LSDA18 - HW2

Isabela Blucher

May 21, 2018

1 Fun with Runtimes

1.1 Exercise 1

To rank the functions in the assignment we remember that to assess big-O domination, we can see that if $f(n)$ is dominated and $g(n)$ is dominating ($f(n) \in O(g(n))$ the following is satisfied

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

.

Having that in mind the rank is as follows

$$f_4 \leq f_2 \leq f_1 \leq f_7 \leq f_8 \leq f_3 \leq f_6 \leq f_5$$

1.2 Exercise 2

In order for $2^{2n} \in O(2^n)$, then there must exist a constant c and an integer n_0 such that for all $n \geq n_0$, the inequality $2^{2n} \leq c \cdot 2^n$ holds. By manipulating the inequality we obtain

$$2^n \cdot 2^n \leq c \cdot 2^n \implies 2^n \leq c, \forall n \geq n_0$$

Since n can't be bounded by a constant, we get a contradiction, which proves that $2^{2n} \notin O(2^n)$.

1.3 Exercise 3

To prove that $\sum_{i=1}^{n} log_2 i \in O(nlog_2 n)$, there must exist a constant c and an integer n_0 such that for all $n \geq n_0$, the inequality $\sum_{i=1}^{n} log_2 i \leq c \cdot nlog_2 n$ holds. We can see that

$$\sum_{i=1}^{n} log_2 i \leq \sum_{i=1}^{n} log_2 n = nlog_2 n$$

This means that for $c = 1$ and $n_0 = 1$ the inequality above holds and the sum is upper-bounded by $nlog_2 n$. Thus, we have proven $\sum_{i=1}^{n} log_2 i \in O(nlog_2 n)$.

1.4 Exercise 4

Let $T(n)$ be the runtime of an algorithm. Since we know that for $n \geq 2, T(n) = 4 \cdot T(\frac{n}{2})$, we can expand the recurrence. Assuming, without loss of generality, that n is a power of 2 ($n = 2^k$).

$$T(n) = 4 \cdot T(\frac{2^k}{2}) = 4^2 \cdot T(\frac{2^k}{2^2}) = ... = 4^k \cdot T(\frac{2^k}{2^k}) = 4^k \cdot T(1) = 4^k$$

Since $n = 2^k$, we have that $k = log_2 n$. We can replace k on the expansion of $T(n)$ to get $T(n) = 4^k = 2^{2k} = 2^{log_2 n^2} = n^2$. Thus, we have proven that $T(n) = n^2 \in O(n^2)$.

2 K-D Trees

2.1 Practical Runtimes

For the first experiment we want to analyze how the runtime of the fitting changes for different sized training sets with points of different dimensionality.

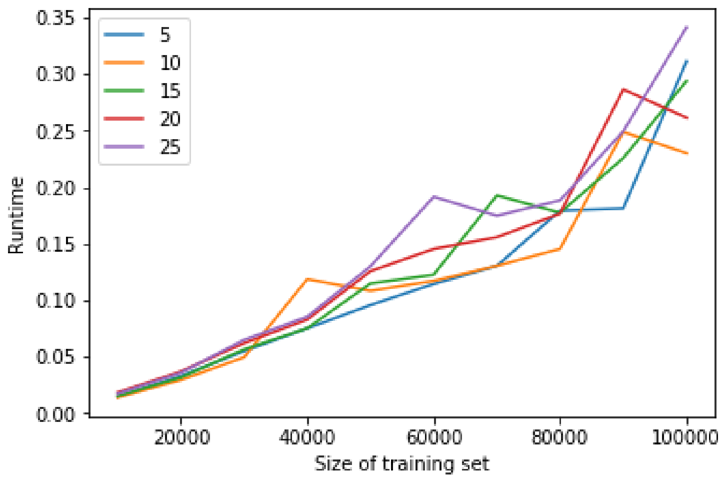

Figure 1: Fitting runtime in seconds in relation to training set size for 5 dimensionalities

From Figure 1 we can observe that the graphs for the different dimensionalities have similar shapes and they evolve similarly for training sets with more instances. This means that the dimensionality of the points isn't an issue when fitting the K-D trees model.

For the second experiment we want to analyze how the runtime of the querying process changes for different sized test sets with points of different dimensionality.

