DLI Accelerated Data Science Teaching Kit

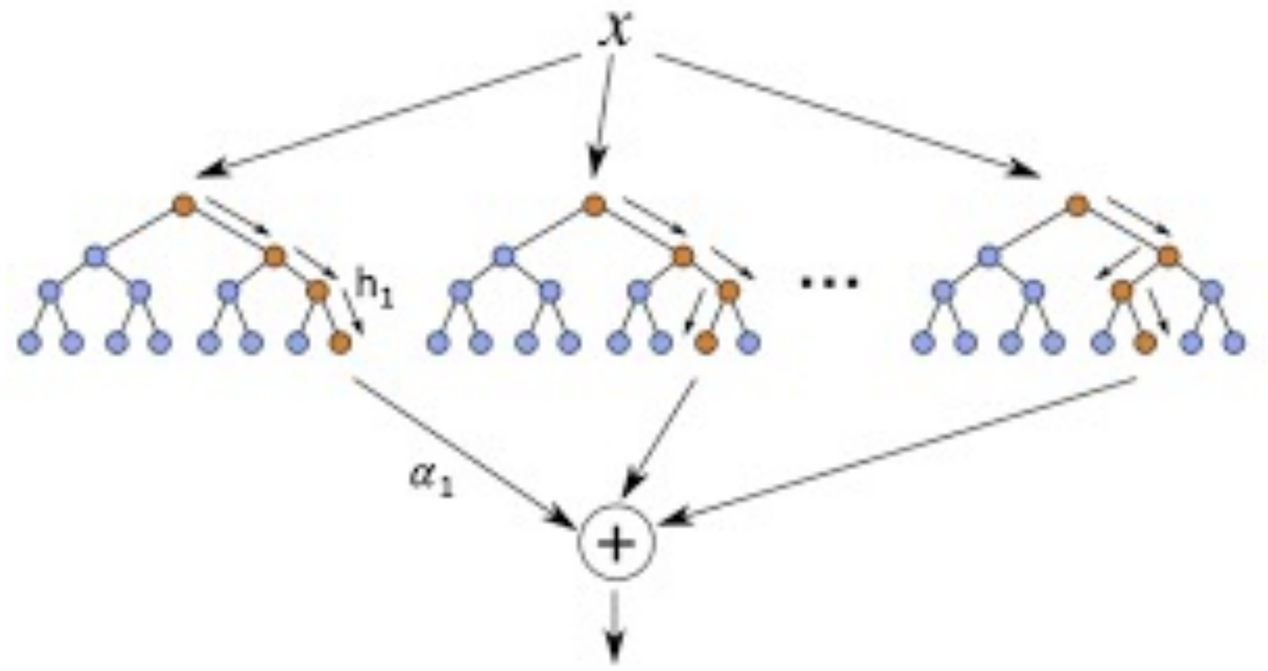
Lecture 14.12 - XGBoost with RAPIDS

What is XGBoosting?

- A more robust and flexible version of gradient boosting
- What does it do?
  - Uses second-order gradients of the loss function in addition to the first-order gradients, based on Taylor expansion of the loss function
- Transforms the loss function into a more sophisticated objective function containing regularization terms

How does XGBoosting improve on Gradient Boosting?

- The extended loss function inhibits overfitting
    - Adds penalty proportional to size of the leaf weights
- This inhibits the growth of the model and prevents it from growing too large
- If the model overfits on the training data, it will not generalize well to new data



DEEP LEARNING INSTITUTE

Example of XGBoost

- Let's consider a scenario in which we are trying to predict the income of an individual
- Instead of following the process used for gradient boosting, we will use quantiles:
  - Cut points that divide a feature into equal-sized groups

Reference: Nvidia RAPIDS

Original Dataset

Instance	Age	Has Job	Owns House	Income
0	12	N	N	0
1	32	Y	Y	90
2	25	Y	Y	50
3	48	N	N	25
4	67	N	Y	35
5	18	Y	N	10

Converted Quantile Dataset

- Reprocess each feature using quantiles:
  - Age separated into strata:
    - <18: One individual
    - <32: Two individuals
    - <67: Two individuals
    - 67+: One individual
  - Separates datapoints into roughly even groups

Instance	Age	Has Job	Owns House	Income
0	0	0	0	0
1	2	1	1	90
2	1	1	1	50
3	2	0	0	25
4	3	0	1	35
5	1	1	0	10

