Homework 07 – Training

Arthur J. Redfern
arthur.redfern@utdallas.edu
Mar 06, 2019

---

0 Outline

1 Logistics

Assigned:	Wed Mar 06, 2019
Due:	Wed Mar 13, 2019
Format:	PDF uploaded to eLearning

2 Reading

1. [Optional] The following papers are part of a **very** recent wave of results that challenges relatively recently (or still) held beliefs. Some of these papers are quite dense and it's easy to get lost in the math (don't get dismayed, this happens to everyone). For our purposes here, focus on the text and key results.

> Dynamical isometry and a mean field theory of CNNs: how to train 10,000-layer vanilla convolutional neural networks
> https://arxiv.org/abs/1806.05393
>
> Rethinking ImageNet pre-training
> https://arxiv.org/abs/1811.08883
>
> How does batch normalization help optimization?
> https://arxiv.org/abs/1805.11604

Fixup initialization: residual learning without normalization
https://arxiv.org/abs/1901.09321

Image classification at supercomputer scale
https://arxiv.org/abs/1811.06992

Complete

3 Theory

2. Rework problems 26, 27 and 28 from Test 01 – Math. Be able to work these confidently without looking at the solutions.

See posted test solution

4 Practice

Training is ultimately about

- Accuracy
- Performance

Accuracy depends on convergence and generalization, performance means speed. This homework will look at both in the context of a slightly larger dataset and also provide some continuing practice with network design. If you have individual access to a system with a fast GPU, use that; if not, use Google Colab with a GPU runtime.

3. Download Tiny ImageNet and convert it into TFRecords where the number of shards is configurable. If you're using Google Colab, put the TFRecords on Google Drive so you don't have to download and convert the Tiny ImageNet dataset at the start of every session. See the provided Tiny ImageNet data script for details.

See posted Tiny ImageNet data script

4. Understand how to incorporate Tiny ImageNet into a network training and testing script. See the Tiny ImageNet code script for details.

See posted Tiny ImageNet vision classification script

5. Modify the sequential networks and ResNet network to add an additional down sampling stage and an additional level of processing. This is practical because CIFAR-10 uses 3x32x32 inputs and Tiny ImageNet uses 3x64x64 inputs. Effectively, a Tiny ImageNet network is like a standard ImageNet network without the tail (which includes 2 levels of down sampling). Parameterize the number of repeats of the 3rd level of the newly added residual blocks identically to levels 0, 1 and 2. Optionally, also make a 1/2 width version.

See posted Tiny ImageNet vision classification script

6. Add the ability to save the validation loss and validation accuracy on a per epoch basis. Additionally, add a plot of these values vs epoch after training. This is useful for optimizing training hyper parameters.

This question should have read "save the **training** loss and validation accuracy" (sorry about that)

7. It's a good idea to have a rough idea of how long an epoch of training should take, this is ideally estimated vs a known value. Practice this by estimating how much slower you expect an epoch of training to be for the Tiny ImageNet dataset and network vs the CIFAR-10 dataset and network. Consider input data volume and network size as a 1st pass.

CIFAR-10 is 50000 training images of size 3x32x32, Tiny ImageNet is 100000 training images of size 3x64x64. Additionally, a Tiny ImageNet network will likely have 1 additional module vs a CIFAR-10 network (e.g., 4 vs 3).

An estimate for how much longer an epoch of Tiny ImageNet than CIFAR-10 is:

(4 from image size)*(2 from number of images)*(1.33 from network complexity) = 10.67

So if we're ball park estimating and not accounting for fixed times, you'd expect an epock to take ~ 10x more time for Tiny ImageNet than CIFAR-10.

8. Verify that training starts and there are no bugs, then kill training after an epoch or 2 completes. Now you have a base to work with and can proceed to the following questions.

See posted Tiny ImageNet vision classification script

9. You want training to be as efficient as possible. How many shards (and what shard size) is ideal for you system? Try a few values and see if there's a noticeable difference in time. If you're using Google Colab and you copy the TFRecord shards from Google Drive to the Google Colab runtime before starting training does that improve performance?

Optimal number of shards = ?

Is it better to copy the TFRecords from Google Drive to Google Colab before training?

10. Little details matter, therefore automate them so you don't hard code them incorrectly. As an example, automate the following 1x activities that are done before training:
    a. Computing the number of images in the training and testing data sets (used as parameters in the main training and testing code)
    b. Computing the per channel mean and variance calculation (used in pre processing)

An alternative place for these calculations would be in the script that creates the TRRecord shards.

See posted Tiny ImageNet data script

11. Data matters, especially as networks get larger. Add additional data augmentation methods into the training pre processing. How do they affect performance? As a suggestion, look to the ResNet paper for inspiration and also do general ImageNet data augmentation searches.

A few data augmentation methods are shown in the script; something to think about as you add additional methods is how they affect the mean and std dev of the data if using batch norm for training as you're using different pre processing from training vs validation and potentially different input statistics

12. Training hyper parameters matter a lot and unfortunately can be tricky to tune. What is the optimal batch size? Try a few. What is the optimal initial learning rate and schedule? Try a few. Does adding L2 regularization help? Try a few weightings. Does adding dropout to the decoder help? Try a few dropout percentages. Note that some of these are easier to add / try than others.

This is experimentation for the particular problem of interest; the default settings in the script were relatively quickly generated, it's likely that there's a better default combination

13. What is your overall configuration that results in the highest accuracy? What is the accuracy?

The example network gets to the mid to high 50s % top 1 accuracy training from scratch with the provided parameters in ~ 8 hours on Google Colab

14. [Optional] Add and demonstrate check pointing to allow training to be interrupted and restarted. In a practical development flow where trainings can last multiple days, this is incredibly important.

15. [Optional] Add and demonstrate saving the trained network, then loading it and applying it to test data. In a practical development flow, this is how you distribute your network to others.

16. [Optional] Go back and create a similar download script, application integration code and explore training again for CIFAR-10 and associated networks.