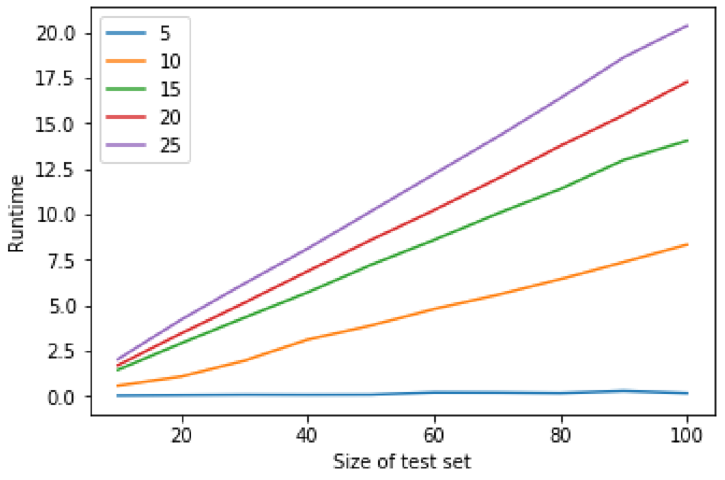

Figure 2: Querying runtime in seconds in relation to test set size for 5 dimensionalities

From Figure 2 we can observe that the graphs for the different dimensionalities have different shapes and evolve in very different ways. This means that the dimensionality of the points and the runtime of the querying of the K-D trees algorithm are very correlated, as a higher dimensional dataset can significantly increase the runtime.

2.2 Runtime Profiling

Making use of the cProfile module to profile the query phase with 10000 training instances, 100 test instances with points in a 50-dimensional space, we get the following output:

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1    0.002    0.002   23.448   23.448 <ipython-input-4-5973250f7d7a>:108(query)
1000000   17.590    0.000   17.970    0.000 <ipython-input-4-5973250f7d7a>:36(_distance)
```

For the query call itself the overall runtime is 23.488 seconds, and for the distance computations (function `_distance` the cumulative time is 17.970 seconds. Comparing both, we can see that 76,5% of the overall runtime is used on the distance computations.

2.3 Faster Distance Computations

By using the `numpy.linalg.norm` function on the x, y vectors we compute the euclidean distance between them and make the computation time faster. This happens because the original function was iterating through all the elements in the vectors, making the computation time $O(n)$. By profiling the query phase with cProfile we get the following output:

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1    0.002    0.002   13.702   13.702 <ipython-input-3-c79dd7673c78>:108(query)
1000000    1.530    0.000    8.419    0.000 <ipython-input-3-c79dd7673c78>:38(_distance)
```

For the query call itself the overall runtime 13.702 seconds, and for the distance computations the cumulative time is 8.419 seconds. Comparing these results to the previous runtimes we can see that there is a significant decrease in time.

2.4 Leaf Visits

The variant of early stopping the algorithm was implemented and with L = 5 we get the following runtime plot for the querying phase.

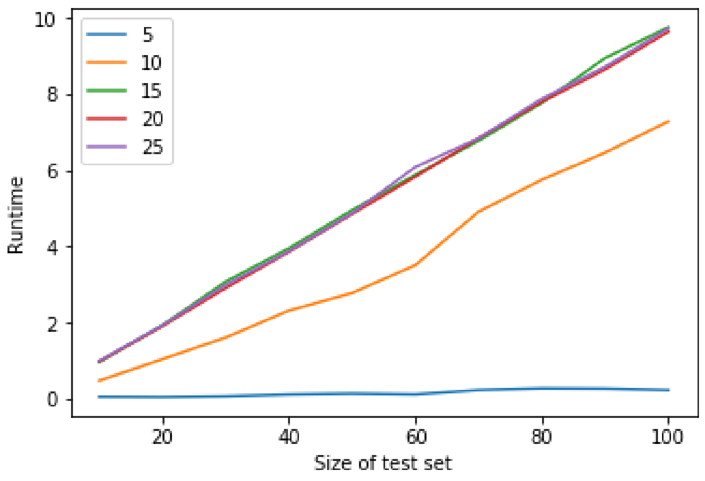

Figure 3: Runtime in seconds in relation to test set size for 5 dimensionalities with early stopping on the querying process

As expected, if we allow less leaves per querying point, the runtime for all dimensions gets smaller. Since the modifier forces the tree to stop adding neighbors after reaching a maximum leaf number, the nearest neighbors set will not be the exact same as the case without the modifier.

As for the worst case asymptotic runtime, we have to think of how the querying process would work on a tree where all the queries were always the worst possible. In order to find a nearest neighbor, the worst case is that the algorithm traverses almost the whole tree, so $O(n)$, so per query the worst case asymptotic runtime would be linear, meaning having to search all the n training points used to build the tree.

