其他

Tensorflow

林楚铭

cmlin17@fudan.edu.cn

Name Scope

变量复用

```
def foo():
  with tf.variable_scope("foo", reuse=tf.AUTO_REUSE):
    v = tf.get_variable("v", [1])
  return v

v1 = foo()  # Creates v.
v2 = foo()  # Gets the same, existing v.
assert v1 == v2
```

Name Scope

网络复用

```
class vgg16():
    def build(self, inp, num_classes=20, training=True, name='vgg16'):
        with tf.variable_scope(name, reuse=tf.AUTO_REUSE):
            x = conv2d(inp, 64, name='conv1_1')
            x = conv2d(x, 64, name='conv1_2')
            x = tf.layers.max_pooling2d(x, 2, 2, 'same')
            x = conv2d(x, 128, name='conv2_1')
…
```

```
inp1 =  tf.placeholder(tf.float32, [1, 224, 224, 3])
inp2 =  tf.placeholder(tf.float32, [1, 224, 224, 3])
x1 = net.build(inp1)
x2 = net.build(inp2) # reuse network
```

Name Scope

通过命名找到tensor

```
a = tf.add(1, 2, name='add')
b = tf.Graph.get_tensor_by_name(tf.get_default_graph(), 'add:0')
assert a == b
```

设置name scope还可以方便获取某一父命名下的所有变量或可训练的变量

```
tf.global_variables(scope=None)
tf.trainable_variables(scope=None)
```

BatchNorm

```
x_norm = tf.layers.batch_normalization(x, training=training)

 # ...

 update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
 with tf.control_dependencies(update_ops):
   train_op = optimizer.minimize(loss)
```

1. 使用BatchNorm时，必须执行update_ops
2. denpendency可以保证update_ops先执行，后再执行train_op
3. 当batch_normalization的training为False时，update_ops为空

Focal Loss

- Focal loss用来解决分类问题中样本不均衡的问题
- 以softmax为例

```python
def softmax_focal_loss(logits, one_hot_label, beta=2):
    with tf.name_scope("softmax_focal_loss"):
        l_softmax = tf.nn.softmax(logits=logits, axis=-1)
        l_log = -1.0 * tf.log(tf.clip_by_value(l_softmax, 1e-10, 1.0))
        l_focal = l_log * (1 - l_softmax) ** beta
        return tf.reduce_sum(one_hot_label * l_focal, axis=-1)
```

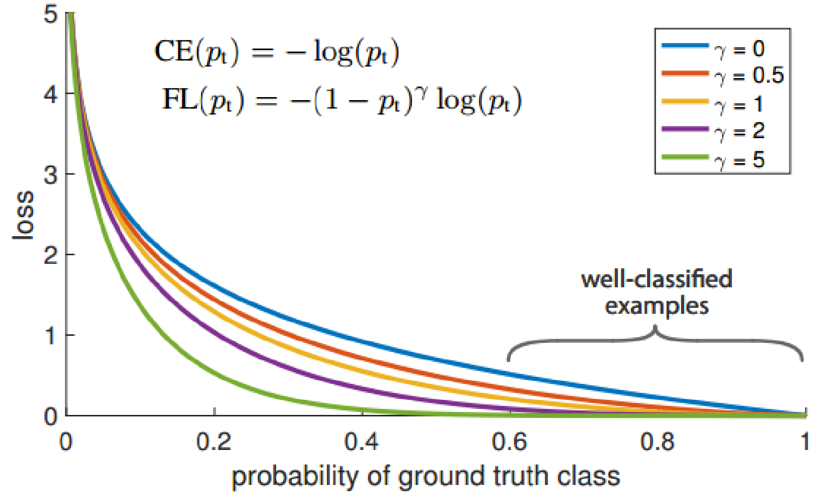
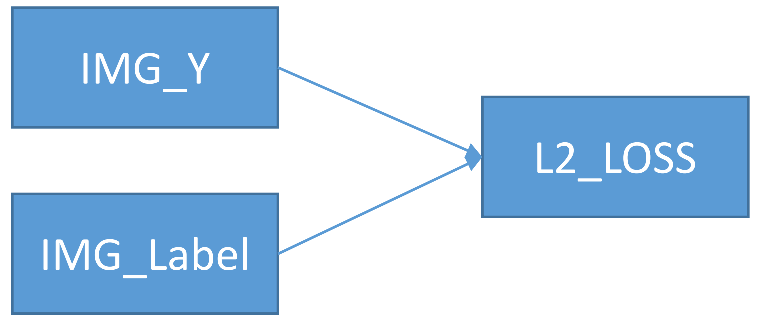
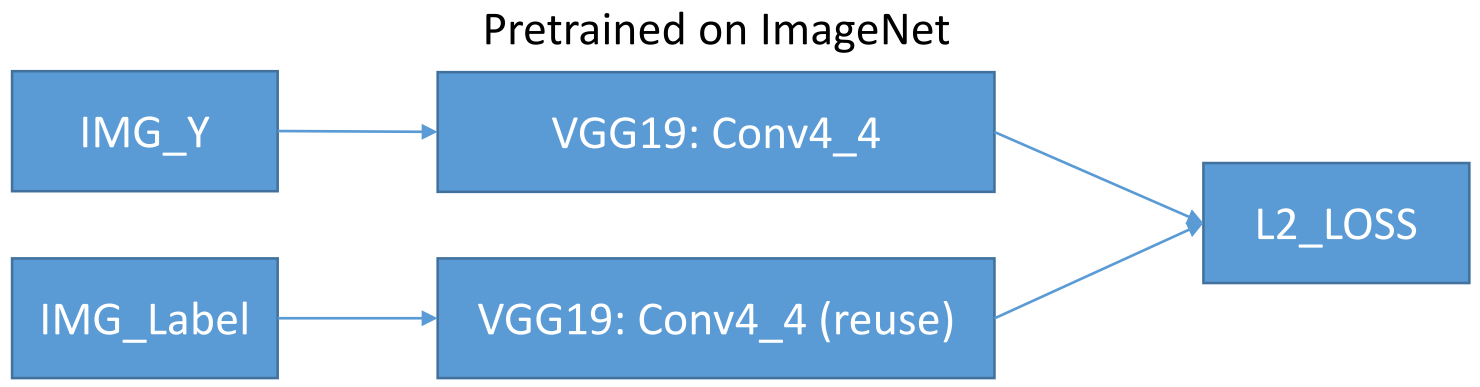


