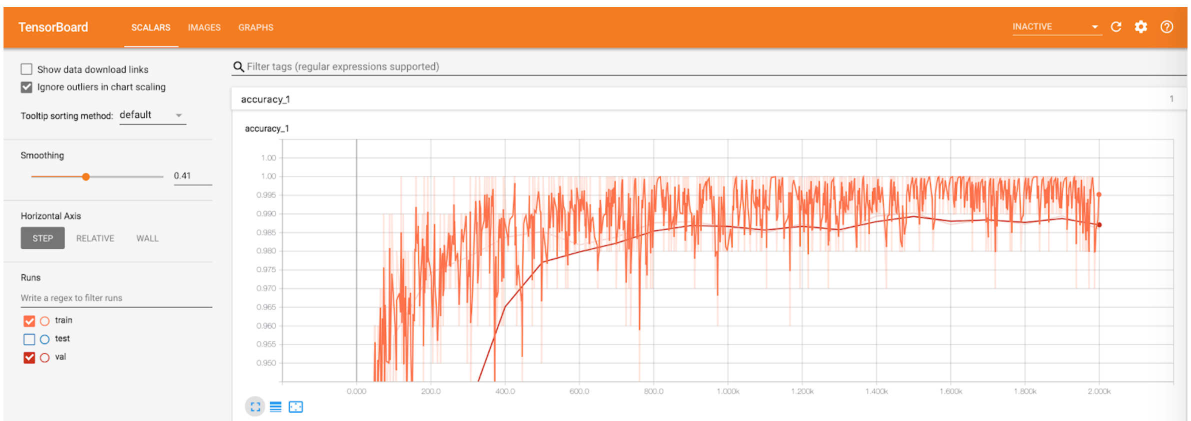
CS498 AML HW10

Huiyun Wu (hwu63)

Yidi Yang (yyang160)

Page 1: (MNIST with TUTORIAL)

1. Accuracy plot (train and val, with smoothing = 0.41)



Where the orange line represents train accuracies (logged every batch), and the red smooth line represents the validation set accuracy (logged every 100 batches).

2. Accuracies (train, test, val) from the 'best' model with a brief explanation.

   Training set: ~0.995

   Testing set: 0.9877

   Validation set: 0.9859

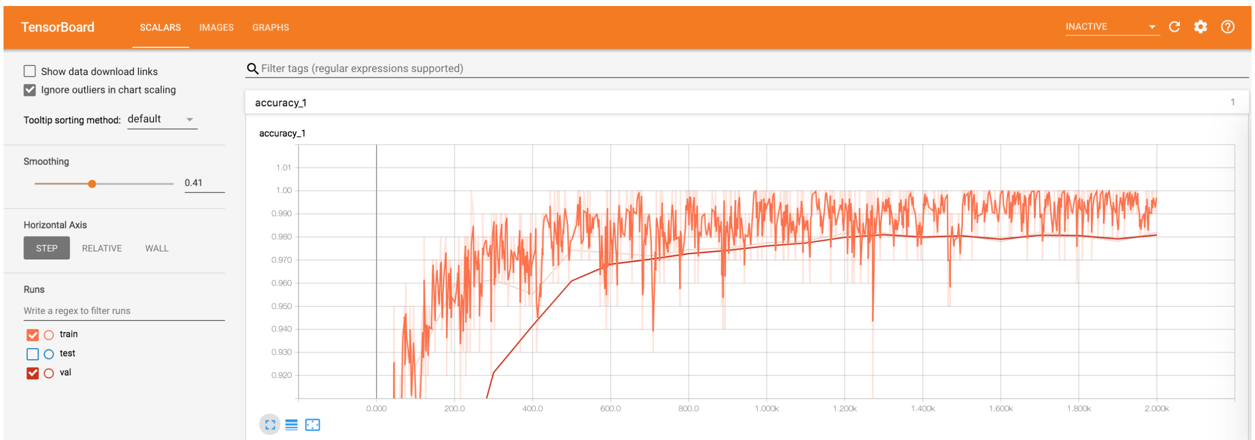
   The model we use here is the original one from the MNIST tutorial. We trained it with 2001 batches and reached this accuracy.

Page 2: (MNIST w MODIFIED)

1. Description of changes

   We added another convolutional layer (with filter = 32 originally to 4) and removed all the poolings, so that the model has three layers in total with filter equals 4 and no max poolings.