3 Neural networks with TensorFlow

3.1 Standard Deep Neural Networks

3.1.1 Adam

The lines of code added to the `MLPMiniBatchWeedTensorFlowTensorBoardDataAPIAssignment.py` file to add the hidden layer and to initialize the Adam optimizer were:

```
# Add hidden layer
y_2 = tf.layers.dense(y_1, FLAGS.no_hidden2, bias_initializer=tf.
    truncated_normal_initializer(stddev=0.1), kernel_initializer=tf.
    truncated_normal_initializer(stddev=0.1), activation=tf.sigmoid, name='layer_2')

# Declare optimizer
my_opt = tf.train.AdamOptimizer(FLAGS.lr)
```

The initial learning rate is 100 and the training and validation accuracy for this learning rate are, respectively, 0.73375 and 0.755. By testing the optimizer with learning rates from 10 to 10^{-5}, the one that yielded the best results was 10^{-4}. The best training and validation accuracies for this case were 0.95875 and 0.96, respectively.

3.1.2 Increase the Throughput

By reading the TensorFlow documentation and testing different values for the buffer sizes of the `prefetch` and `shuffle_and_repeat` the performance couldn't be significantly improved.

The documentation states that the order of the data pipeline is very important for computational speed. For example, while `shuffle_and_repeat` combines both shuffling and repeating to provide good performance and strong ordering, doing one before the other could slow down performance or not guarantee a good ordering.

3.2 Deep convolutional neural networks

3.2.1 Shortcut with pooling

The lines of code added to the `CNNMNISTTensorFlowAssignmentDataAPI.py` were:

```
# Parallel pooling layer
pp = tf.layers.max_pooling2d(inputs = c1,
                 pool_size = 4,
                 strides = 1,
                 padding = 'valid',
                 name = 'pool_parallel')
```

This section of the code creates a new 4×4 max-pooling layer, with the same parameters as the first 2×2 max-pooling layer, even the same input.

The output of the "pool_parallel" layer is then flattened and concatenated with the output from the second 2×2 max-pooling layer. This combined output of the two max-pooling layers is then used as input for the final standard layer of the network. The following section shows this process.

```
# Combine pooling layers outputs
pp_flat = tf.layers.flatten(pp)
pnew = tf.concat([p2_flat, pp_flat], 1)

# Fully connected layer (additional input here)
f1 = tf.layers.dense(pnew, 1024, activation=tf.nn.relu, use_bias=True, name="fc_1")
```

3.2.2 CIFAR10

The `CNNMNISTTensorFlowAssignmentDataAPI.py` code was adapted to work on the CIFAR10 dataset. Using the `tf.keras.datasets.cifar10.load data()` function and applying the recommended modifications to the original code, running on the full dataset was taking too long. The training data was reduced using the `tf.data.Dataset.take()` function, which takes a scalar as parameter and creates a new dataset with the amount of instances specified by the scalar. The test dataset was reduced using Numpy's random choice function to get a random set of test instances. The training dataset was reduced to 5000 and the test to 2000. Since it was still taking about 2.9 to 3.2 seconds per update step, the number of steps was reduced to 5000, so the whole process will take about 4 hours to complete.

After having completed the training and testing process, the test accuracy obtained was 0.5035. If compared to the past assignment there is an increase in accuracy, though a higher score could probably be obtained with different network architectures.

The code added to modify the original network is as follows:

```
# Import CIFAR10 data
data_train, data_test = tf.keras.datasets.cifar10.load_data()

# Here the modifier is the ds.take, which subsamples the training dataset
ds = tf.data.Dataset.from_tensor_slices(data_train)
ds = ds.take(5000) # subsample data (10% of original dataset)
ds = ds.apply(tf.contrib.data.shuffle_and_repeat(10*FLAGS.batch_size, count=FLAGS.epochs))
ds = ds.batch(FLAGS.batch_size)
ds = ds.prefetch(FLAGS.prefetch)

# Resize test dataset to 2000 random instances
ids = np.random.choice(len(data_test[0]), 2000, replace = False)
xtest = data_test[0][ids, :]
ytest = data_test[1][ids]

# Define input and output placeholders. Resizing the x placeholder to have three channels
    and 32-by-32 pixels as to accommodate the CIFAR10 dataset. We also apply the
    modifications recommended to the y placeholder.
x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
y = tf.placeholder(tf.int64, shape=[None, 1])
y_flatten = tf.reshape(y, [-1])
```