11-695: AI Engineering
Assignment 1: Supervised Learning

Spring 2020

Abstract

In this assignment, you will implement an image classification system in TensorFlow [Abadi et al., 2015]. The starter code is provided for you, where a naive classifier has been implemented. Your job is to understand the starter code and implement two models of your choice: a feed-forward neural network, and a convolutional neural network. For each network, you have the freedom to explore different architectures, to implement, to train, and to test your model. As a starting point, you will be working with the CIFAR-10 dataset [Krizhevsky, 2009], where we expect your best method to reach 80% accuracy. Next, you will be dealing with a little more challenging problem by training the Tiny Image-Net[1] dataset. Finally, you are required to work with a more real-world dataset called IMGS.

1 Code and Dataset

Install TensorFlow (of version at least 2.0). If you have not already, please navigate to TensorFlow's site and follow their instructions to install the framework. The instructions are at

`https://www.tensorflow.org/install`

Their instructions should be sufficient to install TensorFlow and all of its dependencies on your system. It is possible to complete this assignment without using any GPU, so you do not need to install CUDA or anything related to GPU programming. However, if you wish to, you are encouraged to install TensorFlow GPU (See "Additional setup" section from the link above), which will make your implementation much faster, hence saving much of your time. Otherwise, another fast and convenient solution is to use some public deep learning AMI on AWS.

Download the CIFAR-10 Dataset. You will be working with the CIFAR-10 dataset, which is available at

`https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz`

This dataset consist of $50,000$ training images and $10,000$ testing images. All images are RGB images with the size of 32×32. Each image is assigned one of 10 pre-defined labels. You can find out more about the CIFAR-10 dataset in the paper by Krizhevsky [2009].

One of the best performances on the CIFAR-10 dataset is 97.7% accuracy, which is achieved by Neural Architecture Search [Zoph et al., 2017]. In this assignment, your best model is required to reach 80% accuracy.

The starter code implements some data preprocessing procedures for you. In particular, we reserve $5,000$ training images for validation. Thus, there are $45,000$ images for training. We have also subtracted the channel mean and divided the channel standard deviation from all images. These are the standard data preprocessing steps for the CIFAR-10 dataset.

[1] https://tiny-imagenet.herokuapp.com/

Download the Tiny Image-Net Dataset. The second dataset you will be working with is the Tiny Image-Net Dataset, which is available at

https://tiny-imagenet.herokuapp.com

This dataset consists of 200 classes, 500 training images and 50 validation images per each class. To make the problem easier for you, we narrow the scope down to 50 classes and already parse the data into 450 train images, 50 validation and 50 testing images per class.

IMGS Dataset. The final dataset IMGS is saved into a pickled file in a separate folder `imgs`, so you do not have to download it. We also have the dataset loader available for you in the `data_utils.py` file.

Starter code. The starter code for your project is located at

http://aieng.cs.cmu.edu/assignment.html

After downloading and unzipping the code and the CIFAR-10 data, you can navigate to your code. To make things simple, please place your uncompressed CIFAR10 data into the folder `cifar10` and Image-Net data into the folder `imagenet`. The respective drivers for each code are the corresponding `main.py` files.

The starter code (for each dataset) implements an extremely simple baseline algorithm: a naive classifier. This algorithm performs the following steps:

- Flattens each CIFAR-10 images into a vector of 3072 numbers,
- Applies a matrix multiplication to turn the image into a logit vector of 10 numbers,
- Applies the softmax function on the logits to obtain the output class distributions.

This softmax classification is trained using stochastic gradient descent (SGD), with a fixed learning rate, which is not a very good setting for SGD. If you run the starter code, which trains this model for only 1000 steps, you will get around 26% accuracy on the test data.

2 Your Work

The driver of the program is in the file `main.py`. From this file, the relevant computational graphs are from `models.py`. It is your responsibility to read the code and figure out all the relevant points. We note the sections to implement/change by the `#TODO` signals. You are free to implement however you want, but it is *prohibited* to use the contained implemented networks such as `LeNet, VGG16, VGG19, ResNet, etc.` from internet. Likewise, you are expected to implement each FC, Conv2d, Pooling and Dropout Layers completely by yourself, using the provided TF libraries (including–and strongly recommended–`tf.keras` layers).

