模板索引与知识梳理

陆恒 luhengok@163.com

2018 年 4 月 17 日

1 Tensorflow 基础知识

第一章包含了一些基础的 tensorflow 使用模板，代码位于 basictensor_handson，具体参看 logistic_fancy.py

1.1 Graph 与 Session

如何开始一个 session

```
with tf.Session() as sess:
    sess.run(init)
```

如何管理多张 graph

```
graph = tf.Graph()
with graph.as_default():
    with tf.Session() as sess:
        sess.run(init)
```

意思是将本来的一个图里所有的代码，包在一个 with 里面，就可以区别出多个图

1.2 值的生命周期

```
x = tf.Variable(3,name='x')
x = x+2
y = x+5
z = x*3
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    print(y.eval())
    print(z.eval())

with tf.Session() as sess:
    sess.run(init)
    y_eval,z_eval = sess.run([y,z])
    print(y_eval)
    print(z_eval)
```

1.3 placeholder

```
X = tf.placeholder(tf.float32, shape=(None, n), name="X")
```

1.4 save 与 restore

```
saver = tf.train.Saver()
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for epoch in range( n_epochs):
        if epoch % 100 == 0:
            saver.save(sess, checkpoint_path)
    saver.save(sess, final_model_path)
with tf.Session() as sess:
    saver.restore(sess,final_model_path)
```

1.5 tensorboard

```
with tf.name_scope("train"):
    loss = tf.losses.log_loss(y, y_proba, scope="loss")
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=
                                    learning_rate)
    training_op = optimizer.minimize(loss)
    loss_summary = tf.summary.scalar('log_loss', loss)
from datetime import datetime
def log_dir(prefix=""):
    now = datetime.utcnow().strftime("%Y%m%d%H%M%S")
    root_logdir = "tf_logs"
    if prefix:
        prefix += "-"
    name = prefix + "run-" + now
    return "{}/{}/".format(root_logdir, name)
logdir = log_dir("logreg")
file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for epoch in range( n_epochs):
        loss_val, summary_str = sess.run([loss, loss_summary],
                                    feed_dict={X: x, y: y})
        file_writer.add_summary(summary_str, epoch)
```

1.6 name_scope

像这样用一个 name_scope 将一个代码块包起来

```
with tf.name_scope("save"):
    saver = tf.train.Saver()
```

1.7 sharing variable 与 variable_scope

to do!!!!!!
在学习到 rnn 的时候，补上 get_variable()

1.8 模块化

将代码包在函数里面，实现模块化
to do!!!!!!!!!
如何在模块化的函数里面加入 name_scope

```
he_init = tf.contrib.layers.variance_scaling_initializer()
xavier = tf.contrib.layers.xavier_initializer()
with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1",
                                    kernel_initializer=he_init,
                                    activation=tf.nn.relu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
                                    kernel_initializer=he_init,
                              activation=tf.nn.relu)
    logits = tf.layers.dense(hidden2, n_outputs, name="outputs",
                                    kernel_initializer=he_init)

def rnn(inputs):
    hidden1 = tf.layers.dense(inputs, n_hidden1, name="hidden1",
                                    kernel_initializer=he_init,
                              activation=tf.nn.relu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
                                    kernel_initializer=he_init,
                              activation=tf.nn.relu)
    logits = tf.layers.dense(hidden2, n_outputs, name="outputs",
                                    kernel_initializer=he_init)
    return logits
```

2 Tensorflow 参数调整

第二章会包含调参的一些知识。to do!!!!!!!!!!!!!!

2.1 如何调参

TO DO

3 deeplearning 知识点和 tensorflow 相应代码

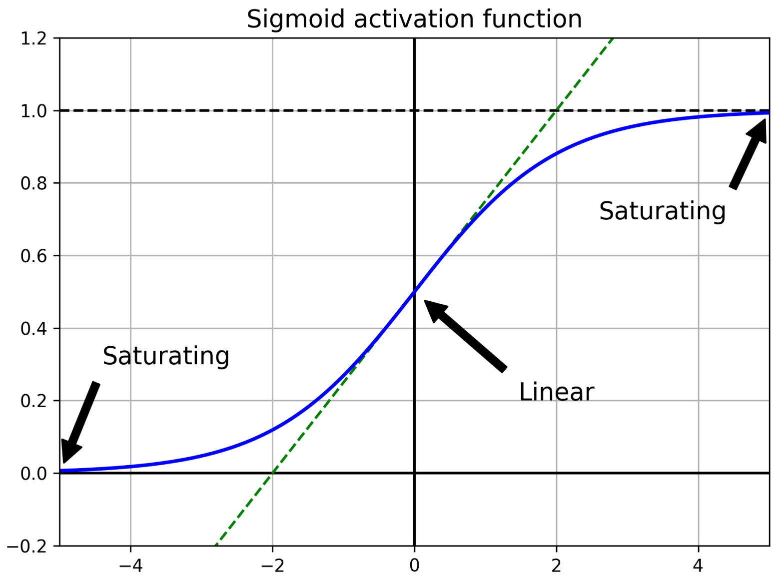
第三章包含一些 deeplearning 的知识点，来源于 Andrew Ng 和 tensorflow 代码，来源于 handson 书