2. Accuracy plot (train and val)



3. Accuracies (train, test, val) from the 'best' model with a brief explanation.

   Training set: 0.9971

   Testing set: 0.985

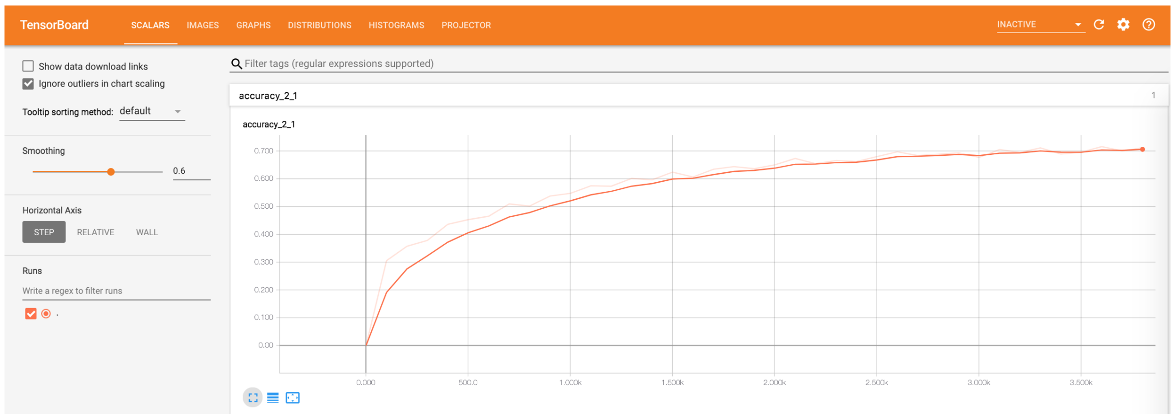
   Validation set: 0.982

   The model here is modified as explained above. We runned it for 2001 batches. The accuracy here dropped by 1% compared with the original model, we think this is due to the filter change.

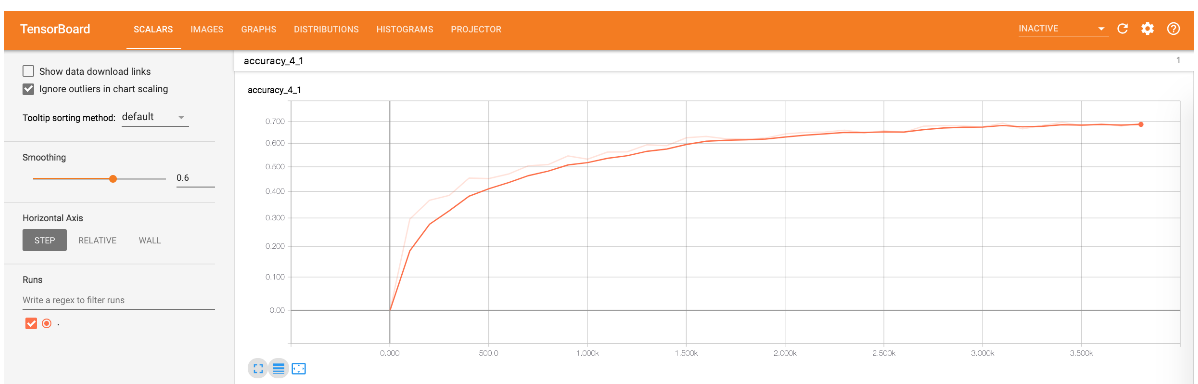
Page 3: (CIFAR w TUTORIAL)

1. Accuracy plot (train and val)

   Train:



   Val:



2. Accuracies (train, test, val) from the 'best' model with a brief explanation.

   We runned the model for around 3900 batches. We don't have GPU so that we couldn't run the model for sufficiently many batches. But we believe the accuracy would be around the desired level (0.83+) if running it with more batches.

