The number of substrate atoms is fixed at 8480, and the number of substrate layers is fixed as 5. The number of depositions is set to be 1000 (unclear what the definition of a deposition is, but hopefully it will become clear later), and each of these depositions is given 1M turns. Directories list will be a list of directories for the different simulations, indexed by a temperature value and one of the film_index indices. So, we will go ahead and make these directories in brian_pvd. This list is opened as, f, in master_pvd.py. We then loop over Tg_coefficients, which is listed (hard-coded) in write_pvd.py. These values are 0.7, 0.8, 0.9, 1.0, 1.1, along with a Tg listed as 0.335. So, I assume that Tg coefficients are multiplied by Tg to give us different Tg values that we are simulating (?). The comments here say "we sample from an array of substrate temperatures: most notably, 0.7, 0.8, 0.9, 1.0, and 1.1 Tg. for each of these, we make 5 films." However, I think that since film indices are only 0, 1, we just make 2 films for each. So, the substrate temperature perhaps is the glass transition temperature. So, back in master_pvd.py, we enumerate(Tg_coefficients), and then also loop over enumeration of the film indices. So, this is going to give us 10 films in total. thisTg, the glass transition temperature/substrate temperature (???), is set as the original Tg, 0.335, times item, which is the Tg_coefficient. thisName is set as pvd_Tgcoefficient, i.e. pvd_0.7, pvd_0.8, etc. initialXyzName is init_0.7 etc. same for intermediate and restart. dirName is given as expected. Then, we call the function write_pvd_script, from write_pvd.py, which will write the script used to actually perform one of these 10 simulations. The arguments are the number of substrate atoms, number of substrate layers, the substrate Tg, the directory name, thisName, .py extension, Xyz and restart names, and deposition/time limit info. We will see what this all leads to later. We then write sbatchName in pvd_0.7_0 or the like as init_sbatch.sh. job name is 0.7_0. Then call write_restart_sbatch. So, the initial files we need are master_pvd.py, write_pvd.py, write_restart_pvd.py, write_restart_sbatch.py.

- (3) After running master_pvd.py, we have directories.list and each directory contains init_sbatch.sh, pvd_0.7.py, write_pvd.py, write_restart_pvd.py, and write_restart_sbatch.py.
- (4) Now, we will analyze write_pvd.py. This file first imports write_restart_pvd, which we will analyze later. Brian's description: "This is a tool for writing the scripts used to simulate the LJ PVD system in 3 dimensions. We first include some system constants - notably, the LJ parameters, and the Tg specific to this system. In the function we also provide default arguments that were found to be good - e.g., 8480 substrate atoms, and 5 layers of substrate (sono atoms punch through). This permits the function to be called from another script for high-throughput." Then, we have the list of Tg_coefficients as well as the Tg value (?? still a question here where these values came from...maybe Dan's paper has a clue). Then, the LJ interaction parameters are set up between 1, 2, and s particles. Z dimension of substrate is then set to be between 0 and 14.5. Tsub is the temperature the substrate will be maintained at. Based on the input to the function write_pvd_script in master_pvd.py, Tsub is set to be thisTg, i.e. the substrate temperature IS the glass transition temperature (should read up theory for why this is...). Inside write_pvd_script, we first check what the directory