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

Perceptual Loss





与分类器训练类似，从slim model中下载VGG19模型，使用返回的end_points['vgg_19/conv4/conv4_4']
Perceptual loss不一定要使用VGG模型，其他在ImageNet上的分类模型也是可以的

Optimizer

- 需求1：只想训练Net_A的参数，Net_B的参数不参与训练
- 注意：不是改变参数的trainable属性，而是让优化器指定要更新的参数

```
lr = 1e-4
optimizer = tf.train.AdamOptimizer(lr)
train_op = optimizer.minimize(loss, global_step,
                              var_list=tf.trainable_variables('Net_A'))
```

Optimizer

- 需求2：不同变量使用不同的学习率进行训练
- 例子：var_list1内的变量学习率为1e-5，其余1e-4
- minimize = compute_gradients + apply_gradients

```
lr = 1e-4
optimizer = tf.train.AdamOptimizer(lr)
capped_grads_and_vars = []
grads = optimizer.compute_gradients(loss)
for grad, var in grads:
    if var in var_list1:
        capped_grads_and_vars.append((0.1 * grad, var))
    else:
        capped_grads_and_vars.append((1 * grad, var))
train_op = optimizer.apply_gradients(capped_grads_and_vars, global_step)
```

Warping

- 学习使用tf来编写复杂的操作
- Wraping: $h(x, y) = f(x + u(x, y), y + v(x, y))$
- u, v可以表示h到f的光流(optical flow)
- 令$x' = x + u(x, y), y' = y + v(x, y), \lceil x' \rceil = \lfloor x' \rfloor + 1, \lceil y' \rceil = \lfloor y' \rfloor + 1$
- $f(x', y') =$
  $(x' - \lfloor x' \rfloor)(y' - \lfloor y' \rfloor) f(\lceil x' \rceil, \lceil y' \rceil)$
  $+(x' - \lfloor x' \rfloor)(\lceil y' \rceil - y') f(\lceil x' \rceil, \lfloor y' \rfloor)$
  $+(\lceil x' \rceil - x')(y' - \lfloor y' \rfloor) f(\lfloor x' \rfloor, \lceil y' \rceil)$
  $+(\lceil x' \rceil - x')(\lceil y' \rceil - y') f(\lfloor x' \rfloor, \lfloor y' \rfloor)$

