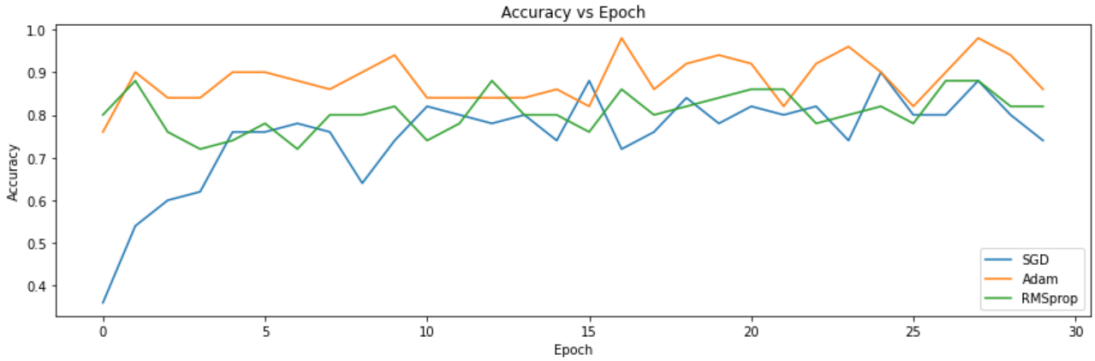
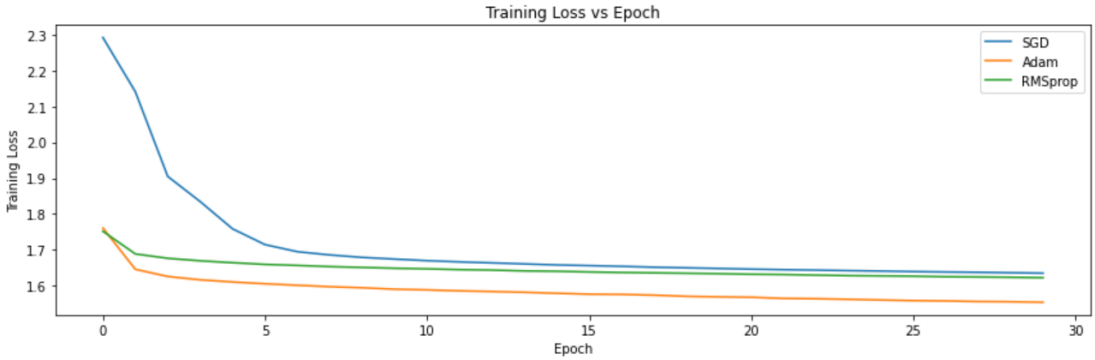
Eric Gagliano (egagli)
Inferring Structure from Complex Systems

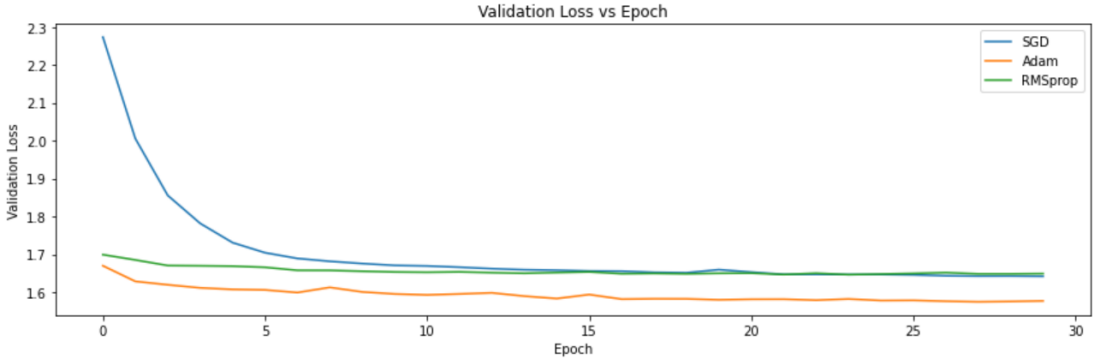
Lab 3

(All of the lab problems in markdown with code and better graphics are in the notebook file, but summarizing here as requested)

1. Try different optimizers including RMSProp, Adam and SGD(You can find the corresponding functions in the torch.optim library). Log your training loss and test accuracy. Comparing these optimizers, how do they work? Which one is the best for this task and try to explain it.