is. IN master, it was something like pvd_0.7_0. If the directory does not exist, make it; if it does, then label pathToFile as pvd_0.7_0/pvd_0.7.py, i.e. the script we are going to generate will be pvd_0.7.py. We then open that python file in write mode, and write the following stuff to create pvd_0.7.py. import sys, then put the appropriate path (we'll need to change to kswanson), import DASH and other generic stuff, import the other write files we will analyze later. Now, we define a function called heightOfFilm(). Create a list of z coordinates of all of the atoms in state, called zCoords. zTop is the highest z value in the simulation. create a histogram using the number of bins as the floor of zTop. Then, get the binned z coordinate at which the largest number of particles appear, called zcoordMax. zcoordMax saves the z coordinate, currentIndex saves the index, and currentCount saves the largest number of particles in one of the histogram bins. If the count here is greater than 100, then search for an index at higher z value until the current count is less than 100, at which point we choose that zcoord and add 5. If we reach the end of the histogram, then add 2. These results are called the filmHeight. Comments label this result as arbitrary. We'll see what it's used for later. Define deleteAtomsAbove(zPosition) function, which deletes atoms that have a z coordinate above the argument. Returns a list of the types of these atoms, which will need to be re-added to "preserve the prescribed 80:20 stoichiometry." set startTime, timeLimit (2 days), prefix (pvd_0.7). In DASH, set rCut to be 2.5. Saving some info from master such as initialXyzName etc. Now, we initialize the system. Since there are 8480 atoms in 5 layers, there are 8480/5 in each layer. In a given layer, we place atoms on a grid at equal distances: so, we take the square root of 8480/5 to find a side length of one layer, i.e. the X and Y dimensions. The bounds are then set at (0, 0, 0) and (L, L, 50). Add the type substrate, 1, and 2 species, all with the same unit mass. Add lJcut fix. Add the lJcut parameters, which are the same as labeled above but are re-hardcoded here. substrateInitBounds are set to be (0, 0, 0) to (L, L, 14.5), where 14.5 was previously labeled subInitZHi or something. Then we populate atoms within this substrate region, nAtoms being the number of substrate atoms 8480, with a minimum distance separation of 0.6. The random population is done using InitializeAtoms.populateRand. Then these atoms are initialized at the substrate temperature, subTemp, which is Tsub, which is thisTg. Then, we define an atom subgroup using state.createGroup('sub'), and state.addToGroup('sub'). So, the substrate atoms are part of 'sub' subgroup. Now, defining a spring function called springFunc(atom), which sets the z coordinate of the atom to be halfway between the lower z bound and upper z bound of the substrate bounds, and then returns the atom position vector. springFuncEquiled(atom) is a function that simply returns the atomic position vector. Then, we add fixSpring, which applies a harmonic potential tethering atoms to user-defined coordinates.

ANISOTROPIC SPRINGS INCLUDING ONE AND TWO DIMENSIONAL SPRINGS CAN BE CREATED BY SETTING DIFFERENT COEFFICIENTS FOR DIFFERENT DIMENSIONS.

Springs differ from external potentials in that springs can tether each atom to a different position, while external potentials simulate all atoms within the relevant group in

the same external field. tetherFunc is set to springFunc, because this returns the vector position to which the atom will be tethered.

- (5) Questions/Comments: Change path name to kswanson,

21. 7/6/2018

- (1) Continuing from before...

(2) dt is set to 0.005. Using a Nose Hoover, fixVNT is set for the 'sub' group, i.e. the substrate atoms. the temperature is set to the subTemp, which is also Tsub, which is also thisTg. IntegratorGradientDescent is applied, I guess as a way to minimize configurations at certain points, and it is labeled integratorRelax. Initial writer, called writerxyz, with the filename from initial xyz filename thingy, is then established, which writes xyz trajectories, every 10,000 steps. Before any runs, integratorRelax is run for 500,000 steps. fixSpring.k is set to 1, before it was 10. The integrator relax is done again for 450,000 steps. Then atoms are reinitialized at substrate temperature, atoms are tethered to the position they fell into, spring constant is increased massively to 1000, and it is made to be isotropic. temperature initialized again. From the documentation, we now apply fix wall harmonic: implements a harmonic potential energy wall at a specified origin with force in the vector direction forceDir. The cut off distance is specified by dist. The spring constant associated with the wall is denoted by the k parameter. So, here topWall creates a wall that applies to EVERY atom, since groupHandle='all', and the origin is (0, 0, zhi). So, this wall essentially acts to push things in the negative z direction away from the top wall. I assume this will help enforce the deposition of the gaseous particles that will be added. We do a final 50,000 Verlet integrator run for a total of 1M substrate steps. We set stoichiometric to [80, 20], which I assume will be the type 1 type 2 ratios. depositionRuns is set to numberOfDepositions, which is set to 1000. Creating a label newVaporGroup='vapor'. creating a variable vaporTemp=0.5 (which is higher than Tg, so this makes sense.) LJ units are the default. Create a new group called 'film'. deactivate the writerxyz. create empty variable atomTypesToAdd. Now we enter the loop of the simulation. 1000 deposition runs, 1M turns per deposition run. Loop over the 1000 deposition runs. print the deposition step out of 1000. Learned about deepcopy. So, this is interesting. toDeposit is a deep copy of the list stoichiometric, which was set to be [80, 20]. For now, let's skip over if atomTypesToAdd, because it is initially empty. Print maxZ, which is the maximum z position of atoms currently in state. print toDeposit, which is the deep copy. newTop is the maximum of the highest z coordinate of atoms plus 20 plus wallDist which is set to 10, or the upper z bound of state. hi is the hi bounds vector. the upper z bound is set to newTop, and the origin of the FixWallHarmonic is readjusted accordingly. Creating a new set of Bounds called populateBounds, which is set with the z low as newTop minus the wallDist minus 12, and the hi as newTop minus the wallDist minus 8. We then initialize randomly, using InitializeAtoms.populateRand, within these populateBounds, 80 type 1 atoms and 20 type 2 atoms, with a minimum distance