Warping

- tf可以通过构造下标索引，获取某矩阵对应索引的值

```
a = tf.random_uniform([2, 36, 24, 3])
N, H, W, C = tf.unstack(tf.shape(a))
n, h, w = tf.meshgrid(tf.range(0, N), tf.range(0, H), tf.range(0, W), indexing='ij')
n = tf.expand_dims(n, axis=-1)
h = tf.expand_dims(h, axis=-1)
w = tf.expand_dims(w, axis=-1)

b = tf.gather_nd(a, tf.concat([n, h, w], -1))
sess = tf.Session()
out1, out2 = sess.run([a, b])
assert np.all(out1 == out2)
```

Warping

```python
def tf_inverse_warp(input, flow):
    shape = tf.shape(input)
    i_H = shape[1]
    i_W = shape[2]
    shape = tf.shape(flow)
    N = shape[0]
    H = shape[1]
    W = shape[2]

    N_i = tf.range(0, N)  # [0, ..., N-1]
    W_i = tf.range(0, W)
    H_i = tf.range(0, H)

    n, h, w = tf.meshgrid(N_i, H_i, W_i, indexing='ij')
    n = tf.expand_dims(n, axis=3)  # [N, H, W, 1]
    h = tf.expand_dims(h, axis=3)
    w = tf.expand_dims(w, axis=3)

    n = tf.cast(n, tf.float32)
    h = tf.cast(h, tf.float32)
    w = tf.cast(w, tf.float32)
```

Warping

```
v_col, v_row = tf.split(flow, 2, axis=-1)  # split flow into v_row & v_col
""" calculate index """
v_r0 = tf.floor(v_row)
v_r1 = v_r0 + 1
v_c0 = tf.floor(v_col)
v_c1 = v_c0 + 1

H_ = tf.cast(i_H - 1, tf.float32)
W_ = tf.cast(i_W - 1, tf.float32)
i_r0 = tf.clip_by_value(h + v_r0, 0., H_)
i_r1 = tf.clip_by_value(h + v_r1, 0., H_)
i_c0 = tf.clip_by_value(w + v_c0, 0., W_)
i_c1 = tf.clip_by_value(w + v_c1, 0., W_)

i_r0c0 = tf.cast(tf.concat([n, i_r0, i_c0], axis=-1), tf.int32) # [N, H, W, 3]
i_r0c1 = tf.cast(tf.concat([n, i_r0, i_c1], axis=-1), tf.int32)
i_r1c0 = tf.cast(tf.concat([n, i_r1, i_c0], axis=-1), tf.int32)
i_r1c1 = tf.cast(tf.concat([n, i_r1, i_c1], axis=-1), tf.int32)
```

