Missing Object Orientation?

Contrast the model code presented in this architecture with the policy code from the previous architecture. Note how the introduction of the `Layer` object allows for dramatically simplified code with concomitant improvements in readability. This sharp improvement in readability is part of the reason most developers prefer to use an object-oriented overlay on top of TensorFlow in practice.

That said, in this chapter, we use raw TensorFlow, since making classes like `TensorGraph` work with multiple GPUs would require significant additional overhead. In general, raw TensorFlow code offers maximum flexibility, but object orientation offers convenience. Pick the abstraction necessary for the problem at hand.

Training on Multiple GPUs

We instantiate a separate version of the model and architecture on each GPU. We then use the CPU to average the weights for the separate GPU nodes (Example 9-3).

Example 9-3. This function trains the Cifar10 model

```
def train():
  """Train CIFAR10 for a number of steps."""
  with tf.Graph().as_default(), tf.device('/cpu:0'):
    # Create a variable to count the number of train() calls. This equals the
    # number of batches processed * FLAGS.num_gpus.
    global_step = tf.get_variable(
        'global_step', [],
        initializer=tf.constant_initializer(0), trainable=False)

    # Calculate the learning rate schedule.
    num_batches_per_epoch = (cifar10.NUM_EXAMPLES_PER_EPOCH_FOR_TRAIN /
                             FLAGS.batch_size)
    decay_steps = int(num_batches_per_epoch * cifar10.NUM_EPOCHS_PER_DECAY)

    # Decay the learning rate exponentially based on the number of steps.
    lr = tf.train.exponential_decay(cifar10.INITIAL_LEARNING_RATE,
                                    global_step,
                                    decay_steps,
                                    cifar10.LEARNING_RATE_DECAY_FACTOR,
                                    staircase=True)

    # Create an optimizer that performs gradient descent.
    opt = tf.train.GradientDescentOptimizer(lr)

    # Get images and labels for CIFAR-10.
    images, labels = cifar10.distorted_inputs()
```

```
batch_queue = tf.contrib.slim.prefetch_queue.prefetch_queue(
      [images, labels], capacity=2 * FLAGS.num_gpus)
```

The code in Example 9-4 performs the essential multi-GPU training. Note how different batches are dequeued for each GPU, but weight sharing via `tf.get_variable_score().reuse_variables()` enables training to happen correctly.

Example 9-4. This snippet implements multi-GPU training

```
# Calculate the gradients for each model tower.
tower_grads = []
with tf.variable_scope(tf.get_variable_scope()):
  for i in xrange(FLAGS.num_gpus):
    with tf.device('/gpu:%d' % i):
      with tf.name_scope('%s_%d' % (cifar10.TOWER_NAME, i)) as scope:
        # Dequeues one batch for the GPU
        image_batch, label_batch = batch_queue.dequeue()
        # Calculate the loss for one tower of the CIFAR model. This function
        # constructs the entire CIFAR model but shares the variables across
        # all towers.
        loss = tower_loss(scope, image_batch, label_batch)

        # Reuse variables for the next tower.
        tf.get_variable_scope().reuse_variables()

        # Retain the summaries from the final tower.
        summaries = tf.get_collection(tf.GraphKeys.SUMMARIES, scope)

        # Calculate the gradients for the batch of data on this CIFAR tower.
        grads = opt.compute_gradients(loss)

        # Keep track of the gradients across all towers.
        tower_grads.append(grads)

# We must calculate the mean of each gradient. Note that this is the
# synchronization point across all towers.
grads = average_gradients(tower_grads)
```

We end by applying the joint training operation and writing summary checkpoints as needed in Example 9-5.

Example 9-5. This snippet groups updates from the various GPUs and writes summary checkpoints as needed

```
# Add a summary to track the learning rate.
summaries.append(tf.summary.scalar('learning_rate', lr))

# Add histograms for gradients.
for grad, var in grads:
  if grad is not None:
```

```
    summaries.append(tf.summary.histogram(var.op.name + '/gradients', grad))

# Apply the gradients to adjust the shared variables.
apply_gradient_op = opt.apply_gradients(grads, global_step=global_step)

# Add histograms for trainable variables.
for var in tf.trainable_variables():
  summaries.append(tf.summary.histogram(var.op.name, var))

# Track the moving averages of all trainable variables.
variable_averages = tf.train.ExponentialMovingAverage(
    cifar10.MOVING_AVERAGE_DECAY, global_step)
variables_averages_op = variable_averages.apply(tf.trainable_variables())

# Group all updates into a single train op.
train_op = tf.group(apply_gradient_op, variables_averages_op)

# Create a saver.
saver = tf.train.Saver(tf.global_variables())

# Build the summary operation from the last tower summaries.
summary_op = tf.summary.merge(summaries)

# Build an initialization operation to run below.
init = tf.global_variables_initializer()

# Start running operations on the Graph. allow_soft_placement must be set to
# True to build towers on GPU, as some of the ops do not have GPU
# implementations.
sess = tf.Session(config=tf.ConfigProto(
    allow_soft_placement=True,
    log_device_placement=FLAGS.log_device_placement))
sess.run(init)

# Start the queue runners.
tf.train.start_queue_runners(sess=sess)

summary_writer = tf.summary.FileWriter(FLAGS.train_dir, sess.graph)

for step in xrange(FLAGS.max_steps):
  start_time = time.time()
  _, loss_value = sess.run([train_op, loss])
  duration = time.time() - start_time

  assert not np.isnan(loss_value), 'Model diverged with loss = NaN'

  if step % 10 == 0:
    num_examples_per_step = FLAGS.batch_size * FLAGS.num_gpus
    examples_per_sec = num_examples_per_step / duration
    sec_per_batch = duration / FLAGS.num_gpus

    format_str = ('%s: step %d, loss = %.2f (%.1f examples/sec; %.3f '
```

```
                      'sec/batch)')
        print (format_str % (datetime.now(), step, loss_value,
                             examples_per_sec, sec_per_batch))

      if step % 100 == 0:
        summary_str = sess.run(summary_op)
        summary_writer.add_summary(summary_str, step)
      # Save the model checkpoint periodically.

      if step % 1000 == 0 or (step + 1) == FLAGS.max_steps:
        checkpoint_path = os.path.join(FLAGS.train_dir, 'model.ckpt')
        saver.save(sess, checkpoint_path, global_step=step)
```

Challenge for the Reader

You now have all the pieces required to train this model in practice. Try running it on a suitable GPU server! You may want to use tools such as `nvidia-smi` to ensure that all GPUs are actually being used.

Review

In this chapter, you learned about various types of hardware commonly used to train deep architectures. You also learned about data parallel and model parallel designs for training deep architectures on multiple CPUs or GPUs. We ended the chapter by walking through a case study on how to implement data parallel training of convolutional networks in TensorFlow.

In Chapter 10, we will discuss the future of deep learning and how you can use the skills you've learned in this book effectively and ethically.