between each of 2. Ah, so it looks like we add 100 new particles each deposition, so in total we will add 100,000 vapor particles for deposition. By contrast, all I want is 4230! So, perhaps these simulations will run quickly. We then create an empty list called newAtoms, to which we append the most recently added atoms, i.e. a number toDeposit[0] of type 1 and a number toDeposit[1] of type 2. We then print their ids as well as their positions. We create a new group using the newVaporGroup tag, i.e. 'vapor', to which we add these newAtoms. We also add newAtoms to the 'film' subgroup, which is a cumulative group. new vapor group are initialized at vaporTemp, 0.5. newVaporGroup is then deleted, i.e. its purpose is to initialize the new toDeposit vapor atoms at a specific temperature before turns. The velocity in the z direction is then enforced to be at least -0.2. I guess this gives us an assurance that they will begin traveling towards the substrate. then, we compute the temperature as a function of velocity using $v = \sqrt{3kT/m}$, where k and m are 1 b/c LJ units. Thermal velocity. We then rescale the velocity so that we get back the vapor temp. Essentially, what we are doing here is initializing the vapor particles with an overall motion in the -z direction but also with an overall temperature that is explicitly specified. We then run the integrator for all but 100 of the 1M turns for a single deposition. toDepIdx, which was initialized as 0, is incremented up by 1, and then also set as itself mod len(toDeposit), which is 2. Not sure what the point of this variable is, as it doesn't show up anywhere else in this file! We then write the xyz configuration, called intermediaXyz_deposition number, every single time, for the final 100 turns. For the final turn, restart file is also written, given restart file name + _ + deposition number. writers are deactivated after the final turn. heightOfFilm is then computed and saved as filmHeightEndOfRun, which is then used to delete Atoms above that height. So basically, we delete atoms above the film, which I assume are atoms that were not successfully deposited. We need to keep track of the types of these deleted atoms so that we can add them back in during the next deposition. So that is how we save atomTypesToAdd. we then print the current temperature, current time, and elapsed time. Let's briefly go back to check what happens if there are atomTypesToAdd. If it is not empty, then count the number of type1's and type2's that need to be added, and add these to the toDeposit deep copy, so that we may be adding more than 100 particles at a given round. If the elapsed time has exceeded 98% of the time limit, then do the following. Increment the restart index. set the restart file name, exactRestartFile. write a restart pvd script, with a new Script Name, new sbatch name, new job name, and new output name. write_restart_sbatch creates a new sbatch. Then, have the program submit the sbatch.

- (3) Now, we will investigate the write_restart_pvd script. It gets passed the restartIndex, which becomes part of the script, batch and output names, the number of (1000), the next to be done deposition run ($i + 1$), turns per deposition, Tsub, prefix which is the original name for these files, restart name, intermediate xyz name, the most recent restart file, and time limit. Then, a lot of stuff is the same as write_pvd.py. In line 104, we use the given restartIndex instead of setting to zero. We also load the most recent restart file. Deposition runs is numberOfDepositions - depositionFromRestart, since we wrote restart file prior to deleting

the number of atoms that may have been too far above the film, delete them now. `j` just gives us the actual index, given that we need to add `depositionFromRestart` and are not starting from index 0. Rest of the code is essentially the same.