Warping

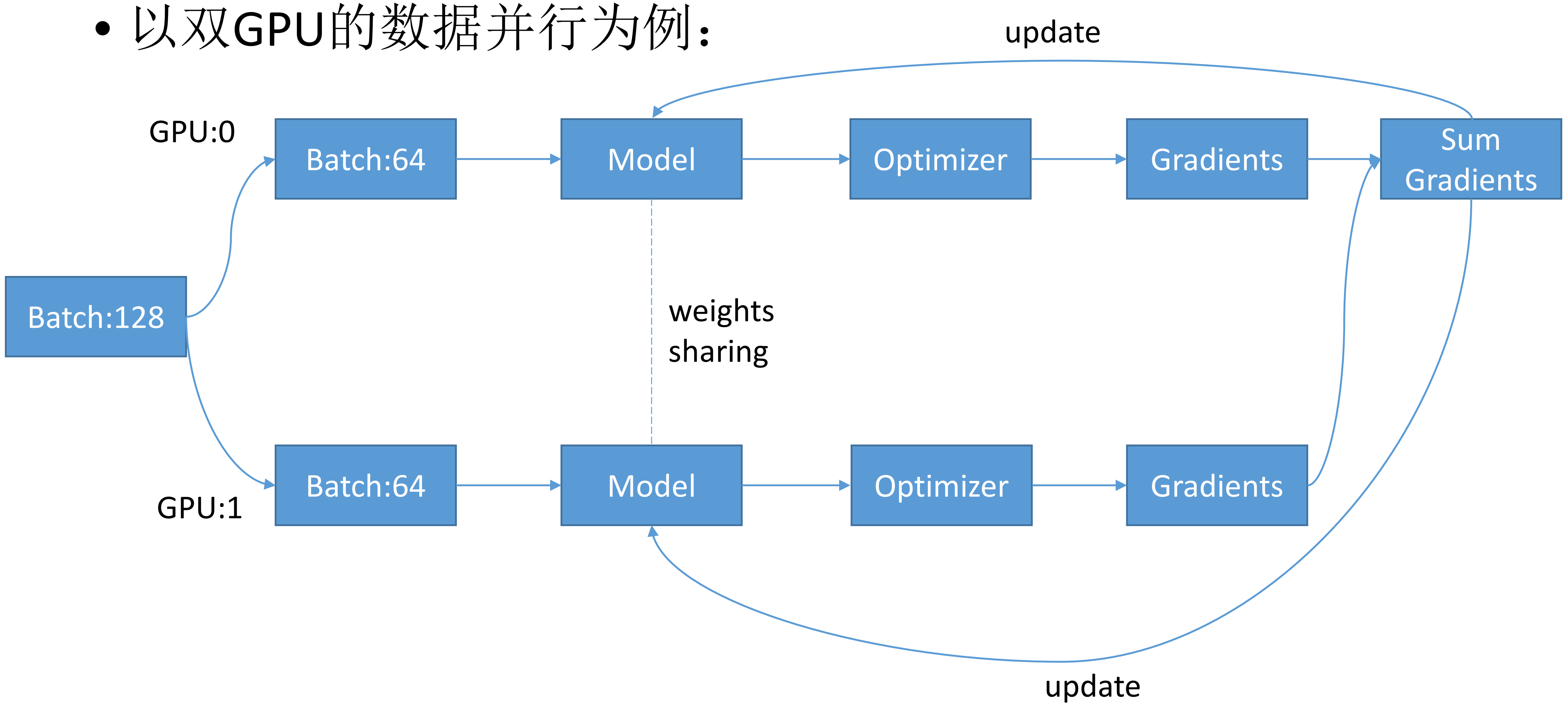
```
""" take value from index """
f00 = tf.gather_nd(input, i_r0c0) # [N, H, W, C]
f01 = tf.gather_nd(input, i_r0c1)
f10 = tf.gather_nd(input, i_r1c0)
f11 = tf.gather_nd(input, i_r1c1)

""" calculate coeff """
w00 = (v_r1 - v_row) * (v_c1 - v_col)
w01 = (v_r1 - v_row) * (v_col - v_c0)
w10 = (v_row - v_r0) * (v_c1 - v_col)
w11 = (v_row - v_r0) * (v_col - v_c0)

out = w00 * f00 + w01 * f01 + w10 * f10 + w11 * f11
return out
```

并行训练

- 为了加快大模型的训练，会采用多张GPU进行模型训练的加速
- 并行训练的模式：
  - 数据并行
  - 模型并行
  - 混合并行
- 数据并行比较常用，用于训练大的BatchSize，在一些比赛中可能可以提高性能

并行训练：数据并行

- 以双GPU的数据并行为例：



并行训练：数据并行

```
device_string = '/gpu:%d'
num_towers = 2
tower_label_losses = []
tower_gradients = []
f1s, f2s, gts = batch_data(batch_size)
tower_f1s = tf.split(f1s, num_towers)
tower_f2s = tf.split(f2s, num_towers)
tower_gts = tf.split(gts, num_towers)
optimizer = tf.train.AdamOptimizer(learning_rate)
for i in range(num_towers):
    with tf.device(device_string % i):
        m = net(tower_f1s[i], tower_f2s[i], training=True)
        loss = tf.reduce_mean(tf.abs(m.logits - tower_gts[i])) * 1e3
        tower_label_losses.append(loss)
        gradients = optimizer.compute_gradients(loss, colocate_gradients_with_ops=False)
        tower_gradients.append(gradients)

loss = tf.reduce_mean(tf.stack(tower_label_losses))
merged_gradients = combine_gradients(tower_gradients)
train_op = optimizer.apply_gradients(merged_gradients, global_step=global_step)
```

并行训练：数据并行

```python
def combine_gradients(tower_grads):
  """Calculate the combined gradient for each shared variable across all towers.

  Note that this function provides a synchronization point across all towers.

  Args:
    tower_grads: List of lists of (gradient, variable) tuples. The outer list
      is over individual gradients. The inner list is over the gradient
      calculation for each tower.
  Returns:
     List of pairs of (gradient, variable) where the gradient has been summed
     across all towers.
  """
  filtered_grads = [[x for x in grad_list if x[0] is not None] for grad_list in tower_grads]
  final_grads = []
  for i in xrange(len(filtered_grads[0])):
    grads = [filtered_grads[t][i] for t in xrange(len(filtered_grads))]
    grad = tf.stack([x[0] for x in grads], 0)
    grad = tf.reduce_sum(grad, 0)
    final_grads.append((grad, filtered_grads[0][i][1],))

  return final_grads
```