   Training set accuracy: 0.7117

   Testing set (Evaluation set) accuracy: 0.647

   Validation set accuracy: 0.6924

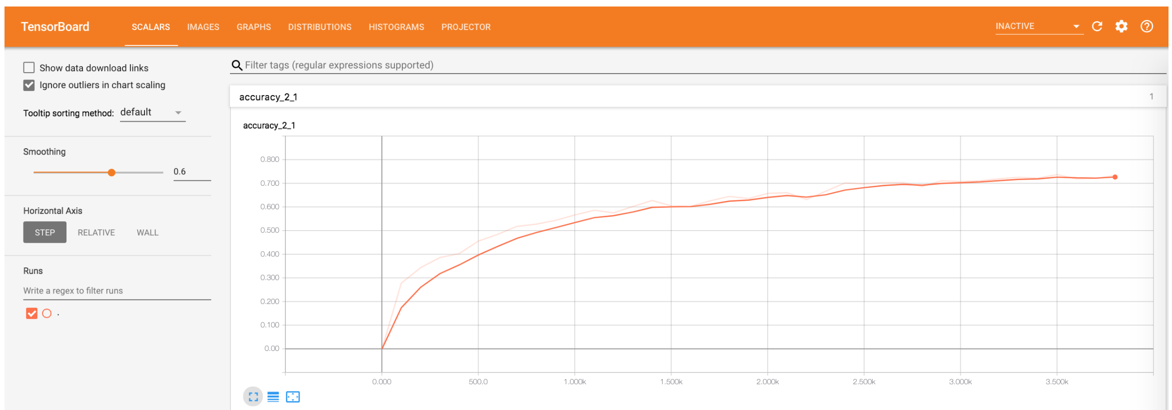
Page 4: (CIFAR w MODIFIED)

1. Description of changes

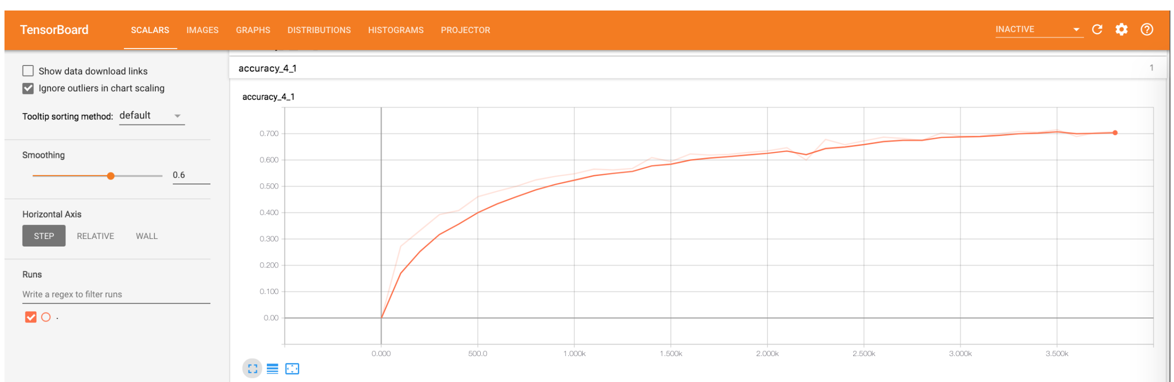
   We added another convocational layer between conv2 and normalization 2.

2. Accuracy plot (train and val)

   Train:



   Val:



3. Accuracies (train, test, val) from the 'best' model with a brief explanation.

   Training set accuracy: 0.7330

   Testing set accuracy: 0.6769

   Validation set accuracy: 0.7070

   To make a valid and fair comparison, we also runned it for 3900 batches. The accuracies of three datasets all increased. Because we added another convolutional layer in the model.

Page 5-6: (CODE SNIPPETS - MNIST MODIFIED)

1. Creating the model/graph: Corresponds to the cnn_model_fn in the mnist tutorial.

```
        tf.summary.scalar('stddev', stddev)
        tf.summary.scalar('max', tf.reduce_max(var))
        tf.summary.scalar('min', tf.reduce_min(var))
        tf.summary.histogram('histogram', var)

conv1 = tf.layers.conv2d(
    inputs=image_shaped_input,
    filters=4,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)

conv2 = tf.layers.conv2d(
    inputs=conv1,
    filters=4,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)

conv3 = tf.layers.conv2d(
    inputs=conv2,
    filters=4,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
#
conv3_flat = tf.reshape(conv3, [-1, 7 * 7 * 64])

dense = tf.layers.dense(inputs=conv3_flat, units=1024, activation=tf.nn.relu)


with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    tf.summary.scalar('dropout_keep_probability', keep_prob)
    dropout = tf.layers.dropout(inputs=dense, rate=keep_prob)
```