- (4) Now, we will investigate `write_restart_sbatch`. This is used to create the initial sbatch files (line 38 in `master_pvd.py`), as well as the subsequent sbatch files for restarts. It is straightforward.
- (5) Now, our goal is to actually try one of these simulations. In order to do so, we will first have to change the name for filepaths in several places. Let's start by going into the `pvd_0.7_0` folder. Made this change to `init_sbatch.sh` and `pvd_0.7.py`.
- (6) Questions/Comments: Change path name to `kswanson`, temperature of substrate is glass transition temp? The stoichiometry here is 80/20 instead of 65/35, so we may need to change this. Don't want algorithm learning for stoichiometry. We may need to make sure we don't delete atoms at the end of the run? Or perhaps, if we take a middle slice....Why do we need to delete atoms

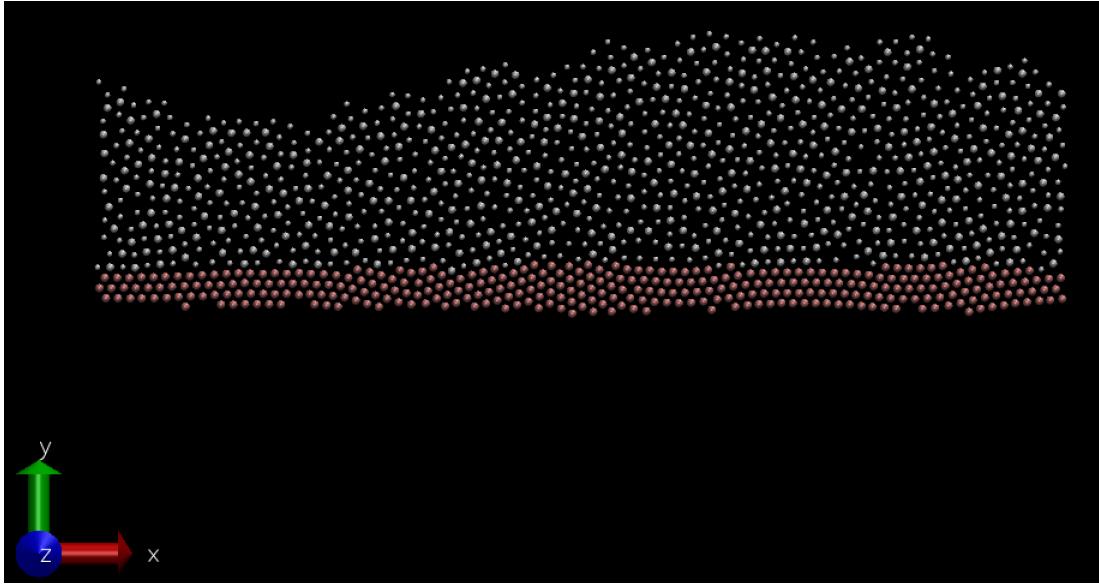
22. 7/11/2018

- (1) Now that we see the 3D PVD glasses code working, we will try to implement 2D.
- (2) Canceling 3D job now.
- (3) Created, in `machine-learning-glasses/brian_pvd`, a directory called `pvd_2D`, in which we place `init_sbatch.sh`, `pvd_0.7.py`, `write_pvd.py`, `write_restart_pvd.py`, `write_restart_sbatch.py`.
- (4) Our goal is not to fully understand PVD at the moment..just see if we can implement any kind of 2D simulation.
- (5) Let's first see if we can just simulate a 2D substrate. For 8480 total atoms in 5 layers, each layer has 1696 atoms. in 3D, giving each atom a spot on a unit lattice, this would mean $\sqrt{1696} \approx 42$ atoms along a given side. So, to keep the same size image (for now), we would just do 5 layers of 42 atoms, or 210 total atoms.
- (6) Based on `src/Bounds.cpp`, line 28, `handle2d` puts `lo[2]`, the z bounds `lo`, at -0.5, and `rectComponents[2] = 1.0`, where `rectComponents` is previously defined as `hi-lo`, which indicates that for 2D simulations, the z bounds are -0.5 to 0.5, so might make sense to initialize bounds as such. Setting the `substrateInitBounds` to be 0 to 0 in the z dimension. Essentially, we will have periodic boundaries in the x dimension (?) but not y? Unclear how this works.....
- (7) Based on `State.cpp`, line 131, if `is2d` is true, then the z position of a newly added atom is set to zero. Also, the number of degrees of freedom is set to 2 rather than 3.
- (8) Perhaps we want exact placement of substrate atoms such that we can choose randomly ourselves....or how do we make one dimension NOT periodic in DASH?

23. 7/12/2018

- (1) Starting over with trying to do 2D simulation.
- (2) Phil says we can use `bounds` object to do `setPeriodic`, check `bounds.cpp`.

- (3) Discovered that we need to set state.is2d=True IN ADDITION to the Fix2d to do a 2d simulation in DASH. Also, have to set z dimension nonperiodic by state.setPeriodic(2, False).
- (4) This worked! in pvd_2D, running pvd_2d.py.