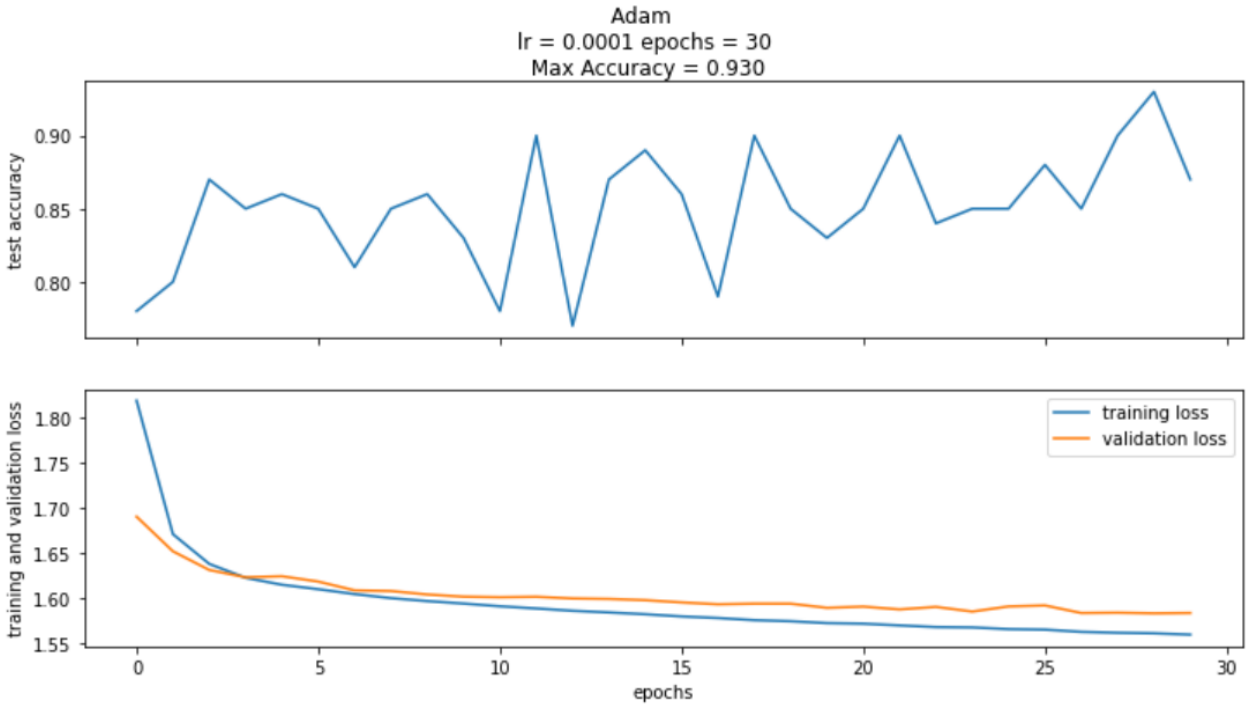




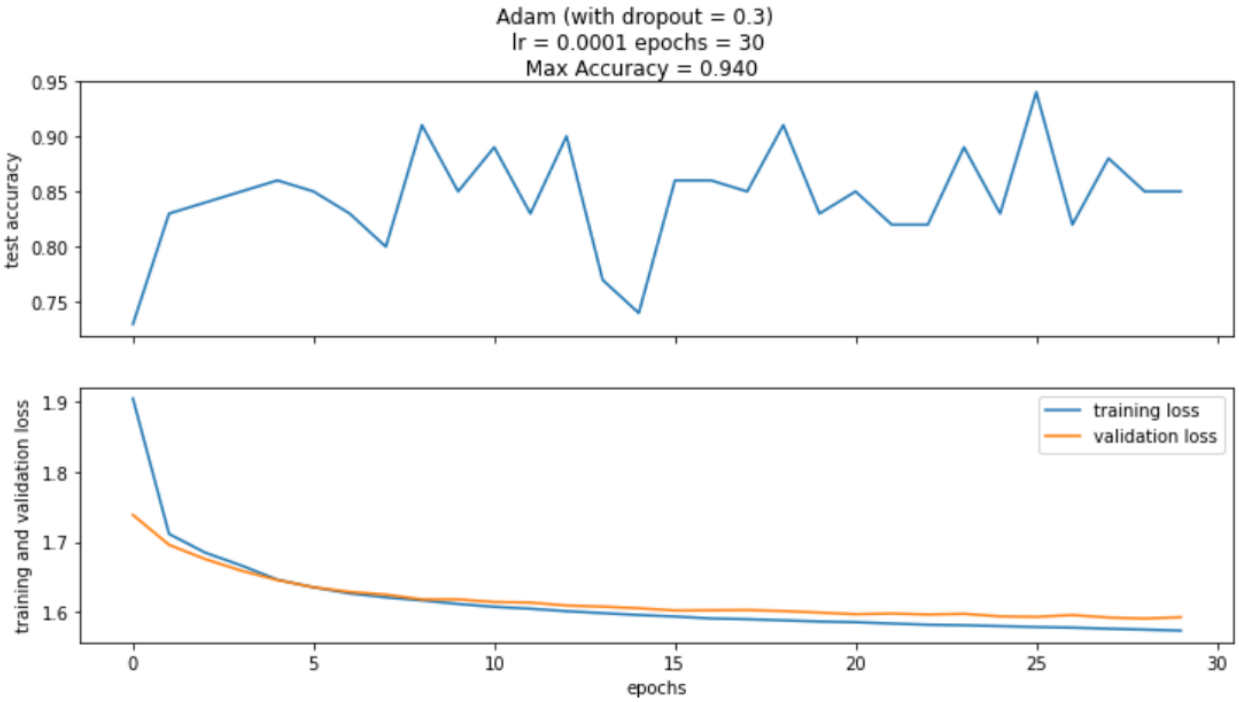


The above figure shows the performance of three optimizers. Out of the 3 optimizers tested (SGD w/ lr=0.01, Adam w/ lr=0.0001, RMSprop w/ lr=0.0001), Adam performs with the best accuracy across almost all epochs. Also, the training and validation loss is consistently lower using Adam. The times were all comparable, with SGD taking 5:53, Adam taking 6:32, and RMSprop taking 6:12. For the increased accuracy, an extra 30 seconds is a small price to pay. For this task, (given limited testing time and exploration of parameter combinations), I'll choose to evaluate Adam going forward.

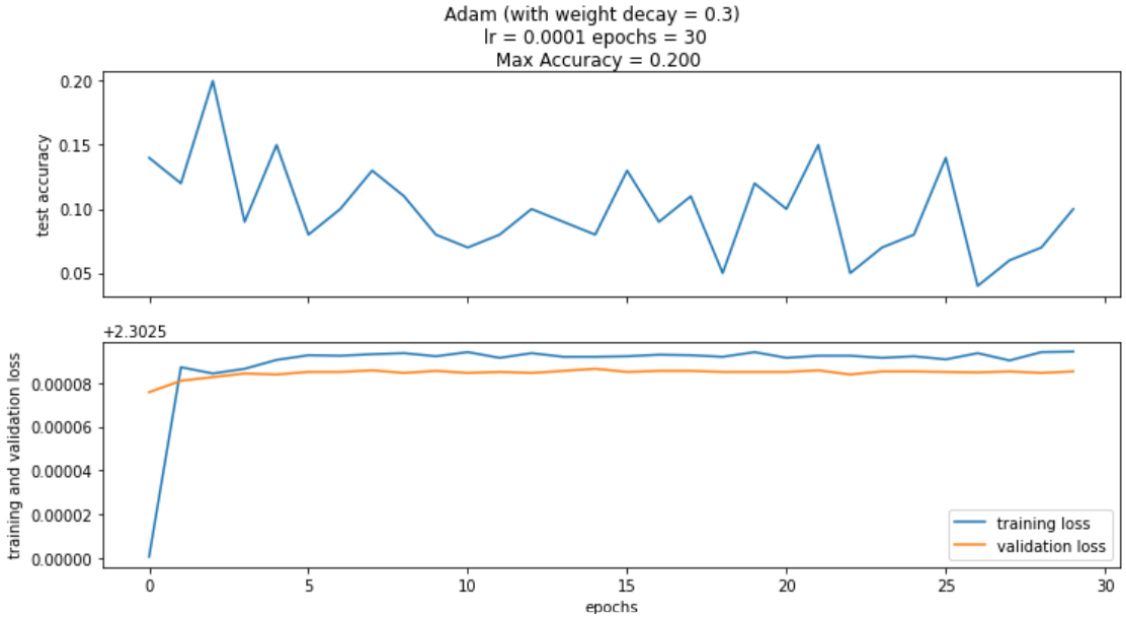
2. Analyze the overfitting/underfitting situation of your model. Try to use regularization like L2/L1, Dropout layers, etc. to improve your performance. How does your regularization work? And how do you balance your regularization and the loss optimization?



Based on the graph above, it seems we are overfitting. This is apparent where the validation loss stagnates and the training loss continues to decrease. This is a telltale sign of overfitting, as the model cannot reproduce its improved training accuracy in the validation set. This begins around epoch 4. To combat this, we regularize. We will utilize L2 regularization and dropout to try to minimize overfitting. First let's try dropout.



