Assignment 2

Shaochen (Henry) Zhong
COMP 576, Fall 2022, by Prof. Patel

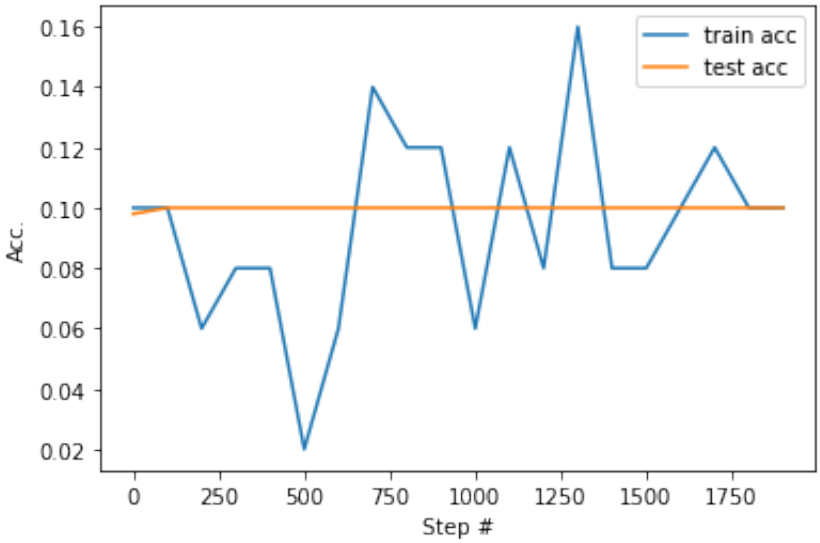
1. Visualizing a CNN with CIFAR10

a) CIFAR10 Dataset

Please refer to `trainCifarStarterCode.py`. Note I tweaked the data-loading process a bit to accelerate the speed, but I have confirmed the loaded data are identical.

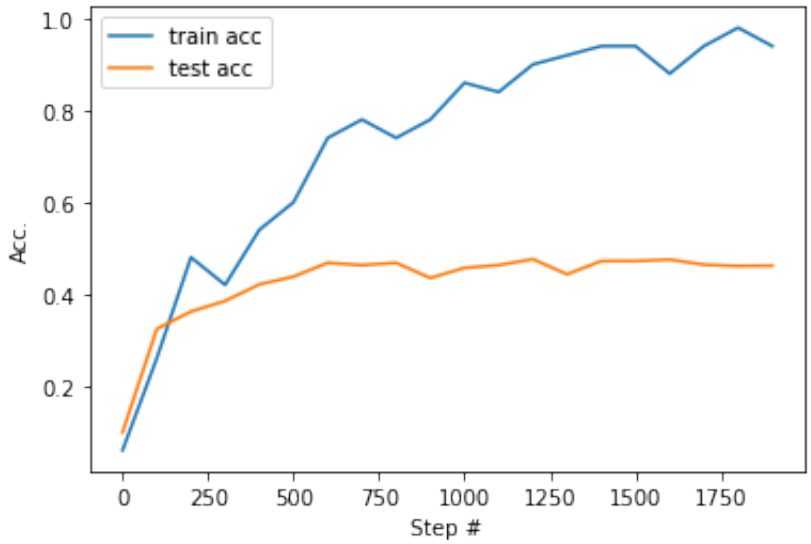
b) Train LeNet5 on CIFAR10

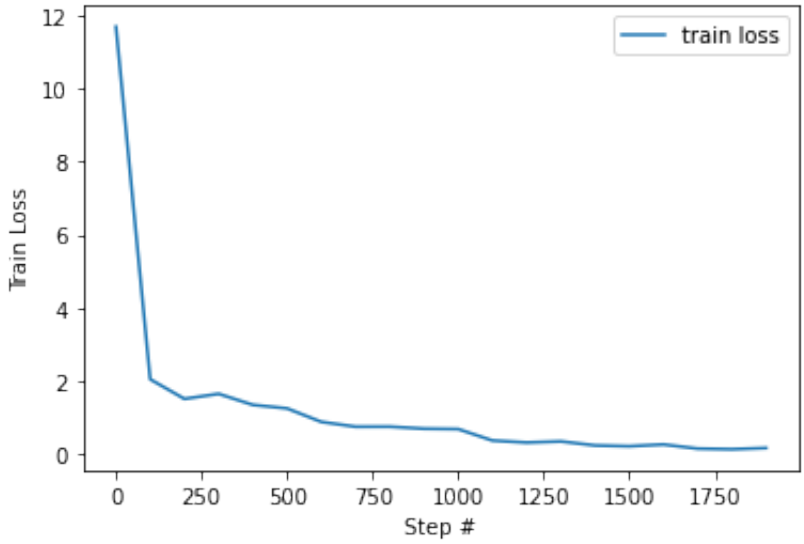
`optimizer = adam, lr = 1e-2`
Final test acc is `0.1`



The train loss are all `nan` so I won't show here.