3.1 梯度爆炸和梯度消失

深度学习会遭受的梯度消失和梯度爆炸问题 (rnn)
下图是 sigmoid 函数的激活函数图，可以看到在两端，sigmoid 函数趋向饱和 (saturating)
sigmoid 函数作为激活函数有两个缺点，一个是刚说的它是饱和的函数。另一个是它的中心点是 0.5，不是 0
所以去解决梯度爆炸梯度消失问题，有两个思路，一个是激活函数，另一个是参数初始化的时候选用不同的策略

3.2 参数初始化

介绍几种参数初始化的方法，来应对梯度爆炸或消失的问题
1,Xavier initialization
Xavier initialization 泽维尔 Xavier Glorot ,Yoshua Bengio.2010



Relu, Gaussion distribution: $\sigma = \sqrt{2}\sqrt{\frac{2}{n_{inputs}+n_{outputs}}}$
2,He initialization
He initialization Kaiming He et al.2015 何恺明
Relu, Gaussion distribution: $\sigma = \sqrt{2}\sqrt{\frac{1}{n_{inputs}}}$

```
he_init = tf.contrib.layers.variance_scaling_initializer()
Xavier = tf.contrib.layers.xavier_initializer()
```

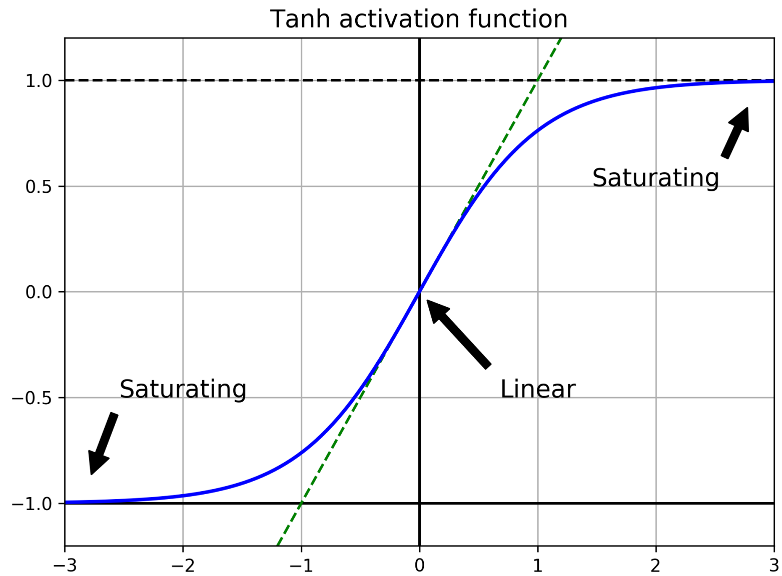
3.3 非饱和激活函数

之前经常用的函数，除了 sigmoid，还有 tanh，都属于饱和的激活函数，附上 tanh 函数的图

1,relu

$relu(z) = max(0, z)$

```
hidden = tf.layers.dense(X, n_hidden, activation=tf.nn.relu, name="
                                          hidden")
```