So it seems dropout reduces overfitting! This is evident in the crossover event between training and validation loss that has been delayed to epoch 7. Further dropout seems to continue to reduce overfitting. Now let's try L2 regularization.

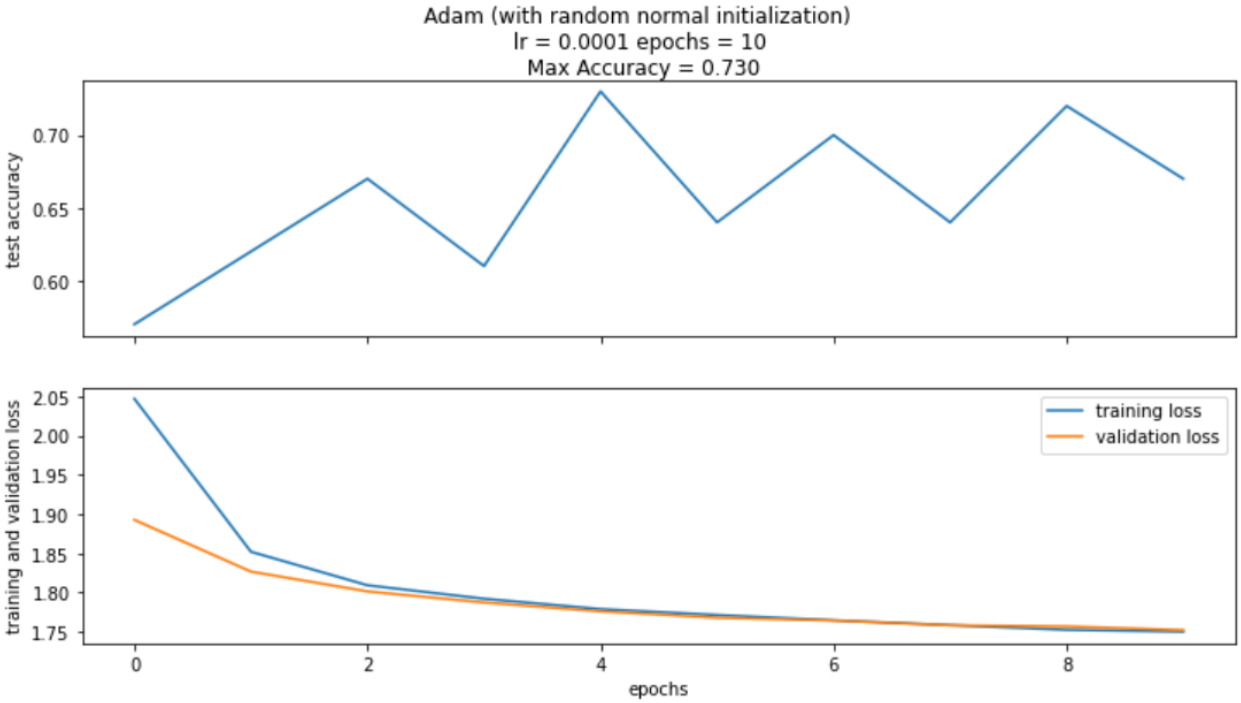


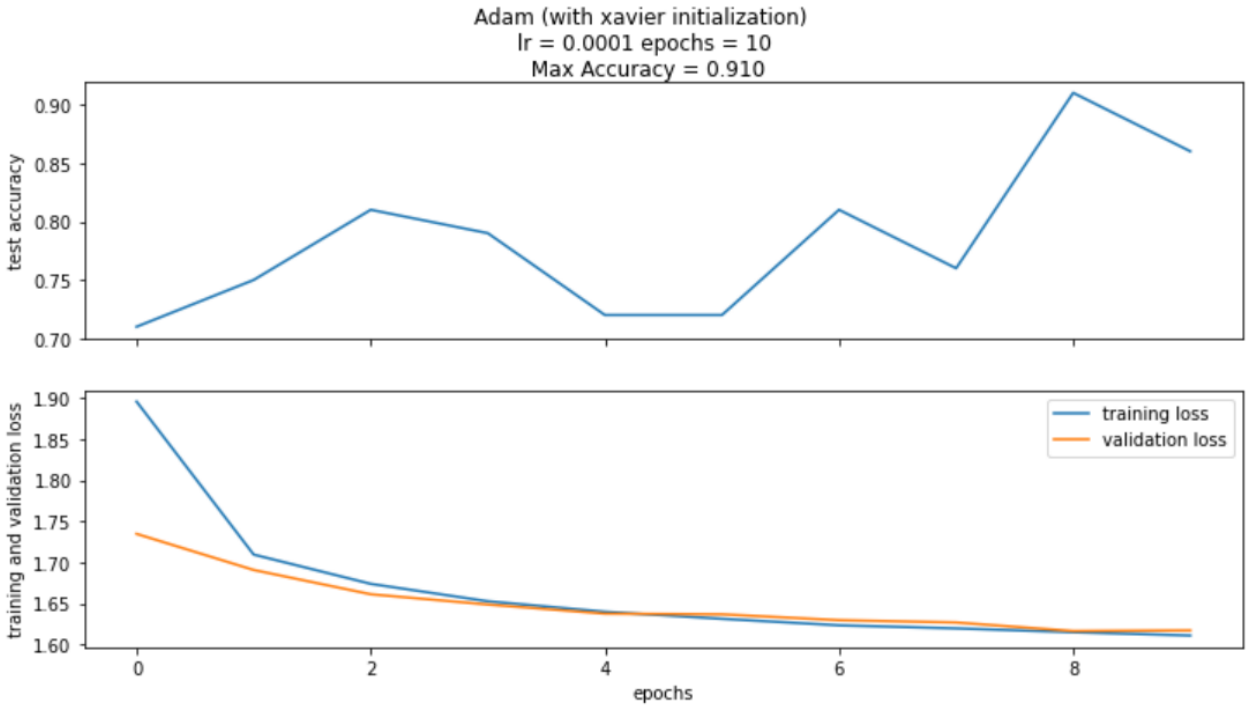
I applied L2 regularization in the Adam definition, setting the weight decay option which acts as an L2 penalty. So it seems L2 regularization (is supposed to) have a similar effect as dropout but took much longer and must