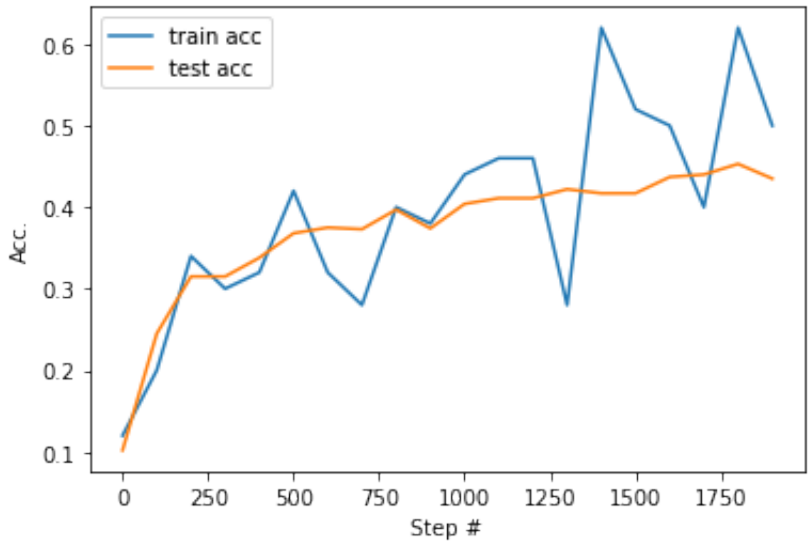
`optimizer = adam, lr = 1e-3`
Final test acc is `0.461`