2,leaky_relu

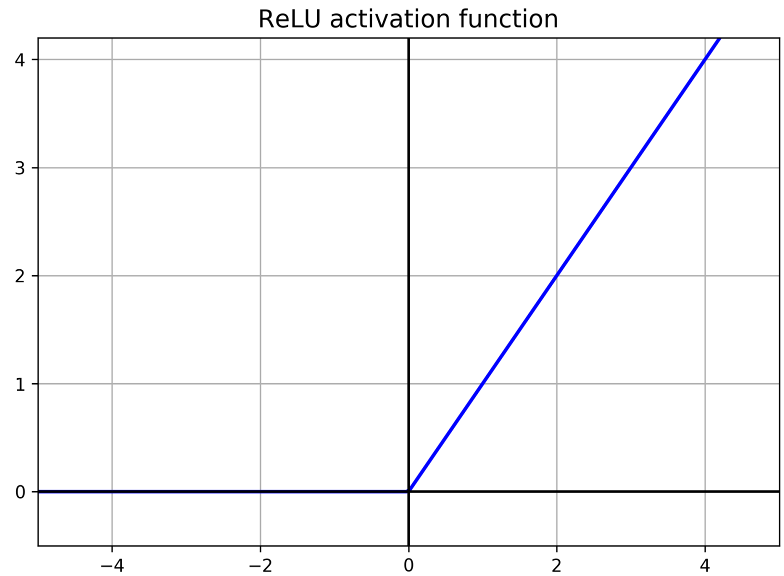
$relu(z) = max(\alpha z, z)$

```
def leaky_relu(z,name=None):
    return tf.maximum(0.01*z, z, name=name )
hidden = tf.layers.dense(X, n_hidden, activation=leaky_relu, name="
                                    hidden")
```

3,ELU

$$ELU_\alpha(z) = \begin{cases} \alpha(exp(z) - 1) & z < 0 \\ z & z > 0 \end{cases}$$

```
hidden = tf.layers.dense(X, n_hidden, activation=tf.nn.elu, name="
                                    hidden")
```



3.4 Batch Normalization

normalize inputs, 对输入值进行正则化，自然语言处理和图像一般用不到，但是其他的情况可以试试

那么在神经网络的第二层开始，可以把第二层输出（也就是第三层的输入）的值进行 normalize.

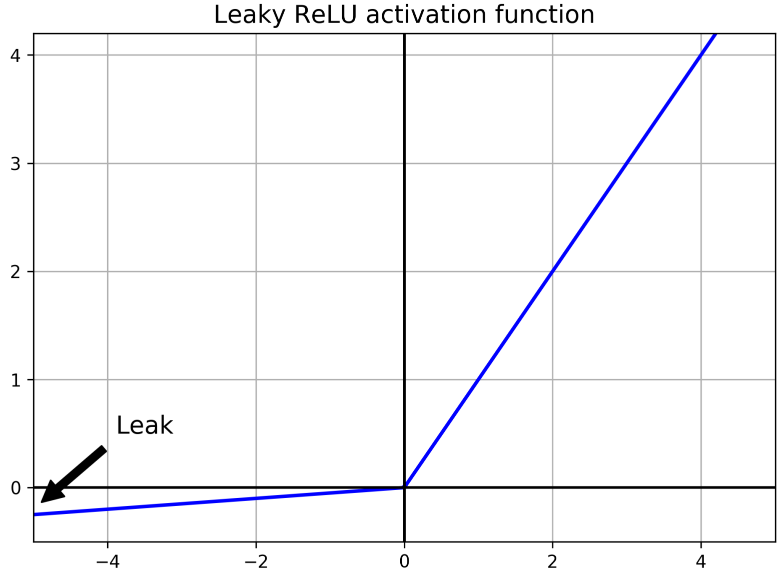
$$\mu = \frac{1}{m}\Sigma_i z^i$$
$$\sigma^2 = \frac{1}{m}\Sigma_i (z_i - \mu)^2$$
$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$
$$\tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$$

