1 Tiny Image Net

- What design that you tried worked the best? This includes things like network design, learning rate, batch size, number of epochs, and other optimization parameters, data augmentation etc. What was the final train loss? Test loss? Test Accuracy? Provide the plots for train loss, test loss, and test accuracy.

1. architecture

A series of convolutional layers (2D), with maxpooling, batch normalization, and a leaky RELU activation function in between. Also, two linear layers at the end instead of one (also separated by a leaky RELU activation).

2. hyperparameters

As mentioned in class, we also lower learning rate over time by multiplying learning rate by 0.9 every epoch (in essence, reducing it by 10% each time). We started learning rate at 0.05 to barrel towards the minimum quickly at the beginning.

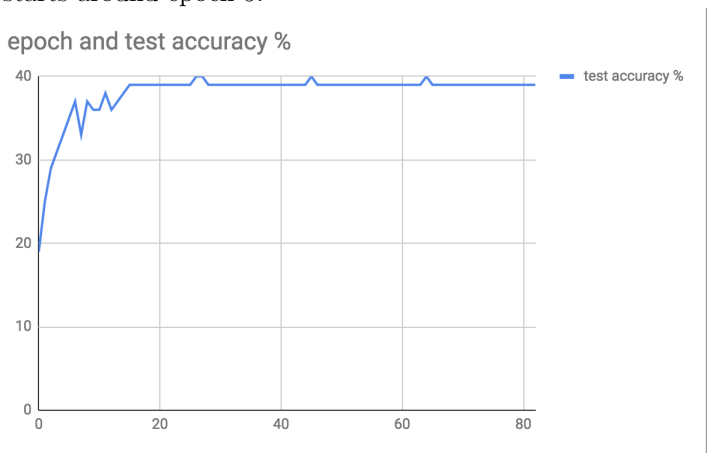
Additionally, we found it suuuuper necessary to use logsoftmax instead of softmax – for whatever reason (we are having trouble reasoning through why), softmax kept us stagnant at 5% but switching to logsoftmax immediately got us hitting 35%. We also tried out some data normalization, but didn't end up using it.

We didn't play around with other hyperparameters too much. We left momentum, batch size, and number of epochs alone because we didn't find a huge difference messing with them on the trycifar and trynmnist datasets from their default values (although it's fair that hyperparameter differences might have been amplified in imagenet!)

Google collab killed our longest run (running out of space), but we honestly could've run it for fewer epochs.

3: graphs

We aimed to train for 100 epochs, although it was killed at epoch 83 by collab for running out of memory. It seems clear that theres some overfitting that starts around epoch 5!



Weirdly enough, around epoch 5 the accuracy is **not** capping out. We reach a threshold of 39%, and occasionally bump up to 40%!

4. final stats

Training Loss	Test Loss	Test Accuracy
0.034331	3.2329	39%

- What design worked the worst (but still performed better than random chance)? Provide all the same information as question 1.

We tried a ton of things and stayed stagnant at 5% for a really long time. This was better than random chance (0.5%), but still quite worse than the expected $\sim 40\%$.

This was mentioned in Q1, and **everything** else was the same in Q1 sans batch normalization and logsoftmax (we used vanilla softmax). Changing from softmax to logsoftmax immediately upped the asymptotic accuracy of our model from 5% to 35%!

- Why do you think the best one worked well and the worst one worked poorly.

Truthfully, not super sure on the softmax $\rightarrow$ logsoftmax. Other things: decreasing learning rate allows us to trust the knowledge we've acquired the longer we've been learning for.

We reduced the number of layers and played around with dimensions, trying to balance speed of training with avoiding losing too much information.

We experimented a lot with maxpooling, as again reducing dimensions (increasing speed) but being careful with it (limiting information loss).

We found the batch normalization was important to avoid outliers in a given batch skewing the results a ton.