Project Proposal: Using Bayesian Optimization to Find Good Augmentation Policies from Data

NAMES: Samuel Frommenwiler, Gian König, Colin Kälin
NETHZ: fsamuel, koenigg, ckaelin
EMAIL: {fsamuel, koenigg, ckaelin}@student.ethz.ch
ID: 08-738-601, 09-913-245, 14-935-118

Abstract

This document states the proposed content of the project for the deep learning course at ETH as required in [?]. The goal is to investigate Baysian Optimization to tune hyperparameters for automated data-augmentation policies.

1. AutoAugment with Reinforcement Learning

Machine learning algorithms usually achieve better results with more data, however acquiring this additional data can be expensive and time-consuming. A common trick to increase the amount of training data is to add copies of existing data with small perturbations to the training set. For a dataset of natural images dataset augmentation methods include random cropping, image mirroring, rotation, color shifting and color whitening. Picking a good combination of these methods is usually done by hand and requires expert knowledge and time.

Therefore, an automated approach was introduced in order to find the best policies [?]. Cubuk et. al. present a process of finding an efficient data augmentation policy, in which each policy contains possible augmentation operations. Each operation contains an image processing function (e. g. translation, rotation or color normalization) combined with a probability that this function is applied with a corresponding magnitude. To find the best choices of these functions and suitable scaling factors, Cubuk et. al. use a reinforcement learning search algorithm such that a neural network, trained on these hyperparameters, yields the best validation accuracy. Geng et al. [?] extended on this idea using augmented random search to efficiently find augmentation policies in a continuous hyperparameter search space. Tran et al. [?] follow a slightly different approach where they use GANs to find valid augmented data. While Fawzi et al. [?] pursued a worst-case augmentation approach where they generate augmented data which gives the biggest loss for the current classifier.

2. AutoAugment with Baysian Optimization

We intend to use Bayesian optimization, a technique already used for tuning hyperparameters, to automatically select a good combination of augmentation policies. And to compare this to the approach used in [?]. Specifically, we are going to investigate the Bayesian optimization [?], [?] approach with a Tree Parzen Esimator [?] by using the Hyperopt library [?] with the help of [?]. For this specific task, [?] gives us a guideline on how to integrate Bayesian optimization into the data augmentation task. We will focus on using a single network architecture (WideResNet), using the same hyperparameters as the authors of the original paper [?], changing the search space from discrete to continuous and utilizing Bayesian optimization to select the optimal combination of augmentation policies. The questions we would like to answer are:

- How does our approach compare to the 2.68% baseline on CIFAR-10 with WideResNet that Cubuk et al. achieved and to the result that Geng et al. got?
- Can we find a comparable augmentation policy in less iterations or are we able to surpass the result that they got?

We started to investigate the code and already run a few tests with the CIFAR-10 dataset and the Wide-ResNet architecture [?]. On a MacBook Pro (macOS Sierra 10.12.5, 2.4 GHz Intel Core i7, 16 GB 1600 MHz DDR3, Intel HD Graphics 4000 1536 MB), one epoch takes around 5.5 hours to complete. On the Google Cloud [?] using one Tesla P100 GPU we were able to achieve the baseline presented by Cubuk et. al. in 9h15min.

The next steps will be to make the search space continuous and integrate the Hyperopt library. Once the setup is working we want to see if we can find an equally good augmentation policy in less iterations or running the process for a longer period if we can find a better optimum in our continuous search space.

3. Infrastructure

A first attempt to run the code on the CPU: MacBook Pro (Retina, 15-inch, Early 2013) with the following specifications 2.4 GHz Intel Core i7, 16 GB 1600 MHz DDR3, Intel HD Graphics 4000 1536 MB, macOS Sierra 10.12.5. This resulted in a Training Time: INFO:tensorflow:Finished epoch: 0, INFO:tensorflow:Epoch time(min): 332.059691219.

We tried to execute the code on the Leonhard Cluster[1]. For this the following commands were necessary. 1) ssh koenigg@login.leonhard.ethz.ch 2) Create folders and git clone and upload the data from PC rsync -Pav /Dropbox/CAS/DL/DL18/autoaugment/tmp/data/ koenigg@login.leonhard.ethz.ch:/cluster/home/koenigg/DL_Project 3) module load python_gpu/2.7.13 (module list: Currently Loaded Modules: 1) StdEnv 2) gcc/4.8.5 3) openblas/0.2.19 4) cuda/8.0.61 5) cudnn/6.0 6) jpeg/9b 7) libpng/1.6.27 8) python_gpu/2.7.13) 4) Execute the code: bsub -n 20 -R ”rusage[mem=4500,ngpus_excl_p=1]” python train_cifar.py –model_name=wrn –checkpoint_dir=tmp/training/ –data_path=tmp/data/ –dataset=’cifar10’ –use_cpu=0. It turns out on the Leonhard Cluster we can only use 1 GPU which is not enough for our dataset.

[koenigg@lo-login-02 model]$ ls -l total 570652 -rw-rw—- 1 koenigg koenigg-group 127 Jan 8 13:49 checkpoint -rw-rw—- 1 koenigg koenigg-group 289836892 Jan 8 13:44 model.ckpt-40.data-00000-of-00001 -rw-rw—- 1 koenigg koenigg-group 9868 Jan 8 13:44 model.ckpt-40.index -rw-rw—- 1 koenigg koenigg-group 1163579 Jan 8 13:44 model.ckpt-40.meta -rw-rw—- 1 koenigg koenigg-group 289836892 Jan 8 13:49 model.ckpt-41.data-00000-of-00001 -rw-rw—- 1 koenigg koenigg-group 9868 Jan 8 13:49 model.ckpt-41.index -rw-rw—- 1 koenigg koenigg-group 1163579 Jan 8 13:49 model.ckpt-41.meta

[1] https://scicomp.ethz.ch/wiki/Leonhard