Reference: Nvidia RAPIDS

Finding Splits in Decision Trees

- We want to optimize trees that limit the depth of the overall model
- To achieve this, we need to build nodes that split the data well by minimizing error, in this case sum of squared error (SSE)
  - We can reformat the SSE equation into a split loss equation that can be used in our RAPIDS code

$$Split\ Loss\ (R_a,\ R_b,\ n_a,\ n_b)\ =\ -\frac{1}{2}\left(\frac{R_a^2}{n_a}\right) - \frac{1}{2}\left(\frac{R_b^2}{n_b}\right)$$

Reference: Nvidia RAPIDS

Apply Split Loss Function to Dataset

The split (<18) has the greatest reduction in the SSE function

Quantile	*n*	Quantile Sum *r*	Inclusive Scan *n*	Inclusive Scan *r*	*Split Loss*
<18	1	0	1	0	4410
<32	2	60	3	60	4350
<67	2	115	5	175	3675
67+	1	35	6	210	

Performing XGBoost on RAPIDS

- To run XGBoost efficiently using CUDA, RAPIDS uses bit compression and memory compression
  - Implements parallel primitives for processing sparse CSR (Compressed Sparse Row) format input
  - Uses symbol compression to store the quantised input matrix
- Next, we will see an example of XGBoost on the UCI Higgs dataset

Reference: Nvidia RAPIDS

Example: Setting up GPUs

```
import csv
import numpy as np
import os.path
import pandas
import time
import xgboost as xgb
import sys
if sys.version_info[0] >= 3:
    from urllib.request import urlretrieve
else:
    from urllib import urlretrieve

data_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00280/HIGGS.csv.gz"
dmatrix_train_filename = "higgs_train.dmatrix"
dmatrix_test_filename = "higgs_test.dmatrix"
csv_filename = "HIGGS.csv.gz"
train_rows = 10500000
test_rows = 500000
num_round = 1000

plot = True
```

Download the Higgs Dataset

Example: Load Higgs Dataset

```
# return xgboost dmatrix
def load_higgs():
    if os.path.isfile(dmatrix_train_filename)
      and os.path.isfile(dmatrix_test_filename):
        dtrain = xgb.DMatrix(dmatrix_train_filename)
        dtest = xgb.DMatrix(dmatrix_test_filename)
        if dtrain.num_row() == train_rows and dtest.num_row() == test_rows:
            print("Loading cached dmatrix...")
            return dtrain, dtest

    if not os.path.isfile(csv_filename):
        print("Downloading higgs file...")
        urlretrieve(data_url, csv_filename)

    df_higgs_train = pandas.read_csv(csv_filename, dtype=np.float32,
                                     nrows=train_rows, header=None)
    dtrain = xgb.DMatrix(df_higgs_train.ix[:, 1:29], df_higgs_train[0])
    dtrain.save_binary(dmatrix_train_filename)
    df_higgs_test = pandas.read_csv(csv_filename, dtype=np.float32,
                                    skiprows=train_rows, nrows=test_rows,
                                    header=None)
    dtest = xgb.DMatrix(df_higgs_test.ix[:, 1:29], df_higgs_test[0])
    dtest.save_binary(dmatrix_test_filename)

    return dtrain, dtest
```

Load the Higgs dataset

Split dataset into training and test

Example: Run on CPU and GPU

```
dtrain, dtest = load_higgs()
param = {}
param['objective'] = 'binary:logitraw'
param['eval_metric'] = 'error'
param['tree_method'] = 'gpu_hist'
param['silent'] = 1

print("Training with GPU ...")
tmp = time.time()
gpu_res = {}
xgb.train(param, dtrain, num_round, evals=[(dtest, "test")],
          evals_result=gpu_res)
gpu_time = time.time() - tmp
print("GPU Training Time: %s seconds" % (str(gpu_time)))

print("Training with CPU ...")
param['tree_method'] = 'hist'
tmp = time.time()
cpu_res = {}
xgb.train(param, dtrain, num_round, evals=[(dtest, "test")],
          evals_result=cpu_res)
cpu_time = time.time() - tmp
print("CPU Training Time: %s seconds" % (str(cpu_time)))
```

Run XGBoost on GPU

Run XGBoost on CPU

Example: Plot CPU/GPU Results

```python
if plot:
    import matplotlib.pyplot as plt
    min_error = min(min(gpu_res["test"][param['eval_metric']]),
                    min(cpu_res["test"][param['eval_metric']]))
    gpu_iteration_time =
        [x / (num_round * 1.0) * gpu_time for x in range(0, num_round)]
    cpu_iteration_time =
        [x / (num_round * 1.0) * cpu_time for x in range(0, num_round)]
    plt.plot(gpu_iteration_time, gpu_res['test'][param['eval_metric']],
             label='Tesla P100')
    plt.plot(cpu_iteration_time, cpu_res['test'][param['eval_metric']],
             label='2x Haswell E5-2698 v3 (32 cores)')
    plt.legend()
    plt.xlabel('Time (s)')
    plt.ylabel('Test error')
    plt.axhline(y=min_error, color='r', linestyle='dashed')
    plt.margins(x=0)
    plt.ylim((0.23, 0.35))
    plt.show()
```