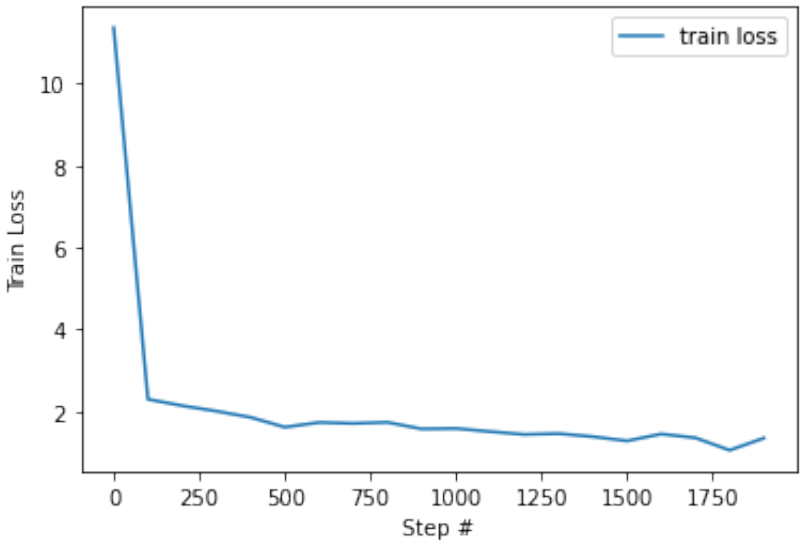




`optimizer = adam, lr = 1e-4`

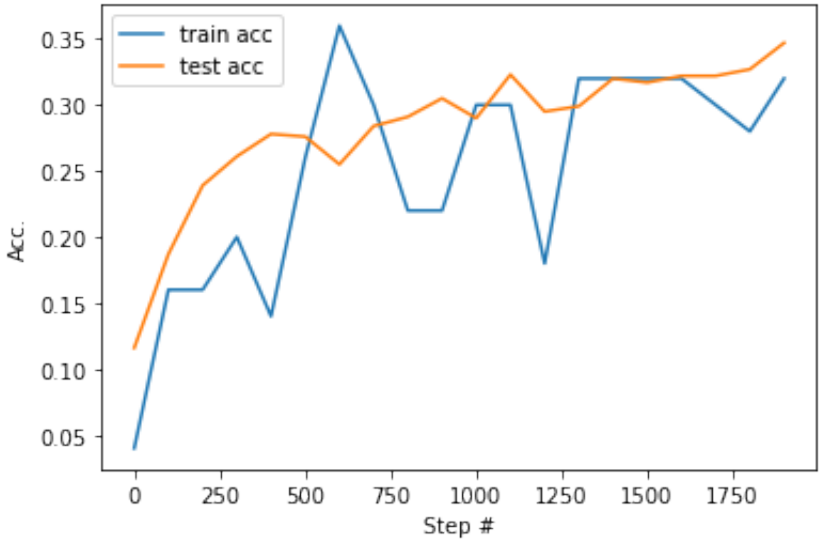
Final test acc is `0.448`

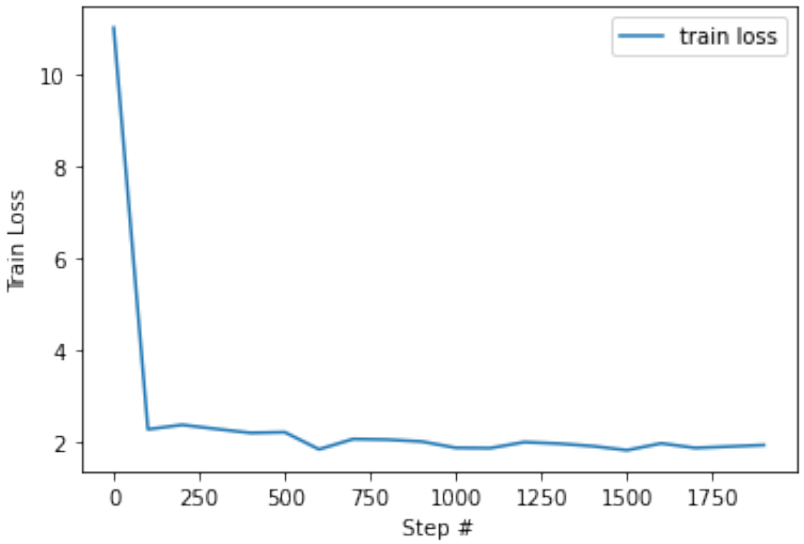




`optimizer = SGD, lr = 1e-2`

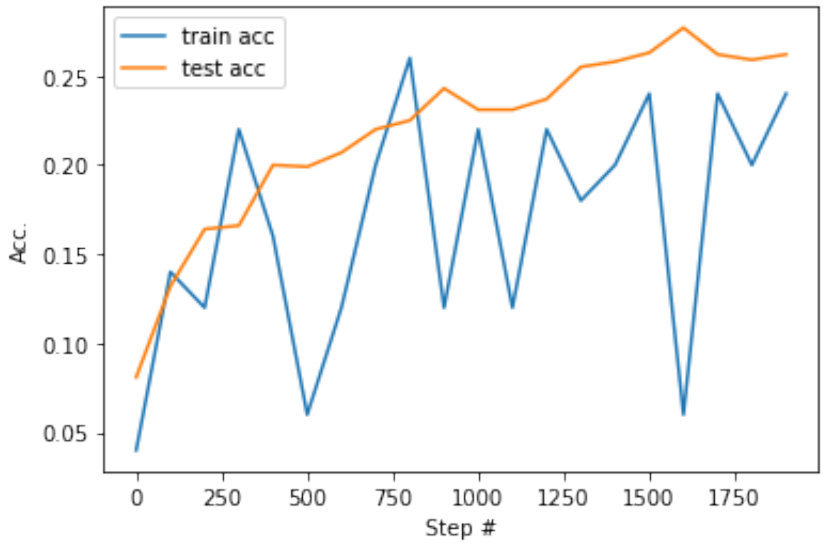
Final test acc is `0.359`

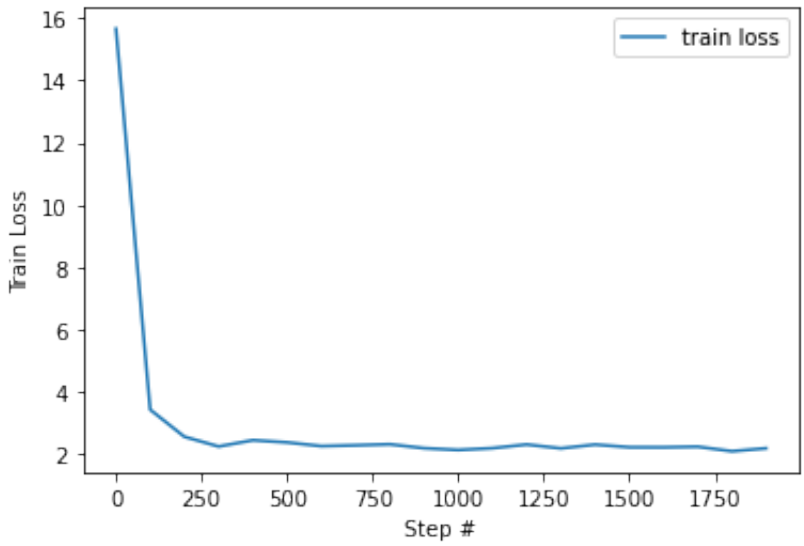




`optimizer = SGD, lr = 1e-3`

Final test acc is `0.267`

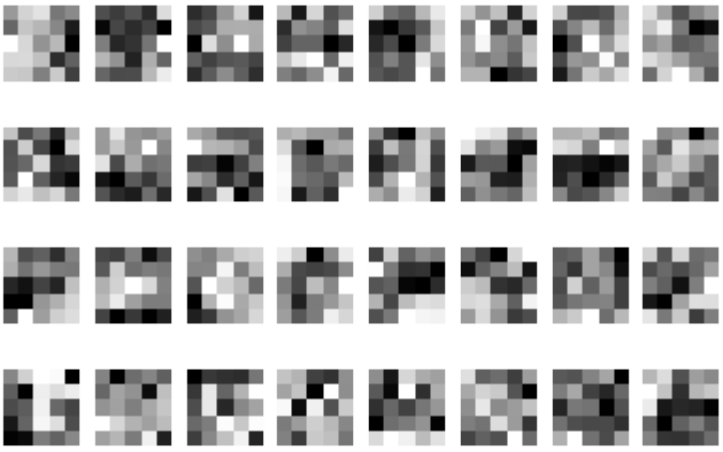




The best tried optimizer-hyperparameter combination seems to be `optimizer = adam, lr = 1e-3`. I also noticed the model will not converge if given a much larger epoch #, which suggests some sort of adaptive learning rate settings should help.

c) Visualize the Trained Network

Visualization of weights within the first `conv` layer:



Statistics of activations:

```
h_conv1_stats: mean = 0.09040263295173645; var = 0.016252201050519943
h_conv2_stats: mean = 0.014666423201560974; var = 0.009365998208522797
```

The above plot/stats are generated based upon the configuration of `optimizer = adam, lr = 1e-3`.

2. Visualizing and Understanding Convolutional Networks

The presented paper is an early work within the field of CNN visualization/explanation, the motivation is to provide a better understanding and potentially diagnose the seemly blackbox CNN network.

The authors utilized deconvnet: a convnet model consisting of standard convnet operations with the original purpose being reversing a standard convnet, a.k.a. doing the opposite of mapping pixel to features. The authors attached deconvnets upon every layer of the target CNN model, and by passing the feature map of each particular conv layer into their attached deconvnets, the proposed method may obtain a continuous path to image pixel.

The experiments are conducted upon some early CNN models and ImageNet 2012. Visualization upon feature evolution is shown, which suggests early layers of CNN recognize basic features (e.g., edges & patterns) whereas final layers recognize shapes correlated to the prediction label. The author also conducted interesting experiments to explore convnet behaviors under some special setups, such as feature occlusions.