Batch norm 的反向传播

在每一个隐藏层，除了参数 w 之外，反向传播还要更新的有每一层上的 γ,β

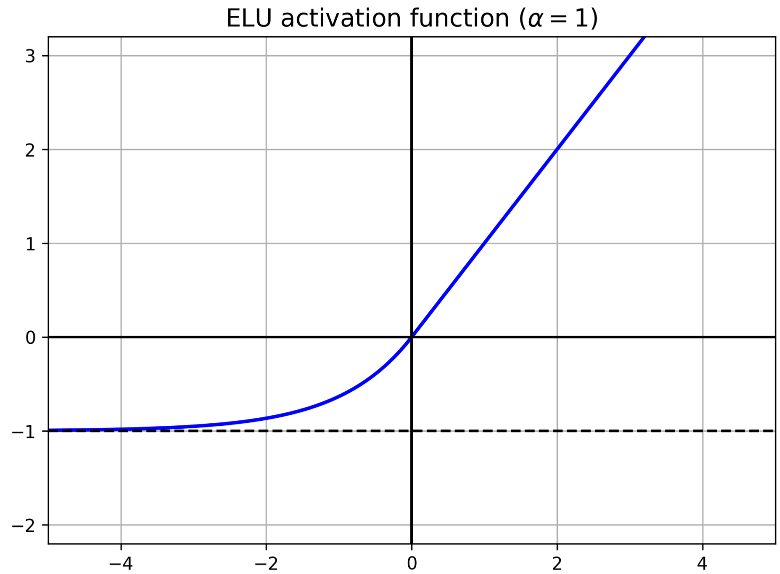
测试时候的 Batch norm

在训练的时候，在每一个 batch 上，对 l 层的 $z_{norm}^{(i)(l)}$,$\tilde{z}^{(i)(l)}$ 做指数加权平均来一直保



持追踪，最后在测试的时候，用指数加权平均后的 $z_{norm}^{(i)(l)}$,$\tilde{z}^{(i)(l)}$，还有当前已经训练好的 w,γ,β 来预测.

```
n_inputs = 28 * 28
n_hidden1 = 300
n_hidden2 = 100
n_outputs = 10
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
training = tf.placeholder_with_default(False, shape=(), name='training'
                                       )
hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1")
bn1 = tf.layers.batch_normalization(hidden1, training=training,
                                    momentum=0.9)
bn1_act = tf.nn.elu(bn1)
hidden2 = tf.layers.dense(bn1_act, n_hidden2, name="hidden2")
bn2 = tf.layers.batch_normalization(hidden2, training=training,
                                    momentum=0.9)
bn2_act = tf.nn.elu(bn2)
logits_before_bn = tf.layers.dense(bn2_act, n_outputs, name="outputs")
```




```
logits = tf.layers.batch_normalization(logits_before_bn, training=
                                       training,
                                          momentum=0.9)
```

3.5 梯度剪裁

虽然现在大家都更喜欢用 Batch Normalization, Gradient Clipping, 还是一个需要掌握的技术.

在 Tensorflow 中,

```
learning_rate = 0.01
threshold = 1.0
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
# training_op = optimizer.minimize(loss) #相当于把这一步拆开
grads_and_vars = optimizer.compute_gradients(loss)
capped_gvs = [(tf.clip_by_value(grad, -threshold, threshold), var) for
                                       grad, var in grads_and_vars]
training_op = optimizer.apply_gradients(capped_gvs)
```

3.6 迁移学习

1, 使用之前训练好的层，并缓存下来加速运行
下面一共有三段代码，第一段，建立重用 3 层，并 freeze 两层的结构；第二段，告诉 tensorflow 我要 restore 的参数；第三段，cache，代码里 cache，hidden2，就可以把 hidden1，hidden2，都可以缓存下来

```
with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.relu,
                              name="hidden1") # reused frozen
    hidden2 = tf.layers.dense(hidden1, n_hidden2, activation=tf.nn.relu
                              ,
                              name="hidden2") # reused frozen & cached
    hidden2_stop = tf.stop_gradient(hidden2)
    hidden3 = tf.layers.dense(hidden2_stop, n_hidden3, activation=tf.nn
                              .relu,
                              name="hidden3") # reused, not frozen
    hidden4 = tf.layers.dense(hidden3, n_hidden4, activation=tf.nn.relu
                              ,
                              name="hidden4") # new!
    logits = tf.layers.dense(hidden4, n_outputs, name="outputs") # new!

reuse_vars = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES,
                               scope="hidden[123]")
                               # regular expression
reuse_vars_dict = dict([(var.op.name, var) for var in reuse_vars])
restore_saver = tf.train.Saver(reuse_vars_dict) # to restore layers 1-3

with tf.Session() as sess:
    init.run()
    restore_saver.restore(sess, "./my_model_final.ckpt")
    #下面的两行代码就把之前的隐层结果缓存了下来
    h2_cache = sess.run(hidden2, feed_dict={X: mnist.train.images})
    h2_cache_test = sess.run(hidden2, feed_dict={X: mnist.test.images})
```

