Exercise 4 Report **Ahmed Abdelhadi**
Deep Learning Lab Course
Lab Date: 04/12/2018
Due Date: 08/01/2019

Introduction

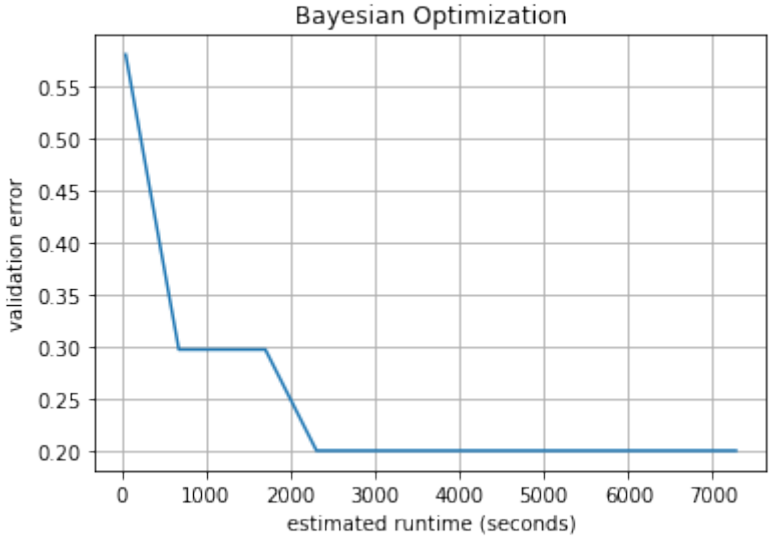
In this exercise, we explored both Bayesian optimization and Hyperband and the methods that we can use to combine them to optimize the hyperparameters of a convolutional neural network (CNN) on CIFAR-10 dataset. The architecture consists of 3 convolutional layers (with RELU activations and batch norm) and one final fully-connected layer. We use Adam to optimize the weights of the network.

Due to the fact that training CNNs is computationally expensive, it was decided to optimize a so-called surrogate benchmark instead of the original benchmark. A surrogate benchmark is basically just a regression model (in our case a random forest) that was trained on a large set of randomly sampled hyperparameter configurations of the original benchmark (CNNs in our case).

Bayesian Optimization

The main loop for Bayesian Optimization consisted of: 1. optimizing the acquisition function, 2. evaluating the objective function 3. augmenting our dataset and updating our model. Additionally we will also keep track of the current best solution we have found (dubbed incumbent) which BO will ultimately return to the user. The incumbent in our case is simply the best observed configuration.

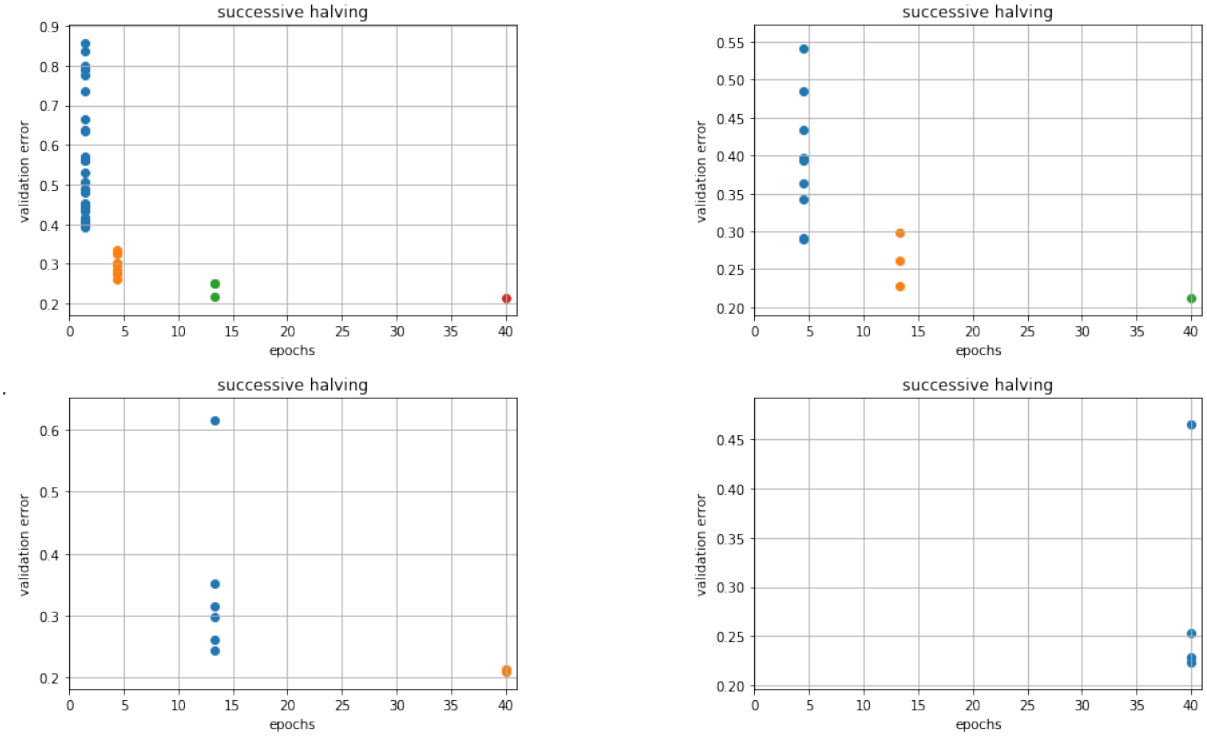
The following graph shows the error for the incumbent against the time needed to reach it.



Hyperband

Hyperband draws random configurations on a specific schedule of iterations per configuration, using earlier results to select candidates for longer runs.[1]

In the exercise, we use successive halving which is a simple bandit strategy to allocate resources to a fixed set of configurations.



[1] http://fastml.com/tuning-hyperparams-fast-with-hyperband/

BOHB

One of the drawbacks of Hyperband is that it draws configurations randomly and hence might take exponentially long to approach the global optimum. In the last part of the exercise, we will combine Hyperband with a kernel density estimator that models the distribution of the good and the bad configurations in the input space. By sampling from this model instead of a uniform distribution we can find good configurations much faster.