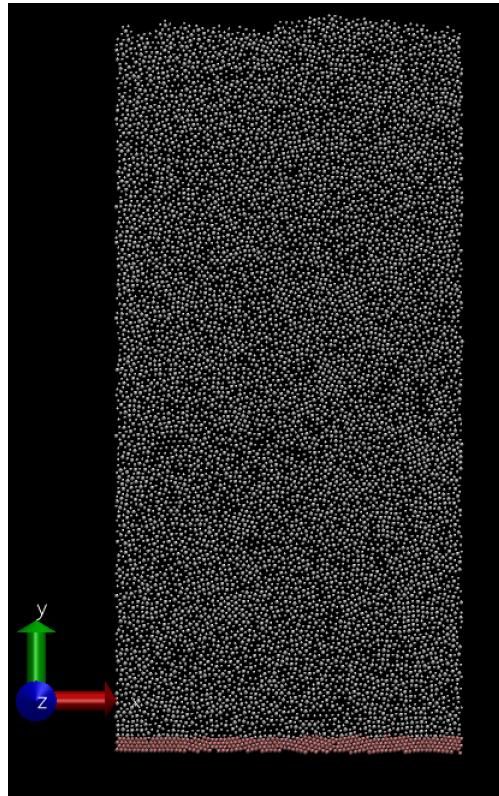
- (5) We will continue pvd_2D.py doing 100 depositions. There are 5 layers of substrate atoms, 330 in total. For each deposition, we do 100 atoms, 65 type 1 35 type 2.
- (6) Moving back to MPNN, simply tried on depablo-gpu module load singularity then singularity shell --nv /software/src/singularity_images/PyTorch_cuda_9.0.simg, but I got the following error:

```
[swansonk1@midway-l13b-20 demos]$ singularity shell --nv /software/src/singularity_images/PyTorch_cuda_9.0.simg
ERROR : Image path /software/src/singularity_images/PyTorch_cuda_9.0.simg doesn't exist: No such file or directory
ABORT : Retval = 255
[swansonk1@midway-l13b-20 demos]$
```

- (7) Bullshit. In the meantime, got on to gpu2 and trying out https://github.com/rusty1s/pytorch_geometric
- (8) on gpu2, in /home/swansonk1, did module load Anaconda3/5.0.1, module load cuda/8.0, module unload gcc, according to https://git.rcc.uchicago.edu/ivy2/PyTorch_midway2. Then, created virtual environment via cfonda create -n geometric-env, according to <https://uoa-eresearch.github.io/eresearch-cookbook/recipe/2014/11/20/conda/>. Following installation instructions now. pip install cffi, already satisfied. pip install --upgrade torch-scatter, error on no permissions, so we do pip install --user --upgrade torch-scatter, and it works, as well as pip install --user --upgrade torch-unique, "" torch-cluster, ""torch-spline-conv, pip install --user torch-geometric.
- (9) Continuing to read documentation. COO from Wikipedia says stores a list of (row, column, value) tuples.
- (10) data.contains_isolated_nodes() will be a great way to check for connectivity.

24. 7/13/2018

- (1) The 2D ran completely:



- (2) Still missing some functions that we should test. Also, based on Reid's paper, we should be about 10 angstroms away from the substrate and the top wall for bulk material. Our 100 depositions led to a y height of 197. That means each deposition (ignoring substrate) took up around 2 Angstroms. So, this means we need about 5 depositions before counting anything, then about 44 depositions to get the 4320 atoms we need, then another 5 depositions for padding. This gives us a total of about 55 depositions. Takes about 45 minutes to do 33 depositions, so it should take about 75 minutes to do the whole thing. So, that means we can do approximately 19 samples on one GPU card per day. If we have 10, then 190 per day. This means about 53 days. Not terrible, but not great either.
- (3) Now we are going to add back in the delete atoms above certain film height command (not sure if this is necessary...), running 55 depositions, printing elapsed-Time at the end. Should take around 75 minutes.
- (4) Run failed. Got to the point, after 3 depositions, where it wanted to add around [171, 75] atoms, because it had removed so many. Perhaps this removing of atoms is not necessary. Code seems to suggest it is arbitrary anyway. So, let's re-do the 55 run and remove unnecessary code.
- (5) Back to Igor and singularity. Creating the shell works, i.e. in /home/swansonk1, module load singularity, singularity shell -nv /software/src/singularity_images/PyTorch_cuda_9.0.sif Then, ran conda create -ic rdkit -n my-rdkit-env-singularity rdkit.