2, 重用其它框架的参数

3.7 优化函数

以下给出了几个最优化算法来应对传统的批梯度下降的缺点
一，指数加权平均
$v_t = \beta v_{t-1} + (1-\beta)\theta_t$，其中 $v_{t-1} \approx \frac{1}{1-\beta}$ 个之前的数据的指数平均
指数加权平均为什么叫" 指数"

$$
\begin{aligned}
&v_{100} = 0.9v_{99} + 0.1\theta_{100} \\
&v_{99} = 0.9v_{98} + 0.1\theta_{99} \\
&v_{98} = 0.9v_{97} + 0.1\theta_{98} \\
&v_{97} = 0.9v_{96} + 0.1\theta_{97} \\
&\ldots\ldots \\
&v_1 = 0.9v_0 + 0.1\theta_0 \\
&v_0 = 0 \\
&v_{100} = 0.9(0.9v_{98} + 0.1\theta_{99}) + 0.1\theta_{100} \\
&\ldots\ldots \\
&\beta^{\frac{1}{1-\beta}} \approx \frac{1}{e} \\
&\lim_{\epsilon \to 0} 1 - \epsilon^{\frac{1}{\epsilon}} = \frac{1}{e}
\end{aligned}
$$

二，偏差修正
为什么我们需要偏差修正?
$v_1 = 0.9v_0 + 0.1\theta_0$
$v_0 = 0$ 那么最开始的指数加权平均会有偏差
修正前：$v_t = \beta v_{t-1} + (1-\beta)\theta_t$
修正后：$v_t = \beta v_{t-1} + (1-\beta)\theta_t$，$v_t := \frac{v_t}{1-\beta^t}$
一般来讲，这个修正不会用，原因是在现实情况下，不会关心最开始的几个数据

三，momentun

$$
\begin{aligned}
v_{dw} &= \beta v_{dw} + (1-\beta)dw \\
v_{db} &= \beta v_{db} + (1-\beta)db \\
w &:= w - \alpha v_{dw} \\
b &:= b - \alpha v_{db}
\end{aligned}
$$

做个假设，假设 w 是最优化方向的梯度群，b 是非最优化方向的梯度群 (震荡)，那么指数加权平均可以通过平均来抵消震荡，并在最优化方向加速。$\beta = 0.9$

四，RMSprop
Root mean squared

$$
\begin{aligned}
s_{dw} &= \beta s_{dw} + (1-\beta)dw^2 \\
s_{db} &= \beta s_{db} + (1-\beta)db^2 \\
w &:= w - \alpha \frac{dw}{\sqrt{s_{dw}}} \\
b &:= b - \alpha \frac{db}{\sqrt{s_{db}}}
\end{aligned}
$$

假设 w 是最优化方向的梯度群，b 是非最优化方向的梯度群 (震荡), 那么我们想要 w 更新的更快，那就要 s_{dw} 小，dw 小，在实际操作中，防止 s_{dw} 为 0，修改为:

$$
\begin{aligned}
w &:= w - \alpha \frac{dw}{\sqrt{s_{dw} + \epsilon}} \\
\beta &= 0.999, \epsilon = 10^{-8}
\end{aligned}
$$

五，Adam

Adaptive moment estimation

$$
\begin{aligned}
&v_{dw} = \beta_1 v_{dw} + (1-\beta_1)dw \\
&v_{db} = \beta_1 v_{db} + (1-\beta_1)db \\
&s_{dw} = \beta_2 s_{dw} + (1-\beta_2)dw^2 \\
&s_{db} = \beta_2 s_{db} + (1-\beta_2)db^2 \\
&{v_{dw}}^{correct} = \frac{v_{dw}}{1-\beta_1^t} \\
&{v_{db}}^{correct} = \frac{v_{db}}{1-\beta_1^t} \\
&{s_{dw}}^{correct} = \frac{s_{dw}}{1-\beta_2^t} \\
&{s_{db}}^{correct} = \frac{s_{dw}}{1-\beta_2^t} \\
&w := w - \alpha \frac{v_{dw}^{correct}}{\sqrt{s_{dw}^{correct} + \epsilon}} \\
&b := b - \alpha \frac{v_{db}^{correct}}{\sqrt{s_{db}^{correct} + \epsilon}} \\
&\beta_1 = 0.9 \\
&\beta_2 = 0.999 \\
&\epsilon = 10^{-8}
\end{aligned}
$$

```
learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
optimizer = tf.train.MomentumOptimizer(learning_rate)
optimizer = tf.train.RMSPropOptimizer(learning_rate)
optimizer = tf.train.AdamOptimizer(learning_rate)
```

3.8 学习率衰减

注意, AdaGrad, RMSProp, Adam 不需要学习率衰减

$\alpha = \frac{1}{1+rate_{decay}*epoch}\alpha_0$

```
with tf.name_scope("train"):
    initial_learning_rate = 0.1
    decay_steps = 10000
    decay_rate = 1/10
    global_step = tf.Variable(0, trainable=False, name="global_step")
    learning_rate = tf.train.exponential_decay(initial_learning_rate,
                                    global_step,
                                           decay_steps, decay_rate)
    optimizer = tf.train.MomentumOptimizer(learning_rate, momentum=0.9)
    # 注意, AdaGrad RMSProp Adam 不需要learning_rate_decay
    training_op = optimizer.minimize(loss, global_step=global_step)
```