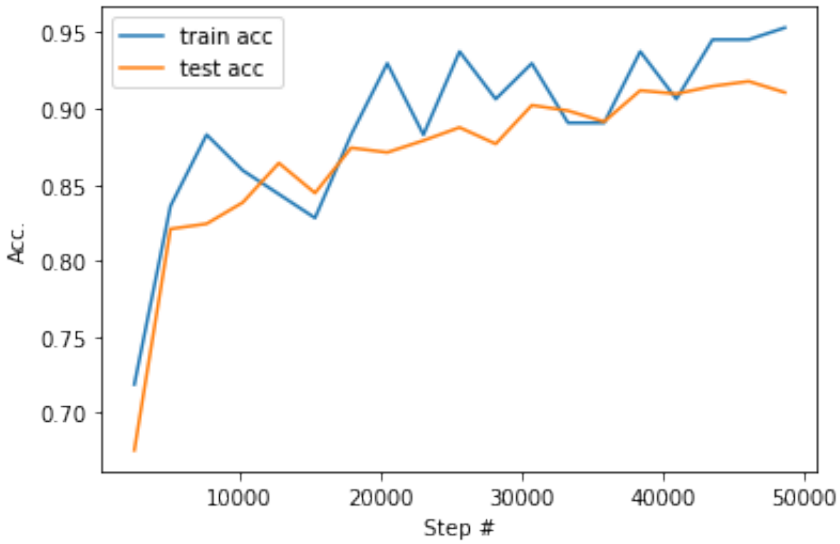
3. Build and Train an RNN on MNIST

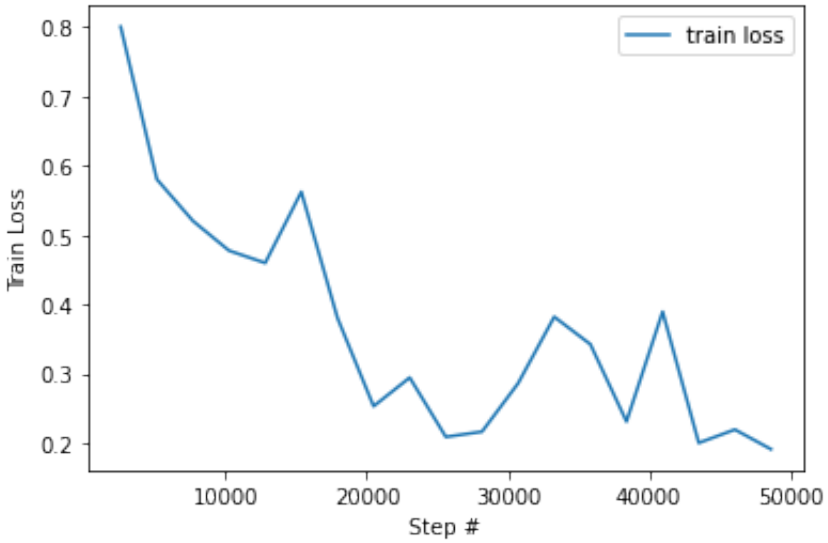
a) Setup an RNN

Please refer to `lstmMNISTStarterCode.py`

b) How about using an LSTM or GRU

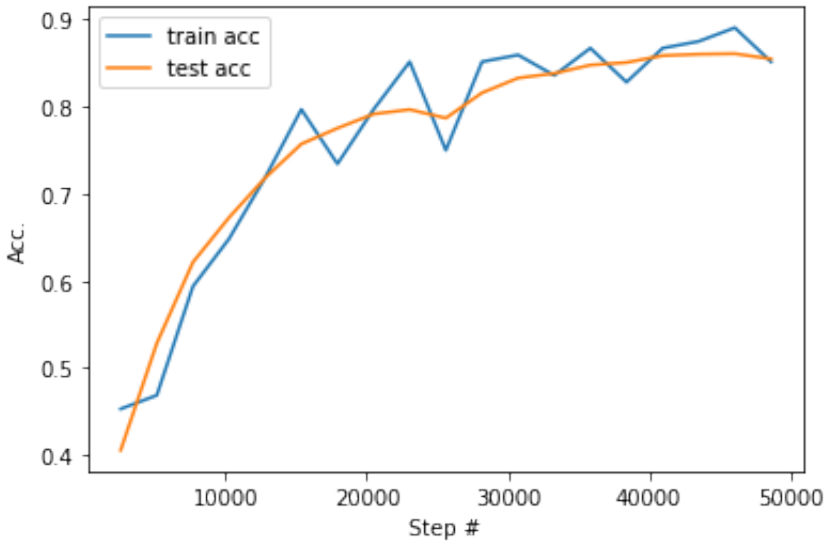
`BasicRNN, Adam, lr = 1e-3, nHidden = 256`
Final test acc is `0.9092`