- (6) Tried running the glass classificaiton demo, but said no module torch. so tried conda install pytorch=0.1.12 -c soumith inside the environment.
- (7) So, now when we type python, import torch, it works. However, with inside source activate my-rdkit-env-singularity, we get some random error.
- (8) Interestingly enough, we are able to source activate my-rdkit-env-test with singularity, but the run fails after a while. On gpu2, using my-rdkit-env-test, program runs briefly but then we get a new error. Ok, I have no idea what the fuck is going on with this code. Perhaps we should try the new code...
- (9) Takes about 4311 seconds to generate a complete film with 55 depositions of [65, 35] particles on a substrate with 330 atoms arranged in approx. 5 layers, no removing of atoms or adjusting film height.

25. 7/23/2018

- (1) Is the inherent structure energy of the PVD film from the bulk section or from the entire thing? It would make sense to just do the bulk section...From Dan's paper. "The inherent structure energy EIS of a configuration, used to quantify its stability, is the potential energy of a configuration brought to its local energy minimum." Later on, in Figure 2 caption, "Only atoms in the bulk region shown are used in calculation unless otherwise specified. We define the bulk region to be several σ_{AA} away from where bulk composition and density properties are reached to ensure that edge effects are not present in the data. Error bars represent 95% confidence intervals." Later, "The energetic properties of PVD glasses are determined using only particles in the 'bulk' region of the films, which is highlighted in Figure 2. It corresponds to a wide domain of constant density and composition...The inherent structure energy, EIS, is an effective measure of the position of a glass on the potential energy landscape."
- (2) So, clearly, what we need to do is figure out how to use DASH to calculate the potential energy of a system within a certain region of a configuration.
- (3) Trying to use state.dataManager.recordEnergy on the pvd_2D.py script. Renaming the version pvd_2D-energy-test.py.

26. QUESTIONS FOR PVD

- (1) Questions/Comments: Change path name to kswanson, temperature of substrate is glass transition temp? The stoichiometry here is 80/20 instead of 65/35, so we may need to change this. Don't want algorithm learning for stoichiometry. We may need to make sure we don't delete atoms at the end of the run? Or perhaps, if we take a middle slice....Why do we need to delete atoms. HOW are periodic conditions set in DASH? How were they set in Brian's simulations?
- (2) We can probably compute based on what group the atoms are in. Could create a separate group during the simulation, perhaps, to compute energies. Pausing this for now, but the file we are using to run these tests is pvd_2D-energy-test.py.
- (3) Paper Review: Neural Message Passing for Quantum Chemistry, by J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, 2018.

Abstract. Supervised learning on molecules has incredible potential to be useful in chemistry, drug discovery, and materials science. Luckily, several

promising and closely related neural network models invariant to molecular symmetries have already been described in the literature. These models learn a message passing algorithm and aggregation procedure to compute a function of their entire input graph. At this point, the next step is to find a particularly effective variant of this general approach and apply it to chemical prediction benchmarks until we either solve them or reach the limits of the approach.

Introduction. The past decade has seen remarkable success in the use of deep neural networks. Although chemists have applied machine learning to many problems over the years, predicting the properties of molecules and materials using machine learning (and especially deep learning) is still in its infancy. To date, most research applying machine learning to chemistry tasks has revolved around feature engineering. MD has been able to generate data at an unprecedented rate. Need to find models with suitable inductive biases. The symmetries of atomic systems suggest neural networks that operate on graph structured data and are invariant to graph isomorphism might also be appropriate for molecules. Graph isomorphism: two graphs G and H are isomorphic if any two vertices are adjacent in G iff they are also adjacent in H . Adjacent: the relation between two vertices that are both endpoints of the same edge. We want graph isomorphism, because if we permute the atom labels we have the same molecule. Sufficiently successful models could someday help automate challenging chemical search problems in drug discovery or materials science. In this paper, our goal is to demonstrate effective machine learning models for chemical prediction problems that are capable of learning their own features from molecular graphs and are invariant to graph isomorphism. To that end, we describe a general framework for supervised learning on graphs called Message Passing Neural Networks (MPNNs) that simply abstracts the commonalities between several of the most promising existing neural models for graph structured data. QM9 lets us consider both the setting where the complete molecular geometry is known (atomic distances, bond angles, etc.) and the setting where we need to compute properties that might still be defined in terms of the spatial positions, but where only the atom and bond information (i.e. graph) is available as input. In the latter case, the model must implicitly fit something about the computation used to determine a low energy 3D conformation and hopefully would still work on problems where it is not clear how to compute a reasonable 3D conformation. In this paper, they achieve state-of-the-art results on all 13 targets and predicts DFT to within chemical accuracy on 11/13 targets. They develop several different MPNNs which predict DFT to within chemical accuracy on 5 out of the 13 targets while operating on the topology of the molecule alone (with no spatial information as input). We develop a general method to train MPNNs with larger node representations without a corresponding increase in computation time or memory. We believe our work is an important step towards making well-designed MPNNs the default for supervised learning on modestly sized molecules.