be paired with a lower learning rate, as my attempt above took 27 minutes (and it didn't work that well :( ). Maybe if I try changing the learning rate with L2 regularization I will not have this error. In general, one thing I wonder is perhaps combining both

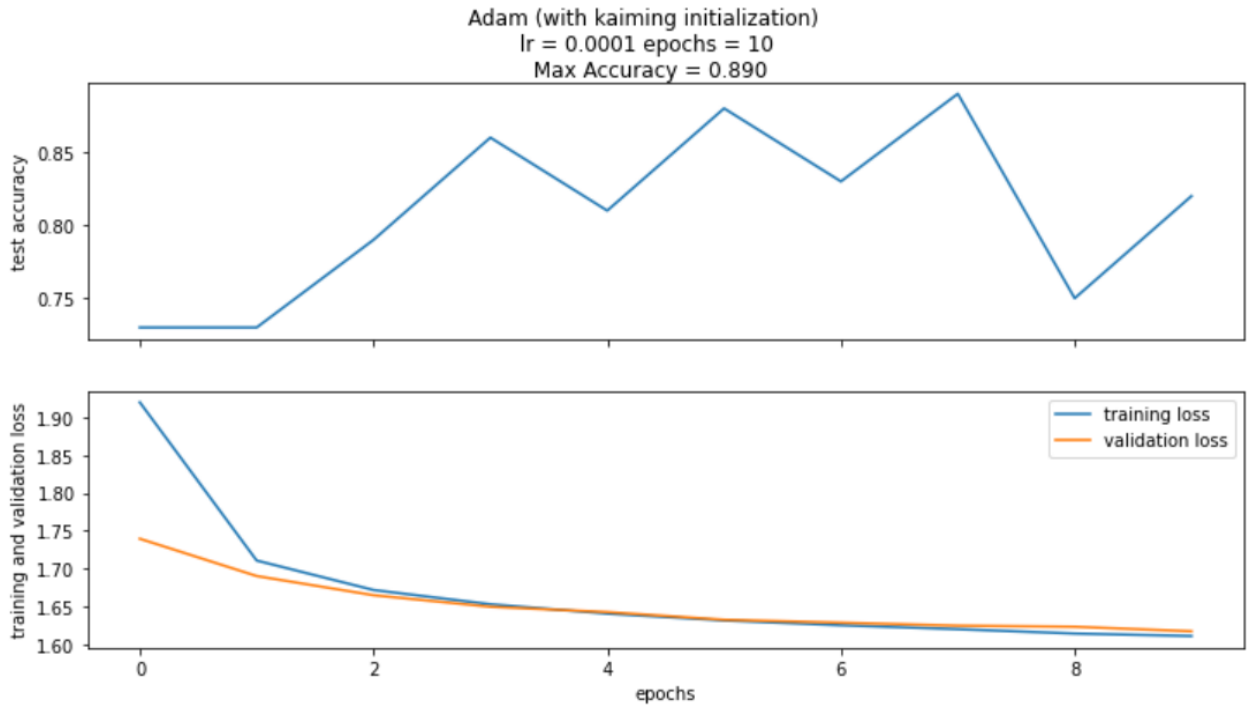
dropout and L2 regularization could further reduce overfitting? Regularization and loss optimization must be tricky to balance, there is a sweet spot. Making sure the model is not overfit is the priority, then we can focus on loss optimization.