2. Defining loss & optimizer: Part where you set the loss (Corresponds to line 100 in the mnist tutorial) and optimizer.

```
with tf.name_scope('cross_entropy'):
    # The raw formulation of cross-entropy,
    #
    # tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.softmax(y)),
    #                               reduction_indices=[1]))
    #
    # can be numerically unstable.
    #
    # So here we use tf.losses.sparse_softmax_cross_entropy on the
    # raw logit outputs of the nn_layer above, and then average across
    # the batch.
    with tf.name_scope('total'):
        cross_entropy = tf.losses.sparse_softmax_cross_entropy(
            labels=y_, logits=y)
tf.summary.scalar('cross_entropy', cross_entropy)
```

3. Training: Part where you either manually defined a train loop or where you used a built-in train function to train the model.

```
for i in range(2001):

    if i % 100 == 0:  # Record summaries and test-set accuracy
        print("running test and val...")
        summary, acc = sess.run([merged, accuracy], feed_dict=feed_dict(0))
        test_writer.add_summary(summary, i)
        print('Test Set Accuracy at step %s: %s' % (i, acc))
        summary2, acc2 = sess.run([merged, accuracy], feed_dict=feed_dict(2))
        val_writer.add_summary(summary2, i)
        print('Val Set Accuracy at step %s: %s' % (i, acc2))

    else:  # Record train set summaries, and train
        summary, _ = sess.run([merged, train_step], feed_dict=feed_dict(1))
        train_writer.add_summary(summary, i)
train_writer.close()
test_writer.close()
val_writer.close()
```

4. Any other snippet you found relevant

```python
train_images, val_images, train_labels, val_labels = \
    train_test_split(mnist.train.images, mnist.train.labels,test_size=0.2)

def next_batch(num, data, labels):
    '''
    Return a total of `num` random samples and labels.
    '''
    idx = np.arange(0, len(data))
    np.random.shuffle(idx)
    idx = idx[:num]
    data_shuffle = [data[i] for i in idx]
    labels_shuffle = [labels[i] for i in idx]

    return np.asarray(data_shuffle), np.asarray(labels_shuffle)

def feed_dict(train):
    """Make a TensorFlow feed_dict: maps data onto Tensor placeholders."""
    if train == 1 or FLAGS.fake_data:
        xs, ys = next_batch(100, train_images, train_labels)
        k = FLAGS.dropout
    elif train == 0:
        xs, ys = mnist.test.images, mnist.test.labels
        k = 1.0
    else:
        xs, ys = val_images, val_labels
        k = 1.0
    return {x: xs, y_: ys, keep_prob: k}
```

Validation data set splitting and modified feed_dict.

Page 7-8: (CODE SNIPPETS - CIFAR MODIFIED)

1. Creating the model/graph: Corresponds to the cnn_model_fn in the mnist tutorial.

```
def inference(images):
    """Build the CIFAR-10 model...."""
    ...
    with tf.variable_scope('conv1') as scope:
        kernel = _variable_with_weight_decay('weights',
                                             shape=[5, 5, 3, 64],
                                             stddev=5e-2,
                                             wd=None)
        conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
        biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
        pre_activation = tf.nn.bias_add(conv, biases)
        conv1 = tf.nn.relu(pre_activation, name=scope.name)
        _activation_summary(conv1)

    # pool1
    pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                           padding='SAME', name='pool1')
    # norm1
    norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                      name='norm1')

    # conv2
    with tf.variable_scope('conv2') as scope:
        kernel = _variable_with_weight_decay('weights',
                                             shape=[5, 5, 64, 64],
                                             stddev=5e-2,
                                             wd=None)
        conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
        biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.1))
        pre_activation = tf.nn.bias_add(conv, biases)
        conv2 = tf.nn.relu(pre_activation, name=scope.name)
        _activation_summary(conv2)
```

```
    with tf.variable_scope('conv3') as scope:
        kernel = _variable_with_weight_decay('weights',
                                             shape=[5, 5, 64, 64],
                                             stddev=5e-2,
                                             wd=None)
        conv = tf.nn.conv2d(conv2, kernel, [1, 1, 1, 1], padding='SAME')
        biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.1))
        pre_activation = tf.nn.bias_add(conv, biases)
        conv3 = tf.nn.relu(pre_activation, name=scope.name)
        _activation_summary(conv3)
    # norm2
    norm2 = tf.nn.lrn(conv3, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                      name='norm2')
    # pool2
    pool2 = tf.nn.max_pool(norm2, ksize=[1, 3, 3, 1],
                           strides=[1, 2, 2, 1], padding='SAME', name='pool2')
```