Plot the Results for GPU and CPU

Difference in Performance

- 4.15x speed improvement for the GPU algorithm with the same accuracy as the CPU algorithm

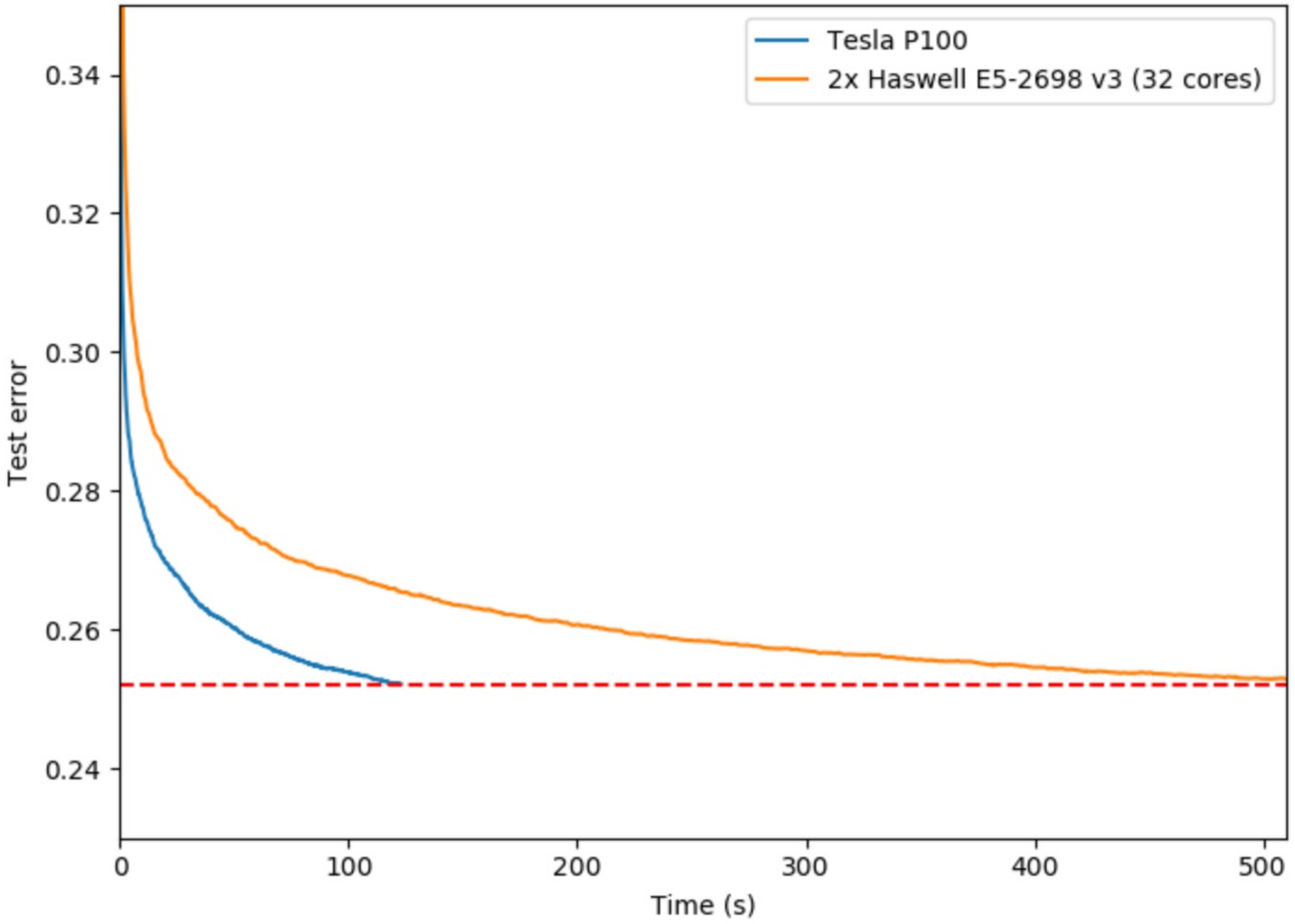


Figure 3. Test error over time for the Higgs dataset, 1000 boosting iterations.

DLI Accelerated Data Science Teaching Kit

Thank You