3. Try different initialization ways like random normal, Xavier and He (Kaiming) etc. How can these initialization affect your training process and performance?

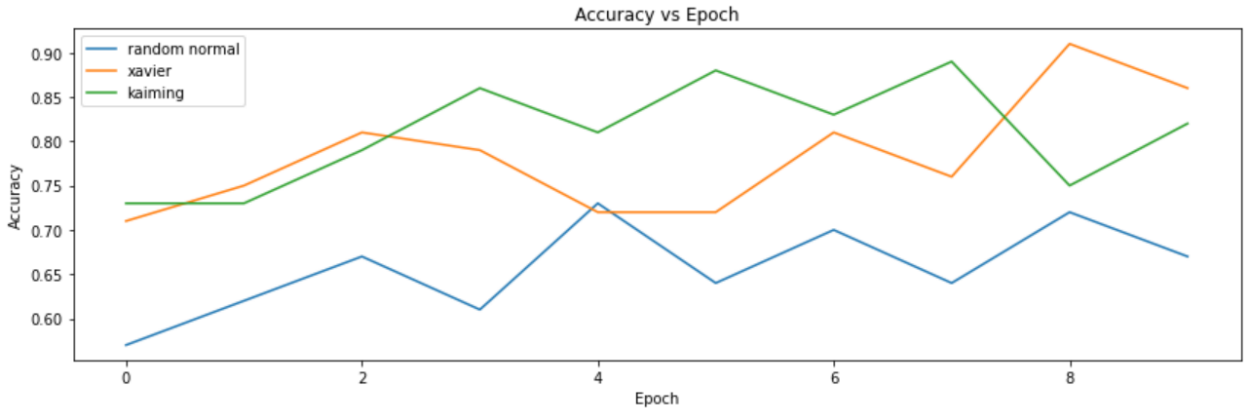
Trying below:

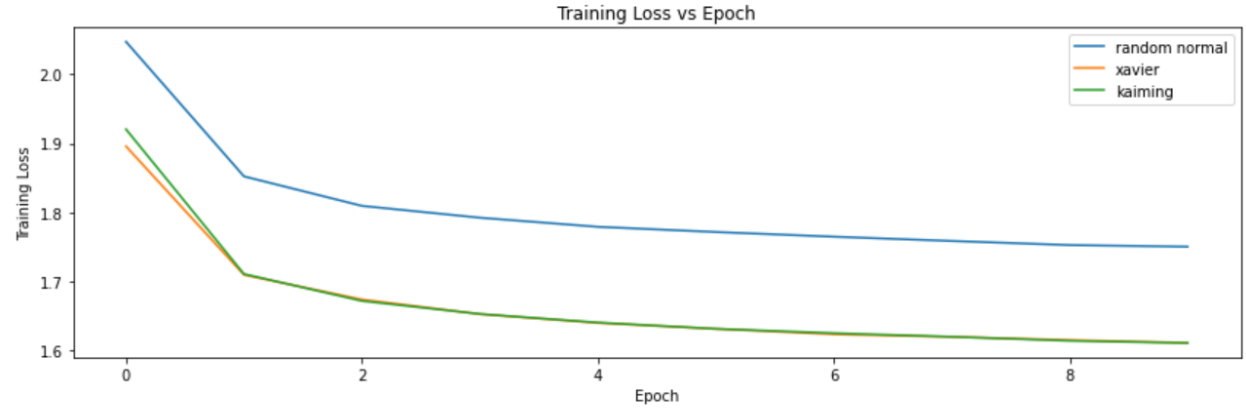


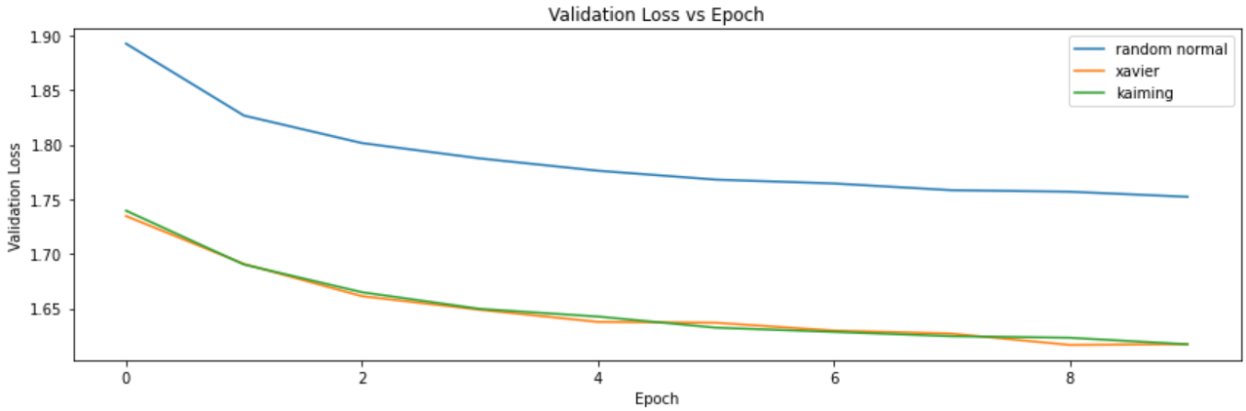




And now let's show everything:

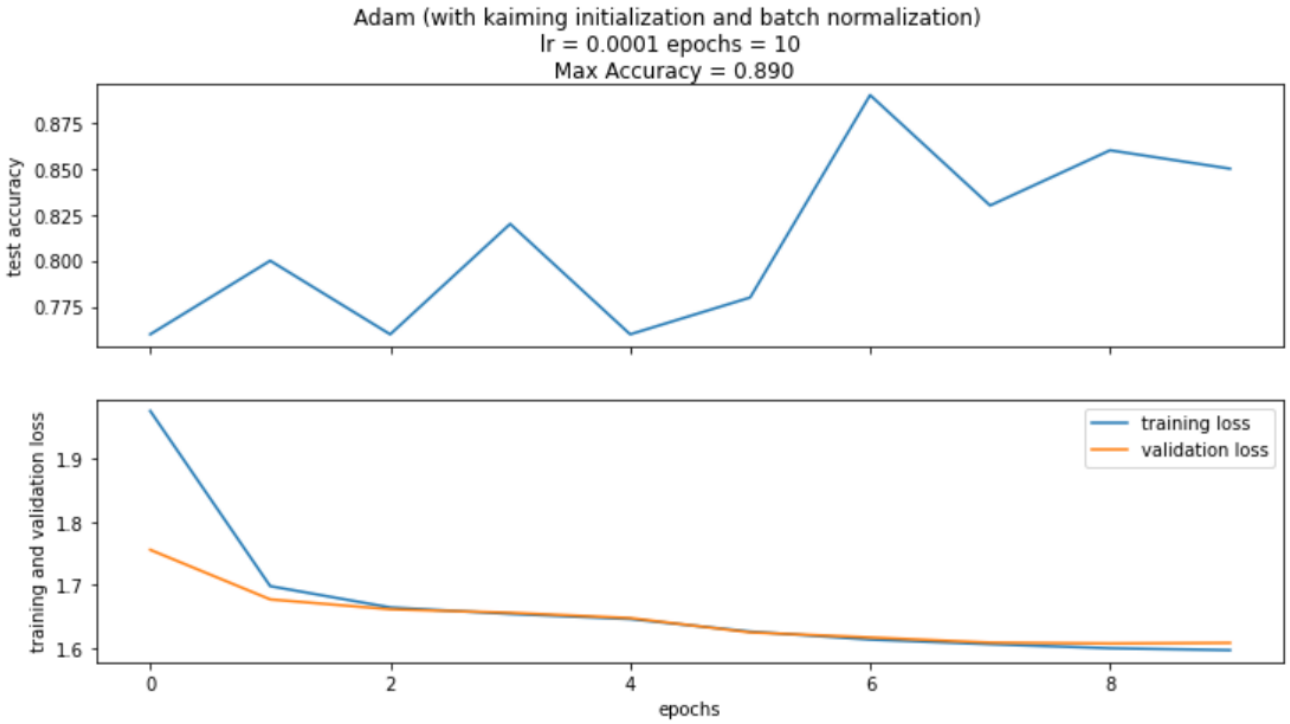






I have combined the three initializations in the plot above for better intercomparison. It looks as if random normal does not perform well compared to xavier and kaiming initialization. It is surprising to me that weight initialization actually has a significant effect on training. But I guess it also has to do with a combination of optimizer and initialization, though I'm sure there are a ton of permutations to explore. This translates to a tangible difference in the accuracy of the model. I will use kaiming initialization going forward for this task. I wonder what the difference between xavier and kaiming is, will look into later.

