Deep Learning Lab Course
Machine Learning and Computer Vision Track
Exercise 5

Aaron Klein
(send the report to kleinaa@cs.uni-freiburg.de)

Due: December 19, 2017

In this exercise we will use random search and Bayesian optimization to find a good hyperparameter setting $\mathbf{x} \in \mathcal{X}$ for a 3 layer convolutional neural network on CIFAR 10. As in the previous exercises, please send us a small report (1 or 2 pages) which contains the below described plots as well as a link to your code. Send us an email if you encounter any issues with installing RoBO or using the surrogate.

Training a convolutional neural network on CIFAR10 can be quite expensive. In order to avoid that you wait for several hours until the hyperparameter optimization has finished, we will make use of a surrogate in this exercise. The idea of a surrogate is to first evaluate N hyperparameter configurations $y = f(\mathbf{x})$ in an offline step to generate a dataset $D = \{(\mathbf{x}_0, y_0), \ldots, (\mathbf{x}_N, y_N)\}$ and then to fit a regression model $\hat{f}(\mathbf{x})$ on D that approximates the true objective function $f(\mathbf{x})$. Afterwards, we can evaluate the regression model (i.e surrogate) which takes only a few seconds rather than evaluating the true objective function (which may take several hours). Of course, the surrogate is just an approximation of the true objective function and, hence, most likely contains some mismatch. But to understand how hyperparameter optimization works and develop new methods, this approximation is usually sufficient.

To generate this surrogate, we sampled several hundred random hyperparameter configurations, used them to train a convolutional neural network and saved the validation error as well as the runtime. We trained one random forest to predict the validation error given a hyperparameter configuration and one random forest to predict the runtime given the hyperparameter configuration. You can download the surrogates as well as a python function that uses the surrogate to simulate the training and validation of the convolutional neural network here.

The hyperparameters of the convolutional neural network and their lower and upper bounds are:

- learning rate $\in [10^{-6}, 10^{0}]$

- batch size $\in \{32, 512\}$
- number of filters layer 1 $\in \{2^4, 2^{10}\}$
- number of filters layer 2 $\in \{2^4, 2^{10}\}$
- number of filters layer 3 $\in \{2^4, 2^{10}\}$

Note: The learning rate and the number of filters usually varies logarithmically and thus we search on a log scale. This means the bounds for the learning rate will be $[-6, 0]$ and $[4, 10]$ for the number of filters respectively.

1 Random Search

The first part of the exercise is to implement random search. In each iteration i, random search samples uniformly at random a hyperparameter configuration $\mathbf{x_i} \sim \mathcal{U}$, evaluates $\hat{f}(\mathbf{x}_i)$ and checks if $\mathbf{x}_i$ improved about the current best seen configuration $\hat{\mathbf{x}}_\star^{(i)}$ (called incumbent), i.e $f(\mathbf{x}_i) < f(\mathbf{x}_\star^{(i)})$. Implement random search, let it run for 50 iterations and save the performance of the incumbent after each iteration.

Every time you run random search the optimization trajectory will look slightly different, due to the random sampling. To get a more significant estimate of its performance, run it 10 times (with 50 iterations each) and plot the mean performance of the incumbents after each iteration.

2 Bayesian Optimization

As we have seen in the lecture, Bayesian optimization usually works better than random search since it maintains a model of the objective function that guides the search. We will use RoBO to do Bayesian optimization. Check our RoBO's fmin example and adapt it to optimize your objective function. Run Bayesian optimization also for 10 independent runs and put the trajectory of the mean performance of the incumbents together with random search's trajectory in one plot. Describe what you see in this plot. How many function evaluations does Bayesian optimization need to become better than random search? Why is it not better right from the beginning?

Compute the runtime of each hyperparameter configuration that has been evaluated by Bayesian optimization and random search and plot the cumulative runtime after each iteration. It suffices if you do this only for a single run. How expensive would it have been if you would have evaluated the true objective function?

Note: RoBO returns you a dictionary, which contains all the hyperparameters that have been evaluates (key: 'X'). You can use numpy's cumsum function to compute the cumulative runtime.