Message Passing Neural Networks. There are at least eight notable examples of models from the literature that we can describe using our Message Passing Neural Networks (MPNN) framework. For simplicity we describe MPNNs

which operate on undirected graphs G with node features x_v and edge features e_{vw} . The forward pass has two phases, a message passing phase and a readout phase. The message passing phase runs for T timesteps and is defined in terms of message functions M_t and vertex update functions U_t . During the message passing phase, hidden states h_v^t at each node in the graph are updated based on messages m_v^{t+1} :

$$\begin{aligned} m_v^{t+1} &= \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \\ h_v^{t+1} &= U_t(h_v^t, m_v^{t+1}) \end{aligned} \quad (26.1)$$

where in the sum, $N(v)$ represents the neighbors of v in graph G . So basically, the message function is a function of the current node and its neighbor nodes and edges. The output of this function is then an argument to the vertex update function. The readout phase computes a feature vector for the whole graph using some readout function R according to

$$\hat{y} = R(\{h_v^T | v \in G\}) \quad (26.2)$$

In what follows, we define previous models in the literature by specifying the message function M_t , vertex update function U_t , and readout function R used. Note that one could also learn edge features in an MPNN by introducing hidden states for all edges in the graph $h_{e_{vw}}^t$ and updating them analogously to equations 1 and 2. Of the existing MPNNs, only Kearnes et al. (2016) has used this idea...This work deviates slightly from other MPNNs in that it introduces edge representations e_{vw}^t which are updated during the message passing phase. NOTE: One downside of all of these approaches is computation time. Recent work has adapted the GG-NN architecture to larger graphs by passing messages on only subsets of the graph at each time step.

3. Related work. Although in principle quantum mechanics lets us compute the properties of molecules, the laws of physics lead to equations that are far too difficult to solve exactly. Therefore scientists have developed a hierarchy of approximations to QM with varying tradeoffs of speed and accuracy, e.g. DFT, the GW approximation, and Quantum MC. Two more recent approaches by Behler and Parrinello (2007) and Rupp et al. (2012) attempt to approximate solutions to quantum mechanics directly without appealing to DFT. In the first case single-hidden-layer neural networks were used to approximate the energy and forces for configurations of a Silicon melt with the goal of speeding up MD simulations. The second paper used Kernel Ridge Regression to infer atomization energies over a wide range of molecules. In both cases hand engineered features were used that built physical symmetries into the input representation. Parrinello's work used a representation that was manifestly invariant to graph isomorphism, but has difficulty when applied to systems with more than three species of atoms and fails to generalize to novel compositions.

To investigate the success of MPNNs on predicting chemical properties, we use the publicly available QM9 dataset. Has about 134k druglike organic molecules that span a wide range of chemistry. For each molecule DFT is used

to find a reasonable low energy structure and hence atom "positions" are available. Additionally a wide range of interesting and fundamental chemical properties are computed. Given how fundamental some of the QM9 properties are, it is hard to believe success on more challenging chemical tasks will be possible if we can't make accurate statistical predictions for the properties computed in QM9. Properties include atomization energy, zero point vibrational energy, HOMO, LUMO, polarizability, spatial distribution of electrons.