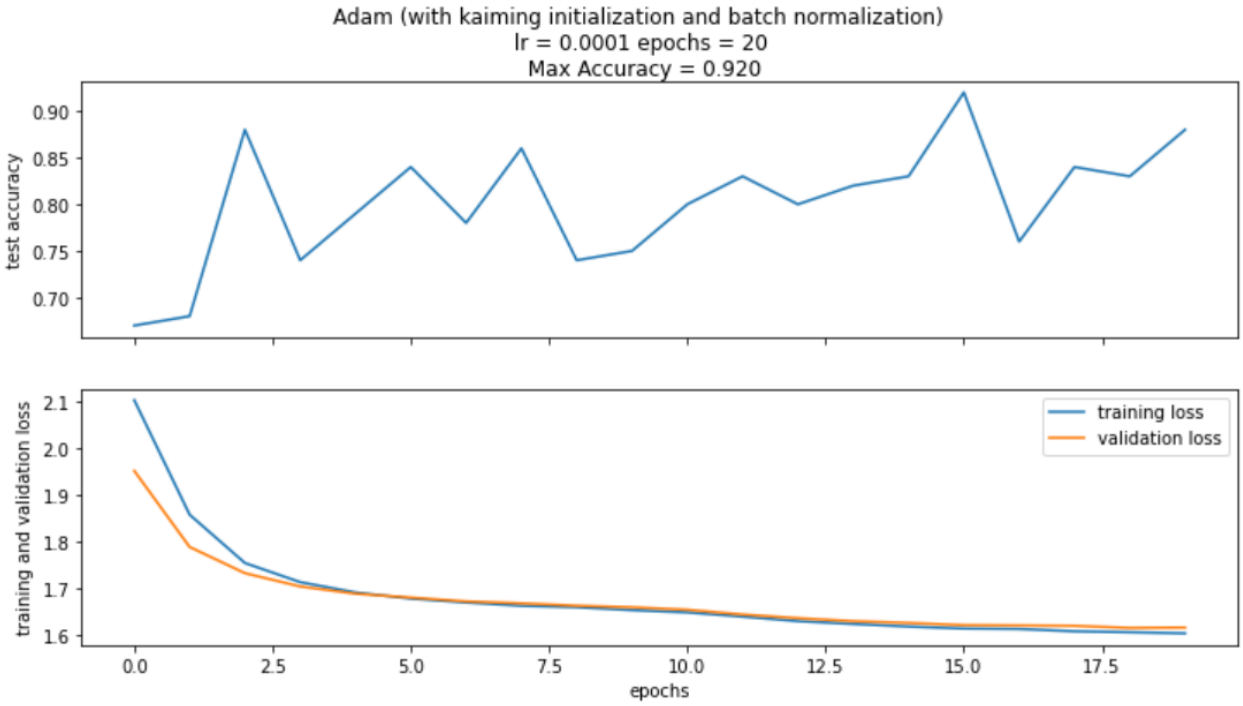
4. Try normalization like batch normalization or layer normalization, whether these normalization methods can help your training or improve your performance?



As shown above, it seems like batch normalization does indeed have a slight? positive effect on training. More than anything else it looks like this sped things up, I got a 20 second speedup. If nothing else, I will keep batch normalization going forward for the speed increase. Googling, it seems like there should also be an increase in model performance using batch normalization, especially as a form of regularization, though that wasn't super evident for my current combination of hyperparameters. It looks like the order in which I do batch normalization will have an effect, so maybe I should more rigorously evaluate where to batch normalize.

5. To further improve the performance, you can also choose your own hyperparameters, including: Number of layers, Number of neurons in each layer, Learning rate, Number of training epochs. What was the accuracy on the test set for your best-performing network?

The accuracy for my best performing network was ~92%. I achieved this through 3 linear layers, with 300,150,75 neurons using relu. I used Adam with lr = 0.0001 and 20 epochs, dropout of 0.4, kaiming initialization and batch normalization. It took 6:11 to run. This model is shown below.



Code is in the attached ipynb, thank you!