2. Defining loss & optimizer: Part where you set the loss (Corresponds to line 100 in the mnist tutorial) and optimizer.

```
    # Calculate loss.
    loss = cifar10.loss(logits, labels)
```

```python
def train(total_loss, global_step):
    """Train CIFAR-10 model...."""
    # Variables that affect learning rate.
    num_batches_per_epoch = NUM_EXAMPLES_PER_EPOCH_FOR_TRAIN / FLAGS.batch_size
    decay_steps = int(num_batches_per_epoch * NUM_EPOCHS_PER_DECAY)

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

    # Generate moving averages of all losses and associated summaries.
    loss_averages_op = _add_loss_summaries(total_loss)

    # Compute gradients.
    with tf.control_dependencies([loss_averages_op]):
        opt = tf.train.GradientDescentOptimizer(lr)
        grads = opt.compute_gradients(total_loss)

    # Apply gradients.
    apply_gradient_op = opt.apply_gradients(grads, global_step=global_step)

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

    # Add histograms for gradients.
    for grad, var in grads:
        if grad is not None:
            tf.summary.histogram(var.op.name + '/gradients', grad)
```

```python
    # Track the moving averages of all trainable variables.
    variable_averages = tf.train.ExponentialMovingAverage(
        MOVING_AVERAGE_DECAY, global_step)
    with tf.control_dependencies([apply_gradient_op]):
        variables_averages_op = variable_averages.apply(tf.trainable_variables())

    return variables_averages_op
```

3. Training: Part where you either manually defined a train loop or where you used a built-in train function to train the model.

```python
    class _LoggerHook(tf.train.SessionRunHook):
        """Logs loss and runtime."""

        def begin(self):...

        def before_run(self, run_context):
            self._step += 1
            return tf.train.SessionRunArgs(loss)  # Asks for loss value.

        def after_run(self, run_context, run_values):
            if self._step % FLAGS.log_frequency == 0:
                current_time = time.time()
                duration = current_time - self._start_time
                self._start_time = current_time

                loss_value = run_values.results
                examples_per_sec = FLAGS.log_frequency * FLAGS.batch_size / duration
                sec_per_batch = float(duration / FLAGS.log_frequency)

                format_str = ('%s: step %d, loss = %.2f (%.1f examples/sec; %.3f '
                              'sec/batch)')
                print(format_str % (datetime.now(), self._step, loss_value,
                                    examples_per_sec, sec_per_batch))

    with tf.train.MonitoredTrainingSession(
            checkpoint_dir=FLAGS.train_dir,
            hooks=[tf.train.StopAtStepHook(last_step=FLAGS.max_steps),
                   tf.train.NanTensorHook(loss),
                   _LoggerHook()],
            config=tf.ConfigProto(
                log_device_placement=FLAGS.log_device_placement)) as mon_sess:
        while not mon_sess.should_stop():
            mon_sess.run(train_op)
```

4. Any other snippet you found relevant

```python
def inputs(eval_data, data_dir, batch_size):
    """Construct input for CIFAR evaluation using the Reader ops.

    Args:
      eval_data: bool, indicating if one should use the train or eval data set.
      data_dir: Path to the CIFAR-10 data directory.
      batch_size: Number of images per batch.

    Returns:
      images: Images. 4D tensor of [batch_size, IMAGE_SIZE, IMAGE_SIZE, 3] size.
      labels: Labels. 1D tensor of [batch_size] size.
    """
    # if not eval_data:
    #   filenames = [os.path.join(data_dir, 'data_batch_%d.bin' % i)
    #                for i in xrange(1, 6)]
    #   num_examples_per_epoch = NUM_EXAMPLES_PER_EPOCH_FOR_TRAIN
    if eval_data == 1:
        filenames = [os.path.join(data_dir, 'data_bach_%d.bin' % i)
                     for i in range(1, 5)]
        num_examples_per_epoch = NUM_EXAMPLES_PER_EPOCH_FOR_TRAIN
    elif eval_data == 2:
        filenames = [os.path.join(data_dir, 'data_batch_%d.bin' % i)
                     for i in range(5, 6)]
        num_examples_per_epoch = NUM_EXAMPLES_PER_EPOCH_FOR_VAL
    else:
        filenames = [os.path.join(data_dir, 'test_batch.bin')]
        num_examples_per_epoch = NUM_EXAMPLES_PER_EPOCH_FOR_EVAL
```

Modified inputs in cifar_input.py to include the validation.