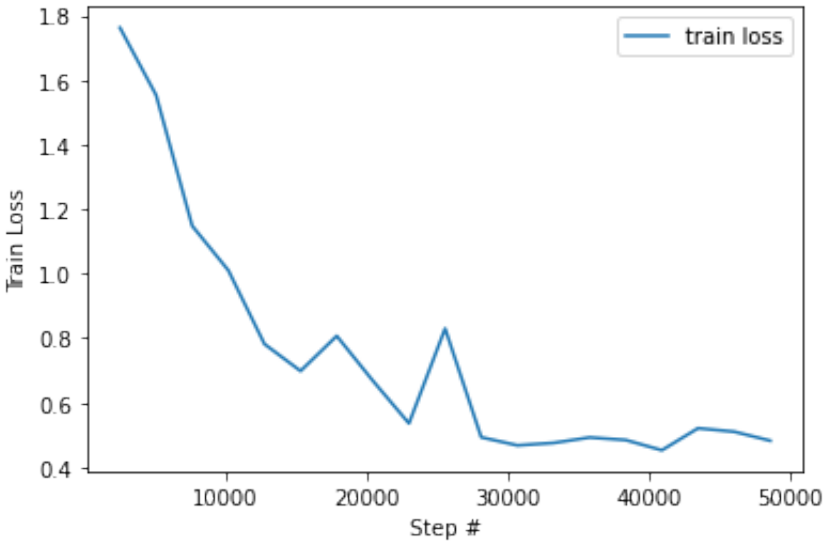




`BasicRNN, Adam, lr = 1e-3, nHidden = 64`

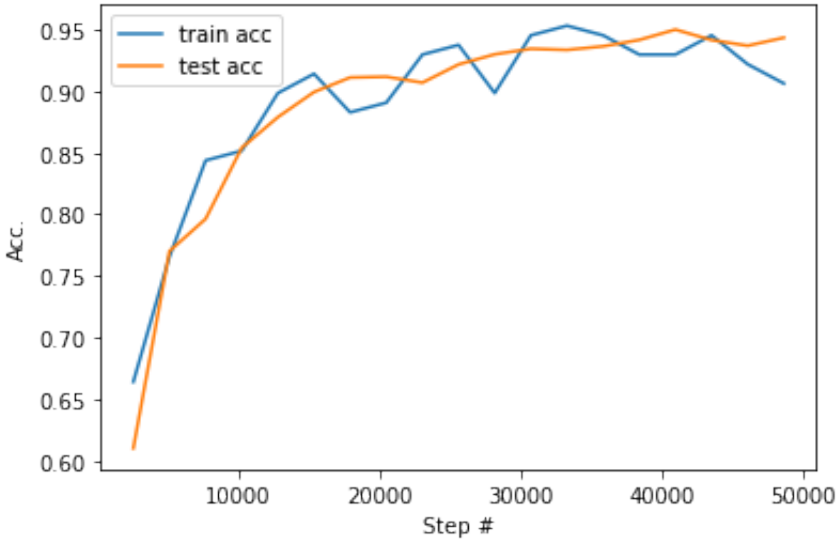
Final test acc is `0.8713`

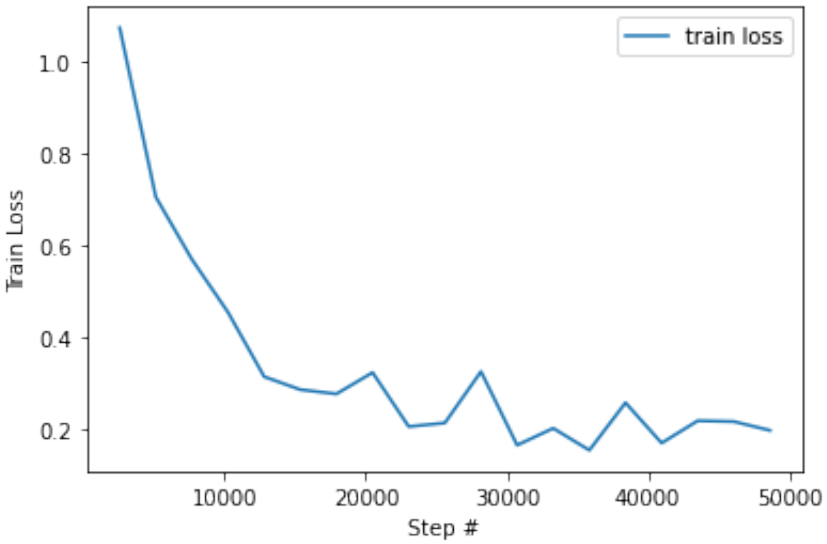




`LSTM, Adam, lr = 1e-3, nHidden = 256`

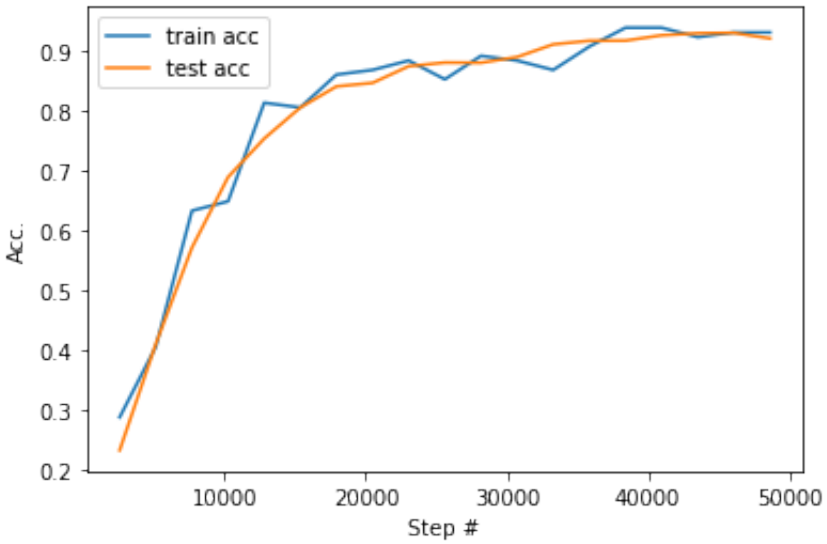
Final test acc is `0.9533`

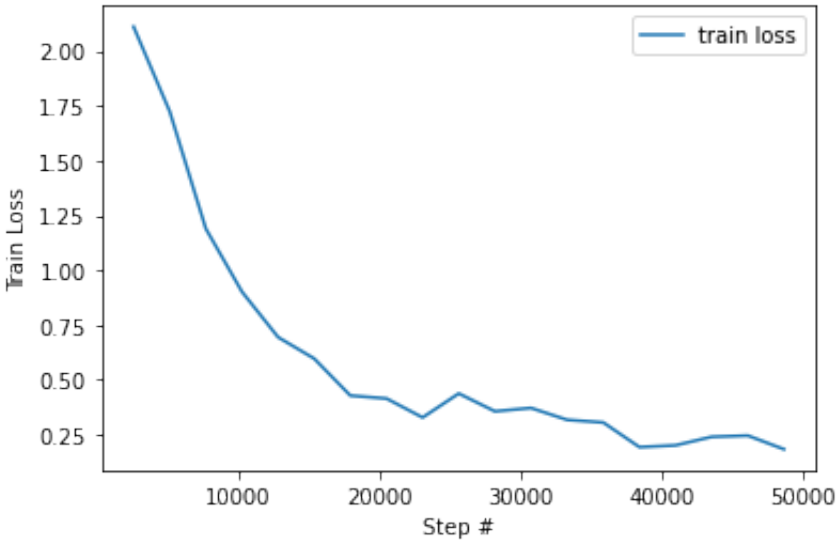




`LSTM, Adam, lr = 1e-3, nHidden = 64`

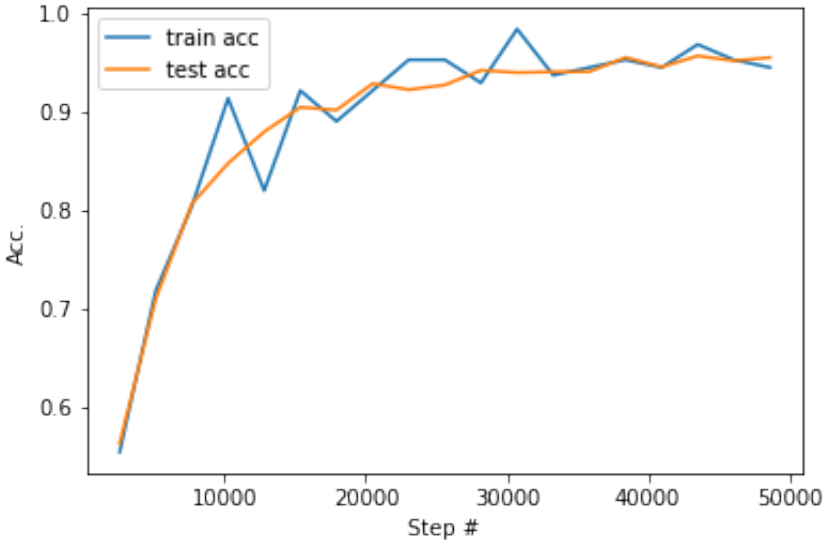
Final test acc is `0.9255`

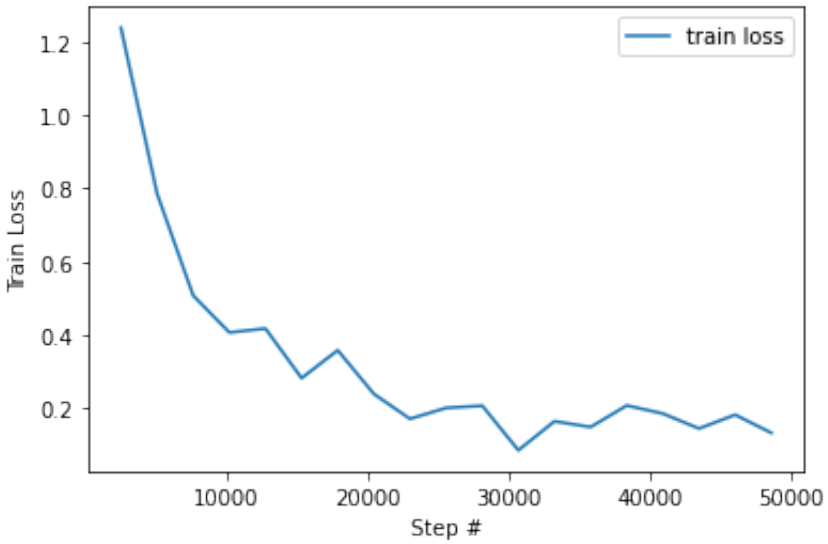




`GRU, Adam, lr = 1e-3, nHidden = 256`

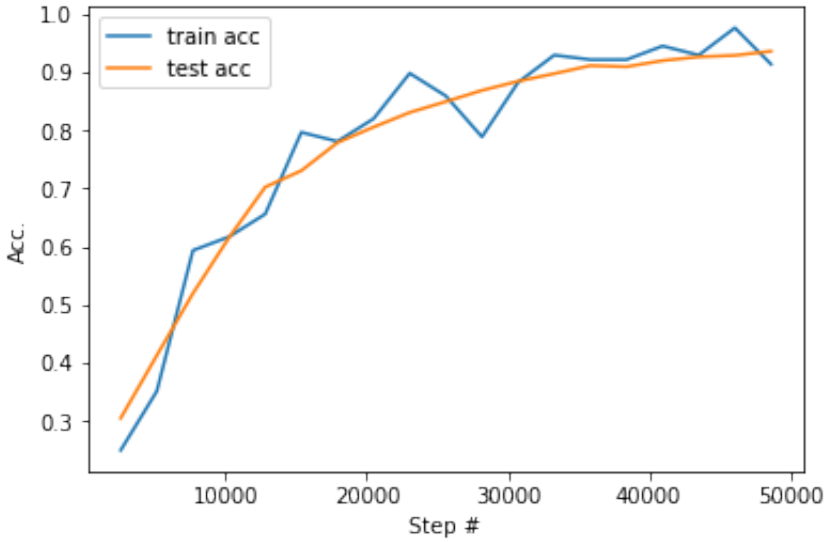
Final test acc is `0.9595`

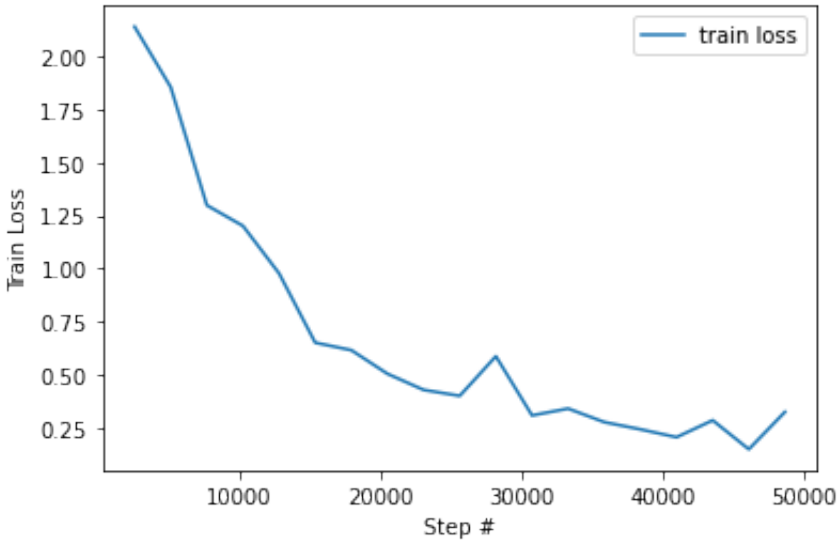




`GRU, Adam, lr = 1e-3, nHidden = 64`

Final test acc is `0.9333`





It looks like `GRU` and `LSTM` are indeed better than the standard `RNN` with the best configuration being `GRU, Adam, lr = 1e-3, nHidden = 256`. The increase of `nHidden` also helps provide better generalizability.

(Presented test accuracies are per each key iteration, the original code only test the final one but here I monitor this throughout the training)

c) Compare against the CNN

Training-wise, it looks like RNN is a lot more stable to train. As you get to play with some aggressive hyperparameter settings yet the network will still nicely converge.

Mechanism-wise, I think the similarity lies upon the fact that both CNN and RNN introduce inductive bias to limit their search space: where CNN assumes locality/translation invariant yet RNN assumes connection between nearby inputs. The two main differences are probably CNN is feed-forward where RNN is not. Also, RNN can deal various types of input, where a CNN is molded to a particular input size/format.

Reference

I have referred to the following two repos (1, 2) for intuitions and debugging reference. All materials presented here are work of my own, please refer to my github repo should you need more information for grading.