There are 3 files to implement in general for each dataset:

1. `data_utils.py`: implement TF batched dataset object
2. `models.py`: implement inference model, i.e. taking the images tensor and return the corresponding logits.
3. `main.py`: implement the driver for dataset, training and evaluation.

You are required to implement at least two models: feedforward network (or `mlp`) and a convolutional network to make the model have as high accuracy as possible on validation and test datasets. You can use any techniques that you desire, including but not limited to:

- ℓ_1-regularization, ℓ_2-regularization, weight decaying, ...
- Activation functions: `tf.sigmoid`, `tf.tanh`, `tf.nn.relu`, etc.
- Momentum training [Nesterov, 1983]. The `TF` code for this is: `tf.train.MomemtumOptimizer`,
- DropOut [Srivastava et al., 2014],
- Batch Normalization [Ioffe and Szegedy, 2015]. For `TF` code, you can use `tf.nn.fused_batch_norm`,
- Residual connections [He et al., 2016], Highway connection [Greff et al., 2017],

You can also use any number of layers and architectures. Remember that networks with more layers are harder to train, but if you can train them appropriately, you can *usually* get a better performance.

In summrary:

- For `Cifar10`: you are required to have 2 separate models: `mlp` and `conv`. You have to pass the baselines of 45% and 80%, respectively, for top 1 accuracy.
- For `Tiny Image-Net`: you are only required to have 1 model, which is expected to be a deep CNN. *Hint*: you can adjust VGG16 to a shorter version to cope with the small resolution. Also note that we are requiring you to train on 50 classes, and you have to pass the baseline of 40% on the test dataset for top 1 accuracy. You can also increase the number of class (the constant `N_CLASSES_LIMIT`) to more than 50 to be considered for bonus points.
- For `IMGS` dataset: this one is more challenging but we also require you to do the followings:
  - Analyze the dataset and report the statistics (with tables, charts, ...)
  - Implement one model as in `Tiny Image-Net` and have to pass the same baseline
  - Report your techniques to deal with this dataset
  - Report your results, important observations and comparisons with `Tiny Image-Net`.

Important Note: Please be noticed that for each dataset, you are required to report both **top1** and **top5** accuracies.

3 Reports

After you have done, you should have respective train log files and more importantly, we expect you to turn in a report detailing the results and analysis. The report is of free format, but ideally should have a summary in tables for respective validation and test accuracies for each model, per each dataset.

To simplify your work, please limit your report to 3 pages maximum.

4 Gradings

The grading breakdowns are as follows with totally 100 points with potential 10 extra credits:

- CIFAR10 Feed-forward network: **15**
- CIFAR10 Convolutional network: **25**
- Tiny-ImageNet network: **20**
- IMGS network: **15**
- Report on top 1 and 5 accuracies for 3 datasets: **5 + 5 + 5**
- Report on IMGS techniques: **10**

As usual, we urge you to start early and acquire help from instructors soon.

5 Submission

All submissions must be to Canvas individually, including:

- Your folder of code, excluding dataset.
- Your real log of running each of your models. You can export log of running simply by using `> log_cnn.txt` after your command, or by using this syntax: `python3 main.py 2>&1 | tee log_cnn.txt`.
- Your report.

Note: Your code must be runnable directly with Python3 and Tensorflow 2.0. Any extra packages might be super useful but is prohibited, and should not be needed. But if you insist to use any of new packages on the internet, check with the instructors first. Furthermore, if your code has a special instruction to run, please detail it. We will give you zero for each part where your code is not runnable. The TAs will be running each of your code so please be considerate to them as well.

6 Academic Integrity.

As normal, you are encouraged to discuss with your friends and the instructors. Anything they tell you, you can use. However, looking at other's code should not happen at all cost.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Klaus Greff, Rupesh K. Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *ICLR*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CPVR*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Yurii E. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 1983.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. In *JMLR*, 2014.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *Arxiv, 1707.07012*, 2017.

Revision

1. Feb 4: First release