3.9 Early Stop

```
best_loss = np.infty
epochs_without_progress = 0
max_epochs_without_progress = 50
checkpoint_path = "./tmp/my_logreg_model.ckpt"
checkpoint_epoch_path = checkpoint_path + ".epoch"
final_model_path = "./my_logreg_model"
#中断重启机制
with tf.Session() as sess:
    if os.path.isfile(checkpoint_epoch_path):
        # if the checkpoint file exists, restore the model and load the
                                                    epoch
        # number
        with open(checkpoint_epoch_path, "rb") as f:
            start_epoch = int(f.read())
```

```
        print("Training was interrupted. Continuing at epoch",
                                                    start_epoch)
        saver.restore(sess, checkpoint_path)
    else:
        start_epoch = 0
        sess.run(init)
    init.run()
    for epoch in range(n_epochs):
        ......
        #early stop
        if epoch % 5 == 0:
            saver.save(sess, checkpoint_path)
            with open(checkpoint_epoch_path, "wb") as f:
                f.write(b"%d" % (epoch + 1))
            if loss_dev < best_loss:
                saver.save(sess, final_model_path)
                best_loss = loss_dev
            else:
                epochs_without_progress += 5
                if epochs_without_progress
                    > max_epochs_without_progress:
                    print("Early stopping")
                    break

    os.remove(checkpoint_epoch_path)
```

3.10 L1,L2 正则

```
from functools import partial
import tensorflow.contrib as tc
scale = 0.001
training = tf.placeholder_with_default(False, shape=(), name='training'
                                    )
my_dense_layer1 = partial(
    tf.layers.dense, activation=tf.nn.relu,
```

```
    kernel_regularizer=tf.contrib.layers.l1_regularizer(scale))
my_dense_layer2 = partial(
    tf.layers.dense, activation=tf.nn.relu,
    kernel_regularizer=tc.layers.l2_regularizer(scale))

dropout_rate = 0.5
with tf.name_scope("dnn"):
    hidden1 = my_dense_layer1(X, n_hidden1, name="hidden1",
                               activation=tf.nn.relu)
    hidden2 = my_dense_layer1(hidden1, n_hidden2, name="hidden2",
                               activation=tf.nn.relu)
    hidden2_drop = tf.layers.dropout(hidden2, dropout_rate, training=
                                        training)
    logits = my_dense_layer2(hidden2_drop, n_outputs, name="outputs")

with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
                                        logits=logits)
    #difference
    base_loss = tf.reduce_mean(xentropy, name="avg_xentropy")
    reg_losses = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
    loss = tf.add_n([base_loss] + reg_losses, name="loss")
```

3.11 Dropout

上述的代码里已经有了 dropout

```
training = tf.placeholder_with_default(False, shape=(), name='training'
                                    )
dropout_rate = 0.5
...
with tf.name_scope("dnn"):
    hidden2 = my_dense_layer1(hidden1, n_hidden2, name="hidden2",
                               activation=tf.nn.relu)
```

```
    hidden2_drop = tf.layers.dropout(hidden2, dropout_rate, training=
                                    training)
...

    sess.run(training_op, feed_dict={training:True,X: X_batch, y:
                                    y_batch})
```

3.12 Max-Norm 正则

先定义一个 max-Norm 函数，然后在定义的层里面 kernel_regularizer 加上
这边需要和梯度剪裁对比

```
def max_norm_regularizer(threshold, axes=1, name="max_norm",
                         collection="max_norm"):
    def max_norm(weights):
        clipped = tf.clip_by_norm(weights, clip_norm=threshold, axes=
                                  axes)
        clip_weights = tf.assign(weights, clipped, name=name)
        tf.add_to_collection(collection, clip_weights)
        return None # there is no regularization loss term
    return max_norm

max_norm_reg = max_norm_regularizer(threshold=1.0)

with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1",
                              activation=tf.nn.relu,kernel_regularizer=
                                                                    max_norm_reg
                                                                    )
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
                              activation=tf.nn.relu,kernel_regularizer=
                                                                    max_norm_reg
                                                                    )
    logits = tf.layers.dense(hidden2, n_outputs, name="outputs")
```