5. MPNN Variants. We began our exploration of MPNNs around the GG-NN model which we believe to be a strong baseline. We focused on trying different message functions, output functions, finding the appropriate input representation, and properly tuning hyperparameters. Treat undirected as directed, so size of message channel doubles. The input to the MPNN model is a set of feature vectors for the nodes of the graph, an adjacency matrix with vector valued entries to indicate different bonds as well as spatial distance between two atoms. The initial hidden states are set to the the atom input feature vectors. One issue with MPNNs is scalability. In particular, a single step of the message passing phase for a dense graph requires $O(n^2d^2)$ floating point multiplications. To address this issue we break the d dimensional node embeddings h_v^t into k different d/k dimensional embeddings and run a propagation step on each of the k copies separately, using separate message and update functions for each copy.

Input Representation. There are a number of features available for each atom in a molecule which capture both properties of the electrons in the atom as well as the bonds that the atom participates in. Graphs have up to 29 nodes including hydrogens. Note that having larger graphs significantly slows training time, in this case by a factor of roughly 10. For the adjacency matrix there are three edge representations used depending on the model: chemical graph (bond types, no distance), distance bins (10 bins using histogram of differences), raw distance feature (when using a message function which operates on vector valued edges, the entries of the adjacency matrix are then 5-dimensional, where the first dimension indicates the euclidean distance and the remaining four are a one-hot encoding of the bond type).

Training. Each model and target combination was trained using a uniform random hyper parameter search with 50 trials. T was between 3 and 8. All models trained using SGD with ADAM optimizer, batch size 20, for 3 million steps (540 epochs). The initial learning rate was chosen uniformly between 1e-5 and 1e-4. We used a linear learning rate decay that began between 10 and 90 percent of the way through training and the initial learning decayed to a final learning. 10,000 samples for validation, 10,000 for testing, 110,462 for training. Used validation set to do early stopping and report scores on the test set. Minimize the mean squared error between model output and target, although they evaluate the mean absolute error. Each sample is a separate molecule. So, we are training on some molecules, and seeing if it can learn on others. Cool!

8. Results. In all of our tables we report the ratio of the mean absolute error (MAE) of our models with the provided estimate of chemical accuracy for that target. Thus any model with error ratio less than 1 has achieved chemical

accuracy for that target. Also, models are trained individually on each target, not all 13 simultaneously. In our experiments, we found that including the complete edge feature vector (bond type, distance) and treating H explicitly was best. Training 1 target rather than 13 joint was also best. We were able to further improve performance on the test set by ensembling the predictions of the five models with lowest validation error. Having a model that extends long-range interactions useful (virtual edges, master node). GG-NN model great baseline.

9. Our results show that MPNNs with the appropriate message, update, and output functions have a useful inductive bias for predicting molecular properties and eliminating the need for complicated feature engineering. Moreover, our results also reveal the importance of allowing long range interactions between nodes in the graph. An important future direction is to design MPNNs that can generalize effectively to larger graphs than those appearing in the training set or at least work with benchmarks designed to expose issues with generalization across graph sizes. Generalizing to larger molecule sizes seems particularly challenging when using spatial information. First of all, the pairwise distance distribution depends heavily on the number of atoms.

- (4) Thought: perhaps we just train on subsets - break down a configuration into smaller elements, train on these all with labels, then for a larger configuration, break it up and take the average prediction on each component.
- (5) Paper Review: recent Shubhendhu materials hierarchy paper

27. 8/1/2018

- (1) https://github.com/rusty1s/pytorch_geometric again. Followed instructions from notes on 7/12/2018, on gpu2 interactive session, module load Anaconda3/5.0.1, module load cuda/8.0, module unload gcc, source activate geometric-env. Now, let's continue the tutorial online.
- (2) Another potential gem: "We can convert the point cloud dataset into a graph dataset by generating nearest neighbor graphs from the point clouds via transforms".

REFERENCES

- [R] D. Reid, I. Lyubimov, M. D. Ediger, J. J. de Pablo, *Age and structure of a model vapour-deposited glass*, Nature Communications, 2016.
- [QM] J. Gilmer, S. Schoenholz, P. Riley, O. Vinyals, and G. Dahl, *Neural Message Passing for Quantum Chemistry*, www.arxiv.org, 2017.
- [GC] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, *Molecular Graph Convolutions: Moving Beyond Fingerprints*, www.arxiv.org, 2017.
- [P] Q. Wei, R. G. Melko, and Jeff Z. Y. Chen, *Identifying polymer states by machine learning*, www.arxiv.org, 2017.

E-mail address: swansonk1@uchicago.edu

INSTITUTE FOR MOLECULAR ENGINEERING, UNIVERSITY OF CHICAGO, 5640 S ELLIS AVE,
CHICAGO, IL 60637